TECHNICAL NOTE 7: ABOUT THE COMM AND THE IECPT

This note describes the interface between the protocol translator (PT) and the controlling station communications hardware, with specifics regarding the IECPT. ASE calls the communications interface the COMM. The COMM is a set of APIs that the user develops according to the specifications provided in this section. (*ASE Protocol Translator Interface Specification* provides detailed information about the COMM.)

Assigning the COMM Handle

The user application is responsible for opening the communications port. The way the user application opens a communications port is completely transparent to the PT; no COMM API supports this function. However, the PT does require a COMM handle.

The user application software assigns a user-defined COMM handle to the PT, using the GPT property IPROP_GPT_COMM_HANDLE. The COMM handle typically points to information about the communications port. In multi-line configurations, the COMM handle usually identifies the communications port that a particular instance of the PT uses. The PT passes the COMM handle to the COMM APIs whenever it calls them.

ReadComm

The PT uses ReadComm to retrieve a PT-specified number of bytes from the communications port. Because the requested number of bytes may be less than the entire message, the user I/O software must continue to receive and buffer incoming data until it receives the complete message. ReadComm does not wait for the I/O to complete; instead, it returns immediately. ReadComm does not queue multiple requests; if a read is already active, it returns an error (GPT_ERROR_IO_ACTIVE).

The user application maintains the status of the last read. If the read does not complete by the time specified by the timeout argument, the read completes with an error (GPT_ERROR_TIMEOUT). A timeout argument of zero indicates no timeout (read status will never be GPT_ERROR_TIMEOUT). The PT uses GetComm to get I/O status.

The PT may optionally supply a pointer to a timestamp buffer. If this argument is present, the user application writes the time that it receives the first byte of the message into the timestamp buffer. The timestamp must be expressed in milliseconds.

```
COMAPI      ReadComm(portHandle,buf,count,timeout,timestamp)

HANDLE      portHandle   /* User-defined COMM handle    */
PVOID       buf          /* Pointer to PT data buffer   */
WORD        count        /* Number of bytes to read     */
DWORD       timeout      /* I/O timeout in milliseconds */
LPGPTTIME   timestamp    /* Pointer to timestamp        */
```

Return code	Description
=0	Success
>0	Request completed; the return value indicates the number of bytes in the PT data

	buffer.
<0	GPT_ERROR_IO_ACTIVE. Read request has already been queued. GPT_ERROR_TIMEOUT. Read did not complete in the specified timeout period.

WriteComm

The PT uses WriteComm to send messages to the controlling station. WriteComm does not wait for the I/O to complete; instead, it returns immediately. WriteComm does not queue multiple requests; if a write is already in progress, WriteComm returns an error (ERROR_IO_ACTIVE).

The user application maintains the status of the last write. If the write does not complete by the time specified by the timeout argument, the write completes with an error (GPT_ERROR_TIMEOUT). A timeout argument of zero indicates no timeout (write status will never be GPT_ERROR_TIMEOUT). The PT uses GetComm to get I/O status.

```
COMAPI      WriteComm(portHandle,buf,count,timeout)

HANDLE      portHandle      /* User-defined COMM handle      */
PVOID       buf             /* Pointer to PT data buffer     */
WORD        count           /* Byte count of PT data buffer  */
DWORD       timeout         /* I/O timeout in milliseconds   */
```

Return code	Description
=0	Success.
>0	Request completed; the return value indicates the number of bytes in the PT data buffer.
<0	GPT_ERROR_IO_ACTIVE. Write request has already been queued. GPT_ERROR_LINE_BUSY. Another station is using the line. GPT_ERROR_TIMEOUT. Write did not complete in the specified timeout period.

WaitComm

WaitComm checks for I/O completion events. It allows the PT to perform concurrent reads and writes using ReadComm and WriteComm. The PT calls WaitComm to see if an I/O has completed. WaitComm returns control back to the PT on any of the following events:

- A read (initiated by a ReadComm call) has completed successfully or with error.
- A write (initiated by a WriteComm call) has completed successfully or with error.
- The timeout argument specified in the WaitComm call has expired with no I/O completion.

The PT passes WaitComm a pointer to an event bit mask. WaitComm sets the read bit (EVT_READ_COMPLETE) if an outstanding read event has completed, and sets the write bit

(EVT_WRITE_COMPLETE) if an outstanding write event has completed. If both an outstanding read and write have completed, WaitComm sets both bits.

WaitComm interprets the value of the timeout argument (expressed in milliseconds) as follows:

Timeout argument	Interpretation
=0	Don't wait; return immediately, whether or not I/O has completed.
>0	Wait for the specified number of milliseconds. Set the I/O completion event mask or return GPT_ERROR_TIMEOUT if no I/O has completed.

Note: *The user can use WaitComm to perform background processing. See Section 5, "Assigning GPT Properties," and Section 6, "Running the Protocol Translator," of* ASE Protocol Translator Interface Specification *for more information.*

```
COMAPI      WaitComm(portHandle,lpEvent,timeout)

HANDLE      portHandle   /* User-defined COMM handle     */
LPDWORD     lpEvent      /* Pointer to event mask        */
DWORD       timeout      /* Time to wait in milliseconds */
```

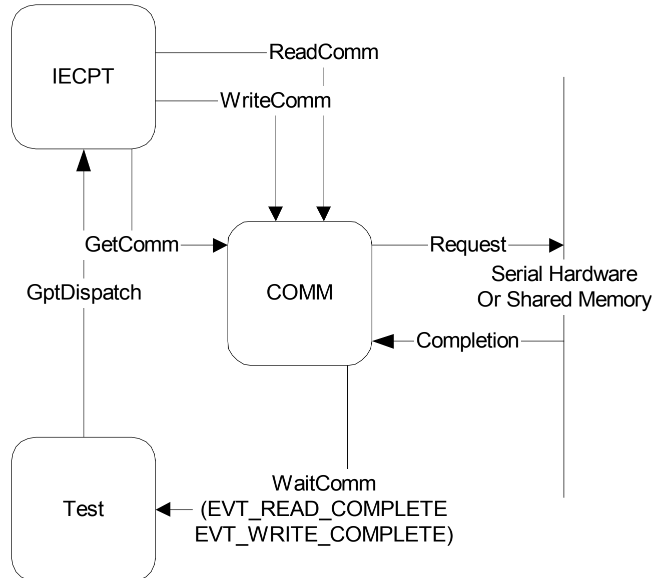
Return code	Description
=0	Success, I/O event detected.
<0	GPT_ERROR_TIMEOUT. No I/O completed in the specified timeout period.

The following sample code illustrates how the PT calls WaitComm:

```
WriteComm(comHndl,...);
ReadComm(comHndl,...);

while(WaitComm(comHndl, &event, 500 ) == GPT_ERROR_TIMEOUT )
{
        /* do some housekeeping every half-second */
}

if ( event & EVT_READ_COMPLETE)
{
        if (GetComm(comHndl,IOC_RXSTATUS) < 0)
                printf("Read failed\n");
        else
                processRead(GetComm(comHndl,IOC_RXLEN));
}

if ( event & EVT_WRITE_COMPLETE)
{
        if (GetComm(comHndl,IOC_TXSTATUS) < 0)
                printf("Write failed\n");
        else
                processRead(GetComm(comHndl,IOC_TXLEN));
```

}

The following figure illustrates the relationship between the IECPT and the user-supported COMM modules.



The figure illustrates the following about the IECPT in either balanced or unbalanced mode:

- When the IECPT needs to read bytes from a remote station, it calls ReadComm to initiate the read. When the IECPT wants to write bytes to a remote station, it calls WriteComm to initiate the read. Neither of these services block, but instead, set up a read or write on the user communication hardware.
- If a read is not active, the user application continues to buffer characters received on the input port.
- The test program that is included in the product package (test.c) checks for I/O completion events by calling the COMM routine WaitComm. WaitComm returns an event mask indicating which pending I/O have completed: EVT_READ_COMPLETE or EVT_WRITE_COMPLETE.
- The IECPT never issues multiple reads or writes. Once a read (ReadComm) or a write (WriteComm) is posted, the IECPT waits for the event EVT_READ_COMPLETE or EVT_WRITE_COMPLETE before issuing another I/O. The IECPT can have both a read and write pending, but it never has more than one read or write pending. To unblock the read/write request, the user sets the corresponding EVT_READ_COMPLETE or EVT_WRITE_COMPLETE bit.
- When the test program detects a COMM completion event, it calls GptDispatch to pass the events on to the IECPT.
- The IECPT confirms the status of the read request by calling GetComm. For reads, the IECPT reads the COMM properties IOC_RXSTATUS and IOC_RXLEN to check the status of the read and the number of bytes read. If IOC_RXSTATUS < 0, then the read

has failed. For writes, the IECPT reads the COMM properties IOC_TXSTATUS and IOC_TXLEN to confirm the status of the write and the length of the write. If IOC_TXSTATUS < 0, the write has failed.
- Test.c does not call ReadComm/WriteComm. The IECPT calls these. Test.c calls WaitComm to check for read/write completion events.
- The communication hardware is transparent to the IECPT. This interface is maintained in the COMM layer. It is the responsibility of the user (COMM) to notify the IECPT when the read/write operations complete.

If the COMM layer does not support the following (logic errors result):

- COMM should never generate a completion event (EVT_READ_COMPLETE or EVT_WRITE_COMPLETE) if a read or write is not active, as illustrated in the following:

```
Gpt()
{
         /* The GPT never has multiple reads outstanding */
         GPTASSERT( read_active == FALSE );
         rc = Call ReadComm(…);
         if ( rc >= 0 ) read_active = TRUE;
}

GptCompletion ( event )
{
   If ( Event & EVT_READ_COMPLETE )
   {
         GPTASSERT( (read_active == TRUE),
               GPT_INTERNAL_LOGIC_ERROR);
         Read_active = FALSE;
         ProcessRead()
   }

}
```

- If the COMM aborts a read or write, a completion event should never be generated for the I/O. With the example above, if the call to ReadComm fails (rc < 0), then the GPT assumes that the read is not active. COMM cannot generate a EVT_READ_COMPLETE for the I/O.