PNSPO

FinsGateway EventMemory

Programmer's Manual

Version 2 8/17/1998

OMRON Corporation

Contents

1 INTRODUCTION....1

2 SETUP....3

2.1 Operating Environment....3

3 EVENTMEMORY....4

3.1 EventMemory Structure....4
3.2 Interface and Data Structure....7

4 PROGRAMMING....11

4.1 Using the EventMemory....11
4.2 Reading or Writing the EventMemory Data....11
4.3 Sending or Receiving Events of the EventMemory....11
4.4 Setting or Clearing Event conditions....11
4.5 Receiving Events by Message-driven Type....12
4.6 Terminating the EventMemory....12

5 ACCESS METHODS....13

5.1 Standard Access Methods....13
5.2 Creating an Independent Access Method....13

6 ERROR SYSTEMS....23

6.1 Error Codes....23

7 API REFERENCE....25

7.1 Summary....25
7.2 Em_requestVersion Function....27
7.3 Em_getVersion Function....28
7.4 Em_openMemory Function....29
7.5 Em_readMemory Function....30
7.6 Em_writeMemory Function....31
7.7 Em_readMemoryEx Function....31
7.8 Em_writeMemoryEx Function....32
7.9 Em_closeMemory Function....34
7.10 Em_openEvent Function....35
7.11 Em_sendEvent Function....36
7.12 Em_receiveEvent Function....37
7.13 Em_closeEvent Function....38
7.14 Em_setCondition Function....39
7.15 Em_setWideCondition Function....40
7.16 Em_clearCondition Function....41
7.17 Em_judgeCondition Function....42
7.18 Em_ setMessageOnArrival Function....43
7.19 Em_ setThreadMessageOnArrival Function....44
7.20 Em_ clearMessageOnArrival Function....45
7.21 Em_getCondition Function....46
7.22 Em_getWideCondition Function....47
7.23 Em_isWideConditionId Function....48
7.24 Em_getConditionList Function....49
7.25 Em_getWideConditionList Function....50
7.26 Em_getLostEventLogs Function....51
7.27 Em_getSystemInfo Function....52
7.28 Em_getMemoryInfo Function....53

7.29 Em_getMemoryPortUsage Function 55
7.30 Em_getEventPortUsage Function 56
7.31 Em_getEventHandle Function 57
7.32 Em_getMutexHandle Function 58
7.33 Em_getBaseAddress Function 59
7.34 Em_swapBytes Function 60
7.35 Em_getLastErrorMessage Function 61
7.36 Em_flushFile Function 62
7.37 Em_getBytesBuffer Function 63

8 DATA STRUCTURE 64

8.1 The EventMemory Address 64
8.2 Event conditions 65
8.3 Acquired Information 67
8.4 Structure Exclusive to the Access Method 68

Revision History

Revision code	Date	Revised content
1.00	August 1998	Original production
2.00	July 2000	Added FinsGateway Version 3 functions

1 Introduction

This manual describes the API for using the EventMemory, which can be shared among two or more processes on the computer where FinsGateway is running. The EventMemory is provided as a component of FinsGateway.

Memory I/O Communication API

The EventMemory offers memory that can be shared among applications. The communication units making up FinsGateway have functions to communicate through shared memory such as DM or CIO. This is how the EventMemory API serves as an API for memory communication between applications. An actual example is the data link of SYSMAC LINK.

Remote Memory

The past EventMemory was a mechanism for memory management that used the OS shared memory. In the expanded FinsGateway Version 3 EventMemory, the memory areas used are not only the shared memory, but it can also access the memory of the PLCs and other network devices that can be accessed by FINS communications. This enables access to the network device memory areas in the same manner as access to the past EventMemory.

Shared Memory of FINS Communication

When CPUs with the server functions of FINS commands/responses share EventMemory, it is used as shared memory on a personal computer during FINS communication. Shared memory areas such as DM or CIO are available for variable reading/writing by FINS.

The major features of EventMemory are as follows:

- It provides a platform upon which to perform communication through memory I/O on a personal computer.
- It provides the functions of an application API such as memory read/write and the function to report memory data updating as an event.
- It provides inter-process communication by event.
- It assures data integrity by exclusive control of access to shared memory from several applications.
- Without any consideration for FINS communications, the memory areas of network devices can be accessed as if it were in the local machine memory.

2 Setup

2.1 Operating Environment

Files Required for Application Development

DLLs	EvtMem32.dll, EmMisc32.dll
Import library	EvtMem32.lib
Include files	EvtMem.h, EmError.h, EmResrc.h, and FgwAccessMethod.h EmError.h, EmResrc.h, FgwAccessMethod.h are included from EvtMem.h.

Files Required for Access Method Development

DLL	EvtMem32.dll, EmMisc32.dll
Import library	EvtMem32.lib
Include file	To implement an independent access method, include private¥EvtMemPrivate.h.

3 EventMemory

3.1 EventMemory Structure

3.1.1 Shared Memory

To share memory with other applications through the EventMemory, the data can be shared by two or more applications which open a shared memory port of the same name. The applications can read and write data, and set and clear event conditions for the shared memory for which a memory port is opened.

A name to be specified for a memory port to be opened is not case-sensitive, so DM, dm, Dm, and dM would all specify the memory port for the same area of shared memory.

Data in shared memory that has a holding file is not lost when the shared memory is unloaded. When shared memory of a specified name is first attached, it is loaded. When it is last detached, it is unloaded.

The size of shared memory defaults to 32,768 (= 0x8000) words.

3.1.2 Remote Memory

To use the EventMemory to access a device memory, define a new EventMemory as a remote memory. To define a new EventMemory, use the FinsGateway Configuration.

Specify the memory name defined as a remote memory. The same as with shared memory, it is necessary to open a memory port. After this is complete, it is possible to use the memory read/write API to access the device memory as EventMemory.

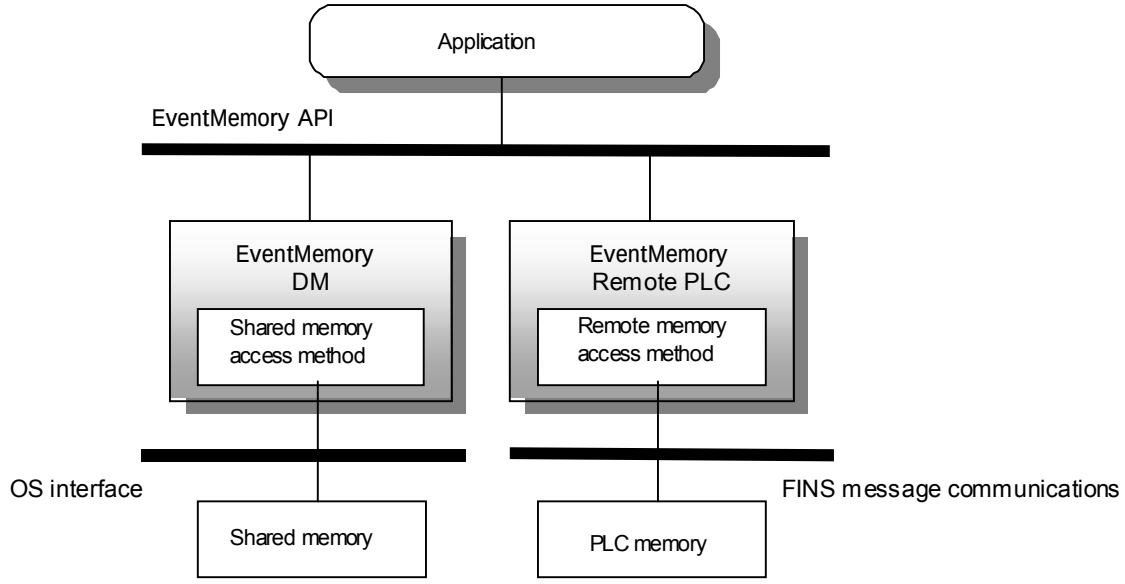
3.1.3 Access Method

EventMemory can access the local computer shared memory or network device memory using the same API. To provide this functionality, an access method module has been implemented.

The EventMemory API, and device access module are separate, and the access methods are used to read/write device data. This makes it possible to absorb the data management differences for each device in the access methods. The various device data can then be handled in the same manner by the EventMemory API.

An access method is provided for each kind of device. When the EventMemory is open, select the appropriate access method.

The shared memory access methods use the OS shared memory service functions. The remote memory access methods use the FinsGateway FINS message service function to access devices.



3.1.4 Events

The EventMemory allows events to be received by opening event ports for an application. Applications opening an event port can send events to an event port opened by other applications. If the destination event port is not open, the send operation fails. Like memory port names, event port names can be specified without making the distinction between uppercase and lowercase letters.

Event ports are different from memory ports in that two or more ports of the same name cannot be opened.

Two or more events are held in an event port used by applications in a FIFO queue. Events held in an event port are all lost when the port is closed.

Event arrival can be reported to applications in a Windows message.

3.1.5 Event Conditions

EventMemory allows events to be sent for the updating of shared memory data. When shared memory data is updated, if the data satisfies specified conditions, an event is automatically sent to a specified event port.

To set event conditions, specify the shared memory area to be allocated and the event conditions. Two or more sets of conditions can be set for an identical shared memory area.

Event conditions include normal and wide-area event conditions.

- Normal event conditions
  These conditions are set for shared memory areas addressed as bytes, words, and double words and various conditions can be set as event conditions.
  Normal event conditions consist of the following information:
- Operation on memory
  Specify an operation that returns a true/false result when executed during shared memory update. For normal event conditions, the operations shown in Table 3-1 Operations on normal event conditions can be used. For a true/false operation, up to two constants can be specified for a comparison operation. Set the necessary comparison constants, depending on the type of operation.
- Determining whether to send an event depending on the transition of the true/false results
  Specify whether an event is to be sent or not for all of the four true/false transition patterns (false -> false, false -> true, true -> true, and true -> false) derived from the operation results from previous memory updating and those upon current memory updating.
- Previous operation result
  When setting event conditions, specify the previous operation results required to determine whether to send an event. For the acquisition of event conditions, the most recent true/false operation results are provided.
- Event information to be sent
  Specify the destination event port, event ID, and the shared memory area (memory name and address) whose data is to be sent. An event ID is defined in each application and is used to identify an event.
- Volatility/non-volatility
  Volatility: Unloading shared memory causes the specified conditions to be lost.
  Non-volatility: The specified conditions are maintained even after shared memory is unloaded.
  A nonvolatile condition can be specified only for shared memory having a normal-condition holding file.

Table 3-1 Operations on normal event conditions

Operation	Description	Comparison constant 1	Comparison constant 2
AND	At least one of the bits corresponding to those of comparison constant 1 is ON.	■	N/A
ANDEQ	Bits corresponding to those of comparison constant 1 are all ON.	■	N/A
AlwaysTRUE	Always true.	N/A	N/A
NOP	The value is not zero.	N/A	N/A
EQ	The value is equal to comparison constant 1.	■	N/A
LT	The value is less than comparison constant 1.	■	N/A
LE	The value is less than or equal to comparison constant 1.	■	N/A
GT	The value is greater than comparison constant 1.	■	N/A
GE	The value is greater than or equal to comparison constant 1.	■	N/A
GELE	The value is greater than or equal to comparison constant 1 and smaller than or equal to comparison constant 2.	■	■
GTLT	The value is between comparison constants 1 and 2.	■	■
GELT	The value is greater than or equal to comparison constant 1 and less than comparison constant 2.	■	■
GTLE	The value is greater than comparison constant 1 and less than or equal to comparison constant 2.	■	■
PrevAND	At least one of the bits corresponding to the previous value is ON.	□	N/A
PrevANDEQ	The bits corresponding to the previous value are all ON.	□	N/A
PrevEQ	Equal to the previous value.	□	N/A
PrevLT	Less than the previous value.	□	N/A
PrevLE	Less than or equal to the previous value.	□	N/A

PrevGT	Greater than the previous value.	□	N/A
PrevGE	Greater than or equal to previous value.	□	N/A

Comparison constants are the values required for various operations, and are specified when making the condition settings.

■: Required for condition settings

N/A: Not required for condition settings

□: Required only when the previous-operation results are set, depending on the memory value specified in the condition settings

Note:

Even if volatile conditions are specified, those conditions are not necessarily lost upon termination of an application under which those conditions were specified.

For example, as long as the CPU is operating, an event-occurrence condition set as DM or CIO remains valid unless the condition is cleared, even after the application under which it was specified is terminated.

In this case, if an identical condition is set again by another application, it follows that two or more identical conditions exist and two or more identical events are sent when the event-occurrence condition is satisfied.

3.1.6 Wide-Area Event conditions

These conditions can be set also for shared memory areas beyond those with addresses specified with double words. As event conditions, the area to which to write data and the occurrence of data updating can be set.

Wide-area event conditions consist of the following information:

- Determining whether to send an event
  Specify whether or not to send an event when data is written to the memory area for which conditions are set or when data in the memory area changes.
- Event information to be sent (likewise normal condition)
  Specify the destination event port, event ID, and the shared memory area (memory name and address) whose data are to be sent. An event ID is defined in each application and is used to identify an event.
- Volatility/non-volatility (likewise normal condition)
  Volatility: Unloading shared memory causes the specified conditions to be lost.
  Non-volatility: The specified conditions are maintained even after shared memory is unloaded.
  A nonvolatile condition can be specified only for shared memory having a wide-area condition holding file.

3.2 Interface and Data Structure

The interface for the EventMemory is compatible with the memory interface for Omron's programmable controller.

Memory configuration

The EventMemory is a string of word data consisting of 16 bits per word. The data structure of one word is as shown below:

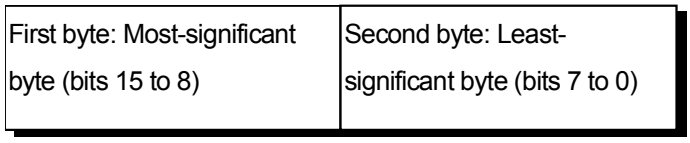


Figure 3-1: Word configuration

Double-word data consists of two contiguous words, as shown below:

First word: Most-significant word	Second word: Least-significant word

Figure 3-2 Double-word configuration

Note:

With personal computers using Intel processors, a one-word data item is referred to as a word value with the first byte as the least-significant byte and the second byte as the most-significant byte, unlike the EventMemory interface. Therefore, a data string read from (or written to) the EventMemory, if referred to as word values without modification, would have different values from those in the EventMemory.

Like the EventMemory interface, a double-word data item is referred to as a value with the first word as the least-significant word and the second word as the most-significant word. However, as with word values, the order of most-significant and least-significant bytes in each word is reversed. Therefore, an item of data string read from (or written to) the EventMemory, if referred to as a double-word value without modification, would have a different value from that in the EventMemory.

To refer to a data string read from (written to) the EventMemory as word values or double-word values, swap the most-significant and least-significant bytes in each word before referring to it. This is also true for the values to be compared with data in shared memory when setting normal event conditions.

Byte strings in the EventMemory have the same arrangement as data read or written.

Addressing

In the EventMemory, to specify the shared memory data area for which read, write, and condition settings are to be performed, specify the following four items:

- Offset (unit: word)
  Specify in words the starting offset of address for which read, write, and condition settings are to be performed. If the data type is double word in the condition settings, the offset must be an even value.
- Element unit
  Specify the element unit (bit, byte, word, or double word) of data for which read, write, and condition settings are to be made. In wide-area conditions settings, only words can be specified as the element unit. In the setting of event conditions, absence of data (EM_NO_DATA) can be specified as the conditions for a shared memory area to which to send data.
- Bit/byte position
  If the data type is bit or byte, specify the starting bit or byte position in the first offset in which a read, write, or condition setting is made.
  Bit: Specify 0 to 15 bits.
  Byte: Specify the most-significant (EM_BYTE_HIGH) or least-significant (EM_BYTE_LOW) byte.
  The bit/byte position specification is valid only in the following cases:
  A read/write setting when the data type is bit or byte
  A condition setting when the data type is byte
- Number of data
  Specify the number of data elements for which a read, write, or condition setting is made. For the setting of normal event conditions, be sure to specify 1 as the number of data.

Examples of Addressing

- Double-word data from offsets 1000 to 1031

Specify 1000 as the offset, double word as the data type, and 16 as the number of data.

- Word data from offsets 1000 to 1031
  Specify 1000 as the offset, word as the data type, and 32 as the number of data.
- Data from the least-significant byte of offset 500 to the most-significant byte of offset 511
  Specify 500 as the offset, byte as the data type, EM_BYTE_LOW(=1) as the byte position, and 22 as the number of data.
- Data from bit 12 of offset 32 to bit 3 of offset 33
  Specify 32 as the offset, bit as the data type, 12 as the bit position, and 8 as the number of data.

3.2.1 Sending or Receiving Data with an Event

Events of the EventMemory are an inter-process communication that enables sending and receiving between applications.

The data configuration of events sent and received through the EventMemory is as shown below.

- Event ID

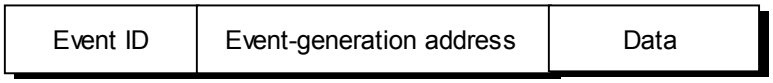


Figure 3-3: Configuration of event data

This is an integer value defined between applications to send and receive events, or between applications to set event conditions or receive events. An event ID is used to give an event a meaning.

- Event address
  When events are automatically reported to applications according to event conditions, an event address is used to indicate the name and address of the shared memory for which the conditions are set. When an event is sent directly from an application, an event address can also be used to pass data in shared memory related to the send event to a receiving application.
- Data
  When events are automatically reported to applications according to event conditions, specification can be made to send the data in a specified contiguous area in shared memory when an event occurs. Specified data can be included in an event for sending or receiving between applications. Up to 2,016 bytes of data can be sent or received for one event.

4 Programming

4.1 Using the EventMemory

Specifying the Operating Version

First, specify the operating version of the EventMemory. Specification of the operating version is made to ensure full compatibility with previous versions of the EventMemory without recompiling for each upgrade.

To specify a version, use the Em_requestVersion function. To specify the operating version of a new release, the macro EM_STARTUP may be used.

4.2 Reading or Writing the EventMemory Data

Opening a Memory Port for Reading or Writing

First, use the Em_openMemory function to open a memory port for reading or writing. Then, perform reading or writing for the opened memory using the handle returned from the Em_openMemory function.

- Read data from memory using the Em_readMemory function.
- Write data to memory using the Em_writeMemory function.

Upon termination of an application, be sure to close the opened memory port using the Em_closeMemory function.

4.3 Sending or Receiving Events of the EventMemory

Opening an Event Port

First, open an event port using the Em_openEvent function so that events can be sent or received. Then, send and receive events using the handle returned from the Em_openEvent function.

- Receive events using the Em_receiveEvent function.
- Send events using the Em_sendEvent function.

Upon termination of an application, be sure to close the opened event port using the Em_closeEvent function.

4.4 Setting or Clearing Event conditions

Opening a Memory Port to Set Event conditions

First, open a memory port to set event conditions using the Em_openMemory function. Then, use the Em_setCondition or Em_setWideCondition function to set event conditions for the opened memory.

- Set normal event conditions using the Em_setCondition function.
- Set wide-area event conditions using the Em_setWideCondition function.
- Clear normal or wide-area event conditions using the Em_clearCondition function.

Upon termination of an application, be sure to close the opened memory port using the Em_closeMemory function.

To receive events sent according to event conditions, open the event port of the event destination specified in the event conditions.

4.5 Receiving Events by Message-driven Type

Opening an Event Port to Set Messages to be Posted

First, open an event port for receiving events. Then, make the settings for posting messages in the window/thread of the application when an event arrives at the event port.

This setting enables applications to receive an event using the Em_receiveEvent function without being blocked after receiving a message posted in the window/thread when an event arrives at the opened event port.

4.6 Terminating the EventMemory

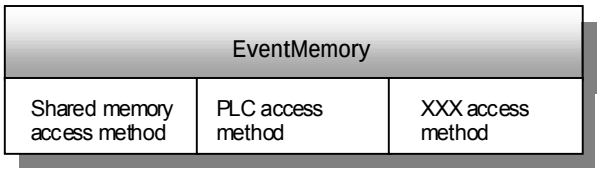
Closing an Open Port

Close all open memory and event ports, then terminate the application. Note that if an event port is not closed, a new event port of the same name cannot be opened.

Normally, if an application is terminated without the memory and event ports being closed, the EventMemory detects that the process is detached, and closes the open ports. However, note that detachment of the process may not be detected, such as when an application is terminated using the debugger.

5 Access Methods

The EventMemory can use the same API to access the local computer shared memory or a network device memory. To provide this functionality, an access method module has been implemented. The access methods are implemented as DLLs. The EventMemory dynamically loads the access method DLL. It is possible to create an independent access method, and integrate it into the EventMemory.



5.1 Standard Access Methods

FinsGateway Version 3 has both the shared memory and remote memory access methods as standard.

5.1.1 Shared Memory Access Method

The AmShmem shared memory access method provides access to the local machine shared memory. At the same time as writing data to the EventMemory, it also has the additional functionality to record the time and other user-specified data to be retained in the shared memory. The additional data can be read during data read, or the area where it was written can be specified directly, and read. When ever data is written to the EventMemory, the previous data and additional data, can be retained in a data history. To use the history, it is necessary to use Em_readMemoryEx() and Em_writeMemoryEx().

5.1.2 Remote Memory Access Method

The AmFinsRemote remote memory access method provides access to the memory of all devices that can communicate by FINS messages. When executing a memory read, the FINS message (0x0101: Data read command) is used to read the device data. When executing a memory write, the FINS message (0x0102: Data write command) is used to write data to the device.

5.2 Creating an Independent Access Method

5.2.1 Access Method Implementation

Create the access method as another DLL. Implement the one and only export function, init(), and the process functions that support the EventMemory API. The relationship with the EventMemory API is as shown below:

EventMemory API	Access Method Functionality
Em_openMemory()	Details the required processing at open.
Em_closeMemory()	Details the required processing at close.
Em_readMemory()	Details the data read processing.
Em_writeMemory()	Details the data write processing.
Em_readMemoryEx()	Details the data read processing that is exclusive to the access method. Implement if required.
Em_writeMemoryEx()	Details the data write processing that is exclusive to the access method. Implement if required.

The functions implemented into the above access method can be freely named. Not everything necessarily has to be implemented. For example, when accessing a read-only device, write processing functions are not needed.

The following is an example of the function processing for an access method designed to access a file on the hard disk:

Function	Access Method Functionality
AmFile_open()	Opens the file.
AmFile_close()	Closes the file.
AmFile_read()	Reads file data.
AmFile_write()	Writes file data.
NULL	Em_readMemoryEx() is not supported yet.
NULL	Em_writeMemoryEx() is not supported yet.

5.2.2 Implementing the init Function

Implement the one and only export function, init().The EventMemory calls the init function before using the access method:

```
DllExport BOOL WINAPI init(
    PFgwEmMemoryAccessMethodRec*    methodRec
    )
{
    static tFgwEmMemoryAccessMethodRec localMethodRec;

    memset(&localMethodRec, 0x00,
sizeof(localMethodRec));
    _tcscpy( localMethodRec.name, _T("AmFile"));
    localMethodRec.accessMethods.open    = AmFile_open;
    localMethodRec.accessMethods.close   =
AmFile_close;
    localMethodRec.accessMethods.read    = AmFile_read;
    localMethodRec.accessMethods.write   =
AmFile_write;
    localMethodRec.accessMethods.readEx  = NULL;
    localMethodRec.accessMethods.writeEx = NULL;
    localMethodRec.accessMethods.judge   = NULL;
    *methodRec = &localMethodRec;

    return TRUE;
}
```

The PfgwEmMemoryAccessMethodRec pointer is passed as a parameter. Declare the actual status of the TfgwEmMemoryAccessMethodRec structure inside the access method as static variable, and substitute that pointer for methodRec.

The init function sets the access method name to the accessMethods structure, name member, as a character string. Next, it sets the pointer for each process function. For the unsupported functions, it sets NULL. This enables the EventMemory to know the address for the process function for each API.

5.2.3 Implementing the open Process Function

If the application has Em_openMemory() implemented, the EventMemory calls the access method open function:

```
// OPEN METHOD
BOOL AmFile_open(
    pFgwEmMemoryHandle hmem,
    LPCTSTR memoryName
    )
{
    pSpecData   specData;

    specData = (pSpecData)malloc(sizeof(tSpecData));
    if (specData == NULL) {
        return FALSE;
    }

    // file open
    if (!getFileName(memoryName, specData)) {
        goto lError;
    }
    specData->file = fopen( specData->fileName, "r+b" );
    if ( specData->file == NULL )
    {
        goto lError;
    }

    hmem->accessMethodHandleData = specData;
    return TRUE;

lError:
    if (specData != NULL) {
        free(specData);
    }
    hmem->accessMethodHandleData = NULL;
    return FALSE;
}
```

The pSpecData structure is a structure to retain the data exclusive to the access method. The access method developer defines it. Substitute the allocated pSpecData structure pointer for the tFgwEmMemoryHandle structure, accessMethodHandleData member variable.

In the above example, it opens the file, and retains the pointer of the opened file:

```
typedef struct {
    TCHAR   fileName[MAX_PATH];
    FILE*   file;
} tSpecData, *pSpecData;
```

5.2.4 Implementing the close Process Function

If the application has Em_closeMemory() implemented, the EventMemory calls the access method close function:

```
// CLOSE METHOD
BOOL AmFile_close(
    pFgwEmMemoryHandle hmem
    )
{
    pSpecData   specData = hmem->accessMethodHandleData;

    if (specData == NULL) {
        return TRUE;
    }

    if (specData->file != NULL) {
        fclose(specData->file);
    }
    free(specData);
    hmem->accessMethodHandleData = NULL;

    return TRUE;
}
```

In the above example, the file descriptor is obtained from the structure set by the open function to hold the data exclusive to the access method, and the file is closed.

The specData memory area secured by the open function is released, and NULL is substituted for accessMethodHandleData.

5.2.5 Implementing the read Process Function

If the application has Em_readMemory() implemented, EventMemory calls the access method read function:

```
// READ METHOD
BOOL AmFile_read(
    pFgwEmMemoryHandle hmem,
    pEM_ADDRESS psAddr,
    PVOID       lpBuffer,
    DWORD       dwNumberOfBytesBuf
    )
{
    longnFileOffset;
    size_t  nNumberOfBytesToRead;
    size_t  nNumberOfBytesRead;
    pSpecData   specData = hmem->accessMethodHandleData;

    if (psAddr->byTypeOfFactor == EM_BYTE_TYPE) {
        nNumberOfBytesToRead =
psAddr->dwNumberOfFactors;
        nFileOffset =   psAddr->dwWordOffset +
                  psAddr->byLocateOnWord;
    } else {
        return FALSE;
    }
    if (nNumberOfBytesToRead == 0 ) {
        return FALSE;
    }

    // move to offset
    if ( fseek(specData->file, nFileOffset, SEEK_SET) )
    {
        return FALSE;
    }

    // read from file
```

```
    memset(lpBuffer, 0x00, dwNumberOfBytesBuf);
    nNumberOfBytesRead = fread( lpBuffer, 1,
nNumberOfBytesToRead,
        specData->file);
    return TRUE;
}
```

The structure indicating the read data address, the pointer to the buffer storing the read data, and the buffer size are passed as parameters.

In the above example, the only data type supported is BYTE. It seeks the file offset and read data size, and reads the file data into the buffer indicated by lpBuffer.

5.2.6 Implementing the write Process Function

If the application has Em_writeMemory() implemented, EventMemory calls the access method write function:

```
// WRITE METHOD
BOOL File_write(
    pFgwEmMemoryHandle hmem,
    pEM_ADDRESS psAddr,
    PVOID       pvData,
    DWORD       dwNumberOfBytesData)
{
    longnFileOffset;
    size_t  nNumberOfBytesToWrite;
    size_t  nNumberOfBytesWrite;
    pSpecData   specData = hmem->accessMethodHandleData;

    if (psAddr->byTypeOfFactor == EM_BYTE_TYPE)
    {
        nNumberOfBytesToWrite =
psAddr->dwNumberOfFactors;
        nFileOffset = psAddr->dwWordOffset +
                  psAddr->byLocateOnWord;
    } else {
        return FALSE;
    }
    if (nNumberOfBytesToWrite > dwNumberOfBytesData ) {
        return FALSE;
    }

    // move to offset
    if ( fseek(specData->file, nFileOffset, SEEK_SET) )
    {
        return FALSE;
    }

    // write to file
    nNumberOfBytesWrite = fwrite( pvData, 1,
nNumberOfBytesToWrite,
        specData->file);
    if (nNumberOfBytesWrite < nNumberOfBytesToWrite) {
        return FALSE;
    }
    fflush(specData->file);
    return TRUE;
```

```
}
```

The structure indicating the write data address, the pointer to the buffer storing the write data, and the buffer size are passed as parameters.

In the above example, the only data type supported is BYTE. It seeks the file offset and write data size, and writes the pvData data into the file.

5.2.7 Implementing the readEx Process Function

If the application has Em_readMemoryEx() implemented, EventMemory calls the access method readEx function. The readEx function is used when supporting read processing exclusive to the access method:

```
// CLOSE METHOD
BOOL File_readEx(
    pFgwEmMemoryHandle hmem,
    pEM_ADDRESS psAddr,
    PVOID       pvBuf,
    DWORD       dwNumberOfBytesBuf,
    PVOID       pvMethodSpec
    )
{
    pSpecData   specData = hmem->accessMethodHandleData;

    // data read
    if (!File_read(hmem, psAddr, pvBuf,
dwNumberOfBytesBuf)) {
        return FALSE;
    }

    // exclusive processing
    if (pvMethodSpec != NULL) {
    {
        …
    }

    return TRUE;
}
```

The structure indicating the read data address, the pointer to the buffer storing the read data, the buffer size, and the data exclusive to the access method are passed as parameters.

To support the readEx function, define a data structure exclusive to the access method. The application sets the necessary data to this structure, and executes Em_readMemoryEx() with the data structure pointer as pvMethodSpec. If pvMethodSpec is NULL, the same processing as the normal read function is performed.

5.2.8 Implementing the writeEx Process Function

If the application has Em_writeMemoryEx() implemented, EventMemory calls the access method writeEx function. The writeEx function is used when supporting write processing exclusive to the access method:

```
// CLOSE METHOD
BOOL File_writeEx (
    pFgwEmMemoryHandle hmem,
    pEM_ADDRESS psAddr,
    PVOID       pvData,
    DWORD       dwNumberOfBytesData,
    PVOID       pvMethodSpec
    )
{
    pSpecData   specData = hmem->accessMethodHandleData;

    // Write the data
    if (!File_write(hmem, psAddr, pvData,
dwNumberOfBytesData)) {
        return FALSE;
    }

    // Peculiar processing
    if (pvMethodSpec != NULL) {
    {
        …
    }

    return TRUE;
}
```

The structure indicating the write data address, the pointer to the buffer storing the write data, the buffer size, and the data exclusive to the access method are passed as parameters.

To support the writeEx function, define a data structure exclusive to the access method. The application sets the necessary data to this structure, and executes Em_writeMemoryEx() with the data structure pointer as pvMethodSpec. If pvMethodSpec is NULL, the same processing as the normal write function is performed.

5.2.9 Implementing the Event Condition Evaluation

The EventMemory can send events according to event conditions. To create an access method that supports event conditions, use the following EvtMem32.dll export functions:

EventMemory API	Functionality
EmCondition_OnWriteMemoryNormal()	Evaluates normal event conditions, and sends the event.
EmCondition_OnWriteMemoryWide()	Evaluates wide event conditions.
EmCondition_sendWideEvents()	Sends wide event condition events.

In the access method write process function, evaluate the conditions and send the event at data write:

```
    // normal conditon
    nNumberOfWordsWrite = nNumberOfBytesToWrite /
sizeof(WORD);
    EmCondition_OnWriteMemoryNormal(hmem,
psAddr->dwWordOffset,
        nNumberOfWordsWrite, pvData);

    // wide condition
    {
        size_t num;
        PWORD   pWordBefore = (PWORD)pBeforeData;
        PWORD   pWordAfter = (PWORD)pvData;
```

```
        DWORD   dwWordOffset = psAddr->dwWordOffset;
        for (num = 0; num < nNumberOfWordsWrite; num++,
            dwWordOffset++) {
            EmCondition_OnWriteMemoryWide(hmem,
dwWordOffset,
                (*pWordBefore != *pWordAfter));
        }
        EmCondition_sendWideEvents(hmem);
    }
```

EmCondition_OnWriteMemoryNormal

Evaluates normal conditions, and sends the event when the conditions are met:

```
void WINAPI EmCondition_OnWriteMemoryNormal(
    pFgwEmMemoryHandle hmem,
    DWORD   offset,
    DWORD   size,
    PVOID   pvWriteData);
```

Parameter	Description
hmem	Specifies the EventMemory handle.
offset	Specifies the data write start word.
size	Specifies the number of write words.
pvWriteData	Specifies the write data buffer address.

The EmCondition_OnWriteMemoryNormal function evaluates the conditions, if normal conditions are set in offset and size. If the send conditions are met, it sends the event.

Normal condition evaluation still requires that the offset and size be set in words, even if the actual write data type is not WORD. To write 10 words of data starting from word 100, set offset to 100 and size to 10.

EmCondition_OnWriteMemoryWide

Evaluates wide conditions. Does not send the event.

```
void WINAPI EmCondition_OnWriteMemoryWide(
    pFgwEmMemoryHandle hmem,
    DWORD   offset,
    BOOLbIsChange);
```

Parameter	Description
hmem	Specifies the EventMemory handle.
offset	Specifies the data write start word.
bIsChange	If the data before writing, and the data after writing are different, this is TRUE; if they are the same, it is FALSE.

The EmCondition_OnWriteMemoryNormal function evaluates the conditions, if wide conditions are set in offset. The evaluation result is retained internally.

Wide conditions are evaluated in word units. To write 10 words of data starting from word 100, execute this function for 10 words from word 100.

To send event for wide conditions, execute the EmCondition_sendWideEvents function after evaluating all the data.

EmCondition_sendWideEvents

Sends events for wide conditions:

```
void WINAPI EmCondition_ sendWideEvents (
    pFgwEmMemoryHandle hmem);
```

Parameter	Description
hmem	Specifies the EventMemory handle.

The EmCondition_ sendWideEvents function sends the event, if the EmCondition_OnWriteMemoryWide function evaluation result was that the conditions were met.

Before executing this function, the EmCondition_OnWriteMemoryWide function must be executed.

5.2.10 Registering the Access Method

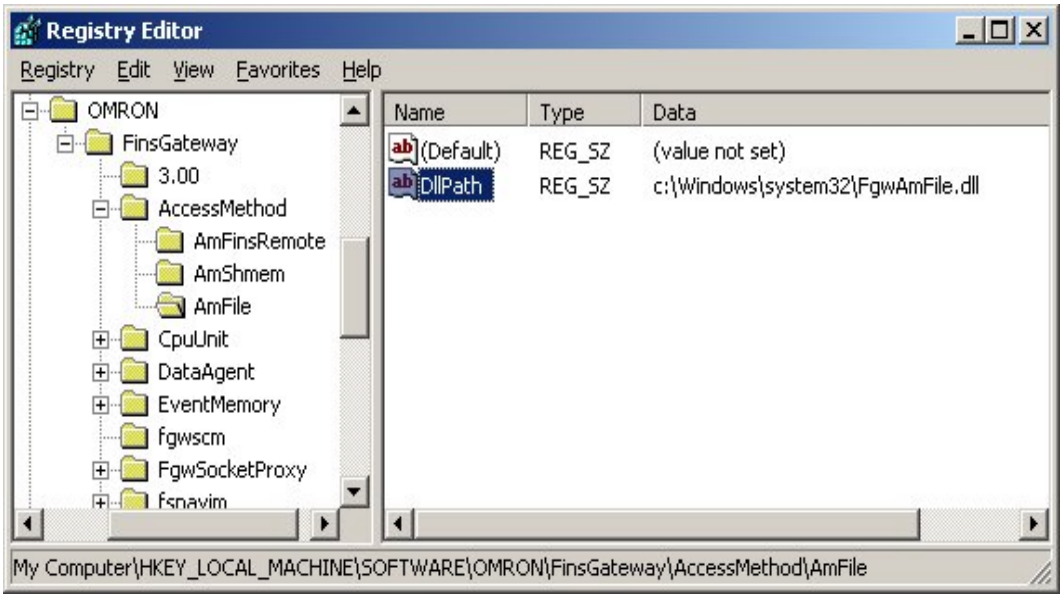
To use the created access method, register an entry for the access method in the registry. The EventMemory looks for access method entries under a fixed key in the registry, and loads the corresponding DLLs.

The registry keys are below the following:

HKEY_LOCAL_MACHINE¥SOFTWARE¥OMRON¥FinsGateway¥AccessMethod

Create a sub-key under this key for the created access method name, and below that create a character string value (REG_SZ or REG_EXPAND_SZ) "DllPath", and specify the DLL file name.

For example, to register an access method called AmFile, the registry configuration would be as follows:



5.2.11 Setting the Registry

The data to determine the EventMemory operations is set to the registry. Normally, the FinsGateway Configuration utility is used to set this data.

The registry keys are below the following:

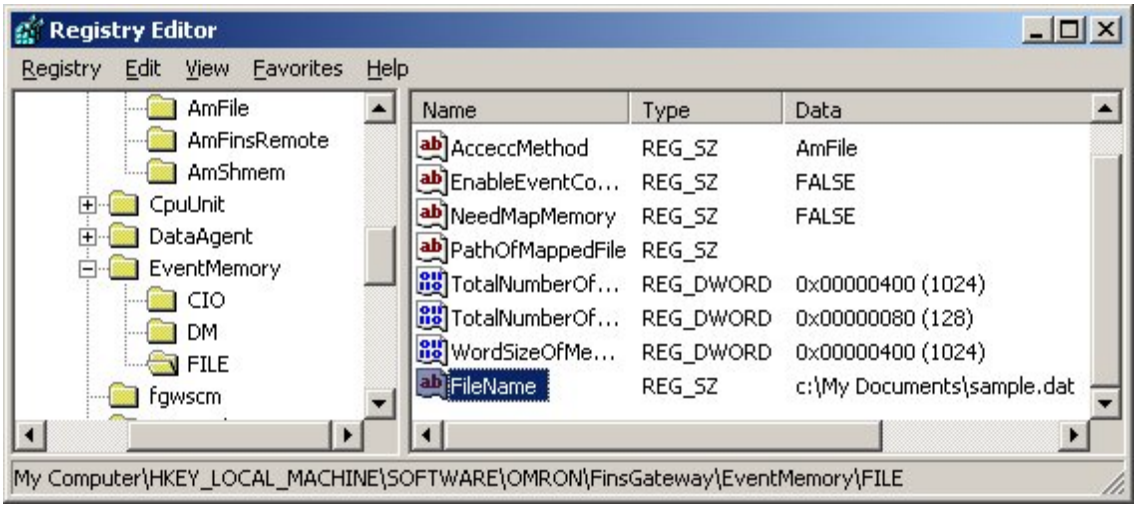
HKEY_LOCAL_MACHINE¥SOFTWARE¥OMRON¥FinsGateway¥EventMemory

Under this key, the memory name sub-keys are listed. The following are examples of values that would be set to these keys as registry entries. Only the main entries are shown here:

Registry Name	Description
AccessMethod	Specifies the access method names to use.
WordSizeOfMemory	Specifies the memory size in words.
NeedMapMemory	Specifying TRUE maps the data area to shared memory. Normally set FALSE for anything other than AmShmem.
EnableEventCondition	Specifying TRUE enables the event condition settings. Set FALSE when not supporting event conditions.

When adding settings exclusive to the access method, those entries cannot be set with the FinsGateway Configuration utility. They must be set with the registry editor.

For example, to add the "FileName" entry under the AmFile access method (the file path), the registry configuration would be as follows:



6 Error Systems

6.1 Error Codes

- Basically, the EventMemory reports an error to an application when a called function returns an error.
- An exception is that when sending an event (as the result of testing an event-occurrence condition) fails, the event is logged. This is because the Em_writeMemory function call to trigger the event send is not always used by a user application; therefore the error cannot be reported to the user application by returning an error by the called function.
- When an error occurs with the EventMemory, the applications can locate the cause of the error by calling the GetLastError function. An error code is a 32-bit value (bit 31 is the most-significant bit), and bit 29 is reserved by Microsoft as an error code for application definition, and is always set for error codes set by the EventMemory. The table below lists the error codes. The error codes in the table are actual error codes whose most-significant two bytes are masked. Actual error codes are the results of a bitwise OR operation with the error codes in the table and 0x20000000.

Table 6-1: Error codes

Code	Definition(EM_ERROR_*)	Description
1	INVALID_EVENT_PORT_NAME	The length of the character string for the event port name is invalid.
2	NO_MORE_EVENT_PORT	The event port has no free space.
3	INVALID_EVENT_HANDLE	The event handle of the EventMemory is invalid.
4	NO_SET_EXECUTE_VERSION	The operation version of the EventMemory is not set.
5	OUTBREAK_OF_EXCEPTION	An exception error occurred during processing.
6	FAIL_IN_DETACH_BASE_OBJECT	A base object could not be opened.
7	OUT_OF_EVENT_RANGE_ON_SEND	The number of bytes that cannot be sent is specified in the event. (During event sending)
8	NO_EVENT_PORT_TO_SEND	The event port of destination was not found.
9	FAIL_IN_SEND_BASE_OBJECT	A base object could not be sent.
10	ILLEGAL_BASE_OBJECT	Invalid data was received as a base object.
11	OUTBREAK_OF_TIMEOUT_FOR_EVENT	A timeout occurred in event reception wait state.
12	FAIL_IN_RECEIVE_EVENT	Event reception wait failed.
13	EXECUTE_VERSION_ALREADY_LOCKED	The operation version is already locked and cannot be changed.
14	NOT_SUPPORTED_VERSION	The version is not supported. The system cannot be activated.
15	INVALID_MEMORY_PORT_NAME	The length of the character string for the memory port name is invalid.
16	ALREADY_OPENED_MEMORY_PORT	The memory port is already open in the process.
17	ILLEGAL_MEMORY_SIZE	An invalid value is set for memory size.
18	FAIL_IN_OPEN_FILE	A file could not be created or opened.
19	NO_MORE_MEMORY_PORT	The memory port has no free space.
20	FAIL_IN_ATTACH_MEMORY	Shared memory could not be attached.
21	ALREADY_EXISTED_MEMORY	The memory area already exists.
22	INVALID_MEMORY_HANDLE	The memory handle for the EventMemory is invalid.
23	FAIL_IN_RELEASE_MEMORY_RECORD	Records of the memory port could not be opened.
24	FAIL_IN_DETACH_MEMORY	Shared memory could not be detached.
25	INVALID_EM_ADDRESS	The structure data used to specify the address of the EventMemory are invalid.

26	FIRST_OFFSET_OUT_OF_MEMORY_RANGE	The first offset of the specified address exceeds the memory size.
27	INSUFFICIENT_READ_BUFFER	The size of the buffer for reading data is insufficient.
28	FAIL_IN_GET_MUTEX	The property rights for Mutex could not be obtained.
29	INSUFFICIENT_WRITE_DATA	Write data are insufficient.
30	BIT_NEITHER_ONE_NOR_ZERO	A value other than 0 or 1 is set for data of bit type.
31	FAIL_IN_SEND_EVENT_ON_WRITE	During data writing, a generated event could not be sent.
32	NOT_WORD_TYPE_OF_FACTOR	In the setting of wide-area event conditions, a data type other than word is specified.
33	NO_MORE_WIDE_CONDITION_POOL	Too many wide-area event conditions.
34	INVALID_EM_ADDRESS_FOR_DATA	The structure data used to specify the EventMemory address sent as event data are invalid.
35	OUT_OF_EVENT_RANGE_ON_CONDITION	The number of bytes that cannot be sent is specified in the event. (Error generate during setting of event conditions)
36	NOT_SET_CONDITION_ID	Processing was requested for a condition ID not set.
37	FAIL_IN_ATTACH_CONDITION_TO_FREE	An event-occurrence condition could not be attached to the free list.
38	NOT_ONE_NUMBER_OF_FACTOR	In the setting for event conditions, 1 was not set as the number of data.
39	BIT_TYPE_ON_CONDITION	In the setting for event conditions, bit is set as the data type.
40	ODD_OFFSET_ON_DWORD_CONDITION	In the setting for double-word event conditions, an odd number is set as the offset.
41	FAIL_IN_ATTACH_CONDITION_TO_MEMORY	Event conditions could not be associated with memory.
42	NO_MORE_NORMAL_CONDITION_POOL	Too many normal event conditions.
43	LAST_OFFSET_OUT_OF_MEMORY_RANGE	The last offset of the specified address exceeds the memory size.
44	NOT_WINDOW_HANDLE	The specified window handle is invalid.
45	ALREADY_SET_TO_POST_THREAD	The setting is already made to post messages to a thread.
46	ALREADY_SET_TO_POST_WINDOW	The setting is already made to post messages to a window.
47	FAIL_IN_POST_THREAD_MESSAGE	A message could not be posted to a thread.
48	FAIL_IN_POST_WINDOW_MESSAGE	A message could not be posted to a window.
49	INVALID_CONSTANT1	The value of true/false evaluation comparison constant 1 is invalid.
50	INVALID_CONSTANT2	The value of true/false evaluation comparison constant 2 is invalid.
51	PROCESS_EVENT_INFO_OUT_OF_ORDER	Event-port management information by process is invalid.
52	FAIL_IN_ALLOC_MEMORY	Memory could not be allocated.
53	TOO_LONG_FILE_PATH	The file path specified in the registry is too long.
54	NOT_EXIST_CONDITION_FILE	A file for setting event conditions does not exist.
55	ALREADY_OPENED_EVENT_PORT	The event port is already open.
56	NO_MORE_EVENT_POOL	Too many events.
57	INSUFFICIENT_RECEIVE_BUFFER	The size of the buffer for receiving data was insufficient.
58	DISABLE_SET_EVENT_COND	This memory cannot have event conditions set.
59	NOT_EXIST_ACCESS_METHOD	Specified access method does not exist.
60	FAIL_OPEN_METHOD	Access method open method execution failure.
61	FAIL_CLOSE_METHOD	Access method close method execution failure.
62	FAIL_READ_METHOD	Access method read method execution failure.
63	FAIL_WRITE_METHOD	Access method write method execution failure.
64	FAIL_READEX_METHOD	Access method readEx method execution failure.
65	FAIL_WRITEEX_METHOD	Access method writeEx method execution failure.
66	FAIL_JUDGE_METHOD	Access method judge method execution failure.

7 API Reference

7.1 Summary

The library contains many functions, as described below.

Version management	
Em_requestVersion	Specifies the operating version.
Em_getVersion	Gets the release version.

Memory read/write	
Em_openMemory	Opens a memory port.
Em_closeMemory	Closes a memory port.
Em_readMemory	Reads from memory.
Em_writeMemory	Writes to memory.
Em_readMemoryEx	Reads from expanded function memory.
Em_writeMemoryEx	Writes to expanded function memory.

Event send/receive	
Em_openEvent	Opens an event port.
Em_sendEvent	Sends events.
Em_receiveEvent	Receives events.
Em_closeEvent	Closes an event port.

Setting and clearing event conditions	
Em_setCondition	Sets a normal event-occurrence condition.
Em_setWideCondition	Sets wide-area event-occurrence condition.
Em_clearCondition	Clears normal or wide-area event-occurrence condition.
Em_judgeCondition	Evaluates event conditions.

Setting or clearing message-driven event reception	
Em_setMessageOnArrival	Sets a message to be posted in a window.
Em_setThreadMessageOnArrival	Sets a message to be posted in a thread.
Em_clearMessageOnArrival	Clears message posting.

Getting internal information	
Em_getCondition	Gets the setting for a normal event-occurrence condition.
Em_getWideCondition	Gets the setting for a wide-area event-occurrence condition.
Em_isWideConditionId	Evaluates a condition ID as normal or wide-area.
Em_getConditionList	Gets the setting list of normal event conditions.
Em_getWideConditionList	Gets the setting list of wide-area event conditions.
Em_getLostEventLogs	Gets lost event logs.
Em_getSystemInfo	Gets the EventMemory system information.
Em_getMemoryInfo	Gets shared memory information.
Em_getMemoryPortUsage	Gets memory port usage status.
Em_getEventPortUsage	Gets event-port usage status.
Em_getEventHandle	Gets a Win32API event object handle.
Em_getMutexHandle	Gets a Win32API Mutex object handle.
Em_getBaseAddress	Gets the starting address of shared memory mapped in a process.

Others	
Em_swapBytes	Byte swap
Em_getLastErrorMessage	Gets error messages.
Em_flushFile	Flush to a file
Em_getBytesBuffer	Gets the number of bytes of a buffer.
Em_clearLostEventLogs	Clears the lost event log.

7.2 Em_requestVersion Function

Function

Specifies the operating version of the EventMemory.

```
BOOL Em_requestVersion(  //Success: TRUE, Failure: FALSE
    BYTEbyMajor,         //Major version
    BYTEbyMinor,         //Minor version
    )
```

Description

This function requests the EventMemory to operate with a specified version. If no operating version is specified by the Em_requestVersion function, the API of the EventMemory cannot be used. The Em_requestVersion function with the release version as an argument is defined by the macro EM_STARTUP.

Argument	Description
byMajor	Major version number. The major version number upon release is defined as EM_CURRENT_MAJOR_VERSION in the header file EvtMem.h.
byMinor	Minor version number. The minor version number upon release is defined as EM_CURRENT_MAJOR_VERSION in the header file EvtMem.h.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function. Specifying a version not found in the release history causes an error.

See Also

Em_*

7.3 Em_getVersion Function

Function

Gets the release version of the EventMemory.

```
EM_VERSION  Em_getVersion (void)
```

Description

This function gets release version. The version information retrieved by this function is independent of the operation version specified in Em_requestVersion.

Argument	Description
None	

EventMemory Version Information (EM_VERSION Structure)

```
typedef struct TagEmVersion {
    BYTEbyMajor;
    BYTEbyMinor;
    BYTEbyRevision;
    BYTEbyReserved;
} EM_VERSION, *pEM_VERSION;
```

Structure member	Description
byMajor	Major version number
byMinor	Minor version number
byRevision	Revision number
byReserved	Reserved area

Return Value

Returns the EM_VERSION structure containing the release version of the EventMemory. This function will not fail.

See Also

7.4 Em_openMemory Function

Function

Opens a memory port for the EventMemory.

```
HANDLE Em_openMemory(          //Memory handle, Failure: NULL
    LPCTSTR lpszMemoryName,    //Memory name
    PVOID   pvBaseAddress,     //Map starting address
    )
```

Description

This function opens a memory port with the specified memory name, and returns the handle.

If the memory is opened for the first time, the necessary profile data is obtained and the memory object is created. If a memory by the same name is already open, the port for the existing memory is returned.

Argument	Description
lpszMemoryName	Pointer to a NULL-terminated character string to specify the name of the EventMemory. Up to 15 characters are allowed. No distinction between uppercase and lowercase letters is made.
pvBaseAddress	Specify NULL.

Return Value

The function returns the memory handle of the EventMemory when it terminates normally. It returns NULL if it does not terminate normally. To get additional error information, use the GetLastError function. If the shared memory of a memory port to be opened already exists before the function is called, the GetLastError function returns EM_ERROR_ALREADY_EXISTED_MEMORY, and the Em_openMemory function returns a handle valid for accessing the existing shared memory. If the specified shared memory area does not exist, GetLastError returns 0.

If a memory port already opened in a process is opened, the GetLastError function returns EM_ERROR_ALREADY_OPENED_MEMORY_PORT, and the Em_openMemory function returns the handle of the memory port already opened.

See Also

Em_closeMemory, Em_readMemory, Em_writeMemory, Em_readMemoryEx, Em_writeMemoryEx, Em_setCondition, Em_setWideCondition, Em_clearCondition, Em_getCondition, Em_getWideCondition, Em_isWideConditionId, Em_getConditionList, Em_getWideConditionList, Em_judgeCondition, Em_getBaseAddress, Em_getMutexHandle, Em_getMemoryInfo

7.5 Em_readMemory Function

Function

Reads the EventMemory data.

```
BOOL Em_readMemory (           //Success: TRUE, Failure: FALSE
    HANDLE         hMemory,    //Memory handle
    pEM_ADDRESS psAddress,     //Read address an specification
    PVOID          pvBuffer,   //Pointer to read from a buffer
    DWORD          dwNumberOfBytesBuffer,
                               //Read buffer size (bytes)
    )
```

Description

Reads the EventMemory data.

Argument	Description
hMemory	Memory Handle of the EventMemory
psAddress	Pointer to the EM_ADDRESS structure to specify the address of the EventMemory to read data to. Specify the offset, data type, bit/byte position, and the number of data.
pvBuffer	Pointer to the data buffer to read data to. If the data type is bit, reading is performed in a way that writes the value 0 or 1 to a 1-byte memory area. Accordingly, a byte array buffer of the same size as the number of bits to be read is required.
dwNumberOfBytesBuffer	Number of bytes of read buffer. If the number of bytes of read buffer is insufficient for the read address specification, the data is not read in buffer.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory, Em_closeMemory

7.6 Em_writeMemory Function

Function

This function writes data to the EventMemory.

```
BOOL Em_writeMemory (          //Success: TRUE, Failure: FALSE
    HANDLE         hMemory,    //Memory handle
    pEM_ADDRESS psAddress,     //Write address an specification
    PVOID          pvData,     //Pointer to write data
    DWORD          dwNumberOfBytesData,
                               //Number of bytes of writes data
    )
```

Description

This function writes data to the EventMemory. If the target is shared memory, when writing data, it evaluates the event conditions set in the write area, and automatically sends the event to the event port. If the target is remote memory, when writing data, it does not evaluate the event conditions. To evaluate the event conditions, execute the Em_judgeCondition function after writing the data.

Argument	Description
hMemory	Memory Handle of the EventMemory
psAddress	Pointer to the EM_ADDRESS structure to specify the address of the EventMemory to write data to. Specify the offset, data type, bit/byte position, and the number of data.
pvData	Pointer to the data buffer to write to. If the data type is bit, writing is performed in a way that writes the value 0 or 1 to a 1-byte memory area. Accordingly, before writing, set write data in a byte array of the same size as the number of bits to be written.
dwNumberOfBytesData	Number of bytes of write data. If the number of bytes of write data is insufficient for the write address specification, the data is not written.

Return Value

The function returns TRUE when data is normally written to the EventMemory. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

If only event automatic send by event-occurrence condition evaluation fails, the GetLastError function returns EM_ERROR_FAIL_IN_SEND_EVENT_ON_WRITE and the Em_writeMemory function returns TRUE. To get log information about the event (lost event) that failed in automatic send by event-occurrence condition evaluation, use the Em_getLostEventLogs function.

If the shared memory port for reading the data to be sent is not opened during event automatic send, the function internally opens the memory port.

See Also

Em_openMemory, Em_closeMemory, Em_setCondition, Em_setWideCondition, Em_getLostEventLogs, Em_judgeCondition

7.7 Em_readMemoryEx Function

Function

Reads data from EventMemory.

```
BOOL Em_readMemory(      //Successful: TRUE; failed: FALSE
```

```
    HANDLE         hMemory,     //Memory handle
    pEM_ADDRESS psAddress,     //Read address specification
    PVOID          pvBuffer,    //Read buffer pointer
    DWORD          dwNumberOfBytesBuffer,
        //Read buffer size (bytes)
    PVOID          pvAccessMethodSpec
        //Pointer to structure exclusive to access method
    )
```

Description

Reads the additional data exclusive to the access method. The additional data read function differs for each access method. Depending on the access method implementation, the additional data read function is not necessarily supported. The pvAccessMethodSpec parameter has a special meaning within each access method. Normally, set the structure pointer.

Argument	Description
hMemory	EventMemory handle.
psAddress	Pointer to the EM_ADDRESS structure, which specifies the data read EventMemory address. Specify the offset, data type, bit/byte position, and number of data.
pvBuffer	Data read buffer pointer. If the data type is bit, the 0/1 value is read and stored into a 1-byte memory area. Therefore, it is necessary to have a byte array the same size as the number of bits to read.
dwNumberOfBytesBuffer	Number of read buffer bytes. If the buffer size is smaller than the read data, the data will not be read into the buffer.
pvAccessMethodSpec	Specifies the pointer to the structure exclusive to the access method. If NULL is specified, it operates the same as the Em_readMemory function.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory, Em_closeMemory, Em_readMemory

7.8 Em_writeMemoryEx Function

Function

Writes data to EventMemory.

```
BOOL Em_writeMemory(     //Successful: TRUE; failed: FALSE
    HANDLE         hMemory,     //Memory handle
    pEM_ADDRESS psAddress,     //Write address specification
    PVOID          pvData,           //Write data pointer
    DWORD          dwNumberOfBytesData,
        //Number of write data bytes
    PVOID          pvAccessMethodSpec
        //Pointer to structure exclusive to access method
    )
```

Description

Writes the additional data exclusive to the access method. The additional data write function differs for each access method. Depending on the access method implementation, the additional data write function is not necessarily

supported. The pvAccessMethodSpec parameter has a special meaning within each access method. Normally, set the structure pointer.

Argument	Description
hMemory	EventMemory handle.
psAddress	Pointer to the EM_ADDRESS structure, which specifies the data write EventMemory address. Specify the offset, data type, bit/byte position, and number of data.
pvData	Data write buffer pointer. If the data type is bit, the 0/1 value is stored into a 1-byte memory area. Therefore, it is necessary to have a byte array the same size as the number of bits to write.
dwNumberOfBytesData	Number of write data bytes. If the write data size is smaller than the write data area, the data will not be written.
pvAccessMethodSpec	Specifies the pointer to the structure exclusive to the access method. If NULL is specified, it operates the same as the Em_writeMemory function.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

If it only fails when sending the event after the event condition evaluation, the GetLastError function returns EM_ERROR_FAIL_IN_SEND_EVENT_ON_WRITE, and the Em_writeMemoryEx function returns TRUE. To get the log data of the events that failed on send (lost events log), use the Em_getLostEventLogs function.

If the shared memory port to read the send data is not open when sending the event, it is opened within the function.

See Also

Em_openMemory, Em_closeMemory, Em_setCondition, Em_setWideCondition, Em_getLostEventLogs

7.9 Em_closeMemory Function

Function

Closes a memory port of the EventMemory.

```
BOOL Em_closeMemory(     //Success: TRUE, Failure: FALSE
    HANDLE hMemory,      //Memory handle
    )
```

Description

This closes an opened memory port. If the number of memory ports opend in a process becomes zero, the shared memory is detached.

Argument	Description
hMemory	Memory Handle of the EventMemory

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory

7.10 Em_openEvent Function

Function

Opens an event port for the EventMemory.

```
HANDLE  Em_openEvent (          //Event handle, Failure: NULL
    LPCTSTR lpszEventName,      //Event port name
    )
```

Description

Opens an event port for the EventMemory.

Argument	Description
lpszEventName	Pointer to a NULL-terminated character string to specify the event name of the EventMemory. Up to 15 characters are allowed. No distinction between uppercase and lowercase letters is made.

Return Value

The function returns the Event handle of the EventMemory when it terminates normally. In other cases, it returns NULL. To get additional error information, use the GetLastError function.

See Also

Em_closeEvent, Em_receiveEvent, Em_sendEvent

7.11 Em_sendEvent Function

Function

This function sends an event of the EventMemory.

```
BOOL Em_sendEvent (            //Success: TRUE, Failure: FALSE
    HANDLE      hEvent,        //Event handle
    LPCTSTR     lpszEventName, //Event destination
    INT         lEventId,      //Event ID
    pEM_AREApsAreaInformed,    //Information indicating
                               //an event-occurrence
    PVOID       pvData,        //Pointer to send data
    DWORD       dwNumberOfBytesData,
                               //Number of bytes of send data
    )
```

Description

This function sends an event of the EventMemory to an event port.

Argument	Description
hEvent	Event Handle of the EventMemory
lpszEventName	Pointer to a NULL-terminated character string to specify the name of an event port to which to send. Up to 15 characters are allowed. No distinction between uppercase and lowercase letters is made.
lEventId	Event ID
psAreaInformed	Pointer to the EM_AREA structure to specify information indicating an event-occurrence area (memory name plus address). This information is not always necessary for the Em_sendEvent function, but can be used to exchange shared memory data indirectly with the application at the event-receive end. When psAreaInformed is NULL, this information is not sent.
pvData	Pointer to the send data buffer. When pvData is NULL, no data are sent.
dwNumberOfBytesData	Number of bytes of send data

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_closeEvent, Em_receiveEvent

7.12 Em_receiveEvent Function

Function

Receives an event of the EventMemory.

```
BOOL Em_receiveEvent (          //Success: TRUE, Failure: FALSE
    HANDLE      hEvent,         //Event handle
    PINT    plEventId,          //Event ID
    pEM_AREApsAreaInformed,     //Information indicating an
                                //event-occurrence area
    PVOID       pvBuffer,       //Pointer to the beginning of
                                //the receive data buffer
    DWORD       dwNumberOfBytesBuffer,
                                //Number of bytes of
                                //the receive data buffer
    PDWORD      pdwNumberOfBytesReceive,
                                //Number of bytes of received data
    DWORD       dwTimeout,      //Receive timeout (milliseconds)
    )
```

Description

Receives an event of the EventMemory.

Argument	Description
hEvent	Event handle of the EventMemory
plEventId	Pointer to the variable to store the ID of a received event
psAreaInformed	Pointer to the EM_AREA structure to specify information indicating an event-occurrence area (memory name plus address). psAreaInformed->lpszMemoryName[0] = EM_NO_ADDR_INFO indicates that no area to be reported is allocated in an event-sending end. When psAreaInformed is NULL, this information is not sent.
pvBuffer	Pointer to the receive data buffer. When pvBuffer is NULL, no data are received.
dwNumberOfBytesBuffer	Number of bytes of the receive data buffer. If the buffer size is smaller than the size of received □data, only the data of the number of bytes specified in dwNumberOfBytesBuffer is transferred, and the □function fails. In this case, part of the data is lost.
pdwNumberOfBytesReceive	Number of bytes of data possessed by a received event.
dwTimeout	Receive timeout (milliseconds). If dwTimeout is INFINITE, the timeout function does not work. If dwTimeout is 0, the function returns immediately, and terminates normally only when a receive event exists.

Return Value

Only when all data sent as events are received does the Em_receiveEvent function succeed. If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_closeEvent, Em_sendEvent, Em_writeMemory

7.13 Em_closeEvent Function

Function

Closes an event port of the EventMemory.

```
BOOL Em_closeEvent(            //Success: TRUE, Failure: FALSE
    HANDLE  hEvent,            //Event handle
    )
```

Description

This closes an open event port.

Argument	Description
hEvent	Event handle of the EventMemory

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent

7.14 Em_setCondition Function

Function

This function sets normal event conditions.

```
BOOL Em_setCondition (          //Success: TRUE, Failure: FALSE
    HANDLE          hMemory,    //Memory handle
    PDWORD          pdwConditionId,   //ID of the set conditions
    pEM_ADDRESS     psAddress,
                    //Address at which conditions are set
    pEM_CONDITION   psCondition,
                    //Normal event conditions
    BOOLEAN         bIsVolatile,//Volatility or non-volatility
                    // of conditions
    )
```

Description

This function sets normal event conditions in shared memory.

Argument	Description
hMemory	Memory Handle of the EventMemory
pdwConditionId	Pointer to the variable to store the ID of the set conditions. Used to clear and get conditions.
psAddress	Pointer to the EM_ADDRESS structure to specify the address of the EventMemory at which condition are set. Specify the offset, data type, byte position, and the number of data. Bit cannot be specified as the data type. The number of element must be 1.
psCondition	Pointer to the EM_CONDITION structure to specify normal event conditions. Specify a true/false evaluation operation, event send evaluation for the true/false transition for operation results, event destination, event ID, and shared memory area (memory name plus address) sent as data. If the data in the shared memory area are not to be sent, set psCondition->sSendObject.sSendArea and .sAddress.byTypeOfFactor=EM_NO_DATA.
bIsVolatile	Specification on volatility or nonvolatility of conditions. Specify the volatility or nonvolatility of conditions to be set. To specify a condition as volatile, set bIsVolatile=TRUE, and to specify it as nonvolatile, set bIsVolatile=FALSE. If volatility is specified, the conditions are lost when the shared memory for which they are set is unloaded. If nonvolatility is specified, the conditions are not lost even if the shared memory is unloaded.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory, Em_clearCondition, Em_getCondition, Em_getConditionList, Em_setMessageOnArrival, Em_setThreadMessageOnArrival, Em_clearMessageOnArrival

7.15 Em_setWideCondition Function

Function

This function sets wie-area event conditions.

```
BOOL Em_setWideCondition (     //Success: TRUE, Failure: FALSE
    HANDLE              hMemory,     //Memory handle
    PDWORD              pdwConditionId, //ID of the set conditions
    pEM_ADDRESS   psAddress,
                        //Address at which conditions are set
    pEM_WIDE_CONDITION  psWideCondition,
                        //wide-area event conditions
    BOOLEAN       bIsVolatile,
                  //Volatility or non-volatility of conditions
    )
```

Description

This function sets wide-area event conditions in shared memory.

Argument	Description
hMemory	Memory Handle of the EventMemory
pdwConditionId	Pointer to the variable to store the ID of the set conditions. Used to clear and get conditions.
psAddress	Pointer to the EM_ADDRESS structure to specify the address of the EventMemory at which condition are set. Specify the offset, data type, and the number of data. The data type must always be word. The byte position, if specified, is ignored.
psWideCondition	Pointer to the EM_WIDE_CONDITION structure to specify wide-area event conditions. Specify an event send evaluation method, event destination, event ID, and shared memory area (memory name plus address) sent as data. If the data in the shared memory area are not to be sent, set psWideCondition->sSendObject.sSendArea and .sAddress.byTypeOfFactor=EM_NO_DATA.
bIsVolatile	Specification on volatility or nonvolatility of conditions. Specify the volatility or nonvolatility of conditions to be set. To specify a condition as volatile, set bIsVolatile=TRUE, and to specify it as nonvolatile, set bIsVolatile=FALSE. If volatility is specified, the conditions are lost when the shared memory for which they are set is unloaded. If nonvolatility is specified, the conditions are not lost even if the shared memory is unloaded.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory, Em_clearCondition, Em_getWideCondition,

Em_getWideConditionList, Em_setMessageOnArrival,

Em_setThreadMessageOnArrival, Em_clearMessageOnArrival

7.16 Em_clearCondition Function

Function

Clears normal or wide-area event conditions.

```
BOOL Em_clearCondition(        //Success: TRUE, Failure: FALSE
    HANDLE              hMemory,          //Memory handle
    DWORD               dwConditionId,    //Condition ID
    )
```

Description

Clears normal or wide-area event conditions.

Argument	Description
hMemory	Memory Handle of the EventMemory
dwConditionId	ID of condition to be cleared

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setCondition, Em_setWideCondition

7.17 Em_judgeCondition Function

Function

Evaluates event conditions.

```
BOOL Em_judgeCondition(  //Successful: TRUE; failed: FALSE
    HANDLE               hMemory,     //Memory handle
    pEM_ADDRESS    psAddr,      //Condition address
    )
```

Return Value

Evaluates the normal event conditions set to the area specified in psAddr, and automatically sends an event to the event port.

This function is generally used for the AmFinsRemote access method (not the EventMemory write functions), for evaluating conditions without writing data. This is for the actual memory of devices that will have data written from other sources. For shared memory, it is also possible just to evaluate the conditions, without writing data.

This function does not evaluate wide conditions.

Argument	Description
hMemory	EventMemory handle.
psAddr	Specifies the event condition evaluation EventMemory address. The data type must always be WORD.

Description

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setCondition, Em_setWideCondition

7.18 Em_ setMessageOnArrival Function

Function

Allows a message to be posted in a window when an event arrives.

```
BOOL Em_setMessageOnArrival(   //Success: TRUE, Failure: FALSE
    HANDLE  hEvent,      //Event handle
    HWNDhWindow,         //Window handle of posting destination
    UINTuMessage,        //Message to be posted
    )
```

Description

This function allows a message to be posted in a window when an event arrives at an event port. In the window that received a message in this way, an event arrives at the event port at the same time and the Em_receiveEvent function succeeds in receiving the event without being blocked.

This function cannot allow messages to be posted in a thread. To change to message posting to a thread, temporarily free the setting using Em_clearMessageOnArrival, then use the Em_setThreadMessageOnArrival function.

When an event arrives, a message is posted in a window only once, and no retry is performed, even if posting fails.

Argument	Description
hEvent	Event handle of the EventMemory. Specify the handle of an event port to set message posting.
hWindow	Window handle for a window in which to post messages
uMessage	Specify the window message to be posted.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_sendEvent, Em_writeMemory,

Em_clearMessageOnArrival

7.19 Em_ setThreadMessageOnArrival Function

Function

Allows a message to be posted in a thread when an event arrives.

```
BOOL Em_setThreadMessageOnArrival (//Success: TRUE, Failure: FALSE
    HANDLE  hEvent,           //Event handle
    DWORD   dwThreadId,       //Thread ID of posting destination
    UINTuMessage,             //Message to be posted
    )
```

Description

This function allows a message to be posted in a thread when an event arrives at an event port. In the thread that received a message in this way, an event arrives at the event port at the same time and the Em_receiveEvent function succeeds in receiving the event without being blocked.

This function cannot allow messages to be posted in a window. To change to message posting to a window, temporarily free the setting using Em_clearMessageOnArrival, then use the Em_setMessageOnArrival function.

When an event arrives, a message is posted in a thread only once, and no retry is performed, even if posting fails.

Argument	Description
hEvent	Event handle of the EventMemory. Specify the handle of an event port to set message posting.
dwThreadId	ID of a thread to in which to post messages
uMessage	Specify the window message to be posted.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_sendEvent, Em_writeMemory,

Em_clearMessageOnArrival

7.20 Em_ clearMessageOnArrival Function

Function

This function clears the setting of message posting when an event arrives.

```
BOOL Em_ clearMessageOnArrival( //Success: TRUE, Failure: FALSE
    HANDLE      hEvent,         //Event handle
    )
```

Description

This function clears the setting of message posting to a window or thread when an event arrives.

Argument	Description
hEvent	Event handle of the EventMemory.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_setMessageOnArrival, Em_setThreadMessageOnArrival

7.21 Em_getCondition Function

Function

This function gets normal event-occurrence condition.

```
BOOL Em_getCondition (          //Success: TRUE, Failure: FALSE
    HANDLE          hMemory,    //Memory handle
    pEM_CND_INFO psCndInfo,     //Setting information items of set
                        //normal event conditions
    DWORD           dwCndId,    //ID of the set conditions
    )
```

Description

This function gets setting information from a set normal event-occurrence condition.

Argument	Description
hMemory	Memory handle of the EventMemory
psCndInfo	Pointer to an EM_CND_INFO structure to store the setting information items of normal event conditions
dwCndId	ID of event-occurrence condition to get setting information

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setCondition, Em_isWideConditionId

7.22 Em_getWideCondition Function

Function

This function gets wide-area event-occurrence condition.

```
BOOL Em_getWideCondition (     //Success: TRUE, Failure: FALSE
    HANDLE               hMemory,     //Memory handle
    pEM_WIDE_CND_INFO    psWideCndInfo,
                         //Setting information items of set
                         // wide-area event conditions
    DWORD                dwCndId,     //ID of the set conditions
    )
```

Description

This function gets setting information from a set wide-area event-occurrence condition.

Argument	Description
hMemory	Memory handle of the EventMemory
psWideCndInfo	Pointer to an EM_WIDE_CND_INFO structure to store the setting information items of wide-area event conditions
dwCndId	ID of event-occurrence condition to get setting information

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setWideCondition, Em_isWideConditionId

7.23 Em_isWideConditionId Function

Function

This function determines whether a condition ID is for wide-area event conditions.

```
BOOL Em_isWideConditionId (    //Success: TRUE, Failure: FALSE
    HANDLE      hMemory,       //Memory handle
    DWORD       dwCndId,       //ConditionID
    PBOOLEAN    pbIsWide,      //Judgment on wide-area conditions
    )
```

Description

This function determines whether a set condition ID is for wide-area event conditions or for normal event conditions.

Argument	Description
hMemory	Memory handle of the EventMemory.
dwCndId	ConditionID
pbIsWide	Pointer to the variable to store the judgment on wide-area conditions. TRUE indicates wide-area conditions, and FALSE indicates normal conditions.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setCondition, Em_setWideCondition

7.24 Em_getConditionList Function

Function

This function gets a list of set normal event conditions.

```
BOOL Em_getConditionList (     //Success: TRUE, Failure: FALSE
    HANDLE          hMemory,    //Memory handle
    pEM_CND_INFO psCndInfo,
                        //Structure to store in a list of
                        //the setting information items of set
                        //normal event conditions
    DWORD           dwNumberOfCndBufs,
                        //Number of items of a structural array
    PDWORD          pdwNumberOfCnds,
                        //Number of set normal event-occurrence
                        //conditions
    )
```

Description

This function gets a list of the setting information items of set normal event conditions, as a structural array by condition ID value in ascending order.

Argument	Description
hMemory	Memory handle of the EventMemory
psCndInfo	Pointer to the beginning of an EM_CND_INFO structure array to store a list of the setting information items of normal event conditions.
dwNumberOfCndBufs	Number of items of a structural array to store in a list. If the number of set normal event conditions exceeds dwNumberOfCndBufs, the number of setting information items to be retrieved is set to dwNumberOfCndBufs.
pdwNumberOfCnds	Pointer to the variable to store the number of normal event conditions set in shared memory.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setCondition

7.25 Em_getWideConditionList Function

Function

This function gets a list of set wide-area event conditions.

```
BOOL Em_getWideConditionList ( //Success: TRUE, Failure: FALSE
    HANDLE          hMemory,    //Memory handle
    pEM_WIDE_CND_INFO   psCndInfo,
                    //Structure to store in a list of
                    //the setting information items of set wide-area
                    //event conditions
    DWORD           dwNumberOfCndBufs,
                    //Number of items of a structural array
    PDWORD          pdwNumberOfCnds,
                //Number of set wide-area event conditions
    )
```

Description

This function gets a list of the setting information items of set wide-area event conditions, as a structural array by condition ID value in ascending order.

Argument	Description
hMemory	Memory handle of the EventMemory
psCndInfo	Pointer to the beginning of an EM_WIDE_CND_INFO structure array to store a list of the setting information items of wide-area event conditions.
dwNumberOfCndBufs	Number of items of a structural array to store in a list. If the number of set wide-area event conditions exceeds dwNumberOfCndBufs, the number of setting information items to be retrieved is set to dwNumberOfCndBufs.
pdwNumberOfCnds	Pointer to the variable to store the number of wide-area event conditions set in shared memory.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_setWideCondition

7.26 Em_getLostEventLogs Function

Function

This function gets log information of lost event.

```
BOOL Em_getLostEventLogs(      //Success: TRUE, Failure: FALSE
    pEM_LOST_EVENT psLostEvents,
                        //Structure to store a log of lost events
    DWORD          dwNumberOfEventBufs,
                        //Number of items of a structural array
    PDWORD         pdwNumberOfEventLogs,
                        //Number of logged lost events
    )
```

Description

This function gets log information of lost event as a structural array on a most recently output basis. Logs of lost events are logs of event-send failure (by event-occurrence condition) after the EventMemory system is loaded. When the EventMemory system has been unloaded, all log information of lost events is lost. The maximum number of items of stored lost events is the same as the total number of lost events log retrieved by the Em_getSystemInfo function.

Argument	Description
psLostEvents	Pointer to the beginning of an EM_LOST_EVENT structural array to store a log of lost events
dwNumberOfEventBufs	Number of items of a structure array to store log items. If the number of logged lost events exceeds dwNumberOfEventBufs, the number of lost events to be retrieved is set to dwNumberOfEventBufs.
pdwNumberOfEventLogs	Pointer to the variable to store the number of logged lost events

Log Information of Lost Event (EM_LOST_EVENT Structure)

```
typedef struct TagEmLostEvent {
    SYSTEMTIME  sSystemTime;
    char        lpszEventName[EM_NAME_LENGTH_MAX];
    INT         lEventId;
    DWORD       dwErrorCode;
} EM_LOST_EVENT, *pEM_LOST_EVENT;
```

Structure member	Description
sSystemTime	Local date and local time when event send failed. For details on the SYSTEMTIME structure, refer to Win32API.
lpszEventName	NULL-terminated character string to indicate the name of an event port to which to send lost events
lEventId	Event ID of a lost event
dwErrorCode	Error code to indicate the cause of failure of an event send

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_getSystemInfo, Em_getLastErrorMessage

7.27 Em_getSystemInfo Function

Function

This function gets information about the entire the EventMemory system.

```
BOOL Em_getSystemInfo (  //Success: TRUE, Failure: FALSE
    pEM_SYSTEM_INFO      psSystemInfo,
                  //Information about the entire EventMemory
    )
```

Description

This function gets information about the entire EventMemory system.

Argument	Description
psSystemInfo	Pointer to the EM_SYSTEM_INFO structure to store information about the entire EventMemory system

Information about the Entire System (EM_SYSTEM_INFO Structure)

```
typedef struct TagEventMemorySystemInformation {
    DWORD   dwTotalNumberOfSharedMemories;
    DWORD   dwTotalNumberOfEventPorts;
    DWORD   dwTotalNumberOfEvents;
    DWORD   dwTotalNumberOfLostEventLogs;
} EM_SYSTEM_INFO, *pEM_SYSTEM_INFO;
```

Structure member	Description
dwTotalNumberOfSharedMemories	Total number of shared memory areas that can be opened in the entire system
dwTotalNumberOfEventPorts	Total number of event ports that can be opened in the entire system
dwTotalNumberOfEvents	Total number of events that can be held in the entire system
dwTotalNumberOfLostEventLogs	Total number of log items for lost events that can be stored in the entire system

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

None

7.28 Em_getMemoryInfo Function

Function

This function gets information on shared memory.

```
BOOL Em_getMemoryInfo (  //Success: TRUE, Failure: FALSE
    HANDLE               hMemory,          //Memory handle
    pEM_MEMORY_INFO      psMemoryInfo,     //Memory-port information
    )
```

Description

This function gets information on shared memory of an open memory port.

Argument	Description
hMemory	Memory handle of shared memory on which information is to be retrieved
psMemoryInfo	Pointer to the EM_MEMORY_INFO structure to store information on shared memory

Information on shared memory (size information and file-path information) (EM_MEMORY_INFO structure)

Information on shared memory consists of size information and file-path information.

```
//Size information on shared memory
typedef struct TagMemorySizeInformation {
    DWORD   dwWordSizeOfMemory;
    DWORD   dwTotalNumberOfNormalConditions;
    DWORD   dwTotalNumberOfWideConditions;
} EM_MEMORY_SIZE_INFO, *pEM_MEMORY_SIZE_INFO;

//Path information about a file holding shared memory data
typedef struct TagMemoryMappedFilePathInformation {
    charlpszPathOfMemoryDataFile[MAX_PATH];
    charlpszPathOfNormalConditionFile[MAX_PATH];
    charlpszPathOfWideConditionFile[MAX_PATH];
} EM_FILE_PATH_INFO, *pEM_FILE_PATH_INFO;

//Information on shared memory (size information and file-path information)
typedef struct TagMemoryInformation {
    EM_MEMORY_SIZE_INFO sMemorySizeInfo;
    EM_FILE_PATH_INFO         sFilePathInfo;
} EM_MEMORY_INFO, *pEM_MEMORY_INFO;
```

Structure member	Description
sMemorySizeInfo .dwWordSizeOfMemory	Memory size (words)
sMemorySizeInfo .dwTotalNumberOfNormalConditions	Total number of normal conditions that can be set on memory
sMemorySizeInfo .dwTotalNumberOfWideConditions	Total number of wide-area conditions that can be set on memory
sFilePathInfo .lpszPathOfMemoryDataFile	NULL-terminated character string to indicate the path of a file holding data in the case of nonvolatile shared memory
sFilePathInfo .lpszPathOfNormalConditionFile	NULL-terminated character string to indicate the path of a file holding normal conditions
sFilePathInfo .lpszPathOfWideConditionFile	NULL-terminated character string to indicate the path of a file holding wide-area conditions

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory

7.29 Em_getMemoryPortUsage Function

Function

This function gets information about memory-port usage status.

```
BOOL Em_getMemoryPortUsage (   //Success: TRUE, Failure: FALSE
    pEM_MEMORY_PORT_USED psUsedMemoryPort,
                //Structure for storing memory-port usage status
    DWORD   dwNumberOfPortBufs,
                        //Number of items of a structural array
    PDWORD  pdwNumberOfUsedPorts,
                        //Number of memory ports used
    )
```

Description

This function gets usage status information about memory ports (shared memories) with different names.

Argument	Description
psUsedMemoryPort	Pointer to the beginning of the EM_MEMORY_PORT_USAGE structural array for storing memory-port usage status.
dwNumberOfPortBufs	Number of structural array items to store memory-port usage status. If the number of memory ports used exceeds dwNumberOfPortBufs, the number of memory port usage statuses to be retrieved is set to dwNumberOfPortBufs.
pdwNumberOfUsedPorts	Pointer to the variable to store the number of memory ports (number of shared memory areas) used.

Information about memory-port usage status (EM_MEMORY_PORT_USED structure)

```
typedef struct TagMemoryPortUsed {
    char    lpszMemoryName[EM_NAME_LENGTH_MAX];
    DWORD   dwProcess;
} EM_MEMORY_PORT_USED, *pEM_MEMORY_PORT_USED;
```

Structure member	Description
lpszMemoryName	NULL-terminated character string to indicate the name of open memory ports.
dwProcess	Indicates the number of processes that open memory ports.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory, Em_closeMemory

7.30 Em_getEventPortUsage Function

Function

This function gets information about event-port usage status.

```
BOOL Em_getEventPortUsage (    //Success: TRUE, Failure: FALSE
    pEM_EVENT_PORT_USED psUsedEventPort,
                //Structure for storing event-port usage status
    DWORD       dwNumberOfPortBufs,
                        //Number of items of a structural array
    PDWORD      pdwNumberOfUsedPorts,
                        //Number of event ports used
    )
```

Description

This function gets information about event-port usage status.

Argument	Description
psUsedEventPort	Pointer to the beginning of the EM_EVENT_PORT_USED structural array for storing event-port usage status.
dwNumberOfPortBufs	Number of structural array items to store event-port usage status. If the number of event ports used exceeds dwNumberOfPortBufs, the number of event port usage statuses to be retrieved is set to dwNumberOfPortBufs.
pdwNumberOfUsedPorts	This function gets the number of event ports used.

Information about event-port usage status (EM_EVENT_PORT_USED structure)

```
typedef struct TagEventPortUsed {
    charlpszEventName[EM_NAME_LENGTH_MAX];
} EM_EVENT_PORT_USED, *pEM_EVENT_PORT_USED;
```

Structure member	Description
lpszEventName	NULL-terminated character string to indicate the name of open event ports.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_closeEvent

7.31 Em_getEventHandle Function

Function

Gets the handle of the Win32API event object used internally by the API function to receive events of the EventMemory.

```
HANDLE Em_getEventHandle(
                        //Win32API Event handle, Failure: NULL
    HANDLE  hEvent,     //Event handle
    )
```

Description

This function gets the handle of the Win32API event object used internally by the Em_receiveEvent function to receive events of the EventMemory. If the event object of the retrieve handle is a signal, an event arrives in Event port and the Em_receiveEvent function succeeds in receiving the event without being blocked.

Argument	Description
hEvent	Event Handle of the EventMemory

Return Value

The function returns the handle of the Win32API event object when it terminates normally. In other cases, it returns NULL. To get additional error information, use the GetLastError function.

See Also

Em_openEvent, Em_receiveEvent

7.32 Em_getMutexHandle Function

Function

This function gets the handle of the Win32API Mutex object used internally by API functions.

```
HANDLE  Em_getMutexHandle (    //Mutex handle, Failure: NULL
    HANDLE  hMemory,           //Memory handle
    )
```

Description

This function gets the handle of the Win32API Mutex object used for exclusive control internally by API functions that read and write through the memory ports of the EventMemory, and set and clear event conditions. Acquisition of the rights to use Mutex objects blocks the functions shown below from processing other threads. Note that acquisition of the rights to use Mutex objects may exert a serious influence on the operation of I/O communication units (service) and CPU units (service) of FinsGateway:

- Em_openMemory, Em_readMemory, Em_writeMemory, Em_closeMemory,
- Em_setCondition, Em_setWideCondition, Em_clearCondition,
- Em_isWideConditionId, Em_getCondition, Em_getWideCondition,
- Em_getConditionList, Em_getWideConditionList,
- Em_openEvent, Em_closeEvent, Em_sendEvent, Em_setMessageOnArrival,
- Em_setThreadMessageOnArrival, Em_clearMessageOnArrival,
- Em_getLostEventLogs, Em_getMemoryInfo, Em_flushFile,
- Em_getMemoryPortUsage, Em_getEventPortUsage

Argument	Description
hMemory	Memory Handle of the EventMemory

Return Value

The function returns the handle of the Win32API Mutex object when it terminates normally. In other cases, it returns NULL. To get additional error information, use the GetLastError function.

See Also

Em_openMemory

7.33 Em_getBaseAddress Function

Function

This function gets the starting address of the map of shared memory.

```
PVOID Em_getBaseAddress (
                    //Starting address of the map of shared memory
    HANDLE  hMemory,    //Memory handle
    )
```

Description

This function gets the starting address at which shared memory is mapped in process address space.

Argument	Description
hMemory	Memory Handle of the EventMemory

Return Value

The function returns the map address of shared memory when it terminates normally. In other cases, it returns NULL. To get additional error information, use the GetLastError function.

See Also

Em_openMemory

7.34 Em_swapBytes Function

Function

This function swaps the most significant and least-significant bytes in a word data array.

```
BOOL Em_swapBytes(
    PWORD   pwData,      //Pointer to a buffer containing swap data
    DWORD   dwNumberOfWords,         //Size (words) of swap data
    )
```

Description

This function swaps the most significant and least-significant bytes of each two-byte data item in a word data array.

Argument	Description
pwData	Pointer to the beginning of a word data array to be swapped
dwNumberOfWords	Number of words to be swapped. **Note that this is not the number of bytes.**

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_readMemory, Em_writeMemory, Em_receiveEvent,

Em_setCondition, Em_getCondition, Em_getConditionList

7.35 Em_getLastErrorMessage Function

Function

This function gets an error message corresponding to the error code

```
int Em_getLastErrorMessage(
                  //Number of bytes transferred to the buffer
    DWORD   dwErrorCode,            //Error code
    LPTSTR  lpBuffer,               //Pointer to a character-string buffer
    int     lNumberOfBytesBuffer,
                            //Number of bytes in the buffer
    )
```

Description

This function gets an error message corresponding to the error code retrieved by the GetLastError function.

Argument	Description
dwErrorCode	Error code retrieved by the GetLastError
lpBuffer	Pointer to top of the character string in which to store an error message
lNumberOfBytesBuffer	Number of character strings in which to store error messages. If the number of character strings for error messages exceeds lNumberOfBytesBuffer, the number of character strings for error messages is set to lNumberOfBytesBuffer-1.

Return Value

Upon normal termination, this function returns the number of bytes transferred to the buffer. In other cases, it returns 0 (zero). To get additional error information, use the GetLastError function.

See Also

GetLastError

7.36 Em_flushFile Function

Function

This function flushes data to a file.

```
BOOL Em_flushFile( //Success: TRUE, Failure: FALSE
    HANDLE  hMemory,    //Memory handle
    )
```

Description

This function flushes shared memory data to a file. In the event of nonvolatile EventMemory, shared memory data is flushed in the following case:

- When a process detaches shared memory (when the number of open memory ports becomes 0 or when a process is terminated)
- Automatic update sequence by Windows NT
- When the Em_flushFile function is called

Since flushing to a file is automatically performed by the operating system, basically the Em_flushFile function need not be called in normal applications. Use this function only when file flushing is required after data updating. If no data are updated, calling the Em_flushFile function does not trigger writing to a file.

Frequent file flushing may adversely affect the system's speed.

Argument	Description
hMemory	Memory handle of the EventMemory. Specify the shared memory area from which data is to be flushed.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_openMemory, Em_writeMemory

7.37 Em_getBytesBuffer Function

Function

This function gets the number of bytes of the buffer required to access shared memory.

```
BOOL Em_getBytesBuffer(        //Success: TRUE, Failure: FALSE
    pEM_ADDRESS psAddress,     //Access address
    PDWORD      pdwNumberOfBytesBuffer,
                               //Number of bytes of the buffer
    )
```

Description

This function gets the number of bytes of the buffer required to access shared memory for a specified access address.

Argument	Description
psAddress	Pointer to the EM_ADDRESS structure to specify the address of the EventMemory to access. Specify the offset, data type, bit/byte position, and the number of data.
pdwNunberOfBytesBuffer	Pointer to the variable to store the number of bytes of the buffer required for access.

Return Value

If the function completes normally, it returns TRUE. Otherwise, it returns FALSE. To get additional error information, use the GetLastError function.

See Also

Em_readMemory, Em_writeMemory

8 Data Structure

8.1 The EventMemory Address

The EventMemory Address (EM_ADDRESS Structure)

```
typedef struct TagEmAddress {
    BYTE    byTypeOfFactor;
    BYTE    byLocateOnWord;
    DWORD   dwWordOffset;
    DWORD   dwNumberOfFactors;
} EM_ADDRESS, *pEM_ADDRESS;
```

Structure member	Description
byTypeOfFactor	Data type. Specify the data types of the data items for read, write, and condition settings. Bit, byte, word, and double word are represented by EM_BIT_TYPE, EM_BYTE_TYPE, EM_WORD_TYPE, and EM_DWORD_TYPE, respectively. The data type used for wide-area condition setting is word only. For the setting of event conditions, no data (EM_NO_DATA) can be specified to specify a shared memory area to send data.
byLocateOnWord	When the data type is bit or byte, specify the starting bit or byte position in the first offset used for read, write, or condition setting. For bit, specify 0 to 15 bits. For byte, specify most-significant (EM_BYTE_HIGH) or least-significant (EM_BYTE_LOW) byte. The bit or byte position has meaning only in the following cases: Read/write setting when the data type is bit or byte Condition setting when the data type is byte
dwWordOffset	Specify in words the starting offset of an address for which read, write, and condition settings are to be performed. If the data type is double word in the condition setting, the offset must be even.
dwNumberOfFactors	Specify the number of data elements for which a read, write, or condition setting is made. For the setting of normal event conditions, be sure to specify 1 as the number of data.

8.2 Event conditions

Normal Event Send-evaluation Information (EM_ESTIMATION Structure)

```
typedef struct TagEmEstimation {
    EM_LOGIC            sLogic;
    EM_ACTION           sAction[TRANSIT];
    EM_PREVIOUS_RESULT  sPreviousResult;
} EM_ESTIMATION, *pEM_ESTIMATION;
```

Structure member	Description
sLogic sLogicType sConst	sLogicType: An operation returning a true/false result that is executed during updating of shared memory. You may set any of the operations shown in Table 3-1 "Operations on Normal Event conditions." sConst.dwConst1/sConst.dwConst2: Constant compared for a true/false operation
sAction: sAction[FtoF] sAction[FtoT] sAction[TtoT] sAction[TtoF]	You may set whether an event is to be sent (ExecuteOnTransit) or not (NothingOnTransit), for all of the four true/false transition patterns ("false -> false," "false -> true," "true -> true," and "true -> false") derived from the operation results from previous memory updating and those upon current memory updating.
sPreviousResult	True/false operation result from previous memory updating. When acquiring event conditions, true or false is obtained as the previous true/false operation results. For setting event conditions, set true, false, or auto as an initial value. The parameter sPreviousResult includes true and false. Auto can be set so that true or false is set depending on the value of shared memory according to the settings for event conditions. This applies only to the setting of event conditions. In the setting of event conditions, when the operation (Prev*) to compare a previous value is performed as a true/false evaluation operation and sPreviousResult=Auto is specified, the previous value for condition setting must be set in sConst.dwConst1 to obtain the previous true/false result.

Event Destination (EM_DESTINATION Structure)

```
typedef struct TagEmDestinationOnEvent {
    charlpszEventName[EM_NAME_LENGTH_MAX];
} EM_DESTINATION, *pEM_DESTINATION;
```

Structure member	Description
lpszEventName	NULL-terminated character string to indicate the name of an event port to which to send. Up to 15 characters are allowed. No distinction between uppercase and lowercase letters is made.

Shared Memory Area Information (EM_AREA Structure)

This information can be used to a specify shared memory area from which data are to be sent as event sent depending on event conditions or to send shared memory data related to events in event send or receive.

```
typedef struct TagEmMemoryArea {
    char        lpszMemoryName[EM_NAME_LENGTH_MAX];
    EM_ADDRESS  sAddress;
} EM_AREA, *pEM_AREA;
```

Structure member	Description
lpszMemoryName	NULL-terminated character string to indicate the name of a shared memory area
sAddress	Address of shared memory

Send Event Information (EM_SEND_OBJECT Structure)

```
typedef struct  TagEmObjectSent {
    INT         lEventId;
    EM_AREA     sSendArea;
} EM_SEND_OBJECT, *pEM_SEND_OBJECT;
```

Structure member	Description
lEventId	Event ID
sSendArea	Area (memory name plus address) of shared memory sent as event data

Normal Event conditions (EM_CONDITION Structure)

Normal event conditions consist of normal event send-evaluation information, event destination, and send event information.

```
typedef struct TagEmCondition {
    EM_ESTIMATION       sEstimation;
    EM_DESTINATION      sDestination;
    EM_SEND_OBJECT      sSendObject;
} EM_CONDITION, *pEM_CONDITION;
```

Structure member	Description
sEstimation	Information for determining whether or not to send an event
sDestination	Event port to which to send an event
sSendObject	Information sent as an event

Wide-area Event conditions (EM_WIDE_CONDITION Structure)

Wide-area event conditions consist of wide-area event send-evaluation information, event destination, and send event information.

```
typedef struct TagEmWideCondition {
    EM_WIDE_ESTIMATION  sWideEstimation;
    EM_DESTINATION      sDestination;
    EM_SEND_OBJECT      sSendObject;
} EM_WIDE_CONDITION, *pEM_WIDE_CONDITION;
```

Structure member	Description
sWideEstimation	The parameter sWideEstimation contains information about whether an event is sent when data on a specified shared memory area changes (EventOnChange) or when data is written to a specified shared memory area (EventOnWrite).
sDestination	Event port to which to send an event
SSendObject	Information sent as an event

8.3 Acquired Information

Setting Information Items of Normal Event Conditions (EM_CND_INFO Structure)

The acquired information on set normal event conditions

```
typedef struct TagEmConditionInfo {
    DWORD          dwCndId;
    EM_ADDRESS     sAddress;
    EM_CONDITION   sCondition;
    BOOLEAN      bIsVolatile;
} EM_CND_INFO, *pEM_CND_INFO;
```

Structure member	Description
dwCndId	ID of event conditions
sAddress	Address at which event conditions are set
sCondition	Normal event-occurrence condition
bIsVolatile	It indicates volatility or non-volatility of conditions.

Setting Information Items of Wide-area Event Conditions (EM_WIDE_CND_INFO Structure)

Acquired information on set wide-area event conditions

```
typedef struct TagEmWideConditionInfo {
    DWORD                dwCndId;
    EM_ADDRESS           sAddress;
    EM_WIDE_CONDITION    sWideCondition;
    BOOLEAN              bIsVolatile;
} EM_WIDE_CND_INFO, *pEM_WIDE_CND_INFO;
```

Structure member	Description
dwCndId	ID of event conditions
sAddress	Address at which event conditions are set
sWideCondition	Wide-area event-occurrence condition
bIsVolatile	It indicates volatility or non-volatility of conditions.

8.4 Structure Exclusive to the Access Method

AmShmem Shared Memory Access Method Additional Data Write

Specify the pointer to this structure for the pvAccessMethodSpec argument of the Em_writeMemoryEx() function.

```
typedef struct _am_shmem_write_entry_tag {
    DWORD       version;
    AMSHMEM_WRITE writeEntry;
} AMSHMEM_WRITE_SPEC, *pAMSHMEM_WRITE_SPEC;

typedef struct _am_shmem_write_tag {
    long*         plTime;
    PVOID       pvAddedData;
    WORD    wAddedDataSize;
    AMSHMEM_WRITE_MODE   fMode;
} AMSHMEM_WRITE, *pAMSHMEM_WRITE;
```

Structure Member	Description
version	Specifies the operating version.
plTime	Specifies the time to be saved in the history as a UTC format long value. If NULL is specified, the system time at data write is automatically written.
pvAddedData	Pointer to the additional write data buffer. If it is not needed, specify NULL.
wAddedDataSize	Number of additional write data bytes.
fMode	Specifies the additional data write processing. The following values can be set: AMSHMEM_WRITE_MODE_ADDED = Write additional data. AMSHMEM_WRITE_MODE_HIST = Write additional data history. AMSHMEM_WRITE_MODE_CLEAR = Clear the additional data and history of the specified area.

Description

Writes data to EventMemory with additional data. To read the data back with the additional data, use the Em_readMemoryEx function. To read only the EventMemory data, the Em_readMemory function ca be used.

If AMSHMEM_WRITE_MODE_HIST is specified for fMode, the data before writing is retained with the additional data in a history. If AMSHMEM_WRITE_MODE_CLEAR is specified, the additional and history data for the area specified in psAddre is all deleted. No data is written in this case.

AmShmem Shared Memory Access Method Additional Data Read

Specify the pointer to this structure for the pvAccessMethodSpec argument of the Em_readMemoryEx() function.

```
typedef struct _am_shmem_read_entry_tag {
    DWORD       version;
    AMSHMEM_READ readEntry;
} AMSHMEM_READ_SPEC, *pAMSHMEM_READ_SPEC;

typedef struct _am_shmem_read_tag {
    short       nHistIndex;
    long*       plTime;
    PVOID       pvAddedDataBuff;
    WORD    wAddedDataBuffSize;
    WORD    wAddedDataSize;
    WORD    wCurHistNum;
    AMSHMEM_READ_MODE   fMode;
} AMSHMEM_READ, *pAMSHMEM_READ;
```

Structure Member	Description
version	Specifies the operating version.
nHistIndex	Specifies the history index for the read data. If 0 is specified, it reads the current data. If 1 is specified, it reads the data from the first history level.
plTime	Specifies the time of data write (or the time specified at data write) as a UTC format long value. If it is not needed, specify NULL.
pvAddedDataBuff	Pointer to additional data read buffer. If it is not needed, specify NULL.
wAddedDataBuffSize	Specifies the number of additional data read buffer bytes.
wAddedDataSize	Specifies the number of additional data write bytes.
wCurHistNum	Specifies the current number of history levels saved.
fMode	Specifies the additional data read processing. The following values can be set: AMSHMEM_READ_MODE_NORMAL = Read additional and history data. AMSHMEM_READ_MODE_CLEAR = Clear the additional data and history of the specified area.

Description

Reads EventMemory data with additional data that was written with the Em_writeMemoryEx function. It cannot read data that was written with the Em_writeMemory function. If an address with multiple data is specified, the additional data of the start offset is read.

AmFinsRemote Remote Memory Access Method Additional Data Read/Write

Specify the pointer to this structure for the pvAccessMethodSpec argument of the Em_readMemoryEx() or Em_writeMemoryEx() function.

```
typedef struct _am_fins_write_entry_tag {
    DWORD               version;
    AMFINS_ERROR_INFO   accessError;
} AMFINS_SPEC, *pAMFINS_SPEC;

typedef struct {
    AMFINS_ERROR_TYPE   errorType;
    struct {
        BYTEMRES;
        BYTESRES;
    } finsResp;
    DWORD   apiLastError;
    TCHAR   description[MAX_PATH];
} AMFINS_ERROR_INFO, *pAMFINS_ERROR_INFO;;
```

Structure Member	Description
version	Specifies the operating version.
errorType	Specifies the error type generated.
MRES	Specifies the FINS response code, if the error type is FINS
SRES	communication error.
apiLastError	Specifies the GetLastError code if the error type is API error.
description	Specifies the error message.

Description

Obtains the error data for read/write errors in EventMemories using the AmFinsRemote access method. FINS communication errors generated when using the Em_readMemoryEx() or Em_readMemoryEx() functions can be obtained.