AppleTalk Wide Area Developer's Toolkit

Toolkit Contents and Overview

The AppleTalk Wide Area Developer's Toolkit contains the following items:

- *AppleTalk Update-Based Routing Protocol: Enhanced AppleTalk Routing*

  This document is the protocol specification for the AppleTalk Update-based Routing Protocol (AURP), which provides wide area routing enhancements to the AppleTalk routing protocols. It is useful for reference, and it provides information router developers need to implement AURP.

- *Apple Internet Router: User-Interface Extensions to the adev File*

  This document provides information needed to support the new adev resource calls. (An adev file is a file that provides support for one or more network connections.) These calls include a hierarchical display call and specific user-interface calls for configuration of AURP adevs.

- *Apple Internet Router: Developing an AURP adev File*

  This document provides information needed to develop an AURP adev. Included are the AURP calls to and from the atlk resource; some clues about the adev resource calls from the Apple Internet Router management application, Router Manager; and hints on using the XTI (X/Open Transport Interface) Shell sample code. The XTI Shell sample code provides a base for writing an AURP adev over XTI and explicit instructions on how to modify it to implement your link.

- *Apple Internet Router: Extending IP Tunnel and DialUp*

  This document provides information needed to enhance the AppleTalk/IP Wide Area Extension (IP Tunnel adev) and the DialUp adev to support your MacTCP mdev file or serial driver.

- *Apple Internet Tunnel Simulator and Apple Tunnel Simulator software*

  Apple Tunnel Simulator is an application that simulates an AURP tunnel over TCP/IP. It is useful in testing your AURP implementations. The document *Apple Internet Tunnel Simulator* explains how to use the Apple Tunnel Simulator software.

How to use the Developer's Toolkit

The AppleTalk Wide Area Developer's Toolkit is intended for two kinds of Apple developers: router developers who wish to develop an AURP-speaking router that interoperates with the Apple Internet Router, and adev developers who wish to develop a network connection file for the Apple Internet Router. Figure 1 illustrates the different parts of the toolkit that developers should reference.

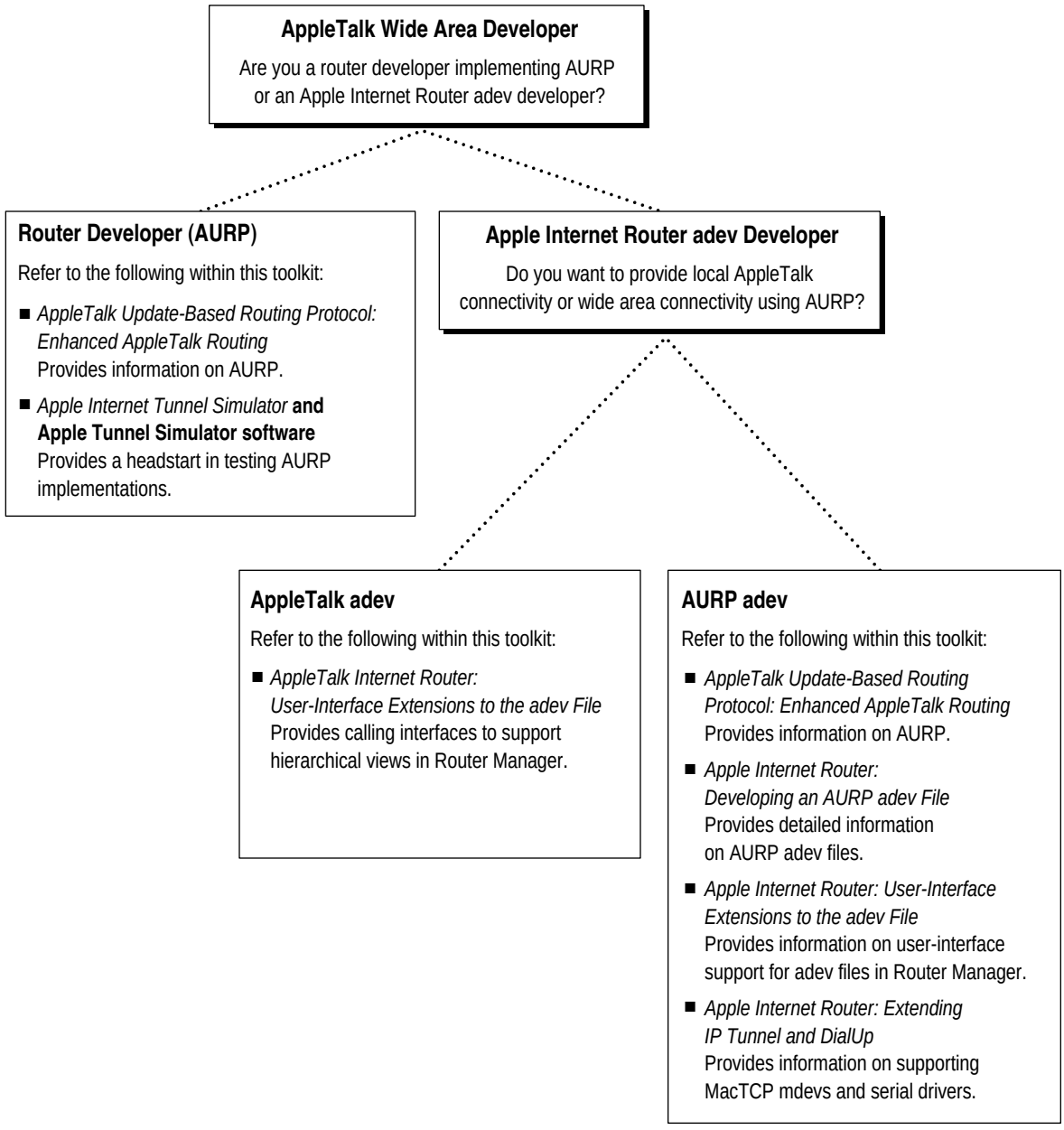


Figure 1 AppleTalk Wide Area Developer's Toolkit road map

Apple Internet Router and adevs

If you are an adev file developer, there are several things you need to consider. As always, you need to implement the standard calls as specified in the *Macintosh AppleTalk Connections Programmer's Guide* (M7056/A). You should fully understand the information in that guide before delving into the materials included in this toolkit. In addition to those standard calls, however, you now have several new options to consider with the Apple Internet Router.

The first of these options is a new feature that allows Router Manager to display adevs in a hierarchical (tree) fashion. The tree levels are device, subport (if any), and method. For instance, within an Ethernet physical port (device), there can be an EtherTalk adev (method) and an IP tunneling adev (method) displayed. Those adevs that do not support this feature will not appear in the Router Manager Setup window in the physical device hierarchy. This option is highly recommended for all adev developers.

The second option is to write an AppleTalk Update-based Routing Protocol (AURP) adev. This enables the Apple Internet Router to provide AppleTalk connectivity by tunneling through a new type of link. The IP Tunnel, X.25, and DialUp adevs are provided with the Apple Internet Router Basic Connectivity Package and Extensions; these tunnel through TCP/IP, X.25, and standard modem phone lines, respectively. Creating new AURP adevs makes it possible to connect remote AppleTalk networks using other types of networks (such as ISDN or T1).

AURP can be used on what are referred to as non-AppleTalk data links. Such links do not provide AppleTalk addressability (that is, they don't have a network range or zone list), although they may provide addressability through another network system (such as TCP/IP). AURP adevs include an atlk resource with significant extensions, providing the functionality needed by the router to implement AURP on top of the adev's data link.

The third option is the ability to append an "extension" to the window used by Router Manager in configuration of a particular port. This window can contain any adev-specific configuration information needed to specify the tunnel fully. AppleTalk adevs should not need to use this feature, because a default window containing the necessary information is provided. This option is provided mainly for AURP adev developers.

Finally, MacTCP mdev developers can easily add their link to the IP Tunnel functionality by creating a resource and adding it to the IP Tunnel adev file. Serial driver developers can also add their link to the DialUp functionality by creating a resource and adding it to the DialUp adev file.

The following table shows which document contains the information needed to accomplish each of the tasks described here.

Task/Option	Document
Writing an AURP adev	Apple Internet Router: Developing an AURP adev File
Displaying adevs in a hierarchical fashion	Apple Internet Router: User-Interface Extensions to the adev File
Creating an adev-specific configuration extension	Apple Internet Router: User-Interface Extensions to the adev File
Updating IP Tunnel or DialUp to support your mdev or serial driver	Apple Internet Router: Extending IP Tunnel and DialUp

Apple Internet Router: User-Interface Extensions to the adev File

Contents

About this document

The user interface for configuring the Apple Internet Router displays available ports in a hierarchical outline similiar to the Finder. To support this, and to allow for the configuration of AppleTalk Update-based Routing Protocol (AURP) ports, the existing adev file-calling interface has been extended. This document describes these extensions.

This document is divided into three parts:

- "Overview" describes the user interface for configuring a router using Router Manager.
- " Supporting the Hierarchical Setup Window" contains all the information needed to develop an AppleTalk adev file.
- "Supporting the Extended Configuration User-Interface Calls" describes the standard routines that have been defined to allow AURP adev file developers to extend the configuration user interface of the Port Info dialog box by displaying port-specific information in the bottom part of the dialog box. *This is required reading for all AURP adev file developers.*

Prerequisite reading

The *Macintosh AppleTalk Connections Programmer's Guide* (available through APDA; order number M7056/A) contains general information about extended and nonextended AppleTalk adev files, the xtlk resource, and other information about AppleTalk programming. Chapter 2 details the procedures for calling adev resources. Make sure you are familiar with the information contained in that guide before proceeding.

Overview

This section presents a general description of the user interface for configuring a router in Router Manager. Detailed step-by-step instructions are provided in the *Apple Internet Router Administrator's Guide.*

Setup window

In Router Manager, configuring a router starts in the Setup window.

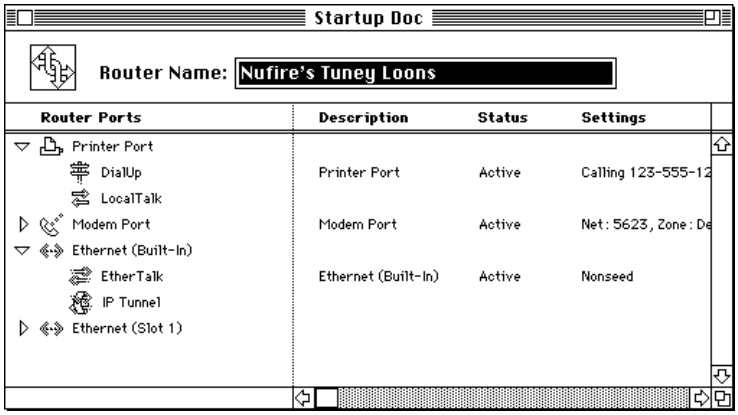


Figure 1 The Setup window

The Setup window contains a hierarchical outline of ports installed on the router Macintosh. This outline consists of a single line item for each physical networking device installed (built-in or slot-based) in the machine. Device-level items can be opened to reveal hardware subdivisions (ports) on the card, if this level is appropriate for the device. The device- and port-level line items can be opened to reveal the link protocols (methods) that are compatible with the physical port.

In the example shown in Figure 1, the printer port has two levels of information: it has a device (Printer Port) and two methods (DialUp and LocalTalk).

The built-in Ethernet port also has two levels of information: it has a device (Ethernet Built-In) and two methods (EtherTalk and IP Tunnel).

An Apple Serial NB Card located in slot 3 of a Macintosh II computer might have three levels of information available: a device (SerialNB slot 3), four ports (1A, 1B, 2A, and 2B), and one method (DialUp).

Port Info dialog box

The method line items can be opened to display a Port Info dialog box that shows the port's current configuration. The Port Info dialog box is divided into two sections. The top section contains information, text fields, and controls common to all port types. The bottom section contains information, text fields, and controls specific to the port type.

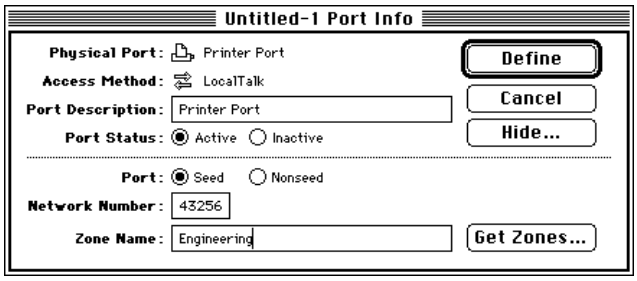


Figure 2 Port Info dialog box (AppleTalk nonextended)

Because the configuration requirements of AppleTalk ports (extended and nonextended) are well known, Router Manager handles the entire Port Info dialog box for these ports on its own. To configure an AURP port, however, Router Manager needs help from the adev file.

Apple has defined a standard set of routines to allow AURP adev file developers to extend the configuration user interface of the Port Info dialog box by displaying port-specific information in the bottom section of the dialog box. Whenever the user opens an AURP Port Info dialog box, Router Manager calls these routines, which allow the user to display and edit port-specific data. The relationship of these routines to Router Manager is roughly analogous to the control panel/cdev file interface, but somewhat more complex due to Router Manager's requirements.

Supporting the hierarchical Setup window

This section provides the information you need to develop an AppleTalk adev file.

xtlk resource

To indicate that an adev file supports the new method-based interface, the xtlk resource, which was introduced in AppleTalk Phase 2, now defines some additional flags. Those adev files that do not contain this resource, or adev files that contain the resource but do not have these flags set (that is, Phase 2 adev files) will be treated as non-method based.

The new xtlk resource format is as follows:

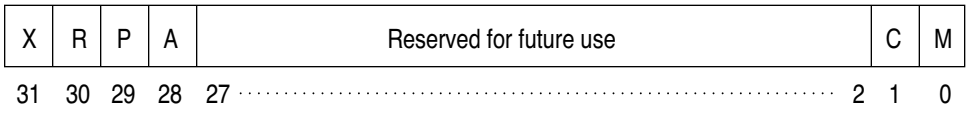


Figure 3 The xtlk resource

The definitions of the xtlk flags are as follows:

31	extendedAddr	For AppleTalk ports: indicates that the port supports Phase 2 addressing.
30	routerOnly	Set by an AURP port to indicate that it is connected to a “Router Only” network (link distance = 0 hops). This flag is usually not set.
29	noRtmpData	Set by ports that implement AURP for transmitting routing data between routers instead of RTMP. This flag should be set for all AURP ports.
28	noAppleTalkPort	AURP ports must set this flag to indicate to Router Manager that they are *not* AppleTalk ports.
27–02		Reserved
01	extendedConfig	Non-AppleTalk method-based ports must set this flag and support the extended configuration user-interface calls from the Router Manager application (see details in “Supporting the Extended Configuration User-Interface Calls” later in this document). For AppleTalk ports, classic and method-based, the configuration user interface is provided by the Router Manager application, and this flag should not be set.
00	methodADEV	Specifies that the adev file supports, as a minimum, the MGetADEV call, making it a method-based adev file. Additionally, AURP adev files must set the extendedConfig flag and support the extended configuration user-interface calls (see details in “ Supporting the Extended Configuration User-Interface Calls” later in this document).

Router Manager tests those adev files that contain an xtlk resource to see if the “method-based adev file” flag is set. If that flag is set, then the adev file supports the MGetADEV method-based adev file call. AppleTalk adev files that support Phase 2 addressing should also set the extendedAddr flag.

AURP adev files should set the noRtmpData, noAppleTalkPort, and extendedConfig flags and support the extended configuration user-interface calls from Router Manager.

Records used by MGetADEV

The old adev file interface allowed for only one global flag (the xtlk resource) and a single icon and string to represent the adev file visually. The extensions to the adev file user interface require additional attributes and visual elements related to the device, port, and method layers supported by the adev file.

The structure used to pass this information between the adev file code (contained in the adev resource) and the calling application is TAdevRec. The Pascal format of this record is:

```
TAdevRec          =  RECORD
     adevRecVers  :  Integer;
     adevId       :  TADevIdRec;
     adevLocals   :  ARRAY [1.. 2] OF LongInt;
     adevDevice   :  TAdevItem;
     adevPort     :  TAdevItem;
     adevMethod   :  TAdevItem;
END;
TAdevRecPtr       =  ^TAdevRec;
```

The adevRecVers field contains the version number of the TAdevRec structure, to detect conflict with future TAdevRec formats. This field is zero for this version of TAdevRec.

The adev file uses the adevId subrecord to store information about a particular port and to match configuration data with a particular adev file. The format of the adevId subrecord is:

```
TADevIdRec      =  RECORD
     aidSize    :  Integer;
     aidData    :  ARRAY [1..8] OF LongInt;
END
TADevIdRecPtr   =  ^TADevIdRec;
```

The aidSize field contains the number of longs in the aidData array. The adevLocals field can be used internally by the adev file. The last three fields, adevDevice, adevPort, and adevMethod, are subrecords that describe the attributes of the device, port, and method levels of the adev file, as well as the icons and text used in representing the adev file visually. The TAdevItem record format is shown in the following display:

```
TAdevItem       =  RECORD
     itemFlags  :  LongInt;
     itemRef    :  LongInt;
     itemIcon   :  Handle;
     itemStr    :  StringHandle;
END;
TAdevItemPtr    =  ^TAdevItem;
```

The itemIcon and itemStr fields contain handles to the icon and string to be associated with the device, port, or method level being displayed. If itemIcon contains a handle to an icon suite (see Tech Note #306: “Drawing Icons the System 7 Way”), then the Router Manager application will draw the icon in color on color monitors.

The itemRef field contains a unique reference code for each device, port, and method returned from the adev file. Using these fields, the Router Manager program can associate link protocol methods from different adev files with the same device- and port-level items.

For example, an EtherTalk adev file would return a TAdevRec containing the following information:

device = ‘Ethernet (slot 1)’; port = *no info*; method = ‘EtherTalk’

An IPTunnel adev file would return a TAdevRec containing:

device = ‘Ethernet (slot 1)’; port = *no info*; method = ‘IP Tunnel’

The adevDevice.itemRef field for ‘Ethernet (slot 1)’ would be the same from both adev files, allowing the Router Manager application to sort both ‘EtherTalk’ and ‘IP Tunnel’ under the same device item.

Similarly, a LocalTalk adev file would return a TAdevRec of:

device = ‘Printer Port’; port = *no info*; method = ‘LocalTalk’

A DialUp adev file would return a TAdevRec containing:

device = ‘Printer Port’; port = *no info*; method = ‘DialUp’

Note that the adevDevice.itemRef field for the Printer Port would be the same from both adev files, allowing the Router Manager application to sort both LocalTalk and DialUp under the same device item.

The reference codes assigned to devices, ports, and methods are administered by Apple Developer Technical Support. Reference codes for existing devices, ports, and methods are defined as follows:

```
{___ Ref codes for itemRef field of TAdevItem record ___}

{___ DEVICES: hi word = hardware, low word = id/slot ___}
kBuiltInRef           = $00010000  {add port id to this number}
kModemPort            = 1          {modem port id}
kPrinterPort          = 2          {printer port id}

kNuBusRef             = $00020000  {add slot # to this number}
kSCSIRef              = $00030000  {add scsi id to this number}

{___ METHODS (use adev res id, or existing ref) ___}
kLocalTalkRef        = 1
kEtherTalk1Ref       = 2
kTokenTalkRef        = 5
kEtherTalk2Ref       = 10
kIPTunnelRef         = 4
kDialUpRef           = 6
kX25TunnelRef        = 25
```

For example, to get your adev file to show up under the printer port, set the itemRef to kBuiltInRef + kPrinterPort. For a slot-based device, set the itemRef to kNuBusRef + slot number (as defined by the Slot Manager in the range $9–$F).

The definition of the itemFlags field will vary, depending on the level of the item (device, port, or method). The general format of the flag word is illustrated in Figure 4.

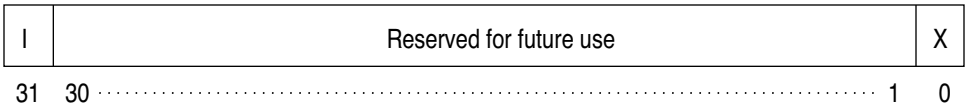


Figure 4 TAdevItem.itemFlags general format

At each level, the general meaning of the flag is the same but only applies to the level of the item being discussed.

31 kAIHasInfo *Definition of adevDevice.itemFlags:* If set to 1, the other device-level fields (itemRef, itemIcon, and itemStr) are significant and should be used to compare and display the device-level information for this adev file. If this level has no significance, the information should be ignored and the item not be displayed.

Definition of adevPort.itemFlags: Same as the adevDevice flag, but pertaining to the port-level fields.This flag is cleared by adev files with single port devices (for example, EtherTalk and TokenTalk).

Definition of adevMethod.itemFlags: Same as the adevDevice flag, but pertaining to the method-level fields.

00	kAIExclusive	*Definition of adevDevice.itemFlags:* If set to 1, the device may not be configured by a different method concurrent with the method for this adev file. Typically used to denote hardware that is incapable of running two data-link protocols on the same device. *Definition of adevPort.itemFlags:* Same as the adevDevice flag, but restricting only the configuration of the port, not the entire device. *Definition of adevMethod.itemFlags:* If set to 1, the method may be configured only once. Typically used for methods like IP Tunnel, which can be configured only once in a router setup document but which show up under multiple devices.
30–01		Reserved.

The kAIExclusive flag is used by Router Manager to arbitrate the configuration of the specific device, port, or method. An adev file would set the device's Exclusive flag if the method required exclusive use of the device, including all available ports on the device. When a user tries to configure a method with this flag set on a device for which another method is already configured (including methods configured on other ports of that device), Router Manager reports the conflict. Similarly, the adev file would set the port Exclusive flag if the method required exclusive use of the port. When a user tries to configure a method with this flag set on a port for which another method is already configured, Router Manager reports the conflict. The adev file would set the method Exclusive flag if the method itself were " exclusive" and could only be configured once.

Note All methods sharing a given device/port level should agree on the exclusivity of that level. To assure that your adev file does not conflict with another third-party developer's adev file, contact Apple Developer Technical Support.

Method-based adev file data interface

All method-based adev files need to support MGetADEV, which provides attribute and icon/string information for all ports supported by an adev file.

MGetADEV (D0=104)

Call D1 (long) Current value of parameter RAM

D2 (long) Value returned from previous MGetADEV call or 0 if this is the first MGetADEV call

A0 (long) Pointer to caller-provided TAdevRec

Return D0 (byte) Status flag

D2 (long) Next value for D2 to call; also used by SelectADEV call

A0 (long) Pointer to TAdevRec (with data filled in by adev file)

MGetADEV is identical to the GetADEV call described in the *Macintosh AppleTalk Connections Programmer's Guide*, with the exception that A0 now points to a TAdevRec to be filled in by the adev file.

In typical usage, after determining that the adev file supports the MGetADEV call, the Router Manager application calls MGetADEV repeatedly until the status flag (D0) returned indicates that the adev file has no additional information. See the definition of GetADEV in the *Macintosh AppleTalk Connections Programmer's Guide* for a general description of this status flag.

Router Manager is responsible for allocating and disposing of the memory for the TAdevRec and passing a TAdevRecPtr to MGetADEV through A0. The adev file should fill in the appropriate TAdevRec fields with data. If icon suites are used, Router Manager also calls DisposeIconSuite (with disposeData set to TRUE) on these handles when it is done with them.

The first MGetADEV call to the adev file contains the current value of parameter RAM in D1 to indicate the currently selected AppleTalk connection, 0 in D2 to indicate that this is the first MGetADEV call, and a pointer to a TAdevRec. The TAdevRec aidSize field is set to 0 initially. The adev file responds to the first MGetADEV call by returning a status-flag value in D0 to indicate whether there are any connections to support.

For AppleTalk adev files, D1 and D2 are used the same way they are used in the GetADEV call as explained in the *Macintosh AppleTalk Connections Programmer's Guide.* AURP adev files can ignore D1 and D2 and use the adevId instead.

Supporting the extended configuration user-interface calls

This section describes the routines that can be used to extend the configuration user interface of the Port Info dialog box in Router Manager.

Figure 5 illustrates a Port Info dialog box for a DialUp port.

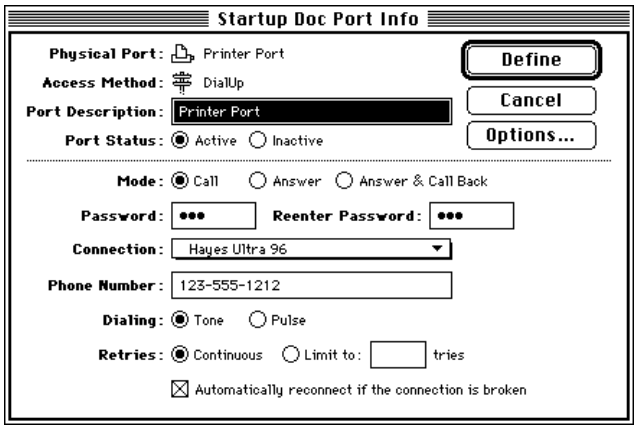


Figure 5 The Port Info dialog box for a DialUp port

AURP adev files are required to provide the configuration user interface. Router Manager leaves the bottom section of the dialog box blank and calls the adev file to fill in the contents of that part of the dialog box.

Because the adev file shares a GrafPort with the Router Manager application, it must cooperate with the application in maintaining the dialog box. This is accomplished through the programmatic interface described in the remainder of this document.

adev file configuration user-interface call overview

The adev file configuration interface is a series of related functions that handle all the events the user generates. The calls fall into three broad categories: initialization/termination, dialog box event processing, and configuration data exchange.

Initialization/termination routines

The following routines are used to create and dispose of the environment used by the adev file during the life of the Port Info dialog box:

MAGetAttribs	Called prior to the creation of the dialog box to determine its size and the size of the configuration record needed by this port.
MAInit	Called to have the adev file allocate and initialize local variables, and to initialize the configuration record for newly created port configurations.
MAKill	Called to have the adev file dispose of local variables.

Dialog box event processing routines

The following routines are used to process events that occur during the life of the Port Info dialog box and are related to the bottom half of the dialog box:

MAActivate	Called whenever the dialog box is activated (becomes the frontmost window) or inactivated (goes behind other windows).
MAClick	Called whenever a mouseDown event occurs in the bottom section of the dialog box. If a user clicks a TextEdit field, activation of the field does not occur until the MASelectTE routine is called (described later).
MADraw	Called whenever an update event occurs.
MAMessage	Called to pass messages to the adev file (similar to an editing command when the user selects an item in the Edit menu).
MAIdle	Called during idle periods to update the visual appearance of the dialog box (flash the insertion bar in text boxes, for instance).
MAKey	Called whenever a keyDown/autoKey event occurs and the adev file has an active TextEdit field. The exceptions to this are the Tab and Return keys, which are used to activate a TextEdit field and click the “default” button. If a TextEdit field will become active when the key is pressed, activation of the field does not occur until the MASelectTE routine is called (described next).
MASelectTE	Called to activate a specific TextEdit field in response to a previous mouseDown or keyDown event. This action is separated from the MAClick and MAKey routines so that the Router Manager can prevent more than one TextEdit field from being active (see the routine for details).

Configuration data exchange routines

The following routines are used to retrieve and set the displayed configuration data:

MAGetConfig	Called to retrieve the current state of the configuration fields, buttons, and so forth. Called whenever the data is to be saved to the Setup document.
MAGetLine	Called to retrieve the current state of the configuration fields, buttons, and so forth in a printable format. Called whenever Router Manager is to print the data.

TAdevWind data structure

Communication of data between the Router Manager application and the adev file is for the most part accomplished though the TAdevWind record structure. The TAdevWind record provides data and events to the various routines and also provides a global environment for the adev file during the life of the Port Info dialog box. The definition of the TAdevWind record (in Pascal) is as follows:

```
TAdevWind   =  RECORD
 awWiPtr : WindowPtr; {dialog where drawing occurs}
 awRect : Rect; {dialog rect where adev file may draw}
 awMessage : LongInt; {event message (varies according to call) }
 awModifiers : Integer; {flags that affect awMessage  }
 awConfig : Handle; {config data filled in by adev file}
 awVars : Handle; {used by adev file's temporary variables}
 awSumStr : Str255; {summary string of current config data}
  awSum2Str  : Str255;   {2nd summary string of config data}
END; {TAdevWind}
TAdevWindPtr = ^TAdevWind;
```

awWiPtr — Points to the WindowRecord of the GrafPort where drawing will occur. The adev file should never directly reference fields in this record. It is provided primarily to be passed into the Control Manager when controls are created so that they may be linked into the current window. (Note that the Router Manager application, not the adev file, will call DrawControls to display all controls linked to the Port Info dialog box).

awRect — Defines the bound where adev file drawing will occur. The adev file should use this rect to determine where to position controls, text fields, and so forth. The adev file should never make any assumptions about the dimensions of this rect. As a general rule, the layout of the adev file configuration section should be designed with a standard compact Macintosh screen (512 × 342 pixels) in mind.

awMessage — Contains a long word value whose meaning is determined by the adev file user-interface routine being called.

awModifiers — Contains an integer value whose meaning is determined by the adev file user-interface routine being called.

awConfig — Contains a handle to the configuration data buffer. The Router Manager application does not inspect the contents of this buffer, but allocates its size according to the buffer size parameter returned from MAGetAttribs(). The adev file receives the initial configuration data (if any exists) though this buffer when the dialog box is opened (MAInit), and returns the current configuration data though this buffer when MAGetConfig() is called. The adev file must never deallocate this handle, but may resize it if needed.

awVars — Defines a field in which the adev file may allocate and store a handle for its local (persistent) variables across calls. The handle may be allocated by the adev file during MAInit() (the field is initially set to NIL by the Router Manager) and must be deallocated by the adev file during MAKill(). The contents of this buffer are never examined by the Router Manager application.

awSumStr — Contains an Str255 string that the adev file uses to provide a descriptive text summary of the configuration data contained in the awConfig handle. The Router Manager application needs this string in order to display decimal summary data for that port (Router Manager does not know what awConfig contains). Router Manager draws this string in the Setup window whenever the configuration data is saved to the Setup window. This string is initially set to blanks by the Router Manager application and must be updated during MAInit() and MAGetConfig(). This field is also used to provide the first column of text to be printed by MAGetLine().

awSum2Str — Contains a second Str255 string that the adev file uses to provide a descriptive text summary of the configuration data contained in the awConfig handle. The Router Manager application needs this string in order to display hexadecimal summary data for that port (Router Manager does not know what awConfig contains). Router Manager draws this string in the Setup window whenever the configuration data is saved to the Setup window. This string is initially set to blanks by the Router Manager application and must be updated during MAInit() and MAGetConfig(). If the adev file does not have a hex mode, this string should be set equal to awSumStr. This field is also used to provide the second column of text to be printed by MAGetLine().

adev file configuration user-interface call syntax

This section contains the assembly language syntax for calling the adev file configuration user-interface routines. Like the existing MGetADEV, GetADEV, and SelectADEV calls, these routines are part of the adev resource of the adev file.

MAGetAttribs (D0 = 106)

Call	A0 (long)	TAdevWind record pointer		
		awRect	→	bottom = Max adev file drawing height
Return	A0 (long)	TAdevWind record pointer		
		awRect	←	bottom = height needed by adev file
		awMessage	←	Minimum awConfig buffer size
	D0 (byte)	Status code:		-1 insufficient drawing height
				0 no errors

This call is made prior to initializing the dialog box to determine the height of the area needed by the adev file for drawing. Upon entry, awRect.bottom contains the maximum available height in the Port Info dialog box for the adev file to draw into. If the available height is not enough to draw the configuration, the adev file returns an error in D0. If it is enough, awRect.bottom should contain the actual height (in pixels) required by the adev file to draw its configuration user interface, and awMessage specifies the minimum size for the awConfig data buffer. This size is used by the Router Manager application to allocate an initial awConfig handle whenever a new port is configured for the first time. The awConfig handle may be resized later by the adev file if the configuration data grows or shrinks during editing.

MAInit (D0 = 107)

Call	A0 (long)	TAdevWind record pointer		
		awWiPtr	→	pointer to Port Info window record
		awRect	→	drawing bounds for adev file
		awModifiers	→	Bit 0: 1 = awConfig needs initialization 0 = awConfig contains valid data Bit 1: 1 = Router active 0 = Router inactive Bit 2: 1 = Display numbers in hex 0 = Display numbers in decimal
		awConfig	→	handle to config data buffer
		awVars	→	nil
		awSumStr	→	" (zero length Pascal string)
		awSum2Str	→	" (zero length Pascal string)
Return	A0 (long)	TAdevWind record pointer		
		awVars	←	handle to adev file's local variables
		awConfig	←	initialized data (if needed)
		awSumStr	←	text summary of config data (decimal)
		awSum2Str	←	text summary of config data (hex)
	D0 (byte)	Status code:		-2 initialization error reported by adev file -1 initialization error not reported by adev file 0 no errors

This routine is called for initialization prior to editing the configuration data. If this is a new port configuration, then awModifiers Bit 0 will be set to signal the adev file that the awConfig handle points to uninitialized data, which the adev file should then set up with default values. When the Router is running awModifiers, Bit 1 will be set. The adev file may also allocate a handle to a local set of variables for its own internal use and store the handle in awVars (Router Manager never references this handle). The adev file must also initialize awSumStr and awSum2Str with a text summary of the configuration data in awConfig. (Note: Make sure you set awSum2Str, even if your adev file doesn't have a hex mode.)

If an error occurs during initialization, return an error code based on whether or not you have reported the error to the user. If you have not reported the error, Router Manager will put up a dialog box for you.

MAKill (D0 = 108)

Call	A0 (long)	TAdevWind record pointer		
		awVars	→	handle to adev file's local variables
		awModifiers	→	Bit 0: 1 = Cannot fail
				0 = Can fail
Return	A0 (long)	TAdevWind record pointer		
		awVars	←	NIL
	D0 (byte)	Status code:		-1 can't close at this time
				0 no errors

This routine is called to terminate editing of the configuration data. The adev file must release any local storage allocated during the life of the dialog box and return a NIL value in awVars. If awModifiers is set to 1, then the Router Manager application is indicating that the adev file will not be called again for this dialog box, even if it returns a failure code in D0. In the current version of Router Manager, this bit is always set, because MAGetConfig is called before MAKill if data is going to be saved. Returning an error from MAGetConfig will cancel the closing of the dialog box.

Note This call must not change the contents of the awConfig buffer or the awSumStr field, because MAGetConfig has already been called.

If, for some reason, you want the adev file to keep the dialog box open and awModifiers is not set, it must return an error status in D0. Except as noted previously, Router Manager will keep the dialog box open in that case.

MAActivate (D0 = 109)

Call	A0 (long)	TAdevWind record pointer		
		awWiPtr	→	pointer to Port Info window record
		awRect	→	drawing bounds for adev file
		awMessage	→	Bit 0: 1 = activating the dialog box 0 = inactivating the dialog box
		awConfig	→	handle to config data buffer
		awVars	→	handle to adev file's local variables
Return	D0 (byte)	Status code:		-1 fatal error(s) occurred 0 no errors

This routine is called whenever the dialog box is made active (becomes the front window) or inactive (goes behind other windows). On entry, awMessage contains a flag: Bit 0 indicates whether the dialog box is being activated (1) or deactivated (0). If the router is active and the Port Info dialog box is in the front window, the adev file may need to change the appearance of various fields in its drawing area to indicate that certain editing functions are disabled.

During this routine, the adev file should change the visual appearance of items to reflect the state of the dialog box. If the adev file currently has an active TextEdit field, its status must be changed to make the flashing insertion bar appear or disappear.

This routine should always return a status code of zero (no errors).

MAClick (D0 = 110)

Call	A0 (long)	TAdevWind record pointer		
		awWiPtr	→	pointer to Port Info window record
		awRect	→	drawing bounds for adev file
		awMessage	→	mouse location of click (Point)
		awModifiers	→	Bit 0: 1 = Shift key pressed 0 = unshifted
		awConfig	→	handle to config data buffer
		awVars	→	handle to adev file's local variables
Return	A0 (long)	TAdevWind record pointer		
		awMessage	←	adev file's TextEdit field ID code
		awModifiers	←	Bit 0: 1 = configuration has changed 0 = no changes made Bit 1: 1 = message OK, TextEdit ID valid 0 = ignore this message Bit 2: 1 = adev file TE has selection 0 = adev file has no selection
	D0 (long)	Status code:		-1 fatal error(s) occurred 0 no errors

This routine is called whenever a mouseDown event occurs in the adev file's portion of the Port Info dialog box. On entry, the mouse location is contained in the awMessage field (it must be cast as a Point record type), and awModifiers contains a flag: Bit 0 indicates the status of the Shift key (1=shift pressed, 0=unshifted).

If the mouseDown event causes the contents of the configuration data to change, the adev file should return a 1 in awModifiers; otherwise, it should be set to 0.

If the click was on an inactive TextEdit field, the adev file should return its nonzero ID for that field in awMessage, and then set or clear the changed bit and the message OK bit in awModifiers. Assuming no other errors occur, the Router Manager application then follows up with a MASelectTE() call (described later in this document) to activate the TextEdit field. If no TextEdit field was clicked, or if an already active TextEdit field was clicked, the adev file should clear Bit 1 of awModifiers. Before returning from this call, the adev file should also deactivate the current TextEdit field.

If this call leaves text selected, it should set Bit 2 of awModifiers so that Router Manager will know to enable the menu items Cut, Copy, and Clear.

Note The adev file must not assume that the TextEdit field will be activated when returning a nonzero field ID. If the Router Manager portion of the dialog box has an active TextEdit field at the time, it may fail to deactivate this field due to a data validation error.

This routine should always return a status code of zero (no errors).

IMPORTANT *Editing while a router is active* The general rule is that no changes in the configuration data should be allowed while the router is active. MouseDown events that are used to inspect the configuration data are permitted, but mouseDown events that alter the configuration data should be disabled.

MADraw (D0 = 111)

Call	A0 (long)	TAdevWind record pointer		
		awWiPtr	→	pointer to Port Info window record
		awRect	→	drawing bounds for adev file
		awConfig	→	handle to config data buffer
		awVars	→	handle to adev file's local variables
Return	D0 (byte)	Status code:		-1 fatal error(s) occurred
				0 no errors

This routine is called whenever an update event occurs. The adev file should use this call to draw all of its visual elements.

Note Because all controls belong to the Port Info dialog box, the Router Manager application will call DrawControls(). The adev file should not make this call.

This routine should always return a status code of zero (no errors).

MAMessage (D0 = 112)

Call	A0 (long)	TAdevWind record pointer		
		awWiPtr	→	pointer to Port Info window record
		awRect	→	drawing bounds for adev file
		awMessage	→	Edit event code: 0 = Undo (not implemented yet) 1 = Cut 2 = Copy 3 = Paste 4 = Clear 5 = Router active 6 = Router inactive 8 = Adjust cursor
		awConfig	→	handle to config data buffer
		awVars	→	handle to adev file's local variables
Return	A0 (long)	TAdevWind record pointer		
		awModifiers	←	Bit 0: 1 = configuration has changed 0 = no changes made Bit 2: 1 = adev file TE has selection 0 = adev file has no selection
	D0 (byte)	Status code:		-1 fatal error(s) occurred 0 no errors

This routine is called for one of three reasons: (1) the Router is being activated or deactivated; (2) an Edit menu command (or its keyboard equivalent) was selected and the adev file has an active TextEdit field; and (3) the cursor needs to be adjusted. When editing occurs, on entry awMessage contains a code that specifies which edit function has been selected: 0=Undo, 1=Cut, 2=Copy, 3=Paste, and 4=Clear. When the Router is being activated or deactivated, awMessage indicates the new Router state: 5=Active, 6=Inactive. When the cursor needs to be adjusted, awMessage is set to 8, and the adev file should get the current mouse location and set the cursor.

If the Edit command results in changes to the configuration data, the adev file should return a 1 in Bit 0 of awModifiers; otherwise, it should return 0.

If this call leaves text selected, it should set Bit 2 of awModifiers to 1 so that Router Manager will know to enable the menu items Cut, Copy, and Clear.

This routine should always return a status code of zero (no errors). If an error is returned, Router Manager will ignore the return bits but will not close the dialog box.

MAIdle (D0 = 113)

Call A0 (long) TAdevWind record pointer

awWiPtr	→	pointer to Port Info window record
awRect	→	drawing bounds for adev file
awConfig	→	handle to config data buffer
awVars	→	handle to adev file's local variables

Return D0 (byte) Status code: -1 fatal error(s) occurred
0 no errors

This routine is called for periodic updating of the dialog box when no other events occur. The adev file should use this time to call TEIdle (if it has an active TextEdit field) or any other periodic function required.

This routine should always return a status code of zero (no errors).

MAKey (D0 = 114)

Call A0 (long) TAdevWind record pointer

awWiPtr	→	pointer to Port Info window record
awRect	→	drawing bounds for adev file
awMessage	→	key character code
awModifiers	→	Bit 0: 1 = Shift key pressed 0 = unshifted
awConfig	→	handle to config data buffer
awVars	→	handle to adev file's local variables

Return A0 (long) TAdevWind record pointer

awMessage	←	adev file's TextEdit field ID code (Tab key events only)
awModifiers	←	Bit 0: 1 = configuration has changed 0 = no changes made Bit 1: 1 = message OK, TextEdit ID valid 0 = ignore this message Bit 2: 1 = adev file TE has selection 0 = adev file has no selection

D0 (byte) Status code: -1 fatal error(s) occurred
0 no errors

This routine is called whenever a keyDown or autoKey event occurs and the adev file has an active TextEdit field. On entry, awMessage contains the character code for the key that was pressed, and awModifiers contains one flag: Bit 0 indicates the status of the Shift key (1=Shift pressed, 0=unshifted). Command keys are *never* passed to this routine. For most keyDown events, the adev file will pass the character code to the current TextEdit field by means of a call to TEKey(). The notable exception to this rule is the Tab key. Processing this key event is described later in "Text Field Selection Using the Tab Key." On exit you should return "message OK" or "ignore message" by setting or clearing Bit 1 of the awModifiers field. No message indicates the returned TextEdit ID should be ignored.

IMPORTANT *Editing while a router is active* The general rule is that no changes in the configuration data should be allowed while the router is active. Key events that are used to inspect the configuration data are permitted, but key events that alter the configuration data should be disabled.

Text field selection using the Tab key A Tab key event indicates the user wishes to select the next TextEdit field. If the adev file portion of the dialog box contains more than one TextEdit field, then it is up to the adev file to determine which field is "next." As a rule of thumb, the next TextEdit field is to the right of (or below, if there is no field to the right) the current field activated for editing. If the Shift key is pressed (test the awModifiers field for Shift key status), then the next field is to the left of (or above, if there is no field to the left) the current field.

Only one TextEdit field may be active in the Port Info dialog box at any one time. Initially, the Router Manager portion of the dialog box contains the active field. When the Tab key is pressed, the Router Manager will determine if it needs to activate one of the fields in its portion of the dialog box. If the Router Manager is positioned at its "last" field (in a given direction of tabbing), then it will pass the Tab key to MAKey(). If the adev file has no TextEdit fields, then it returns a value of zero in awMessage. If it has one or more TextEdit fields, then it returns a nonzero ID to the Router Manager, which will later be passed back to the adev file in MASelectTE() (described next). The ID may be any integer value that the adev file uses to distinguish one field from another.

Note The adev file must not assume that the TextEdit field will be activated when returning a nonzero field ID. If the Router Manager portion of the dialog box has an active TextEdit field at the time, it may fail to deactivate this field due to a data validation error.

Note If the adev file has no currently active TextEdit field and it receives a Tab to activate one of its fields, it should examine awModifiers to see if the Shift key is pressed. If the Shift key is pressed, the adev file should activate its "last" field; otherwise, it should activate its "first" TextEdit field.

If the adev file receives a Tab and its “last” field (in the direction specified by the Shift key status) is the currently active field, then it signals that it is releasing control of text editing back to Router Manager by returning a value of zero in awMessage. When returning control back to Router Manager, the adev file must ensure that all its TextEdit fields are deactivated (no flashing insertion point or selection range). If the adev file cannot deactivate its TextEdit field for some reason (for example, a data validation error), then it should pass the current (nonzero) field ID back to Router Manager. Router Manager will then call MASelectTE() with that ID.

If this call leaves text selected, it should set Bit 2 of awModifiers to 1 so that Router Manager will know to enable the menu items Cut, Copy, and Clear.

The routine should always return a status code of zero (no errors). If an error is returned, Router Manager will ignore the return bits but will not close the dialog box.

MASelectTE (D0 = 115)

Call A0 (long) TAdevWind record pointer

awWiPtr	→	pointer to Port Info window record
awRect	→	drawing bounds for adev file
awMessage	→	adev file’s TextEdit field ID code (from MAClick or MAKey routines)
awConfig	→	handle to config data buffer
awVars	→	handle to adev file’s local variables

Return A0 (long) TAdevWind record pointer

awModifiers	←	Bit 2: 1 = adev file TE has selection 0 = adev file has no selection

D0 (byte) Status code: -1 cannot deactivate the current TE field
0 no errors

This routine is called to activate the adev file TextEdit field indicated by the ID number returned from MAClick() or MAKey() (described previously). If the ID is zero, then the adev file must deactivate its current TextEdit field and relinquish control of text editing back to Router Manager. If the current TextEdit field cannot be deactivated for some reason (for example, a data validation error), then return the status code -1; otherwise, return a status code of 0 (no errors).

If this call leaves text selected, it should set Bit 2 of awModifiers to 1 so that Router Manager will know to enable the menu items Cut, Copy, and Clear.

MAGetConfig (D0 = 116)

Call	A0 (long)	TAdevWind record pointer		
		awWiPtr	→	pointer to Port Info window record
		awRect	→	drawing bounds for adev file
		awConfig	→	handle to old config data buffer
		awVars	→	handle to adev file's local variables
		awSumStr	→	text summary of old config data (decimal)
		awSum2Strr	→	text summary of old config data (hex)
Return	A0 (long)	TAdevWind record pointer		
		awMessage	←	adev file's TextEdit field ID code
		awModifiers	←	Bit 0: 1 = configuration has changed 0 = no changes made Bit 1: 1 = message OK, TextEdit ID valid 0 = ignore this message Bit 2: 1 = adev file TE has selection 0 = adev file has no selection
		awConfig	←	updated configuration data buffer
		awSumStr	←	updated text summary of config (decimal)
		awSum2Str	←	updated text summary of config (hex)
	D0 (byte)	Status code:		-1 invalid configuration field data 0 no errors, save changes

This routine is called to retrieve the current contents of the adev file configuration editing fields before the dialog box is closed. If one or more fields contain invalid data, then return an error code in D0; otherwise, return a status code of zero (no errors). When an error is returned, the configuration dialog box is not closed.

Make sure to set all the bits in awModifiers correctly, especially when returning an error.

Note If your adev file does not have a hex mode, awSum2Str should be set equal to awSumStr.

MAGetLine(D0 = 117)

Call A0 (long) TAdevWind record pointer

awWiPtr	→	nil
awRect	→	undefined
awMessage	→	'line'
awModifiers	→	line number (1,2...n)
awConfig	→	handle to config data buffer
awVars	→	nil

Return A0 (long) TAdevWind record pointer

awSumStr	←	updated text for column 1 of line
awSum2Str	←	updated text for column 2 of line

D0 (byte) Status code:
-1 no more lines to print after this one
0 more lines to come

This routine is called to retrieve the current contents of the adev file configuration in printable format. The adev file is responsible for determining the text to return for column 1 of a printed page in awSumStr, and similarly, if desired, for column 2 in awSum2Str. If more lines remain to be printed, then return zero in D0. When a status code of –1 is returned, there are no more lines to print and this routine is not called again.

Apple Internet Router: Developing an AURP adev File

Apple Computer, Inc.

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

About this document

This document is intended for developers who wish to write an Apple Update-based Routing Protocol (AURP) adev file. As with any adev file, an AURP adev includes two main resources: the adev resource and the atlk resource.

When developing the adev resource, refer to the *Macintosh AppleTalk Connections Programmer's Guide* (available through APDA; order number M7056/A), and the *Apple Internet Router: User-Interface Extensions to the adev File* document in this toolkit. Note that since an AURP adev is a non-AppleTalk adev, it should not appear in the Network cdev. Therefore, calls used explicitly by the Network cdev (GetAdev and SelectAdev) should be implemented but only need to return an error. The XTI Shell sample code briefly covers the adev resource, as does "Developing an adev File Using XTI" section of this document.

This document includes detailed information on developing the atlk resource. Developers are encouraged to develop an AURP adev that uses XTI, the X/Open Transport Interface, to ease the transition to Apple's upcoming Transport Independent Interface (TII). Developers can combine link-specific XTI libraries with a modified XTI adev shell to create an AURP adev that uses XTI. It is envisioned that in the future, the XTI libraries will continue to work with TII with little work required on the developer's part.

In some cases, a developer may not need to implement an AURP adev as described in this document. These cases are:

- If the developer wishes to support AURP over TCP/IP over the developer's data link (for example, SMDS or ISDN) and the developer already has a MacTCP mdev, the developer can use the Apple-provided IP tunneling adev that ships with the Apple Internet Router AppleTalk/IP Wide Area Extension and need only insert certain resources into this adev (see the *Apple Internet Router: Extending IP Tunnel and DialUp* document in this toolkit for details).
- If the developer wishes to support AURP over dial-up modems over alternate serial ports (for example, on a four-port card) and the developer already has an .AIN/.AOUT style driver, the developer will be able to use the DialUp adev provided by Apple. No documentation on this is available, however, as it is not currently supported by the DialUp adev version that ships with the Apple Internet Router Basic Connectivity Package.

Calls to and from the atlk resource

An AURP atlk interfaces with the AURP module within the Apple Internet Router. The programming interface resembles the standard atlk interface, but contains enhancements that facilitate use of AURP (see Figure 1). For more information on AURP, refer to the *AppleTalk Update-Based Routing Protocol: Enhanced AppleTalk Routing* document.

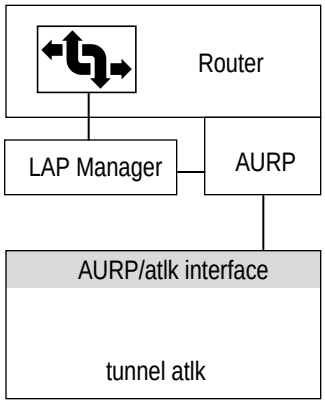


Figure 1 Interface block diagram

This document describes the AURP programming interface (assembly and C calling conventions) and the responsibilities of the AURP atlk code. The atlk is responsible for handling the data link: setting up any data-link connections necessary, and reading, writing, and shutting down the data-link connections if necessary. Procedural access to the AURP module is provided dynamically when the atlk is called initially to install itself. AURPInterface.h, a C include file, and AURPEqu.a, an assembly equates file, contain defines and prototypes for the AURP interface. Routine names preceded by 'A' or 'ATnl' reside in the atlk and are calls made by the AURP module to the atlk; those preceded by 'AURP' reside in the AURP module and are calls made by the atlk to the AURP module.

In order for the router to identify an atlk as an AURP atlk, certain flags must be set in the adev file's xtlk resource. The xtlk flags are defined as follows in Rez format:

```
type 'xtlk' { /* extended AppleTalk adev flags */
      Boolean noExtendedAddr, extendedAddr;
      Boolean noRouterOnly, routerOnly;
      Boolean rtmpData, noRtmpData;
      Boolean appleTalkPort, noAppleTalkPort;
      fill bit[26];
      Boolean noExtendedConfig, extendedConfig;
      Boolean noMethodADEV, methodADEV;
};
```

To identify itself as an AURP atlk (that is, a port that does not use RTMP and is not an AppleTalk port), an AURP atlk must set the flags for noRtmpData and noAppleTalkPort. The xtlk would be defined in Rez format as:

```
resource 'xtlk' (-4032) {
   extendedAddr,     /* bit 31 set = extended address ADEV */
   noRouterOnly,     /* bit 30 clear for no router only */
   noRtmpData,       /* bit 29 set = no RTMPs sent to ADEV */
   noAppleTalkPort,  /* bit 28 set = ADEV does not use AT
                        addresses */
   extendedConfig,   /* bit 1 set = expanded config support */
   methodADEV        /* bit 0 set = supports MGetADEV */
};
```

For more information about the xtlk resource, see the Macintosh AppleTalk Connections Programmer's Guide, and the Apple Internet Router: User-Interface Extensions to the adev File document in this toolkit.

Definition of terms

configuration record Information referenced by the awConfig structure as described in detail in the *Apple Internet Router: User-Interface Extensions to the adev File* document of this toolkit. The adev builds this configuration record from user settings made in the adev's Port Info dialog box under Router Manager.

adev signature Data referenced by the adevID structure as described in detail in the *Apple Internet Router: User-Interface Extensions to the adev File* document in this toolkit. The adev creates its signature and may embed machine-specific port configuration information in the signature.

indicator A four-byte (long) reference to the actual address of an entity that the atlk is sending packets to or receiving packets from (also known as the *next internet router* or *next IR*). The atlk is responsible for defining a value for the indicator that is unique on that port for the given entity.

Overview of AURP interface calls

Administrator action in Router Manager	*AURP call triggered*
Start Router	AInstall, ATnlConnect
Stop Router	ATnlDisconnect, AShutdown
Activate Port	ATnlConnect
Inactivate Port	ATnlDisconnect

When the router is starting up, the AURP module makes the AInstall call so that the atlk may initialize and register its write and readDone routines. The AURP module makes the ATnlConnect call to set up any necessary data-link connections.

If the router wishes to close any open connections, the AURP module makes the ATnlDisconnect call. When the router is shutting down, the AURP module calls ATnlDisconnect, if necessary, and AShutdown to dispose of any memory allocated. Note that while the router is running, the administrator may deactivate an AURP port, resulting in a call to ATnlDisconnect or, conversely, the administrator may activate an AURP port, resulting in a call to ATnlConnect (see Figure 2).

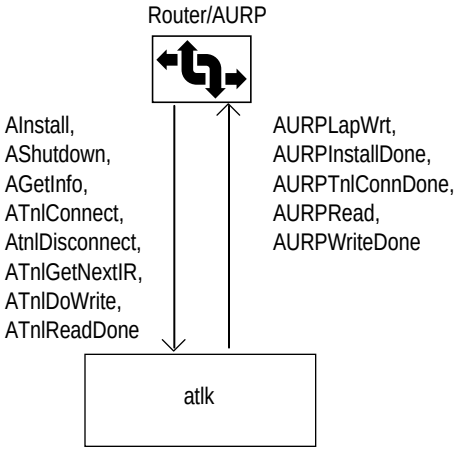


Figure 2 Direction of calls

Because of the varying lengths and formats of next IRs, the atlk is responsible for maintaining its own list of known next IRs and the corresponding indicators. When it makes the AURPInstallDone call after AInstall is called, the atlk passes a list of the indicators to the AURP module. The AURP module references this list for a given next IR. The AURP module may at any time make the AGetInfo or ATnlGetNextIR call to get information about the data link.

The atlk drives the process of receiving incoming packets, while the AURP module drives the process of sending outgoing packets. When the atlk receives an incoming packet, it calls the AURPRead routine (accessed through the AURP dispatch routine passed in AInstall) with a pointer to the packet data following the link-specific headers. After the AURP module is done processing the packet, it indicates it is ready to process the next incoming packet by calling the ATnlReadDone routine passed back in the AInstall call. Note that the atlk cannot free up its read buffer until the ATnlReadDone routine is called (see Figure 3).

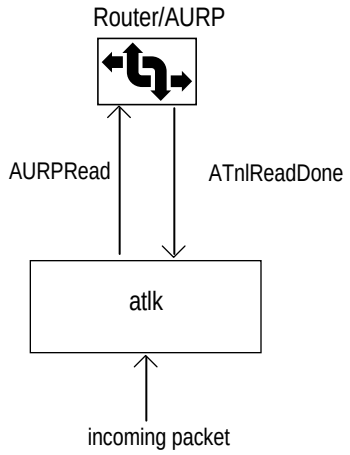


Figure 3 Read mechanism

When the atlk's LAP write code is called, the atlk passes the write on to the AURP module. The AURP module then calls the atlk's ATnlDoWrite routine when it wishes to send a packet out the data link. After processing the outgoing packet, the atlk calls the AURPWriteDone routine (accessed through the AURP dispatch routine passed in AInstall). (See Figure 4.)

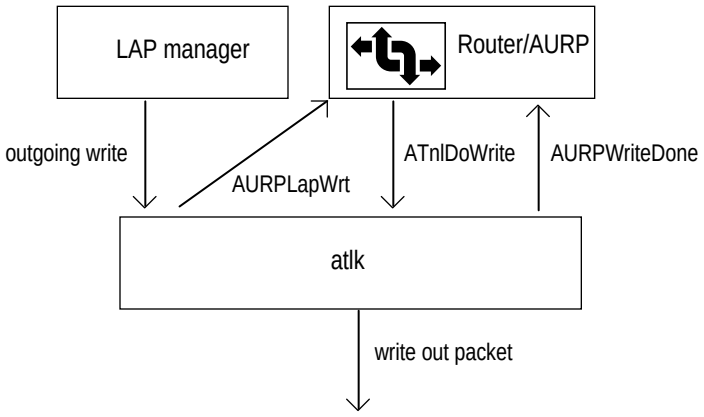


Figure 4 Write mechanism

The atlk should only allocate memory during AInstall. Most of the routines return an error result of type 0Serr, which is defined as a short. A zero result indicates no error; a nonzero result indicates an error. Note that the type Ptr refers to a pointer to a char (unsigned char *). In addition, the type NIRIndicator refers to the indicator of a next IR, a four-byte (long) quantity.

Calls to the AURP atlk module (assembly interface)

AInstall (D0 = 1)

Call	D3 (long)	AURP LAP write code entry point
	D4 (byte)	port number on which to perform the install operation
	A0 (long)	AURP Interface Dispatch Routine
	A1 (long)	pointer to ATnlConfig parameter block
	A2 (long)	AURP refnum
Return	D0 (word)	result code (nonzero if error)
	A1 (long)	pointer to ATnlConfig parameter block

The ATnlConfig parameter block is defined as follows:

```
typedef struct ATnlConfigPBlk          {
short reserved;         /* Reserved -- do not modify */
short hopCntWgt;        /* → Hop count weight specified in
                           Options… window */
long reserved2;         /* Reserved -- do not modify */
Ptr acfgPtr;            /* → Ptr to adev's 'acfg' rsrc */
Ptr paidPtr;            /* → Ptr to adev's 'paid' rsrc */
short maxRPktSz;        /* ← maximum packet size */
long atlkFlags;         /* ← flags for configuration */
Ptr atlkWriteCode;      /* ← address of atlk write code */
Ptr atlkReadDone;       /* ← address of atlk read done */
char srcDILen;          /* ← length of Domain Identifier */
char filler;
short numNextIRs;       /* ← number of nextIRs configured*/
};
```

When the AInstall routine is called, the code allocates its variables, opens an I/O driver (if this is required), and performs any other initialization necessary. The AInstall call does not establish any connections. (Connections may be established in the ATnlConnect call.) This extended AInstall call is different from the AInstall call described in the *Macintosh AppleTalk Connections Programmer's Guide* and is only available for AURP atlks (if the xtlk flags are set appropriately).

The AURP interface dispatch-routine address passed in the AInstall call enables the atlk to call the AURP module. The atlk should store this address and JSR to it on every call to the AURP module except for the LAP write call. (See "Calls From the AURP atlk Module (C Interface)" later in this document for more details.)

The hopCntWgt field in ATnlConfigPBlk refers to the weighting associated with this particular AURP port's data link. The hopCntWgt value plus one should be stored and returned in the bandwidth field of ATnlGetInfoRec (see the description of AGetInfo later in this document).

The adev signature referenced by the paidPtr in ATnlConfigPBlk may contain machine-specific port configuration information necessary for setting up the AURP data link. (Note that the signature is defined in the extensions to the adev interface.) For example, if the atlk needs the slot number of the NuBus™ card chosen by the administrator, the adev signature may contain the specified slot number. The acfgPtr points to a configuration record containing the AURP port's user settings. Since paidPtr and acfgPtr are temporary pointers, the atlk may wish to save their contents locally for use in the ATnlConnect call (described later). Note that both the adev signature and the configuration record are built using the extensions to the adev interface and may vary from adev to adev. The adev signature is defined in the MGetAdev call, and the configuration record is defined in the configuration user-interface calls.

The AURP refnum that is passed in the AInstall call is needed for the AURP module to identify the particular atlk if multiple AURP atlks exist. The atlk should store the AURP refnum and pass it on every call it makes to the AURP module (usually in the variable aurpRef).

Like standard atlks, the AURP atlk should call the LAP Manager to insert the atlk's code in the LAP write hook (using LWrtInsert and the port number passed in D4). Note that the code that is inserted is the beginning of the atlk resource. When the atlk's code is called on a LAP write, the AURP atlk should not process the LAP write itself; instead, the atlk should pass the call to the AURP module by jumping to the code location passed into the AInstall call in D3 (see "Calls From the AURP atlk Module (Assembly Interface)" later in this document).

The atlk code must communicate its initial values for AURP options by returning them in the ATnlConfigPBlk. The maxRPktSz is the maximum AURP routing packet size (in number of bytes) that the atlk will support. (Note that the value for maxRPktSz must be a value from 618 to 4096.) Currently only the high bit of the atlkFlags field is defined. If it is set, the AURP module will only establish AURP connections with routers specified in the initial ATnlNextIRBlk (returned in AURPInstallDone). If it is clear, the AURP module will learn about new routers from received packets and establish AURP connections with them.

The atlk registers its ATnlDoWrite code with the AURP module by passing the address of the routine in atlkWriteCode in ATnlConfigPBlk. The ATnlDoWrite code is then called when the AURP module wishes to write a packet out the atlk's link. When the LAP Manager calls the atlk's code that was installed in the LAP write hook (by LWrtInsert), the atlk passes the call to the AURP module by jumping to the AURP module's LAP write code. After adding its headers to the data, the AURP module calls the atlk's registered write code, ATnlDoWrite, with a parameter list to write the packet out the link.

The atlk also registers its ATnlReadDone code with the AURP module by passing the address of the routine in atlkReadDone.The AURP module calls the ATnlReadDone code after AURPRead is called and after the AURP module is done processing the incoming packet. The AURP module makes the ATnlReadDone call to notify the atlk that it is ready to receive the next incoming packet.

The Domain Identifier (DI) is used to distinguish between connected AppleTalk internets. The format of the DI is a length byte followed by an authority byte, followed by the identifier itself. Used primarily on point-to-point links, a null DI is simply 0x01 00. The IP form of the DI is 0x07 01 00 00 *XX XX XX XX*, where *XX XX XX XX* corresponds to the configured IP address. For more information about DIs and their formats, see the *AppleTalk Update-Based Routing Protocol: Enhanced AppleTalk Routing* document. The length of the DI is passed back in srcDILen in ATnlConfigPBlk.

The atlk is responsible for passing the number of next IRs, numNextIRs, that the AURP module should initially connect to. The full list of next IRs is passed when the atlk calls AURPInstallDone. AURPInstallDone may be called at a later time so that the atlk may asynchronously resolve the addresses or map addresses to indicators as necessary.

Note that errors returned by the AInstall call may cause the router to abort startup.

AShutdown (D0 = 2)

Call	D4 (byte)	port number on which to perform the shutdown operation
Return	D0 (word)	result code (nonzero if error)

When the atlk code is called with an AShutdown call, it issues an LWrtRemove call to the LAP Manager with the port number passed in D4, disposes of its variables, and performs any other necessary operations before the router shuts down. Refer to the *Macintosh AppleTalk Connections Programmer's Guide* for more information.

AGetInfo (D0 = 3)

Call	D1(word)	length (in bytes) of reply buffer
	A1 (long)	pointer to reply buffer
Return	A1 (long)	getInfo record
	D0 (word)	result code (nonzero if error)

The AGetInfo routine is responsible for filling the getInfo record with the appropriate information, if known. If the caller's buffer is not large enough to hold the record, an error is returned in D0 and partial information is returned.

```
typedef struct ATnlGetInfoRec {
     short  version;         /* of AGetInfo, set to two (2) */
     short  length;          /* of this record in bytes */
     long   speed;           /* of link in bits/sec */
     char   bandwidth;       /* link speed weight factor */
     char   reserved;        /* set to zero */
     char   slotNum;         /* physical slot number */
     char   reserved;        /* set to zero */
     char   flags;           /* set = 0x80 for backward
                                compatibility*/
     char   linkAddrSize;    /* of link addr in bytes */
     char   linkAddress[20]; /* link (or logical) address */
};
```

If the link speed is unknown, the maximum possible speed (in bits/sec) should be returned.

Note If links slower than 56 KB per second are supported, it is recommended that the maximum speed returned be less than 56 KB.

The bandwidth field contains the hopCntWgt plus one as described above in the description of AInstall. The slotNum represents the physical slot number that the Slot Manager uses to identify this atlk's port (usually from 0x09 to 0x0F). Set the slotNum to zero for nonslot hardware (for example, built-in networking ports). The linkAddress represents the address of the particular link: the address may be a hardware address (for example, an Ethernet hardware address) or a configured address (for example, an IP address), depending on the nature of the link. If 20 bytes isn't long enough for the link address, the address may be a unique logical address for the port. Refer to the *Macintosh AppleTalk Connections Programmer's Guide* for more information on the AGetInfo call. (Note that the ATnlGetInfoRec structure is slightly different from the AGetInfo record.)

ATnlConnect (D0 = 32)

Call

Return D0 (word) result code (nonzero if error)

The ATnlConnect routine is called when the AURP module wishes to establish an AURP connection. During this routine, the atlk establishes the data-link connection, if necessary. Connect configuration information may be found in the adev signature and in the configuration record passed in the AInstall call.

Because establishing a data-link connection may be an asynchronous activity and may take some time, the atlk must notify the AURP module when the connection has been established. The atlk notifies the AURP module by calling AURPTnlConnDone (see the section "Calls From the AURP atlk Module (C Interface)" later in this document). The AURPTnlConnDone call may be made from this ATnlConnect routine or at some later time after the data-link connection has been established.

Note If an error is returned from this call, the Apple Internet Router will attempt to make the port inactive.

ATnlDisconnect (D0 = 33)

Call

Return D0 (word) result code (nonzero if error)

The ATnlDisconnect routine is called when the AURP module wishes to bring down the AURP connection. After this routine is called, the atlk completes any outstanding writes and closes any open connections. When the connection has been closed, the atlk notifies the AURP module by calling AURPTnlDiscDone (see the section "Calls From the AURP atlk Module (C Interface)" later in this document). The AURPTnlDiscDone call may be made from this ATnlDisconnect routine or at some later time after the data-link connection has been torn down.

ATnlGetNextIR (D0 = 34)

Call D1 (long) indicator of next IR address
A1 (long) location to store next IR string

Return A1 (long) next IR string
D0 (word) result code (nonzero if error)

On entry, A1 holds a pointer to a 32-byte block allocated by the caller for the string (Pascal format).

The ATnlGetNextIR routine provides a mapping between the value of an indicator (next IR) and a string description of the indicator. The AURP module passes in the next IR indicator. The routine is responsible for filling the block referenced by A1 with the string equivalent of the next IR in Pascal format (length byte followed by ASCII characters). The length of the string is limited to a maximum of 31 characters.

Calls to the AURP atlk module (C interface)

ATnlDoWrite

```
OSErr ATnlDoWrite (NIRIndicator dstNextIR, WDSElement
                   *wdsPtr)
```

→ dstNextIR	indicator of the destination router
→ wdsPtr	pointer to array of wds elements containing AURP data to be written out (starting at the domain header)

When sending a packet, the AURP module calls the atlk routine ATnlDoWrite. The atlk write routine asynchronously writes the packet out the data link to the entity corresponding to the given dstNextIR. After the atlk completes sending the packet, it calls the AURPWriteDone routine (accessed through the AURPDispatch routine). If an error occurs, the atlk calls the AURPWriteDone routine with an error and may also return from ATnlDoWrite with an error. The atlk is guaranteed that only one write will be issued at a time; ATnlDoWrite will not be called again until after the AURPWriteDone routine is called. Note that AURPWriteDone must always be called, even if there is an error on the write.

The AURP module provides the indicator of the destination (dstNextIR). The wds elements referenced by wdsPtr are owned by the AURP module and thus should not be modified by the atlk. Note that the AURP module learns about this routine through the AInstall call described previously.

ATnlReadDone

```
void ATnlReadDone (Ptr dataPtr, OSErr readResult)
```

→ dataPtr	pointer to data that was read in
→ readResult	error result of read

After the AURP module is done processing the incoming packet (passed in the AURPRead call), it calls the atlk routine ATnlReadDone (which was passed back in the AInstall call), to notify the atlk that the AURP module is ready to process the next incoming packet. The dataPtr points to the data that was read in by the AURP module and is the same value that was passed in the AURPRead call. The readResult indicates whether the packet was processed successfully. (Refer to the description of the AURPRead call in the section "Calls From the AURP atlk Module (C Interface)" later in this document). The atlk may release the processed read buffer in this routine, but not before. Before returning to the caller, the atlk should restore the interrupt level to the level that was saved before calling AURPRead.

Calls from the AURP atlk module (assembly interface)

AURPLapWrite

Call D1 (long) aurpRef, AURP refnum
D0,D2,D3 (unchanged)
A0–A2 (unchanged)

When the beginning of the atlk's code (the code that is inserted with LWrtInsert) is called, the atlk should pass the LAP write call directly to the AURP module by jumping to this AURP LAP write code (passed in AInstall). The atlk should pass aurpRef (passed in AInstall) in register D1. In addition, the atlk should not modify any of the original input values (in registers A0–A2, D0, D2, or D3). It may, however, use register A3 to do the indirect jump necessary to call AURPLapWrite.

Calls from the AURP atlk module (C interface)

The AURP interface dispatch routine passed in the AInstall call enables the atlk to make calls to the AURP module. Calls to the AURP module follow C calling conventions. The atlk should call the dispatch routine, AURPDispatch, with the appropriate command type and subsequent parameter list. The command types are defined as follows:

```
#define     kAURPInstallDone     1
#define     kAURPTnlConnDone     2
#define     kAURPRead            3
#define     kAURPLogMsg          5
#define     kAURPWriteDone       6
#define     kAURPTnlDiscDone     7
```

AURPInstallDone

```
void AURPDispatch(char cmdType = kAURPInstallDone, long
                  aurpRef, Ptr atlkDIPtr, ATnlNextIRBlkPtr
                  nextIRBlkPtr)
```

→ aurpRef AURP refnum

→ atlkDIPtr pointer to default Domain Identifier for this port (source)

→ nextIRBlkPtr pointer to structure containing information about next IRs

The AURPInstallDone routine is called after the atlk is done installing itself and has acquired its full list of initial nextIRs. The aurpRef value is the same as was passed in AInstall. The atlk passes the default DI to be used for this port in atlkDIPtr. The default DI consists of a length byte followed by that number of bytes (Pascal string format). The length byte of the DI must have the same value as the srcDILen passed back in the ATnlConfigPBlk in the AInstall call. For a description of DIs, refer to the description of AInstall in "Calls to the AURP atlk Module (Assembly Interface)," earlier in this document.

The ATnlNextIRBlk is defined as follows:

```
struct ATnlNextIRBlk     {
short         numNextIRs;  /* The number of next IRs */
NIRIndicator  nextIR;      /* Indicator of next IR */
                           /* Repeat nextIR for numNextIRs */
};
```

The information in ATnlNextIRBlk describes the atlk's initial configuration of nextIRs. The numNextIRs should be equal to the value passed back in AInstall. If the value is not initially known, the numNextIRs should be zero, and the rest of the block is ignored. Note that the nextIR is a four-byte indicator of the next router's actual address.

The atlk must ensure that the pointers for both atlkDIPtr and nextIRBlkPtr remain valid until the first ATnlConnect call is made. The AURP module may reference the blocks until the first ATnlConnect call is made.

AURPTnlConnDone

```
OSErr AURPDispatch(char cmdType = kAURPTnlConnDone, long
                   aurpRef, OSErr connResult)
```

→ aurpRef AURP refnum

→ connResult Zero if no error, nonzero if unable to establish connection

When the atlk successfully establishes the data-link connection (after ATnlConnect is called), the atlk must notify the AURP module so that the AURP module may begin establishing AURP connections over the data link. To notify the AURP module, the atlk calls AURPTnlConnDone with connResult equal to zero (noErr). The aurpRef value is the same as was passed in AInstall. After the data link is set up and any connections are established, the atlk may deliver incoming packets to the AURP module.

If the atlk is unable to establish a data-link connection, the atlk may call AURPTnlConnDone with the connResult equal to a nonzero error code, to notify the AURP module that a data-link connection cannot be established.

AURPRead

```
OSErr AURPDispatch(char cmdType = kAURPRead, long aurpRef,
                   NIRIndicator srcNextIR, short dataLen, Ptr
                   dataPtr)
```

→ aurpRef	AURP refnum
→ srcNextIR	indicator of the source router
→ dataLen	length of received data
→ dataPtr	pointer to received data (at start of AURP data)

Upon receiving a packet, the atlk calls the AURP routine AURPRead to process the packet. The atlk may not release the read buffer while AURP is processing the packet. After processing is complete, the AURP module calls the ATnlReadDone routine passed back in the AInstall routine. If an error occurs, the AURP module calls the ATnlReadDone routine with an error and should also return from AURPRead with an error.

Before calling AURPRead, the atlk should save off its current interrupt level. This precaution is necessary because the router treats the read similarly to AppleTalk reads and changes the interrupt level. (It sets the interrupt level to the VSCCEnable level indicated by MPP.) The atlk should restore the interrupt level when ATnlReadDone is called.

The aurpRef value is the same as was passed in AInstall. The atlk provides an indicator of the sender of the packet (srcNextIR) to the AURP module so that the AURP module can associate the packet with the appropriate AURP connection. The data received is referenced by dataLen, the length of the data, and dataPtr, a pointer to the data itself. Note that the atlk should remove link-specific headers before passing the data to the AURP module; the dataPtr should point to the packet data following the link headers.

AURPLogMsg

```
void AURPDispatch(char cmdType = kAURPLogMsg, long aurpRef,
                  long reserved, Ptr messageStr)
```

→ aurpRef AURP refnum

→ reserved set to zero

→ messageStr Pascal string containing message to be logged

Whenever the atlk wishes to log a message to the Router Manager router log, it calls AURPLogMsg with aurpRef (passed in AInstall) and the messageStr, a pointer to a length byte followed by the log message. The length of the log message is limited to 223 characters. The reserved parameter should be set to zero. When this routine is called, the AURP module posts the given message to the router's log.

Note It is recommended that at install time, the atlk preload the log messages into memory from a resource.

AURPWriteDone

```
OSErr AURPDispatch(char cmdType = kAURPWriteDone, long aurpRef,
                   WDSElement *wdsPtr, OSErr writeResult)
```

→ aurpRef AURP refnum

→ aurpRef wds pointer passed in ATnlDoWrite

→ writeResult error result of read

After the atlk completes the ATnlDoWrite call and is done sending a packet, the atlk calls AURPWriteDone. The aurpRef value is the same as was passed in AInstall. The aurpRef is the pointer to the write structure that was passed in ATnlDoWrite. The writeResult indicates whether the packet was sent successfully.

AURPTnlDiscDone

```
OSErr AURPDispatch(char cmdType = kAURPTnlDiscDone, long
                   aurpRef, OSErr discResult)
```

→ aurpRef AURP refnum

→ discResult zero if no error, nonzero if error in closing connection

When the atlk completes bringing down the data-link connection (after ATnlDisconnect is called), the atlk must notify the AURP module. To notify the AURP module, the atlk calls AURPTnlDiscDone with discResult equal to the error result. The aurpRef value is the same as was passed in AInstall.

Developing an adev file using XTI

This section details the XTI adev Shell sample code. This shell is meant to be a base for writing a point-to-point AURP adev using the X/Open Transport Interface (XTI). If a multipoint adev is desired, this code is still useful in understanding the flow of data between the adev and AURP as well as between Router Manager and the adev. This section includes some hints for multipoint links, but they are by no means comprehensive.

IMPORTANT The sample code in this section is included as a guide only. Apple makes no warranty or representation, express or implied, with respect to the code, its quality, performance, or fitness for a particular purpose.

There are two main reasons the XTI adev Shell exists. The foremost reason is that with the upcoming Transport Independent Interface (TII), programming of the AppleTalk Network System will be moving to an XTI-like standard. If adevs are written to XTI, the migration to TII will likely be straightforward. The second reason this shell exists is to give developers an example of how to use the AURP/atlk interface and, to a somewhat lesser extent, how to use the Router Manager/adev interface (see Figure 1).

This code was written to the MacOSI transport specifications that are currently in step with the TII specifications. This may change before TII is released, however, so care should be taken. Other XTI implementations may not work, because substantial effort has been put into making TII, and hence the XTI shell, capable of operating asynchronously. (Standard XTI is not by nature asynchronous.)

The code is a hypothetical adev that uses an X.25 addressing scheme and appears in Router Manager under the printer and modem ports. (This is admittedly not a real-life application, but it makes a good example!)

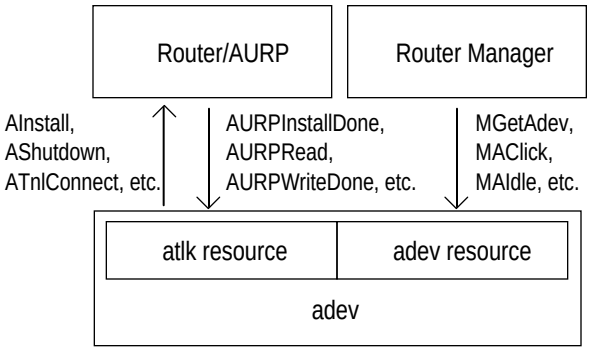


Figure 5 adev interfaces

The best plan of attack is to go through the source code, reading the comments carefully. For each procedure, the comments state whether or not the code may need to be modified and why. Below is a kind of checklist, or map, of the modifications that may need to be made, as well as an overview of some of the more complicated sequences.

Source code map

The table below lists the files provided and gives a brief description of their contents.

Filename	Description
AURPEqu.a	Assembler equates for the AURP/atlk interface.
AURPInterface.h	C include file for the AURP/atlk interface.
TADev.h	C include file that contains definitions needed by all Apple Internet Router adevs.
X25_Addr.h	C include file that contains X.25 addressing defines.
XTI.h	C include file that contains defines for XTI events and errors, and prototypes for XTI calls.
XTIShell.a	The main assembly code for the atlk portion of the adev. The calls from AURP to the atlk are implemented here.
XTIShell.c	The C code that supports the XTIShell.a code. All the routines in this file are called from the .a file.
XTIShell.h	Main C Include file for both adev and atlk portions of the adev.
XTIShell.r	Main resource file.
XTIShellADev.a	Main assembly code for the adev portion of the adev. The calls from Router Manager are received here and passed on to XTIShellADev.c.
XTIShellADev.c	Main C code that supports the XTIShellADev.a code. All the routines in this file are called from the .a file.
XTIShell_r.h	Resource includes.

Modifications that may be needed

The following table indicates what changes may be required for each file.

Filename	What to Change
AURPEqu.a	Nothing.
AURPInterface.h	Nothing.
TADev.h	When an itemRef is assigned to you by MacDTS, it should be entered here.
X25_Addr.h	This file should be replaced by an include file that outlines the structures involved in addressing your particular link.
XTI.h	This is included merely for sake of compilation. Your code would need to include the headers of the XTI implementation you are using (TII, MacOSI, or other).
XTIShell.a	
■ DoInstall	If you would like to specify a Domain Indicator other than 0x0100 (null DI), do so here.
	If you would like to specify your next internet router (nextIR) more explicitly than a code indicating call or answer, do so here.
■ GetNextIR	On a point-to-point link, there is only one nextIR to map, so if you have a multipoint link, you will have to look at the indicator passed in by AURP. On the other hand, if you have a point-to-point link but would like to use something other than the remote address as the string, do so here. An example of this is DialUp using the string “Answering.”
■ FreeReadBuf	If you intend to change the method by which packets are read (see the section “Read Sequence” later in this document), this routine may need to change. This should not be necessary, however.
XTIShell.c	
■ DoXTIInstall	Fill in the line speed and maximum buffer size of your link here.
	Fill in the remote address of the router you are connecting to in the format appropriate to your link.
	If you need memory other than what is already allocated, allocate it here.
■ DoXTIShutdown	If you allocated any extra memory, make sure to dispose of it here.
■ try_open	If you would like to limit the number of times a connection is attempted before giving up, add that capability here.
■ do_open	If you would like to have a provider name other than the slot ID, substitute the name here.
■ do_bind	Change this to use the addressing scheme appropriate for your link.
■ do_connect	Change this to use the addressing scheme appropriate for your link.
■ get_listen	If password protection is desired, add code here to verify the password before accepting the connection.
XTIShell.h	Review all data structures and add variables you need as appropriate.
XTIShell.r	Add any extra log messages in this file as well as any other strings or resources you need.
XTIShellADev.a	Nothing.
XTIShellADev.c	This file is a shell containing place holders for the routines you need to implement to communicate with Router Manager. You will need to modify the entire file.
XTIShell_r.h	Put any resource defines you may add in here.

Open sequence

At install time, the adev allocates memory and initializes variables. When AURP calls the adev to connect, a time manager task is installed (through the Time Manager); it in turn installs a deferred task (through the Deferred Task Manager) that is responsible for the open, bind, and connect (if calling). The flow chart of this deferred task (try_open) is shown in Figure 6.

The try_open routine is confusing because it is called repeatedly to do several tasks. Each time it is called it may be in a different state; the states are defined as follows:

1. The routine has never before been entered; it requires an open, a bind, and a connect.
2. The open is complete; it requires a bind and a connect.
3. The bind has been initiated but not completed; the routine is waiting for the bind to complete.
4. The bind is complete; it requires a connect.

What state the try_open routine is in needs to be determined before action can be taken (see Figure 6).

Read sequence

When a T_DATA event is received, the buffer is checked for availability. If it is available, the get_data routine is called to read the data. If it isn't available, a variable indicating the number of packets waiting is incremented. The get_data routine is responsible for reading all the data currently available on the link or one packet's worth (whichever comes first), where a packet must be less than or equal to the max read packet size (maxRPktSz).

When a T_DATA event occurs is likely dependent on the XTI implementation. This code assumes a T_DATA event will be received for some sequence of fragments that likely add up to a complete packet. A T_DATA event will be generated for a partial packet only if one or more fragments do not arrive in a timely manner.

When a full packet has been read, it is handed off to AURP. When AURP is done with it, the FreeReadBuf routine is called. This routine checks to see if there are any more packets waiting to be read, and if so, it calls the get_data routine to read them. If an error other than T_NODATA occurs, the adev will disconnect, tell AURP the connection went down, and attempt to reconnect.

Figure 7 shows the flow chart of the read sequence.

Write sequence

When the adev is called by AURP to write a packet, it copies the WDS into one buffer and sends it by means of XTI. When the T_SENDCOMPLETE event is received, it checks to see if all the data was sent. If not, it resends the remainder of the data. If all the data was sent, it tells AURP the write completed successfully.

Errors in the write sequence are handled in a similar manner to errors in the read sequence, with the exception of the T_FLOW error. If this error occurs, the link is flow-controlled, and the packet is held until a T_GODATA event is received. Because the write complete routine has not been called, AURP will not send any more data to the adev at this time.

Close/Shutdown sequence

When an error occurs or the port is made inactive, the adev performs its close sequence. If there is an outstanding write, it is canceled by calling the write done routine with an error. The endpoint is then closed, and the file descriptor set accordingly. In the case of a shutdown, memory is also released.

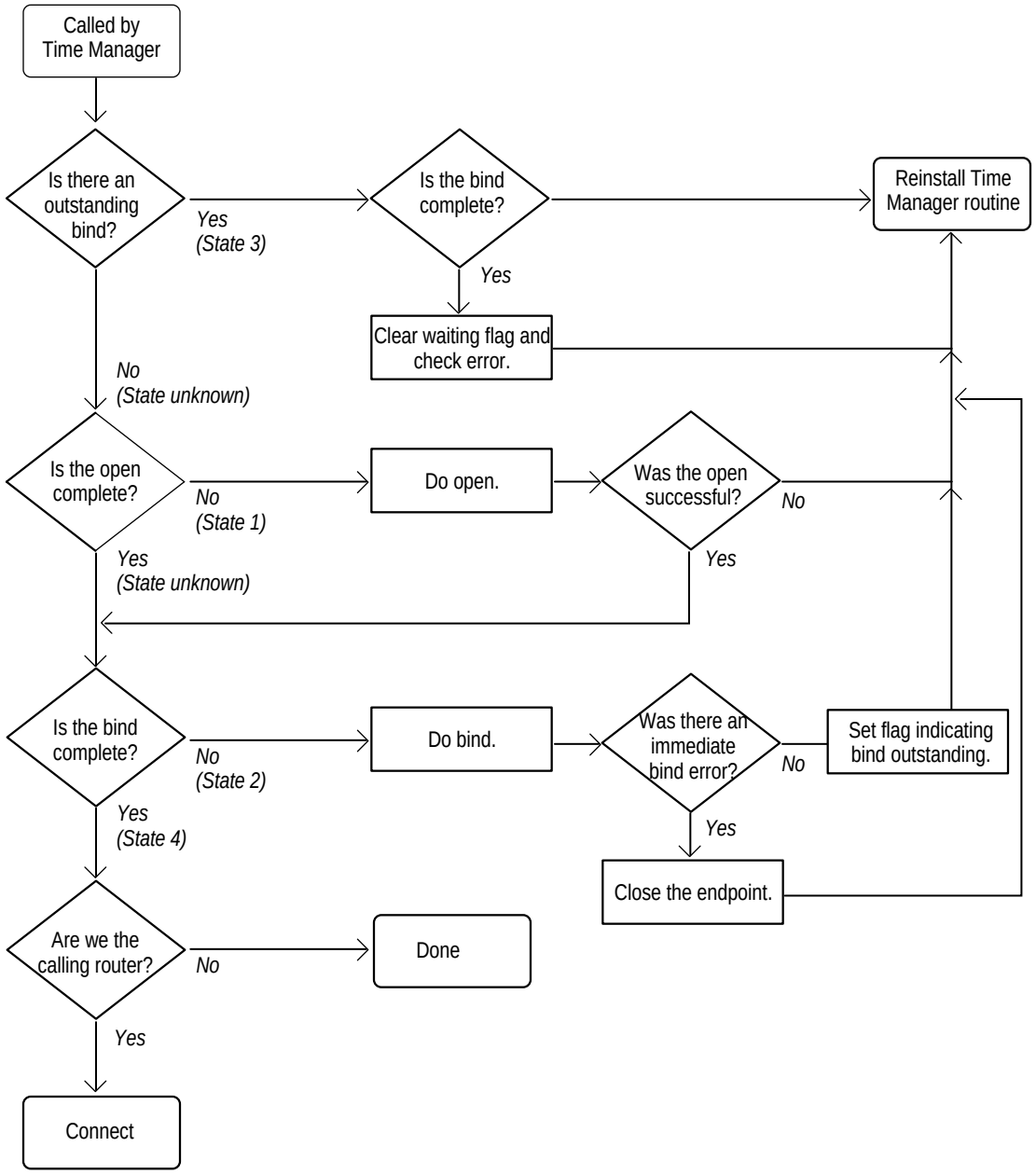


Figure 6 try_open flow chart

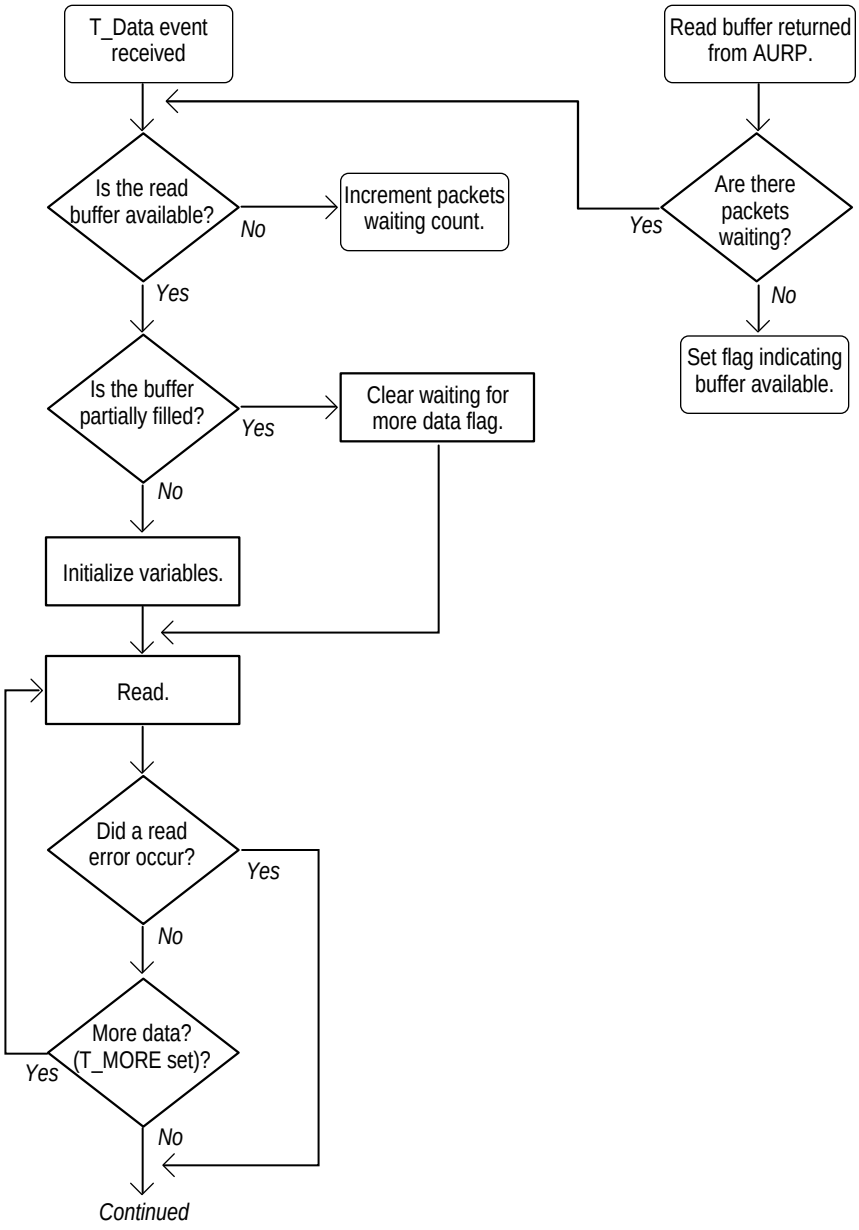


Figure 7 Read sequence flow chart (continued on next page)

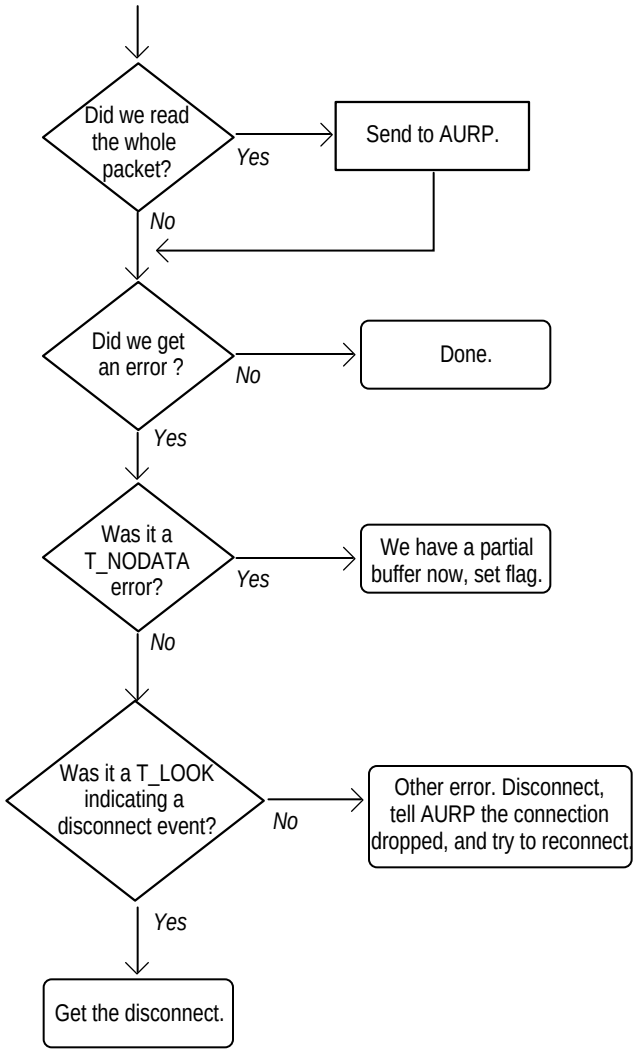


Figure 7 Read sequence flow chart (continued)

Apple Internet Router: Extending IP Tunnel and DialUp

Apple Computer, Inc.

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

About this document

This document outlines the process for extending the IP Tunnel and DialUp adev files to support your mdev or serial driver. To get a copy of the adevs that support this functionality please contact your Apple evangelist.

IP Tunnel

The AppleTalk/IP Wide Area Extension (IP Tunnel adev), which is layered on top of MacTCP, supports different physical media much as MacTCP does (see Figure 1). IP Tunnel contains built-in support for Ethernet and may be extended to run over any other medium.

Note IP Tunnel does not support the concept of subports as defined in the Apple Internet Router.

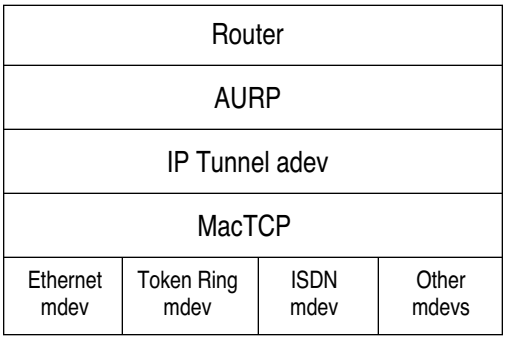


Figure 1 IP tunneling over alternate links

In order to configure IP Tunnel to run over media other than Ethernet, the following building blocks and "glue" pieces are necessary:

- An mdev file, as described in "Building Alternate Link Access Modules for MacTCP," which is available on Essentials•Tools•Objects (E.T.O.) (order number M0895LL). The IP Tunnel adev relies on implementation of the LAP_statistics mdev call to return correctly the "slot" number of the interface for which the mdev is currently configured.
- If the mdev is not slot based, it must only conform to the latest "Building Alternate Link Access Modules for MacTCP," specifically the new definition of the LAP_Statistics call. If the "slot" number returned is 1 or 2, IP Tunnel will include itself underneath the modem and printer ports, respectively. If the slot number is -1 or 3 (SCSI or other), a new device "MacTCP port" will be created containing the IP Tunnel method. No other pieces are necessary.
- If the mdev is slot based (such as Ethernet or Token Ring), an IDv2 resource in the IP Tunnel file with a res ID equal to (-4032 – *x*), where *x* is a unique positive number (0,1,2, . . .) that you can get from Apple Developer Technical Support. The IDv2 resource is formatted as follows:

```
type 'IDv2' {       /* IP Tunnel device block (v2) */
   integer;         /* base res ID for device strings */
   integer;         /* res ID for device 'ICON' */
   integer;         /* spCategory */
   integer;         /* spCType */
};
```

The first field, a word, contains the base res ID for strings ('STR ') describing the device (see details later in this document). The second field, also a word, contains the res ID for the device small icon suite (icn#). The third and fourth fields are the category and type of card as defined by the Slot Manager sResource.

The Token Ring IDv2 is shown below as an example ($x = 0$):

```
#define catNetwork     4
#define typeToken      2

resource 'IDv2' (-4032) {       /* Token Ring */
   -4044,                       /* res id for 'STR ' */
   -4034,                       /* res id for small icon
                                   suite */
   catNetwork,                  /* spCategory */
   typeToken,                   /* spCType */
};
```

- The device string resources ('STR ') starting at the res ID specified in the IDv2. The seven strings must have sequential res IDs. The IDs should start at (-4044 – 7*x*), where *x* is the same unique positive number as described previously (0,1,2, . . .). The IDs should increment in the negative direction. The first string resource contains the name of the device. The remaining string resources contain the name of the device plus "(Slot *n*)," where n is the logical slot number from 1 to 6 (in order).

  The Token Ring STR resources are shown below as examples (x = 0):

```
resource 'STR ' (-4044, "", preload) {
         "Token Ring";
};

resource 'STR ' (-4045, "", preload) {
         "Token Ring (Slot 1)";
};

resource 'STR ' (-4046, "", preload) {
         "Token Ring (Slot 2)";
};

resource 'STR ' (-4047, "", preload) {
         0"Token Ring (Slot 3)";
};
```

```
resource 'STR ' (-4048, "", preload) {
        "Token Ring (Slot 4)";
};
resource 'STR ' (-4049, "", preload) {
        "Token Ring (Slot 5)";
};
resource 'STR ' (-4050, "", preload) {
        "Token Ring (Slot 6)";
};
```

- The device small icon suite (ics#) with the res ID specified in the IDv2. The ID should be equal to (-4034 – *x*), where *x* is the same unique positive number as described previously (0,1,2, . . .). The device icon is a graphical representation of the device. It should be easy for the user to associate the icon with the actual card. The entire small icon suite (ics#, ics8, ics4) should exist so Router Manager can display the appropriate icon based on the current display.

  The Token Ring icon  resource is shown below as an example (*x* = 0):

```
resource 'ics#' (-4034) {
   {
        // icon data here
   }
};
resource 'ics4' (-4034) {
        // icon data here
};
resource 'ics8' (-4034) {
        // icon data here
};
```

DialUp

The DialUp adev is layered on top of the AppleTalk Remote Access Link Tool Manager, which supports Macintosh serial drivers such as the .AIn and .AOut drivers. DialUp contains built-in support for the printer and modem ports, and may be extended to run over slot-based devices with a single port or multiple ports.

In order to configure DialUp to run over media other than the printer and modem ports, the following building blocks and “glue” pieces are necessary:

- Input and output drivers for the card.
- A 'DDev' resource in the DialUp file with a res ID equal to (-4032 – *x*), where *x* is a unique positive number (0,1,2, . . .) that you can get from Apple Developer Technical Support. The DDev resource is formatted as follows:

```
type 'DDev' {  /* DialUp device block */
   integer;    /* base res ID for device strings */
   integer;    /* res ID for device small icon suite */
   integer;    /* res ID for STR# containing driver names */
   integer;    /* spCategory */
   integer;    /* spCType */
   integer;    /* number of ports */
   longint;    /* type of file where INIT code resides */
   longint;    /* creator of above file */
};
```

Field 1 The base resource ID for the strings describing the device and its ports (see details provided later).

Field 2 The base resource ID for the small icon suites for the device and its ports (see details provided later).

Field 3 The resource ID of the string list resource containing the names of the input and output drivers for the device.

Field 4 and 5 The Slot Manager spCategory of the card (this value will probably be 4 – catNetwork) and spCType of the card, respectively.

Field 6 If the card is multiport, this is the number of ports on the card; otherwise, this field should be 0.

Field 7 and 8 If there is any INIT code used to load the drivers, the type and creator of the file containing the INIT code, respectively. The file is assumed to be in the Extensions folder. When the router is set to run at startup, this INIT code will be loaded and run. If no INIT code exists, these fields should be 0.

The Serial NB Card DDev is shown below as an example ($x = 0$):

```
resource 'DDev' (-4032) {
   kSerNBStrRes,
   kSerNBIconRes,
   kSerNBstrlRes,
   6,                    // communications category
   25,                   // Serial NB card type
   4,                    // 4 ports
   'INIT',
   'xxxx'
};
```

- The device string resources ('STR ') starting at the res ID specified in the DDev. The seven strings must have sequential res IDs. The IDs should start at (-4044 – 25*x*), where *x* is the same unique positive number as described previously (0,1,2, . . .), and should increment in the negative direction. The first string resource contains the name of the device. The next 6 string resources contain the name of the device plus "(Slot *n*)," where *n* is the logical slot number from 1 to 6 (in order). Following these should be the string resource containing the port names if the device is multiport.

  The Serial NB STR resources are shown below as examples (*x* = 0):

```
resource 'STR ' (-4044, "", preload) {
        "Serial NB";
};
resource 'STR ' (-4045, "", preload) {
        "Serial NB (Slot 1)";
};
resource 'STR ' (-4046, "", preload) {
        "Serial NB (Slot 2)";
};
resource 'STR ' (-4047, "", preload) {
        "Serial NB (Slot 3)";
};
resource 'STR ' (-4048, "", preload) {
        "Serial NB (Slot 4)";
};
resource 'STR ' (-4049, "", preload) {
        "Serial NB (Slot 5)";
};
resource 'STR ' (-4050, "", preload) {
        "Serial NB (Slot 6)";
};
resource 'STR ' (-4051, "", preload) {
        "Port 1A";
};
resource 'STR ' (-4052, "", preload) {
        "Port 1B";
};
resource 'STR ' (-4053, "", preload) {
        "Port 2A";
};
resource 'STR ' (-4054, "", preload) {
        "Port 2B";
};
```

- The device and port small icon suites (ics#) starting with the res ID specified in the DDev. The ID should be equal to (-4034 – 20*x*), where *x* is the same unique positive number as described previously (0,1,2, . . .). The entire small icon suite (ics#, ics8, ics4) should exist so Router Manager can display the appropriate icon based on the current display. The first small icon suite should be for the device and the remaining should be for the ports if the card is multiport.

  The Serial NB Card icons (ics# only) are shown below as an example *(x* = 0):

```
resource 'ics#' (-4034) {
   {
        // device icon data here
   }
};
resource 'ics4' (-4034) {
        // device icon data here
};
resource 'ics8' (-4034) {
        // device icon data here
};
resource 'ics#' (-4035) {
   {
        // port 1 icon data here
   }
};
```

  other port 1 icons . . .

```
resource 'ics#' (-4036) {
   {
        // port 2 icon data here
   }
};
```

  other port 2 icons . . .

```
resource 'ics#' (-4037) {
   {
        // port 3 icon data here
   }
};
```

  other port 3 icons . . .

```
resource 'ics#' (-4038) {
   {
        // port 4 icon data here
   }
};
```

  other port 4 icons . . .

- A string list resource containing the names of the drivers for each port (or one name if it is a single port device) with resource ID (-4038 – *x*), where *x* is the same unique positive number as described previously (0,1,2, . . .).

  The Serial NB Card driver names are shown below as an example (*x* = 0):

```
resource 'STR#' (-4038) {
   {
         ".SNB0In",
         ".SNB0Out",
         ".SNB1In",
         ".SNB1Out",
         ".SNB2In",
         ".SNB2Out",
         ".SNBC3In",
         ".SNBC3Out"
   }
};
```

Apple Internet Tunnel Simulator

Apple Computer, Inc.

© Apple Computer, Inc., 1993
20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

About this document

The Apple Tunnel Simulator is a tool for testing AppleTalk Update-based Routing Protocol (AURP) compliant routers. It is designed to work specifically with AURP IP tunneling. This document explains how to set up the Tunnel Simulator and describes features of the software.

IMPORTANT The Apple Tunnel Simulator is a test tool to help developers check many of the functions and features of the AURP protocol. It should not be taken as a reference standard. Apple makes no warranty or representation, express or implied, with respect to the program, its quality, performance, or fitness for a particular purpose.

Features

The Apple Tunnel Simulator has the following features:

- It acts as one or more routers.
- It simulates large AppleTalk network topologies.
- It creates multiple AURP connections.
- It connects and exchanges network and zone information, and maintains the AURP protocol link between the routers.
- It displays information on packets received and sent.
- It does sanity checking on packet IDs and sequencing numbers.
- It can simulate randomly changing network topology.
- It can load configurations and topologies from files.

Setup

The Apple Tunnel Simulator is a stand-alone application that can be launched by double-clicking its icon. The following steps show how to set up the simulator.

IMPORTANT The Apple Tunnel Simulator does *not* use MacTCP, and MacTCP must be removed before the simulator is run. The Apple Tunnel Simulator must be run on an independent Macintosh computer connected by Ethernet to the router being tested.

1. **Select the Add Simulated Router command in the Setup menu.**
2. **Type in the IP address for a simulated router.**
3. **Click the Generate Network button to create a default network topology for the simulated router.**

4 **Select the Add Test Router command in the Setup menu.**

5 **Type in the IP address of the router to be tested.**

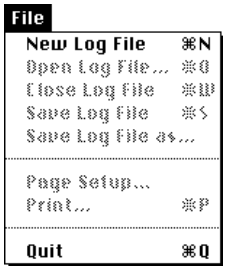
6 **Select the Run command in the Setup menu.**

The Tunnel Simulator activates the simulated router and attempts handshaking with the test router. To initiate a connection, either the test router must have the address of the simulated router, or the test router must be set up to accept connections from any router.

Commands

This section describes in detail the commands in each menu of Apple Tunnel Simulator.

File menu



New Log File

All output to the Log display window is captured in a text file.

Close Log File

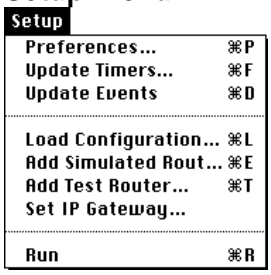
Output capture is turned off and the log file is closed.

Quit

Before Apple Tunnel Simulator quits, it sends a “router down” command on all open AURP connections. The simulator waits 5 seconds for any ACK responses and then quits.

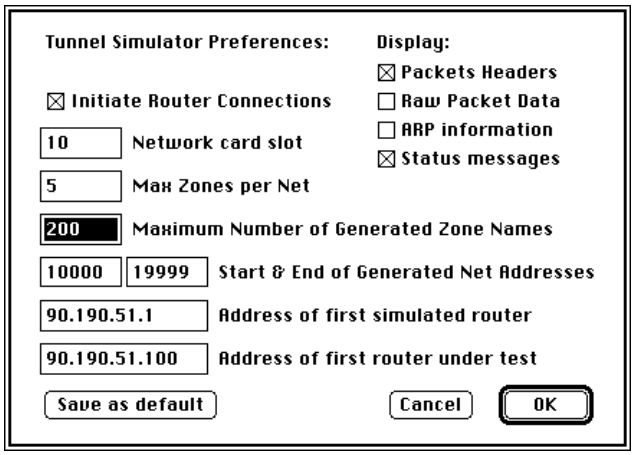
Note Apple Tunnel Simulator does not currently support the Open Log File, Save Log File, and Save Log File as menu commands. The corresponding menu items appear dimmed.

Setup menu



Preferences

Brings up the Tunnel Simulator Preferences dialog box. Each item in the dialog box is described below.



Display Packet Headers (checkbox) When this option is selected, all AURP packets sent and received by the simulator have their header information displayed in the status window.

Display Raw Packet Data (checkbox) When this option is selected, all packets received and sent by the simulator are displayed in the status window. (This option is not implemented in this version of Apple Tunnel Simulator).

Display ARP information (checkbox) When this option is selected, all AURP packets sent and received by the simulator have their header information displayed in the status window.

Display Status messages (checkbox) When this option is selected, status messages such as “router alive” and “update event” are displayed.

Initiate Router Connections (checkbox) When this option is selected, the simulated routers initiate connections to routers under test. If this option is not selected, the test routers must know about the simulated routers in order to open AURP connections.

Network card slot (value) The slot number of the network card to be used by the simulator. This is the physical slot number (9..E) and not the virtual slot number (1..6). If a network card is not found in the slot indicated, a slot with a network card is selected.

Max Zones per Net (value) The maximum number of zones that can be associated with one simulated network. Apple Tunnel Simulator selects a random number from 1 to the value entered here.

Maximum Number of Generated Zone Names (value) The simulated routers randomly generate zone names for their networks. Multiple network ranges may have the same zone name. The number of zones generated by the simulator will not exceed the value specified here. This value should be set to the test router's maximum number of zones *minus* the number of zones the test router has created for itself. If multiple test routers are used, then all the zones they create must be taken into account. This value should be set before any simulated routers are created.

Start & End of Generated Net Addresses (values) These values define a range of network numbers that the simulated routers can create when randomly generating AppleTalk networks. These values must be set before any simulated routers are created. Once a router is created, these values are stored internally and cannot be changed. The network range can be shared by multiple simulated routers without duplication. However, between creating the routers, change these values so that multiple simulated routers have separate ranges. These values do not limit networks loaded by a topology file.

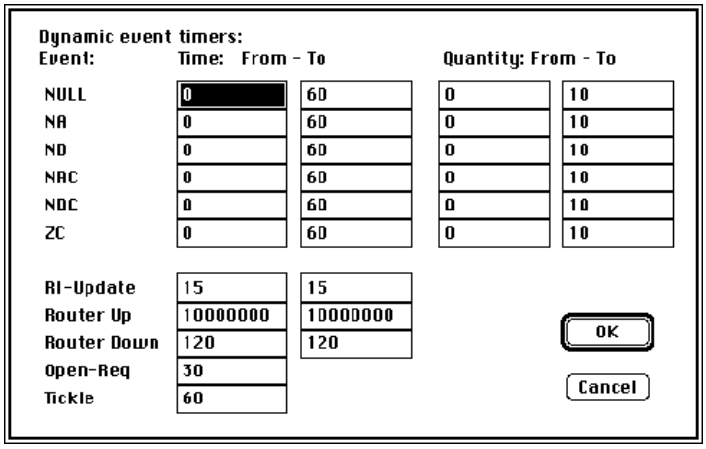
Address of first simulated router (value) Can be used in conjunction with the Save as default button to store the IP internet address of the first simulated router. This value is used by the Add Simulated Router command.

Address of first router under test (value) Can be used in conjunction with the Save as default button to store the IP internet address of the first router to be tested. This value is used by the Add Test Router command.

Save as default (button) When this button is clicked, the values and checkbox states
are saved in a resource of the program and are used as defaults each time the program is run. Clicking this button also saves the values in the Update Timers dialog box (see the next section).

Update Timers

This command displays the dialog box shown below. The values entered in this dialog box are saved as defaults when you click the “Save as defaults” button in the Preferences dialog box.



This dialog box is used to set the quantity and time intervals for random update events. Each event has one to four values. The first two values set the time an event will occur, in seconds. The time will be a randomly selected number between the two values. If there is only one value, then the time will not be random. The third and fourth values set how many of that event will occur. This quantity will also be a random amount between the two values.

The NULL event does not do anything in this revision of the software. The NA event generates new networks in the router, and the ND, NRC, NDC, and ZC events are applied to current networks.

The number of networks actually affected may be less than what is specified. When the RI-Update interval is reached, an RI-Update packet is created with the networks affected by these events.

The Router Up and Router Down intervals control the amount of time that a simulated router is actively talking to the test router and how long it can go down and be inactive. The cycle repeats, based on the quantity values entered in the dialog box.

The Open-Req value is the interval the router polls when a connection is not open, and the Tickle value is the interval that tickle packets are sent on an open connection.

To enable the events, use the Update Events command. The Open-Req and Tickle events are always enabled. To keep an event from occurring, set the event’s time to a large number.

Update Events

Starts the update events based on the timers set in the Update Timers dialog box.

Load Configuration

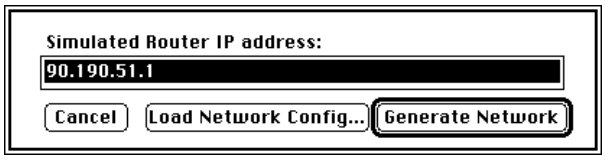
Brings up the directory dialog box, so you can select a configuration file. The configuration file is a text file that gives Apple Tunnel Simulator setup information. The contents and format of the file are as follows:

```
! Commands are the first letter of a line and are as follows:
!
!     ! - comment lines
!     T - Test routers       ie.  T ip-address
!     S - Simulated routers  ie.  S ip-address [topology
!                                                filename]
!     (If no topology filename is given, a default network
!      topology is generated)
!     NR - network range     ie.  NR 10000 19999
!     MZ - maximum zones     ie.  MZ 150
!
! For example:
!
! Routers under test
!
T 90.190.51.200
t 90.190.51.201
!
! Simulated tunnel routers
!
NR 10000 19999
S 90.190.51.1
!
NR 20000 29999
S 90.190.51.2
!
nr 30000 39999
S 90.190.51.5  5Mixed.Ext
s 90.190.51.110 100Mixed.Ext
!
! Set the maximum number of zones
!
MZ 200
!
! end
```

Apple Tunnel Simulator uses two types of files: the configuration file, which provides the simulator with setup information (shown previously), and the network topology file, which provides a simulated router with a network topology. The topology file can be used by the configuration file in the "S" command, or it can be used manually with the Add Simulated Router menu option, which includes the format of the topology file (see the next section).

Add Simulated Router

Creates a simulated AURP router with an AppleTalk network topology. Selecting this command first brings up a dialog box in which you enter the IP address of the simulated router.



The initial address displayed is based on the address given in the Preferences dialog box. The next time the command is invoked, the address is incremented by one from the previous address used. A maximum of 1000 simulated routers can be created.

Clicking the Generate Network button randomly creates a small topology based upon the start and end network numbers set in the Preferences dialog box.

Clicking the Load Network Config button brings up the directory dialog box, so you can select a topology file. The topology file can be used to set up the network numbers and ranges for the simulated AppleTalk network. If you have not specified zone names, Apple Tunnel Simulator randomly generates them and assigns them to the network numbers. The topology file format is as follows:

```
Non/Extended Flag, NetStart, [NetEnd], NetDistance=0,
   [Zone Name]
Number of Network Ranges
Non/Extended Flag, NetStart, [NetEnd], NetDistance,
   [Zone Name]
<Repeat the network range descriptions for the number of
 network ranges>
```

Following is an example of data within a topology file:

```
1     158     160          0
3
1     7000    7010         10
1     7011    7017         6     West Zone
0     7018                 4
```

A network can be either nonextended or extended. Zero represents a nonextended network, and 1 represents an extended network.

NetEnd is required when entering extended networks.

The zone name is optional. If it is not specified, it will be randomly generated. Only one zone name can be used. The zone name consists of the first nonspace character following the network distance (NetDistance) and all the characters to the end of the line.

The Number of Network Ranges is the number of networks (or network ranges) that are defined in the list that follows.

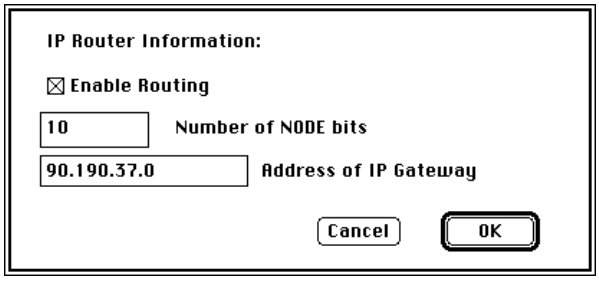
Note The first network specified in the topology file must have a distance of zero. The file format is the same as the format used by the BridgeSim test tool, except that zone names may be added. Gaps between the numbers of the file can be either spaces or tabs.

Add Test Router

Brings up a dialog box in which you enter the IP address of the router under test. The initial address displayed is based on the address given in the Preferences dialog box. If the command is invoked again, the default address is the previous address plus one. The maximum number of routers that can be specified is 50.

Set IP Gateway

Displays the IP Router Information dialog box (shown here), which allows Apple Tunnel Simulator to communicate to IP gateways and routers.



Since Apple Tunnel Simulator can create several IP addresses, it does not use MacTCP and thus does not have the information in the MacTCP control panel. The IP Router Information dialog box is the micro substitution for the control panel. Selecting the Enable Routing option causes Apple Tunnel Simulator to poll for the gateway and direct all traffic to IP addresses out of its network to the gateway. “Number of NODE bits” defines what is in Apple Tunnel Simulator’s local network. “Address of IP Gateway” is the IP address of the local gateway.

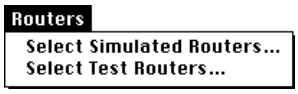
Run

Starts the simulation. If Initiate Router Connections is selected in the Preferences dialog box, then all of the simulated routers try to establish AURP connections with all of the test routers. Additional simulated routers and test routers may be added when the simulation is running.

The Run command toggles with the Stop command. When Stop is selected, Apple Tunnel Simulator does not produce nor respond to any packet data, and the menu command reverts to Run. When the router is restarted, any packets received in the interval between stopping and starting again are purged and lost.

Note Stopping the simulator does not tear down the connections; it freezes them.

Routers menu



The Routers menu is used to select the routers, both simulated and those under test, to be used as source and destination for packets sent by the Send menu. All combinations of addresses are used. If you select two simulated routers and two test routers, then four operations occur.

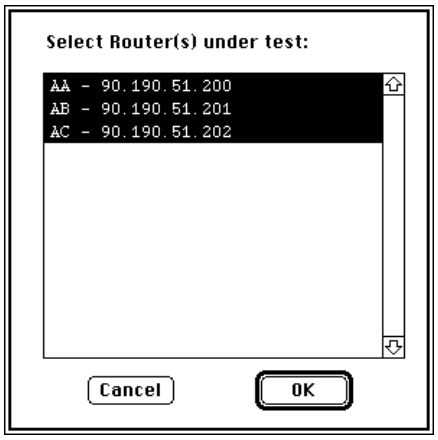
Select Simulated Routers

Brings up a dialog box with a list of IP addresses for all simulated routers. Any highlighted router addresses are used in conjunction with the commands in the Send menu. The dialog box is similar to the Select Test Routers dialog box (see the next section), except that the simulated routers are identified by numbers instead of letters.

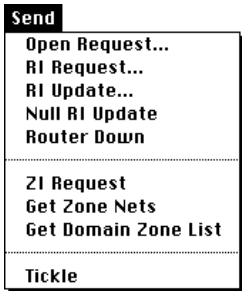
Select Test Routers

Brings up a dialog box with a list of IP addresses for all test routers. Any highlighted router addresses are used in conjunction with the commands in the Send menu.

Below is an example of the Select Test Routers dialog box. The two letters preceding each router's IP address are used to identify the router in the packet headers shown in the log window.



Send menu



Note This menu is used to send AURP packets. The packets are sent from selected simulated routers to selected test routers designated in the Router menu.

Open Request

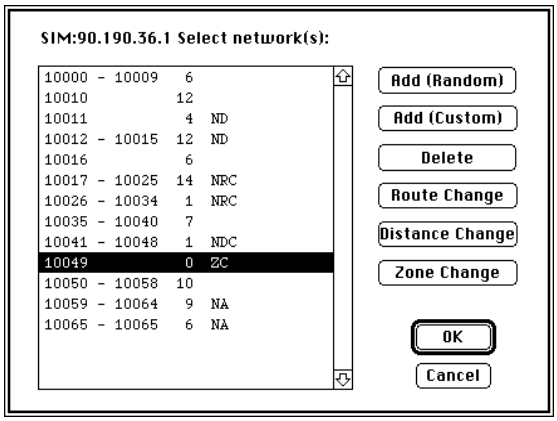
Sends an Open Request packet from all selected simulated routers to all selected test routers.

RI Request

Sends an RI Request packet from all selected simulated routers to all selected test routers.

RI Update

For each selected simulated router, displays a directory dialog box with a listing of all the networks created for that router. The buttons in the dialog box control attributes attached to those networks. Each button is described below.



Add (Random) Asks for and then adds the specified number of networks to the network list. The new networks will have network numbers in the range set in the Preferences dialog box.

Add (Custom) Asks for a network range and a zone name. Any network numbers can be used except numbers already in use by the simulator.

Delete Lets you mark for deletion any networks that are highlighted in the list.

Route Change and **Zone Change** Mark any selected networks for those respective updates. The Zone Change update command has not been fully defined in the AURP specifications, which means that Apple Tunnel Simulator sends the proper data, but the current implementation of the router ignores it.

Distance Change Asks for a new network distance for each highlighted network in the list and marks the networks for this change.

OK When the OK button is clicked, a packet is created based on the network updates selected. The packet is sent to the selected test routers.

Note If the number of update events created exceed one packet, Apple Tunnel Simulator gives a warning. The extra events are not sent but can be sent later by reselecting the RI Update command.

Null RI Update

Sends a Null RI Update packet from all selected simulated routers to all selected test routers.

Router Down

Sends a Router Down packet from all selected simulated routers to all selected test routers. The simulated routers reset to initial preconnection conditions.

ZI Request

Sends a ZI request packet from all selected simulated routers to all selected test routers. (This command is not implemented in this release.)

Get Zone Nets

Sends a GetZoneNets packet from all selected simulated routers to all selected test routers. Displays a dialog to enter the zone name to use in the packet.

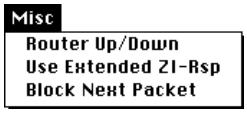
Get Domain Zone List

Sends a GetDomainZoneList packet from all selected simulated routers to all selected test routers. Displays a dialog box in which you enter the starting reference number to use in the packet. The default is 1, which gets the zone list from the beginning.

Tickle

Sends a Tickle packet from all selected simulated routers to all selected test routers

Misc menu



Router Up/Down

Forces each simulated router selected under the Routers menu to change state from up to down or down to up. If the simulated router is up, it sends a Router Down packet to all connected test routers and then ignores open requests from the test routers. If the simulated router is down, then selecting this command allows the simulated routers to connect to the test routers.

Use Extended ZI-Rsp

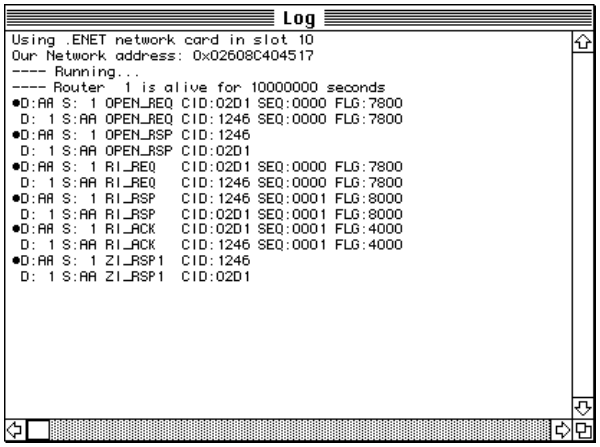
Forces each simulated router to send extended ZI-Rsp packets. The data in all normal ZI-Rsp packets is divided between two extended ZI-Rsp packets.

Block Next Packet

Tells Apple Tunnel Simulator to ignore the next packet sent to it. Apple Tunnel Simulator ignores one packet each time this command is selected. Thus, if this command is activated five times in quick succession, the simulator ignores the next five packets.

Log display

This window displays information text and status messages. It may be output to a long file by using the New Log File command in the File menu. A typical display of packet headers is shown below:



Lines starting with four dashes (----) are information messages. Lines starting with asterisks (****) are error messages.

The bulk of the lines above are packet headers. The format of the packet headers is as follows: A line that starts with a dot (•) is sent from the simulator to a test router. “D” is destination and “S” is source. “AA” signifies the first test router; subsequent routers would be AB, AC, and so forth. Simulated routers are numbered. The next column is the packet type. CID is the AURP connection ID in hex. SEQ is the sequence number in decimal. FLG is the packet flags in hex. If the sequence number and flags are zero, then those columns are not displayed.

Outgoing packet capture

Apple Tunnel Simulator supports outgoing packet capture and manipulation. Any packet that the simulator is about to send can be displayed and then sent as is, modified before being sent, or dropped. To intercept a packet, hold down the ⌘ and Shift keys before the packet is sent. A window containing the packet data in hex appears.

This feature is useful for testing router error detection by corrupting the packet headers and changing the packet content. It also can be used to duplicate or send custom packets.

> **WARNING** The window displays only the first 100 bytes of a packet. If the packet is larger than 100 bytes, you will not be able to modify the data beyond the first 100 bytes. Also, the code that decodes hex back into binary is not extremely robust.

Other features

- Apple Tunnel Simulator continues to run when any dialog box is displayed except for a directory dialog box. The simulator can also run in the background and with other programs, as long as the other programs multitask properly.

Note Do not leave directory dialog boxes open; if they are left open too long, Apple Tunnel Simulator will not handle received packets in a timely fashion.

- Apple Tunnel Simulator allows connections to test routers it does not have listed. If a test router sends an open request to a simulated router and Apple Tunnel Simulator does not know that test router's address, then it will be added. The simulator will eventually attempt to connect to the test router.

Known problems, limitations, and troubleshooting

- Simulated routers cannot be removed while the program is operating; to remove them, you must restart the application.
- You can have duplicate IP addresses for simulated routers (no checking is done).
- Apple Tunnel Simulator does not implement all the timeout features for AURP, and thus can be coerced into a confused state.
- Apple Tunnel Simulator does not work with Token Ring or Macintosh SE and SE/30 Ethernet cards.
- If a packet with a sequence number is dropped, then any further packets with sequence numbers on that channel will be out of sequence and Apple Tunnel Simulator will complain. However, the simulator should be able to resynchronize the packets.
- Apple Tunnel Simulator can run in 512 KB of memory and is able to generate a small network topology. One megabyte of memory is recommended. Larger topologies or multiple simulated routers require greater amounts of memory. Operating Apple Tunnel Simulator under low memory conditions causes unpredictable behavior and crashes.
- The packet display slows Apple Tunnel Simulator considerably. If there is a lot of information to display, connections may timeout or packets may be retransmitted. You can speed up the simulator by reducing the size of the display window or by deselecting Display Packet Headers in the Preferences dialog box.
- If Apple Tunnel Simulator displays "**** ERROR, ARP attach: - 94" when launched, remove MacTCP (or another protocol handler) and restart your computer before running Apple Tunnel Simulator again.
- Apple Tunnel Simulator has not been tested under System 6 and may work only under System 7.