1.3.2 DB_GET_ID

NAME

db_get_id -- "Get an Item Identifier"

SYNOPSIS

```
uint db_get_id ( item_id, &ret_id, class, arg )

        uint item_id;     /* Previous item_id */
                          /* 0 requests first item */
        uint ret_id;      /* Returned item_id - returned by this call */
        uint class;       /* Class of item */
        uint arg;         /* Argument as defined by class */
```

DESCRIPTION

The *db_get_id* directive allows the debug task to receive a unique identifier as defined by *item_id* and *class,* to be returned in *ret_id.*

Item_id must be the unique id of the appropriate type from the list or queue specified by *class,* possibly further qualified by the *arg* parameter. If *item_id* is zero, then an identifier for the first element of the list or queue specified by *class* is returned. If *item_id* is non zero, then the next item past *item_id* is returned in *ret_id.*

Class specifies the list or queue that *item_id* is to be taken from. *Arg* can further specify how the selection is done by selecting a specific list or queue.

Valid class values and the appropriate value for *arg* are given in the following table.

Class Value	Returned item id	Meaning of arg
TASK	task id	
MESSAGE_QUE	message queue id	
SEMAPHORE	semaphore id	
REGION	region id	
PARTITION	partition id	
MESSAGE	message id	message queue id
TASK_IN_MESQ	task id	message queue id
TASK_IN_SEMQ	task id	semaphore id
TASK_IN_SEGQ	task id	region id
SEGMENT	segment id	region id
BUFFER	buffer id	partition id

RETURN VALUE

If *db_get_id* succeeds, the *item_id* for the item in the *class* is returned in *ret_id,* and 0 is returned.

If *db_get_id* succeeds, and there are no more items of the appropriate class, then an error code is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

No more items in this class.

Invalid *class* identifier.

Item_id not in class.

Invalid *arg*.

NOTES

For example, to process a queue, the *get_id* function is called first with a 0 *item_id* to get the first item in the queue. Subsequent calls use the last value of *item_id* in order to get the next item in the queue.

1.3.3 DB_GET_ITEM

NAME

db_get_item -- "Get Information About an Item"

SYNOPSIS

```
uint db_get_item ( item_id, class, buffer, &size )

                    uint item_id;    /* Item_id */
                    uint class;      /* Class of item */
                    char *buffer;    /* address of buffer for returned data */
                    uint size;       /* Size of item - returned by this call */
```

DESCRIPTION

Db_get_item copies an item description into *buffer,* and returns the size of the item description in *size.* The exact format of the data in *buffer* depends on the *class* parameter.

Item_id is a unique identifier for the item within the *class.*

Class specifies the type of item. Valid *classes* are:

Class	returned data
GENERAL	general info block
TASK	task info block
MESSAGE_QUE	message queue info block
MESSAGE	message info block
SEMAPHORE	semaphore info block
REGION	region info block
SEGMENT	segment info block
PARTITION	partition info block
BUFFER	buffer info block

RETURN VALUE

If *db_get_item* is successful, then 0 is returned.

If the call was not successful, an error code is returned.

Buffer is filled in with various structures depending on the *class* parameter. The following information block structures are used:

```
struct  gib     {
        uint    num_tasks;          /* Total number of tasks */
        uint    num_mque;           /* Total number of message queues */
        uint    num_sema;           /* Total number of semaphores */
        uint    num_regions;        /* Total number of regions */
        uint    num_partitions;     /* Total number of partitions */
        uint    num_ready;          /* Sise of ready list */
        uint    num_calls;          /* Total number of RTEID calls made */
        uint    num_inter;          /* Total number of v_returns */
        uint    ticks;              /* Number of ticks on clock */
        uint    min_level;          /* Minimum Processor Mask */
}
```

Figure 1. General Info Block

```
struct  tib     {
        uint    name;               /* Task's name */
        uint    id;                 /* Task's Task id */
        uint    mode;               /* Task's current mode */
        uint    prio;               /* Task's current priority */
        uint    stat;               /* Task's current status */
        uint    events_pending;     /* Events pending for the task */
        uint    events_waiting;     /* Task's event condition from ev_receive */
        uint    signals;            /* Task's pending signals */
        uint    timeout;            /* Task's current timeout value */
        ptf     asr_addr;           /* Task's ASR address */
}
```

Figure 2. Task Info Block

```
struct  mqib    {
        uint    name;           /* Message Queue's name */
        uint    id;             /* Message Queue's id */
        uint    num_mess;       /* Number of messages in queue */
        uint    num_tasks;      /* Number of tasks waiting on messages */
        uint    total_mess;     /* Total messages ever placed in this queue */
        uint    total_urg;      /* Total number of urgent messages */
}
```

Figure 3. Message Queue Info Block

```
struct  message     {
        long        text[4];        /* Message text (16 bytes) */
}
```

Figure 4. Message Info Block

```
struct  smib    {
        uint    name;           /* Semaphore's name */
        uint    id;             /* Semaphore's id */
        uint    value;          /* Semaphore's current value */
        uint    num_tasks;      /* Number of tasks waiting on this Semaphore */
        uint    total_v;        /* Total number of sm_v operations */
        uint    total_p;        /* Total number of sm_p operations */
}
```

Figure 5. Semaphore Info Block

```
struct  rib     {
        uint    name;           /* Region's name */
        uint    id;             /* Region's id */
        uint    page_size;      /* Region's page size */
        uint    paddr;          /* Region's physical start address */
        uint    length;         /* Region's length */
        uint    attributes;     /* Region's attributes */
        uint    num_segs;       /* Number of allocated segments */
        uint    num_tasks;      /* Number of tasks waiting for a segment */
        uint    total_getseg;   /* Total number of rn_getseg */
        uint    total_retseg;   /* Total number of rn_retseg */
}
```

Figure 6. Region Info Block

```
struct  sgib    {
        uint    address;    /* Address of the Segment */
        uint    size;       /* Size of the Segment */
        uint    attrib;     /* Segment Attributes (RDONLY) */
}
```

Figure 7. Segment Info Block

```
struct  pib     {
        uint    name;           /* Name of the Partition */
        uint    id;             /* Id of the Partition */
        uint    bsize;          /* Buffer size */
        uint    bnum;           /* Total number of buffers in the Partition */
        uint    bavail;         /* Number of available buffers */
        uint    paddr;          /* Physical start of the Partition */
        uint    flags;          /* Partitions flags */
        uint    total_getbuf;   /* Total number of pt_getbuf calls */
        uint    total_retbuf;   /* Total number of pt_retbuf calls */
}
```

Figure 8. Partition Info Block