

# New Technical Notes

Macintosh



®

---

Developer Support

## Token Ring Q&As

### Networking

Revised by: Developer Support Center

May 1993

Written by: Developer Support Center

October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As this month:

NetCopy procedure replaces TokenTalk CopyNuBus

---

### **NetCopy procedure replaces TokenTalk CopyNuBus**

Date Written: 9/23/92

Last reviewed: 3/1/93

The TokenTalk Programmer's Guide describes a routine called CopyNuBus. This routine is not in the IPCGlue.o file dated 3/4/92 that I got off a Developer CD. Has this routine been replaced by something else, and if so, what is the calling sequence of the new routine?

---

The TokenTalk Programmer's Guide hasn't been updated to reflect the fact that the CopyNuBus routine has been replaced by the more robust NetCopy procedure. The prototype for the NetCopy routine is

```
short NetCopy(tid_type srcTID, void *srcAddress, tid_type dstTID, void *dstAddress,  
              unsigned long bytcount);
```

Note the addition of the source and the destination TIDs. NetCopy will copy from a source virtual address to a target virtual address and is a safe way to move data across the NuBus. Both the source and destination virtual addresses can be paged out to disk in a VM environment. NetCopy will cause these pages to be brought back into memory before the copy is performed. A result of zero is returned if NetCopy is successful.



This information comes from the A/ROSE Programmer's Guide. You should get a copy of this guide and referring to it regarding A/ROSE programming questions instead of referencing the TokenTalk Programmer's Guide. A/ROSE has been a fast evolving piece of software. Always refer to the latest Developer CD for the latest revision.

## **TokenTalk LLCBadList and LLCTruncated**

Date Written: 9/24/91

Last reviewed: 9/24/91

When a list-directed receive is queued to TokenTalk's implementation of Logical Link Control (LLC) with a total data size less than the active frame size, LLCBadList is returned when the receive completes. If a nonlist-directed receive is used, and the buffer size is less than a frame size, LLCTruncated is returned. Why aren't the same error codes (LLCTruncated) returned as the condition is the same?

—

The LLCBadList error results because of an undocumented limitation imposed on the number of lists allowed in a list-directed receive. It's likely that you've specified a list array with greater than eight elements. When the list is passed to LLCReceive, a check is made on the list array if one is used. If more than eight elements are in the list, the LLCBadList error is returned immediately with no processing of the incoming data.

On the other hand, if a nonlist-directed receive is used, the LLCReceive call is queued into the task list. The present TokenTalk LLC implementation doesn't support partial reads of large data frames, so if it turns out that the buffer size is smaller than the frame size, only the information that fits into the buffer is returned. The remaining information is discarded. The error condition LLCTruncated is returned to indicate that this event occurred.

## **How to get burned-in & locally administered Token Ring addresses**

Date Written: 12/11/91

Last reviewed: 12/12/91

How can I read the "burned-in" address from the TokenTalk NB card? How about for the locally administered address?

—

To access the burned-in address, use the TTGetDefaultParms selector when calling the TTUtil function pointer. On return, the record structure, TRInit, will contain the value of the "burned-in" address in the field NodeAddr. Use the TTGetBootParms selector to obtain the equivalent information from the TokenTalk Prefs file, that information which has been locally set.

Note the following #defines to include in your TTUtil.h header file:

```
#define TTGetDefaultParms 10 /* Return default TRInit parameters */
#define TTGetBootParms 11 /* Return TRInit parameters to use on next boot */

/*
```

- \* The TTGetDefaultParms and TTGetBootParms returns a pointer to the
- \* parameter structure, or zero if no parameters could be returned. The
- \* parameter structure returned should be released by the caller by calling
- \* DisposPtr when done.

\*/

### **TokenRing NB 4/16 Card and “promiscuous” mode support**

Date Written: 12/6/91

Last reviewed: 1/27/92

Can the IBM chipset which is on Apple’s new TokenRing NB 4/16 Card be programmed to go into “promiscuous” mode? I would like to write a Sniffer application which runs on the new card.

—

At present, the present design of the IBM mini-card is such that the chipset cannot be programmed to run in promiscuous mode. A request has been submitted to IBM to allow the chipset to be placed into this mode. There is no date as to when such functionality will become available.

### **TokenTalk maximum transmit buffer size**

Date Written: 12/19/91

Last reviewed: 2/6/92

In reviewing the 'llcp' resource for the new TokenTalk 4/16 NuBus card, I noticed that the maximum receive frames size has been changed to 0x1170 or 4464 bytes. If this is correct, what is the maximum transmit size (used to be 1509 bytes)?

—

The change is for real. It was implemented at the request of developers like yourself. The change affects all Apple’s TokenTalk products provided that TokenTalk Prep 2.4 is present.

The maximum transmit buffer size depends on the 'llcp' resource in the TokenTalk Prep or (TokenTalk Prefs file if one is available). Within the 'llcp' resource is a buffer size value used to initialize the size of the receive/transmit buffers. For file versions 2.2 and earlier, the factory limit was 1509 bytes. For TokenTalk Prep version 2.4, the buffer size is set to 4464 bytes.