

## Technical Note 16: About Master Translation Mode Requests

### TECHNICAL NOTE 16: ABOUT MASTER TRANSLATION MODE REQUESTS

In Master Translation mode, the GPT supports a variety of logical device requests (also called commands) that span a broad range of RTUs. Each protocol component, such as the IECPT, is free to map the request type to any of the operations supported by the protocol. The following table defines the logical device commands supported by the GPT master.

Attribute	Descriptions
Command	Types of Command: <b>Initialize.</b> Initialize a remote device. <b>Counter Control.</b> Freeze counter, reset counter, or freeze and reset counter. <b>Poll.</b> Poll or interrogate in the background, that is, read analog, binary, and counter input data from a remote device. <b>Exception Poll.</b> Poll (foreground), read events from a remote device. <b>Clock Sync.</b> Synchronize time. <b>Read.</b> Read specified variables. <b>Write.</b> Write specified variables. <b>Protocol-specific.</b> Perform protocol-specific function.

The following table shows the relationship between the GPT logical device commands and IEC ASDUs.

GPT Logical Device Command	Equivalent IEC ASDU or Data Link Request
Initialize	IEC reset link data-link request
Counter Control	IEC Counter Interrogate ASDU
Poll	IEC General Interrogate ASDU
Exception Poll	IEC Class 1 or Class 2 request
Clock Sync	IEC Clock Sync ASDU
Read	IEC Read ASDU
Write	IEC Command ASDUs
Protocol Specific.	IEC Test ASDU

Request processing proceeds as follows:

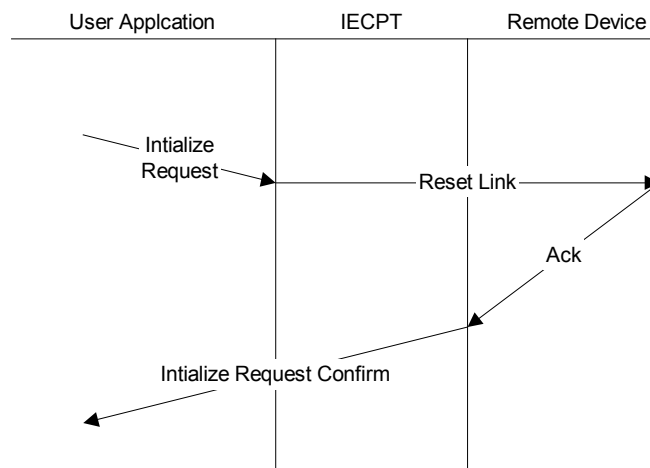
1. The user application formats a device command and provides any required arguments in the request. The user assigns a unique ID or identification to the request.
2. The protocol-specific PT translates the logical request into one or more protocol-specific operations on a device. For example, a control request (logical Write command) may require that the protocol component build separate select and operate messages that are transmitted to the device.

## Technical Note 16: About Master Translation Mode Requests

- The protocol component traces the progress of a request. When a user request completes either successfully or with error, the PT writes the request back to user. The PT communicates the final result of the request to the user application, using the unique identification that the user assigned to the request. Most requests complete synchronously: The user issues the request, the PT sends the request to the remote device, and then returns the result of the operation to the user application. However, some protocols, like IEC, use activation. When a request is activated in the remote device, it may run asynchronously with other commands that the PT sends to the device. For these protocols, the PT hangs onto the request until the remote device terminates the activation. When the operation completes, the PT returns the final result to the user application.

The following figures illustrate some simple sequences among the user application, the IECPT, and the remote device when the IECPT is running in master mode.

### Remote Device Initialization



These are the steps illustrated in the diagram:

- The user application builds an Initialize request.

The request can be built as follows:

```
REQUESTINPUT.Id = transaction number
REQUESTINPUT.Function = DAC_REQ_POLL
REQUESTINPUT.Value.Device = IEC device to which the request is sent
```

*Note:* The user application assigns the transaction. The GPT does not use it. When the request completes, the GPT writes the request back to the user application with a result code. The user application can use the transaction number to identify which request completed.

- The IECPT translates the Initialize request into a Reset Link service, which it transmits to the remote device.
- The remote device acknowledges the reset link.

## Technical Note 16: About Master Translation Mode Requests

- The IECPT confirms the Initialize by writing the original Initialize request back to the user application with a successful result code.

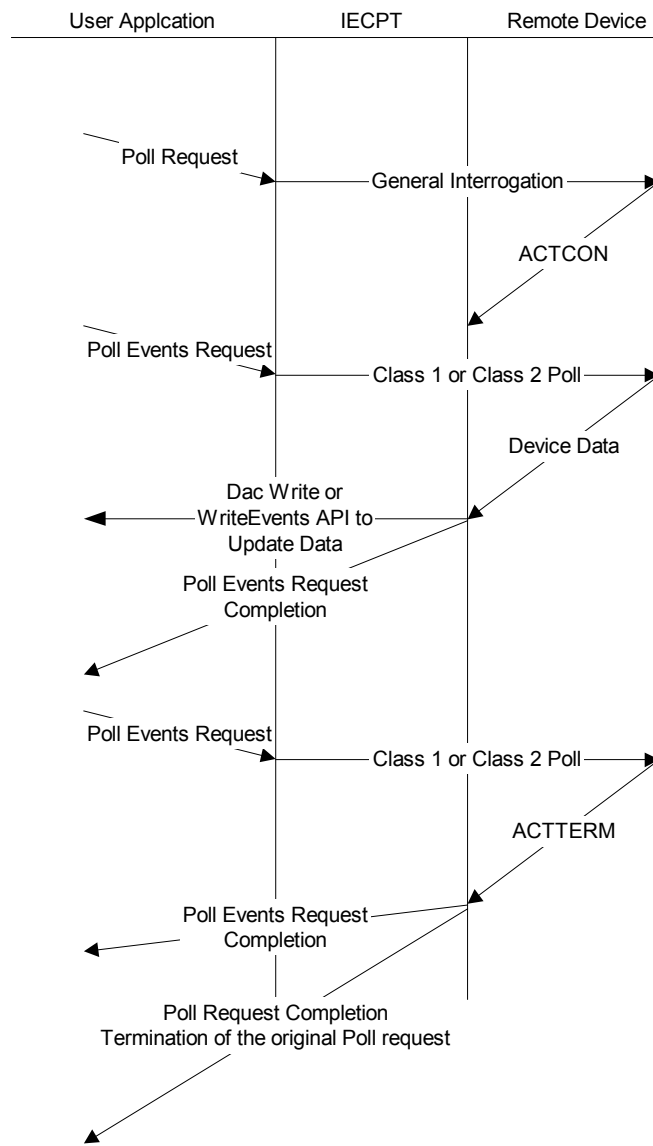
RequestWriteHandler

```

/* The running request is located by its transaction sequence number */
FindRequest( REQUESTINPUT.ID)

If (REQUESTINPUT.Result > 0 ) Then
    Retry(REQUESTINPUT)
End
    
```

### Remote Device Interrogation



## Technical Note 16: About Master Translation Mode Requests

These are the steps illustrated in the diagram:

1. The user application builds a request to poll (interrogate data in the remote device).
2. The IECPT translates this request into a G.I. ASDU. The interrogation is running in the device.
3. The user application builds a Poll Events request to perform class 1 or class 2 data scans as appropriate. The IECPT writes the data returned by the remote device to the user application using DAC Write or WriteEvents APIs on the DAC objects.
4. The event polling continues until the device receives the G.I. ACTERM. At this point, the IECPT writes both the Poll Events request and the original Poll request back to the user with the appropriate completion status. The interrogation has terminated.

### Static and Event Requests

Requests are a DAC object. Like all DAC objects, the Request object supports static and event data. The user typically configures static requests to run at some frequency. The user generates event requests in response to an event, for example, a request to control a single-bit output.

Static requests are configured in an array. In the sample code included in the product package there is one static request created for each device in the system. The request is a Poll Events request and is scheduled to run at some frequency. The following table shows how static requests are setup.

Request Id	IEC DEVICE	Request	Frequency
0	ID of Device 1	DAC_REQ_POLL_EVENTS	0
1	ID of Device 2	DAC_REQ_POLL_EVENTS	3

Terms are defined as follows:

- **Request ID.** Assigned by the user application to identify the request.
- **IEC Device.** ID of the device to which the request is directed.
- **Request.** DAC Request to transmit. DAC\_REQ\_POLL\_EVENTS = Class 1 or Class 2 poll.
- **Frequency.** Frequency in seconds at which the request should be sent. 0 = continuously or as fast as possible.

The GPT uses the following DAC APIs to process static requests:

- **ObjGet.** The GPT uses this API with the property IPROP\_DACOBJ\_STATIC\_COUNT to retrieve the number of static requests configured by the user.
- **Read.** The GPT uses this API to read a static request (REQUESTINPUT). The GPT executes any static requests that have the quality REQUESTINPUT.Status = DAC\_RQ\_QUALITY\_TRIGGERED set.

## Technical Note 16: About Master Translation Mode Requests

- **Write.** The GPT uses this API to write the result of the request execution. If REQUESTINPUT.Result > 0, the request failed; if = 0, the request succeeded. The quality DAC\_RQ\_QUALITY\_TRIGGERED is cleared until the next execution cycle.

The user maintains event requests in a queue, and only queues them as required. For example, the test software included in the product package queues the following event requests:

- **Poll.** General Interrogation.
- **Counter Control.** Freeze, Reset, or Integrated Totals Interrogation.
- **Clock Sync.** Clock Sync.
- **Initialize.** Reset Link.

The GPT uses the following DAC APIs to process event requests:

- **ReadEvents.** The GPT uses this API to read the next event request (REQUESTINPUTEVENT) from the request queue.
- **Ack.** The GPT uses this API to indicate that the request is being processed.
- **WriteEvents.** The GPT uses this API to write the result of the request execution. If REQUESTINPUT.Result > 0, the request failed; if = 0, the request succeeded.

### About DACREQ.C

The module DACREQ.C has been re-written to make request processing easier to follow. DAC requests support static data and events just like any other DAC module. For protocols like IEC, the DAC Request module uses static data to transmit basic Class1 and Class2 scan requests. These requests are performed in the background when no other activity is taking place. DAC Request uses events to transmit activations, such as general interrogations, counter interrogations, and commands. The protocol component configures the DAC request. For IEC, configuration is done in IECCFG.C. The following sections discuss how the DAC request module is initialized.

### *DAC Request Class*

The DAC Request class is defined in the module DACVAR.H (vendor\include). There are several methods supported by the class, such as Umode, Initialize, Poll, Counters, PollEvents, and Command. These are the routines that build the requests that the GPT transmits. They are defined as methods to the DAC Request Class, so that the user application can override these methods to provide a protocol specific implementation.

```
CLASS( DacRequest, DacVariable )

    DACENV          m_Env;          /* DAC Context          */
    GPTBOOL         m_Running;
    GPTBOOL         m_Unsolicited; /* Umode/unbalanced mode */
    GPTWORD         m_Configuration; /* Events supported     */
    GPTDWORD        m_Timers[REQ_MAX_EVENTS];
    GPTDWORD        m_Modifiers[REQ_MAX_EVENTS];
    GPTDWORD        m_Frequency;
    DACOBJREF       m_DevCount;     /* Devices on line     */
    GPTDWORD        m_Sequence;     /* Transaction sequence */
    GPTQUEUE        m_Queues[REQ_MAX_QUEUES];
    DacReq          m_Global;
```

## Technical Note 16: About Master Translation Mode Requests

```
pDacReq      m_Requests;      /* Global line requests */
pDacReq      m_Context;      /* Current device context */

VTABLE( DacRequest, DacVariable )

/* Configuration & Setup */
DACAPI (*Setup)( DacRequest, int, GPTDWORD, GPTDWORD );

/* Online Processing */
DACAPI (*Start)( DacRequest );
DACAPI (*Stop) ( DacRequest );
DACAPI (*Work)( DacRequest );

/* Request Handlers */
DACAPI (*Umode)( DacRequest );
DACAPI (*Initialize)( DacRequest );
DACAPI (*Poll)( DacRequest );
DACAPI (*Counters)( DacRequest );
DACAPI (*PollEvents)( DacRequest );
DACAPI (*Command)( DacRequest );
DACAPI (*ClockSync)( DacRequest );

/* Support */
DACAPI (*Find)( DacRequest, GPTBYTE, DACID, pDacReq * );
DACAPI (*Switch)( DacRequest, GPTBYTE, pDacReq );

METHODS( DacRequest )
    DacRequest DacRequestNew( DacRequest, GPTBYTE, GPTBYTE, GPTBYTE, DACOBJREF, GPTDWORD
);
END_CLASS
```

Following is the protocol-specific definition for IEC in `vendor\include\daccfg.h`:

```
CLASS( IecRequest, DacRequest )

    IecCommand m_Cmd;

VTABLE( IecRequest, DacRequest )
METHODS( IecRequest )
END_CLASS
```

This definition allows the IEC protocol to override any of the general request methods supported by the base class `DacRequest`.

The following section of code defines an IEC-specific implementation of the `DacRequest` class. In addition this implementation overrides the `DacRequest` command method and provides an IEC-specific implementation of this method.

```
static DACAPI locCommand( IecRequest );

/* Override the standard Command Handler in Master Mode */
BEGIN_VTABLE( IecRequest, DacRequest )

    VMETHOD( DacRequest, Command ) = (DACAPI (*) (DacRequest)) locCommand;

END_VTABLE
```

### ***IecRequestNew***

This section steps through the IEC configuration of requests and explains how the configuration is performed. The module `IecRequestNew` in `\vendor\config\ieccfg.c` performs this configuration. This module has the following structure:

```
/* *****\
 * @func HANDLE | IecRequestNew | Request constructor
```

## Technical Note 16: About Master Translation Mode Requests

```
*
*   Construct device requests for master translation mode.
*
* @parm IecRequest      | this      | configuration object pointer
* @parm GPTBYTE        | mode   | GPT_MASTER or GPT_RTU
* @parm GPTBYTE        | line   | line the protocol is assigned to
* @parm GPTBYTE        | protocol | protocol PT_IEC
* @parm GPTBYTE        | balanced | balanced or unbalanced mode.
*/

IecRequest IecRequestNew( GPTBYTE mode, GPTBYTE line, GPTBYTE protocol, GPTBYTE balanced )
{
    IecRequest      r;
    DACOBJREF       background;

    r = NEW( IecRequest );
}
```

The balanced flag is checked. If balanced is true, then no background requests to perform Class 1 or Class 2 polling are configured. If balanced is false, one background request for each device is configured. The frequency at which the background polls will be sent is LOCAL\_CFGOPT\_BACKGROUND\_FREQUENCY.

```
background = ( balanced ) ? 0 : LOCAL_CFGOPT_DEVICE_COUNT;

DacRequestNew( &r->super, mode, line, protocol, background,
               LOCAL_CFGOPT_BACKGROUND_FREQUENCY );

/* Build requests for device */
if ( mode == GPT_MASTER )
{
}
```

Next, the requests are setup. First come the configuration requests. Configuration requests are run whenever the remote device restarts or the system starts up. The CONFIG\_INIT event sends a RESET\_LINK request to the remote device.

```
/* Configuration Events */
(METHOD( r, DacRequest, Setup ), REQ_EVENT_CONFIG_INIT, 0, 0 );
(METHOD( r, DacRequest, Setup ), REQ_EVENT_CONFIG_POLL, 0, 0 );
(METHOD( r, DacRequest, Setup ), REQ_EVENT_CONFIG_POLL_COUNTERS, 0,
         DAC_REQ_MODIFIER_FREEZE | DAC_REQ_MODIFIER_RESET | DAC_REQ_MODIFIER_POLL );
(METHOD( r, DacRequest, Setup ), REQ_EVENT_CONFIG_CLOCK_SYNC, 0,
         DAC_REQ_MODIFIER_CHAN_DELAY );
```

Next, global requests are setup. Global requests are sent periodically to all remote devices on a circuit. The number after the event is the frequency at which the request is sent in seconds. Following the frequency is the modifier. The modifier allows standard requests to be changed. For example, REQ\_EVENT\_POLL\_COUNTERS is modified to freeze and poll the counter data.

```
/* Global line Events */
(METHOD( r, DacRequest, Setup ), REQ_EVENT_GLOBAL_FREEZE, 60, 0 );
(METHOD( r, DacRequest, Setup ), REQ_EVENT_GLOBAL_CLOCK_SYNC, 60, 0 );
```

Next, device requests are setup. Device requests are transmitted to a specific remote device. The setup is the same as the above. The setup includes an execution frequency or request modifier.

```
/* Device Events */
(METHOD( r, DacRequest, Setup ), REQ_EVENT_DEVICE_POLL_COUNTERS, 60,
```

## Technical Note 16: About Master Translation Mode Requests

```
        DAC_REQ_MODIFIER_FREEZE| DAC_REQ_MODIFIER_POLL );
(METHOD( r, DacRequest, Setup ), REQ_EVENT_DEVICE_CLOCK_SYNC, 60,
        DAC_REQ_MODIFIER_CHAN_DELAY );
(METHOD( r, DacRequest, Setup ), REQ_EVENT_DEVICE_POLL, 60, 0 );

/* For Testing periodically send a command */
(METHOD( r, DacRequest, Setup ), REQ_EVENT_DEVICE_COMMAND, 10, 0 );
}
```

Finally, request overrides are defined. In this example, the IEC configuration module overrides the Command request method.

```
        VHOOK( r, IecRequest );

        return r;
}
```

### *IEC Commands*

This section looks at the local code in IECCFG.C that overrides the standard DAC Request method for transmitting commands. On entry, the locCommand module is passed a handle to the IecRequest class.

```
/******\
* @func HANDLE | locCommand | Construct an Iec Command
*
* Construct a command in master mode
*
* @parm IecRequest | this | configuration object pointer
*/
static DACAPI locCommand( IecRequest this )
{
    pDacReq rp = this->super.m_Context;
```

The purpose of this module and all request handlers is to format a REQUESTINPUT structure. This is what the GPT uses to transmit requests to a remote device. Each request handler contains a pointer to a REQUESTINPUT structure by looking at the context pointer in the DacRequest Class.

```
memset( &this->m_Cmd, 0, sizeof(this->m_Cmd) );
```

The REQUESTINPUT structure now contains a pointer to arguments used in the DAC\_REQ\_WRITE\_VARIABLES request. In previous releases, the GPT would call the DAC Static Write API to obtain these parameters. Now the user application can pass the arguments as part of the request. Since the write is to the IEC single-point variables, the DAC BINARYOUTPUT structure is used to hold the arguments. The following code initializes the BINARYOUTPUT structure. A pointer to this structure and the number of structures passed in the request are loaded into the request later in the code.

**Note:** *These arguments cannot be released by the user application until the request terminates. That is, if dynamic memory is used for the request, this memory cannot be released until the request completes.*

```
/* Setup Request arguments */
this->m_Cmd.Cmd.B.Id = 4501;
this->m_Cmd.Cmd.B.Value.Bo.u8 = 1;

/* Setup write request */
rp->Rq.Id = this->super.m_Sequence++;
```

## Technical Note 16: About Master Translation Mode Requests

```
rp->Rq.Function = DAC_REQ_WRITE_VARIABLES;
rp->Rq.Value.ObjId = TEC_SINGLE_COMMAND;
rp->Rq.Value.Modifier = DAC_REQ_MODIFIER_SELECT_EXECUTE;
rp->Rq.Value.Qualifier = DAC_REQ_QUALIFIER_ID_COUNT;
rp->Rq.Value.Start = 4501;
rp->Rq.Value.Count = 1;

rp->Rq.Value.Args = &this->m_Cmd.Cmd;
rp->Rq.Value.ArgCnt = 1;

return TRUE;
}
```