MICROSTAR LABORATORIES™

2265 116th Avenue N.E., Bellevue, WA 98004
Sales & Customer Support: (425) 453-2345
Finance & Administration: (425) 453-9489
Fax: (425) 453-3199
World Wide Web: http://www.mstarlabs.com/

1. Changes to the DAPIO Interface in DAPIO32

Note: This reference document assumes familiarity with the Microstar Laboratories Windows Toolkit and the DAPIO.DLL.

There have been significant changes with the release of the DAPIO32.DLL (32-bit DLL) from its predecessor the DAPIO.DLL (16-bit DLL). Immediately obvious to DAPIO users is the naming convention change. The 32-bit functions use an object oriented naming scheme: Dap<object><action> as in **DapHandleOpen**(). To convert the 16-bit function calls of an application to 32-bit function calls, the symbols must be changed. The association between the function names of the two releases can be found in the Cross Reference section at the end of this document.

Other changes in the new DAPIO32.DLL include the addition of new functions and variants, function removals, and syntax changes. For specific information about the services and their usage, please refer to the Accel32 or DAPIO32 for Windows documentation provided with DAPIO32.

Functional Changes

Changes to the following functions in the DAPIO32.DLL will affect most programs.

DapHandleOpen(. . .) - This function now uses the new UNC (Universal Naming Convention) and requires a pipe direction specification. Handles are no longer bi-directional as with the previous 16-bit DLL, and for this reason, two separate handles must be opened to establish bi-directional communication with the DAP. This is illustrated in the example below which opens handles to the text communication pipe for read and write operations.

```
HDAP hdapSysPut, hdapSysGet;
hdapSysGet = DapHandleOpen("\\\\.\\Dap0\\$SysOut",DAPOPEN_READ);
hdapSysPut = DapHandleOpen("\\\\.\\Dap0\\$SysIn",DAPOPEN_WRITE);
```

Handle names that indicate the direction of the data transfer are recommended to avoid any confusion over the direction of the communication for the handle. For example, **hdapSysGet** retrieves data from the DAP's communication pipe **$SysOut** while **hdapSysPut** transfers data to the DAP.

DapLineGet(. . .) - This function is a replacement for the DAPIO.DLL function **GetDapString**(). **DapLineGet**() gets a string of characters from the DAP with user specified time-out, and returns the number of character read from the DAP excluding line-feed characters.

DapStringGet(. . .) - This function is not the same as the DAPIO.DLL function **GetDapString**(). **GetDapString**() assumes that all carriage-return characters are followed by a line-feed character. The following two lines may be used as a direct replacement for **GetDapString**(). For new code use **DapLineGet**().

```
DapStringGet(hdapSysGet, sizeof(ErrorMsg), ErrorMsg);
DapCharGet(hdapSysGet, TempChar);
```

DapStringPut(. . .) - This function is similar to the 16-bit function call **PutDapString**(), except that a carriage return character (**\r**) must be applied explicitly to the string. The carriage-return must be present for some versions of DAPL to correctly recognize the command. **DapLinePut**() may be a better direct replacement for **PutDapString**() than **DapStringPut**() because it naturally appends a carriage-return to the end of a string. The following two equivalent approaches send a stop command to the DAP.

```
DapStringPut(hdapSysPut, "STOP\r");
DapLinePut(hdapSysPut, "STOP");
```

DapCommandDownload(. . .) - This function now uses a special structure as its parameter and returns TRUE or FALSE upon completion of its task. The **TDapCommandDownload** structure must be fully initialized before a call to **DapCommandDownload**() can be made, as displayed in the following example.

```
TDapCommandDownload dcd;
memset(&dcd, 0,                          sizeof(dcd));
dcd.iInfoSize                            = sizeof(dcd);
dcd.hdapSysPut                           = hdapSysPut;
dcd.hdapSysGet                           = hdapSysGet;
dcd.pszCCFileName                        = pszFile;
dcd.pszCCName                            = NULL;
dcd.iCCStackSize                         = 1000;
if (DapCommandDownload(&dcd))
{...}
```

Implementation Changes

Almost all functions in the new DAPIO32.DLL now return TRUE or FALSE upon completion. TRUE and FALSE are defined in WINDOWS.H or if WINDOWS.H is not used they are defined in DAPIO.H. For this reason, changes should be made to all code that currently checks the return value from a function call. Error messages are now handled through the Win32 API or by calling the function **DapLastErrorTextGet**(). The functions whose responses have changed in the DAPIO32.DLL are listed below with their new return values.

DAPIO32.DLL(32-bit) Function	Return Value	DAPIO.DLL(16-bit) Function	Return Value
DapCharGet	TRUE or FALSE	**GetDapChar**	# Bytes or -1
DapCharPut	TRUE or FALSE	**PutDapChar**	# Bytes or -1
DapConfig	TRUE or FALSE	**ConfigDap**	error codes
DapConfigParamsClear	TRUE or FALSE	**ClearParam**	(none)
DapConfigParamSet	TRUE or FALSE	**SetParam**	(none)
DapConfigRedirect	TRUE or FALSE	**RedirectConfigDap**	(none)
DapHandleClose	TRUE or FALSE	**CloseHandle**	0 or -1
DapHandleOpen	handle or 0	**OpenHandle**	handle or -1
DapInputAvail	# Bytes or -1	**GetDapAvail**	# Bytes or -1
DapInputFlush	# Bytes or -1	**FlushDap**	0 or -1
DapInt16Get	TRUE or FALSE	**GetDapInt**	# Bytes or -1
DapInt16Put	TRUE or FALSE	**PutDapInt**	# Bytes or -1
DapLinePut	TRUE or FALSE	**PutDapString**	# Bytes or -1
DapOutputSpace	# Bytes or -1	**PutDapAvail**	# Bytes or -1
DapStringPut	TRUE or FALSE	**PutDapString**	# Bytes or -1

New DLL Functions

The following functions are available only with the new DAPIO32 library. Additionally, the three functions: **DapComPipeCreate**, **DapComPipeDelete**, and **DapHandleQuery** are available only with the Accel32 (Windows NT) driver. Please refer to the 32-bit DLL documentation to obtain syntax and usage information for these functions.

DapComPipeCreate(. . .)	**Accel32 (WinNT) only**
DapComPipeDelete(. . .)	**Accel32 (WinNT) only**
DapHandleQuery(. . .)	**Accel32 (WinNT) only**
DapBufferGetEx(. . .)	
DapInputFlushEx(. . .)	
DapLastErrorTextGet(. . .)	
DapLinePut(. . .)	
DapStringFormat(. . .)	
DapInt32Get(. . .)	
DapInt32Put(. . .)	

Replaced Function

The function **GetAccelVersion**() has been replaced by a new function **DapHandleQuery**() in the DAPIO32 library. **DapHandleQuery**() provides the same information that **GetAccelVersion**() did in the DAPIO.DLL. Please refer to the 32-bit DLL documentation for information on how to use **DapHandleQuery**().

DAPIO.DLL(16-bit) Function	DAPIO32.DLL(32-bit) Function
GetAccelVersion()	**DapHandleQuery**()

Obsolete Functions

The following functions have become obsolete and have no replacements in the DAPIO32.DLL. Most are low level ACCEL device routines that provide information beyond the need of the typical user or whose functionality has been provided by other routines.

ReadIoCtl(. . .)
WriteIoCtl(. . .)
SetDapInput(. . .)
SetDapOutput(. . .)
IsVxdRunning(. . .)
DownloadList(. . .)

Furthermore, downloading a list of custom commands is no longer supported in the DAPIO32.DLL. Instead, multiple calls to **DapCommandDownload**() can be used to load more than one custom command.

Import Libraries

The DAPIO32.DLL conforms to the Win32 API convention for selecting between ANSI and Unicode versions of functions. When using the import libraries included with DAPIO32, understanding this convention is not necessary. The DAPIO32.DLL does not support Unicode character mapping. However, developers needing to generate their own import library must understand the mapping scheme used in the DAPIO32.DLL.

Functions which support ANSI characters end with 'A', while functions which support Unicode end with 'W'. The base function name is then mapped to the correct version of the function based on a pre-processor symbol. For example, **DapConfig**() is mapped by the DAPIO32.H C header file to **DapConfigA**() using the ANSI convention. Following is a listing of the DAPIO32.DLL functions and structures and their actual ANSI mapping names.

Function Name	Maps To	Actual Function Name
DapCharGet	⇒	**DapCharGetA**
DapCharPut	⇒	**DapCharPutA**
DapCommandDownload	⇒	**DapCommandDownloadA**
DapComPipeCreate	⇒	**DapComPipeCreateA**
DapComPipeDelete	⇒	**DapComPipeDeleteA**
DapConfig	⇒	**DapConfigA**
DapConfigParamSet	⇒	**DapConfigParamSetA**
DapConfigRedirect	⇒	**DapConfigRedirectA**
DapHandleOpen	⇒	**DapHandleOpenA**
DapHandleQuery	⇒	**DapHandleQueryA**
DapLastErrorTextGet	⇒	**DapLastErrorTextGetA**
DapLinePut	⇒	**DapLinePutA**
DapStringFormat	⇒	**DapStringFormatA**
DapStringGet	⇒	**DapStringGetA**
DapStringPut	⇒	**DapStringPutA**

Structure Name		Actual Structure Name
TDapCommandDownload	⇒	**TDapCommandDownloadA**
TDapHandleQuery	⇒	**TDapHandleQueryA**

Cross Reference

The following table relates the DAPIO (16-bit) library functions with the new DAPIO32 (32-bit) functions. The function calls are arranged alphabetically by their DAPIO.DLL name.

DAPIO.DLL (16-bit) Function	DAPIO32.DLL (32-bit) Function
ClearParams	**DapConfigParamsClear**[1]
CloseHandle	**DapHandleClose**[1]
ConfigDap	**DapConfig**[1]
DownloadCmd	**DapCommandDownload**[2]
DownloadList	(none)
FlushDap	**DapInputFlush**[1]
GetAccelVersion	**DapHandleQuery**[1]
GetDapAvail	**DapInputAvail**[1]
GetDapBuf	**DapBufferGet**
GetDapChar	**DapCharGet**[1]
GetDapInt	**DapInt16Get**[1]
GetDapString	**DapStringGet**[2]
IsVxdRunning	(none)
OpenHandle	**DapHandleOpen**[1 2]
PutDapAvail	**DapOutputSpace**[1]
PutDapBuf	**DapBufferPut**
PutDapChar	**DapCharPut**[1]
PutDapInt	**DapInt16Put**[1]
PutDapString	**DapLinePut1** **DapStringPut**[1 2]
ReadIoCtl	(none)
RedirectConfigDap	**DapConfigRedirect**[1]
SetDapInput	(none)
SetDapOutput	(none)
SetParam	**DapConfigParamSet**[1]
WriteIoCtl	(none)

[1] The function's return values have changed with the DAPIO32.DLL library release.
[2] There is a change in the usage of the function call.