

# Network Layer Routines

## *Calling Sequences*

Aug 3, 1989

### *Low Level Interface Routines*

```
Function NetCnct(taskName: LongInt;  
                queueId: LongInt;  
                eventMask: Integer;  
                VAR taskId: Integer): Integer;
```

Connect this task to the network using the 4-byte `taskName` as the name of the task as known by the network (not necessarily the same task name known to the operating system). The second parameter is the message queue id. Messages received from the network which use the Acnet header SAP and which are destined for the task referred to by `taskName` are passed to the `queueId` message exchange. The third parameter is an optional event mask to be used to signal a task when a message is placed in the queue. The fourth parameter is the returned task id to be used in other network calls.

```
Function NetQueue(taskId: Integer;  
                 VAR msgBlk: Integer;  
                 VAR xmitStat: Integer): Integer;
```

Queue the message in `msgBlk` to the network. This call is always asynchronous. A pointer to `msgBlk` will be queued to the network output queue. The `xmitStat` parameter is initialized to pending on return, and it will be set to indicate completion when the frame containing the message has been transmitted.

```
Function NetSend: Integer;
```

Queued messages are not released to the network right away in order to allow frames to be built from multiple messages. This call flushes all queued messages to the network, insuring that they will be transmitted promptly.

```
Function NetCheck(taskId: Integer;  
                 timeOut: LongInt;  
                 VAR msgRef: MsgRefType): Integer;
```

Check the message queue used by the given task for a message block and return a reference to it if present. If not present, wait with given time-out in 100 Hz ticks.

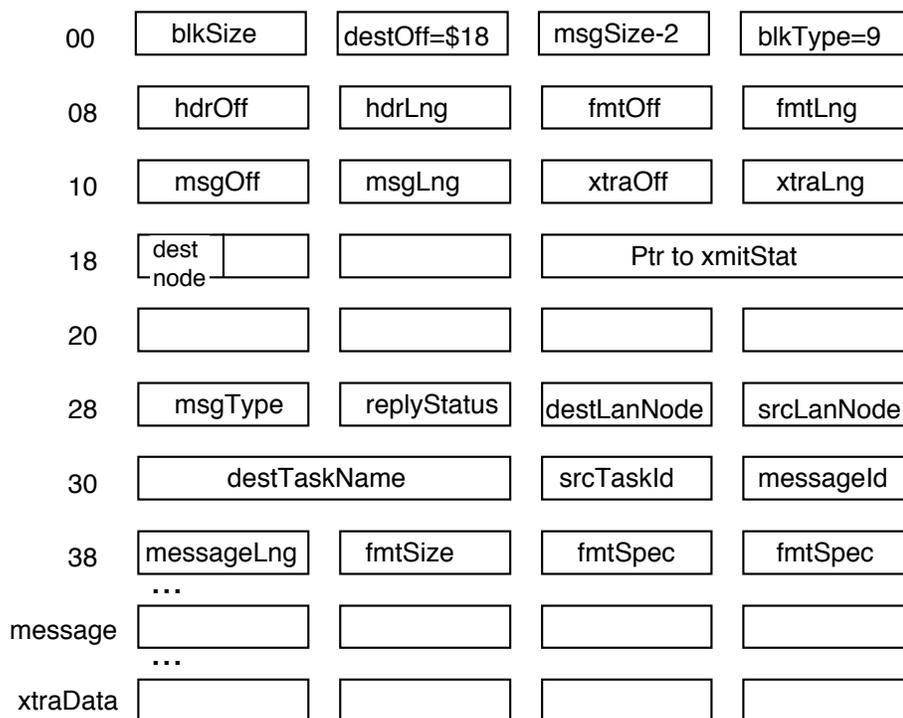
```
Function NetRecv(VAR msgRef: MsgRefType;  
                VAR msg: MsgType;  
                maxSize: Integer): Integer;
```

Copy the received message from the circular buffer to the caller's `msg` buffer with length `maxSize` using the `msgRef` reference block, and decrement the frame's message count word to indicate that the copy of the message in the circular buffer is no longer needed.

```
Function NetDcnt(taskId: Integer): Integer;
```

Disconnect task from the network. Any request messages received directed to the given `taskId` will be ignored, and a reply will be sent to indicate no reply task available.

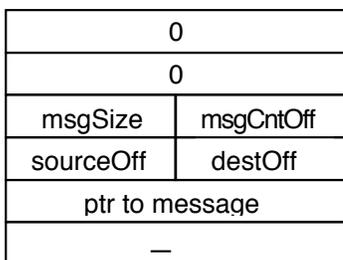
**Message Block Structure**



The blkSize is the length of the entire memory block. The hdrOff, hdrLng, fmtOff, fmtLng, msgOff, msgLng, xtraOff, xtraLng indicate the four components of a network message under Acnet—the Acnet header, the format block, the message itself and an extra component. The entire message is composed of the header block followed by the format block followed by the message block followed by the extra block. These four components are collected together into the frame buffer by NetXmit.

The user need only specify parts of the Acnet header and the four component offsets and lengths. If an offset is negative, then a pointer to the data is expected at -offset bytes from the start of the block. In this way, the entire message need not be put into a single block. The destOff, msgSize-2, blkType, destNode and xmitStat are built by NetQueue. In the case of a request, the srcLanNode and srcTaskId words are also filled in. The component data (or pointers to it) may be placed anywhere in the structure beginning at offset 0x28. Any of the components may be missing, in which case the associated component length is zero.

The msgRef block has the following form, where offsets are referenced to the message ptr.



*Application Program Interface Routines*

In order to present a simple view of the Network Layer to a Pascal application program, the following routines are provided which operate as a layer above the Network Layer. An application program does not conveniently allocate dynamic memory or create message queues. The following four routines invoke the “real” Network Layer interface routines described above to implement a kinder, gentler application interface.

```
Function NetOpen(netName: Longint): Integer;
```

If a queue by `netName` does not exist, create one with 100 entries and in fifo order with unrestricted access. Then invoke `NetCnct` to register the queue to the network using the same name for a `taskName`. (If the `netName` is already registered, `NetCnct` will invoke `NetDcnt` automatically to free it.) The returned `taskId` is not needed by these four routines, as each uses the `netName` to refer to the `NETCT` entry (and the message queue) of interest. The entry is marked with an event number to cause the application task to be promptly invoked upon response to a network message directed to its queue.

```
Function NetWrite(netName: Longint;
                 VAR msg: MsgType;
                 VAR xmitStat: Integer): Integer;
```

Find entry in `NETCT` to get `srcTaskId` to use in case message is a request. Allocate message block from dynamic memory and put a pointer to `msg` into it. There is only one component in this case, the `Acnet` header being the first nine words. The user must prepare part of the header to designate the message type, destination node and task name, message id and size, in the case of a request or a `USM`. For a reply, the user includes the status word; the rest of the header is mostly derived from that received with the request. Invoke `NetQueue` to pass the message block to the network output pointer queue. Then call `NetSend` to flush the queue to the network. The call is always asynchronous. The `xmitStat` variable is likely to indicate “operation pending” upon return. The application may likely exit after this anyway. The `xmitStat` variable can be checked later.

```
Function NetRead(netName: Longint;
                VAR msg: MsgType;
                maxSize: Integer): Integer;
```

Find entry in `NETCT` to get `taskId` to use in call to `NetCheck`. The application program calls this routine in response to its being called with the network event indicated. This happens upon arrival of a message placed into the associated message queue. `NetCheck` is called with no time-out to collect the waiting `msgRef` block that points to the message as it resides in the frame buffer. Check the received size against the `maxSize` argument to insure that the user’s buffer is large enough. Then copy the received message from the frame buffer into the user’s buffer by calling `NetRecv`.

```
Function NetClose(netName: Longint): Integer;
```

Find entry in `NETCT` to get queue id to use to delete message queue. Then call `NetDcnt` to free `NETCT` table entry.

This implementation of application interface routines doesn’t allow for combining multiple messages into network frames. This was done to simplify the interface for the user. If significant use is made of the application interface, this feature may be added.

*Error return codes*

0	No error.	-10	spare
-1	NETCT table doesn't exist.	-11	Bad msgType in Acnet hdr.
-2	Invalid taskId.	-12	Message queue empty.
-3	Task not connected.	-13	Timeout from NetRecv.
-4	Invalid queueId.	-14	Message queue deleted.
-5	NETCT table full.	-15	Queue access violation.
-6	Bad size in msgRef block.	-16	Bad total message size.
-7	OUTPQ full.	-17	No network board.
-8	Can't create message queue.		
-9	Can't allocate message block.		