Open Transport/PPP Client Developer Note

document version 1.0
11/5/96

Comments should be sent to the AppleLink address OPENTPT or Internet address opentpt@applelink.apple.com .

Table of Contents

Revision History

11/05/96	Added sections for new events, reworked sample code and updated options information.
09/27/96	More review, new sample code section
08/12/96	Began post-implementation review
03/15/96	Documentation review, draft 3
02/28/96	Documentation review, draft 2
02/12/96	Documentation review, draft 1
01/30/96	Created

Related Documents

IETF (Internet Engineering Task Force) RFC (Request for Comments) 1662: *PPP in HDLC-like Framing,* W. Simpson, July 1994.

IETF RFC 1661: *The Point-to-Point Protocol (PPP),* W. Simpson, July 1994.

IETF RFC 1332: *The PPP Internet Protocol Control Protocol (IPCP),* G. McGregor, May 1992.

IETF RFC 1334: *PPP Authentication Protocols*, B. Lloyd et al, October 1992.

IETF RFC 1471: *The Definitions of Managed Objects for the Link Control Protocol of the Point-to-Point Protocol,* F. Kastenholz, June 1993.

IETF RFC 1472: *The Definitions of Managed Objects for the Security Protocols of the Point-to-Point Protocol,* F. Kastenholz, June 1993.

IETF RFC 1473: *The Definitions of Managed Objects for the IP Network Control Protocol of the Point-to-Point Protocol*, F. Kastenholz, June 1993.

IETF RFC 1144: *Compressing TCP/IP Headers for Low-Speed Serial Links,* V. Jacobson, February 1990.

Open Transport Client Developer Note , November 30, 1995.

X/Open CAE Specification: *X/Open Transport Interface (XTI),* X/Open Company Limited, 1992.

Open Transport/PPP Overview

This document describes the applications programming interface (API) available for managing Open Transport/PPP (OT/PPP) links. The OT/PPP client API is part of the overall Open Transport client API and is based on Open Transport endpoints. This document should be used along with the *Open Transport Client Developer Note*, which describes general information about Open Transport endpoint libraries. For additional PPP-specific information, please refer to the relevant RFCs listed in the Related Documents section.

Open Transport/PPP is a fully OT-native implementation of the Point-to-Point Protocol. It provides standard PPP functionality along with many of the additional features common to the Apple Remote Access product family.

The current release of Open Transport/PPP supports TCP/IP over PPP. It is a client-only release: it does not answer incoming calls. Support for AppleTalk, additional network protocols, and server functionality will be included in future releases.

Version 1.0 of OT/PPP supports asynchronous framing only. Synchronous framing may be supported in a future release.

This release of OT/PPP does not support CCP (Compression Control Protocol), PPP datalink compression, and reliable transmission (i.e., HDLC LAP B). These features may be supported in future releases.

This release of OT/PPP supports a single active PPP "port" or link and one corresponding physical serial port at any given time. Currently, there is no programmatic way to select a physical port and to make the association between the PPP port and a selected physical port. The selection and association must be made manually with the "Connect via:" menu in the Modem control panel.

This release of OT/PPP does not allow client applications to permanently modify a user's PPP or Modem configurations on disk. This capability may be provided through a separate mechanism in a future release of Open Transport.

Technical Specifications

This document specifies the application programming interface (API) available for managing Open Transport/PPP connections. The OT/PPP API is a part of the overall Open Transport API and is based on Open Transport endpoints. To use this specification you will need some familiarity with endpoints and OT application programming. The document titled*Open Transport Client Developer Note* included with the Open Transport SDK describes how to use the OT API.

The OT/PPP API provides a set of commands to initiate connections and disconnections. It also provides definitions for XTI option management calls to customize the settings and PPP negotiable options used for a connection. Option management calls are not required when using the API: your program can connect and disconnect using the default configurations if custom settings are not required. The Options section below details each of the available configuration options.

The OT/PPP API also provides a set of asynchronous endpoint notification events. Events signal the all of the important activity on OT/PPP endpoints, such as DCE call completion, PPP negotiation completion and connection establishment. The Events section below describes each event and its corresponding notification parameters.

Using the OT/PPP API

As mentioned above, the OT/PPP API is part of the overall Open Transport API. To use it, you need the Open Transport Client SDK, Version 1.1.1 or later. You also need the OT/PPP SDK, which consists of a single header file, OpenTptPPP.h. This file defines all the data constants and structures referenced in the following specifications. There are no additional libraries in the OT/PPP SDK; to use it you simply link with OT client SDK libraries, OpenTransport.o and OpenTransportApp.o.

The OT/PPP API is based on control endpoints. An OT/PPP control endpoint is a generic Open Transport endpoint opened with an OT/PPP configuration. It is called a "control" endpoint because it does not support data transfer: it is used only to initiate and monitor PPP links. Unlike conventional data-transfer endpoints associated with networking protocols, the OT/PPP endpoint does not implement the Transport Provider Interface and does not support the OTBind, OTConnect, OTSnd, OTRcv, and OTDisconnect calls.

Opening OT/PPP Endpoints

To open an OT/PPP control endpoint, use the PPP control endpoint name:

```
#include <OpenTransport.h>
#include <OpenTptPPP.h>

ep = OTOpenEndpoint(OTCreateConfiguration(kPPPControlName), 0, NULL, &err));
```

OT/PPP endpoints may be opened synchronously or asynchronously. OT/PPP 1.0 supports only one logical PPP port at a time, which is associated with a physical hardware port. The physical port is specified by the user's "Connect via:" selection in the current Modem configuration. The number of OT/PPP endpoints opened on the logical PPP port is limited only by the amount of available memory. A PPP port can exist even if no OT/PPP endpoints are opened, as long as a previously initiated connection remains connected.

Sample Code

This document contains a Sample Code section. The code snippets provided in that section demonstrate the use of OT/PPP endpoints to temporarily customize the active PPP and Modem configurations, to start and stop connections, and to monitor the status of a connection. They do not present a complete OT/PPP application or a complete reference for using Open Transport endpoints.

Commands

The two primary OT/PPP actions, connecting and disconnecting, are initiated by issuing IOCTL commands on OT/PPP endpoints. Modem and connect script changes are also made with an IOCTL command. This use of IOCTLs is different from conventional data-transfer endpoints which implement the Transport Provider Interface and use OT calls such as OTConnect and OTDisconnect. There is no data transfer protocol on OT/PPP endpoints--they are used only for control of the PPP links. The IOCTL commands supported on OT/PPP endpoints are described in the sections below.

I_OTConnect

The Open Transport I_OTConnect IOCTL command causes an OT/PPP endpoint to initiate a connection. The connection-establishment process takes place in the background and involves several protocol layers in progressive phases, as well as end-to-end exchanges of control packets over the link. The process includes lower-layer connection establishment (i.e., modem, DSU/CSU, ISDN, etc.) , Link Control Protocol (LCP) connection establishment, authentication, and Network Control Protocol (NCP) connection establishment. If PPP events are enabled on the endpoint, asynchronous notification events will be issued to indicate significant link events during the connection-establishment process.

The I_OTConnect IOCTL call on OT/PPP endpoints usually completes immediately. IOCTL call completion indicates that the connection attempt has been started successfully, or that an error occurred; it does not mean that the PPP connection is fully established. Connection completion is signaled with the kPPPConnectCompleteEvent. You must install an endpoint notifier and handle this event to correctly detect OT/PPP connection completion. Refer to the Events section for the meaning of the various notifier parameters for this event.

If a connection has already been opened when the I_OTConnect command is issued, the command will be rejected, a kOTOutOfState error will be returned to the caller, and the existing connection will remain intact. The caller must tear down the existing connection using the I_OTDisconnect command before establishing a new connection using the I_OTConnect command.

```
//
// I_OTConnect is issued as a transparent IOCTL command
// with no data parameter.
//
err = OTIoctl(ep, I_OTConnect, NULL);
```

I_OTDisconnect

The Open Transport I_OTDisconnect IOCTL command causes an OT/PPP endpoint to initiate the connection-termination process. This process takes place in the background and involves several protocol layers in progressive phases, as well as end-to-end exchanges of control packets over the link. The termination process includes NCP connection termination, LCP connection termination, and lower-layer connection termination. If PPP events are enabled on the endpoint, asynchronous notification events will be issued to indicate significant link events during the connection-establishment process.

The I_OTDisconnect IOCTL call on OT/PPP endpoints usually completes immediately. IOCTL call completion indicates that the disconnection has been started successfully, or that an error occurred; it does not mean that the PPP connection is fully terminated. Termination is signaled with the kPPPDisconnectCompleteEvent. You must install an endpoint notifier and handle this event to correctly detect OT/PPP disconnect completion. Refer to the Events section for the meaning of the various notifier parameters for this event.

If a connection doesn't exist, the kRANotConnected error will be returned in the cookie parameter to the caller's notifier along with kPPPDisconnectComplete event. It isn't necessary to use the same endpoint that initiated a connection to terminate it. A different endpoint opened in the same or a different application can be used.

```
//
// I_OTDisconnect is issued as a transparent IOCTL command
// with no data parameter.
//
err = OTIoctl(ep, I_OTDisconnect, NULL);
```

I_OTScript

The Open Transport I_OTScript IOCTL command causes an OT/PPP endpoint to update either the current modem script or the current connect (terminal-server) script. Modem scripts automate DCE dialing using the Connection Control Language (CCL) introduced in Apple Remote Access 1.0. Connect scripts use the same CCL languages to automate terminal server log-in.

The I_OTScript command does not change the scripts chosen in the modem and PPP control panels; it merely overrides them temporarily. The effect of the new scripts will last until either they are overridden again (by the same or a different client application) or the last endpoint is closed and the connection is inactive.

Note: it is not necessary to issue the I_OTScript command if you don't need to override the scripts selected by the user in the PPP and Modem control panels. The selected scripts are used by default if no I_OTScript command is issued on an OT/PPP endpoint. However, if the user has not configured the Modem control panel, you must set a modem script with the I_OTScript command before issuing the I_OTConnect script or the connection attempt will fail with a kModemPreferencesMissing (-14001) or a kModemScriptMissing (-14002) error. This is because there is no default value for the modem script in the absence of a user configuration for the Modem layer.

The OTScriptInfo structure is defined as follows in Open Transport.h to describe the script you wish to set. You must read the script data into memory completely before issuing the I_OTScript call: OT/PPP endpoints do not read script data from files or other sources. Constants for specifying the type of script to set are defined in OpenTptPPP.h.

```
//
// From OpenTransport.h:
//
struct OTscriptInfo
{
   UInt32      fScriptType;
   void*       fTheScript;
   UInt32      fScriptLength;
};
```

```
//
// From OpenTptPPP.h:
//

#define kPPPScriptTypeModem     1
#define kPPPScriptTypeConnect   2

//
// Sample use of OTScriptInfo and I_OTScript.
//

OTScriptInfo info;
str.fScriptType    = kModemScript;
str.fTheScript     = scriptData;
str.fScriptLength  = scriptDataLen;

err = OTIoctl(ep, I_OTScript, &info);
```

There is a 32K byte limit to size of any script set with I_OTScript on an OT/PPP endpoint. If fScriptLength exceeds this limit, the data pointed to by fTheScript will be truncated to 32K when it is copied by the endpoint. All script data is copied before it is used by OT/PPP; the caller's buffer is not modified.

As with I_OTConnect and I_OTDisconnect, the I_OTScript IOCTL call on OT/PPP endpoints usually completes immediately. IOCTL call completion indicates that the script update has been started successfully, or that an error occurred; it does not mean that the script update is fully completed. Completion is signaled with the kPPPSetScriptCompleteEvent. You must install an endpoint notifier and handle this event to correctly detect OT/PPP script update completion. This is important if you plan to update scripts then start a PPP connection: your application should not issue I_OTConnect until it has received a kPPPSetScriptComplete event for every previous I_OTScript call. Refer to the Events section for the meaning of the various notifier parameters for the kPPPSetScriptComplete event.

Options

Configuration settings such as the user name, password, and the phone number to dial (if using a DCE that requires one) can all be set and examined onOT/PPP endpoints using the OTOptionManagement call. For general information about option management and using he OTOptionManagement call, please refer to the*Open Transport Client Developer Note* and the *X/Open Transport Interface (XTI)* specification.

You can examine the default setting for an OT/PPP option by using the OTOptionManagement call with the T_DEFAULT flag. You can examine the current setting for an OT/PPP option--which may be different from the default setting--by using OTOptionManagement with the T_CURRENT flag. You can temporarily override any or all of the default option settings specified below by using OTOptionManagement calls with the T_NEGOTIATE option flag set.

Most OT/PPP option values are simple long integers. However, a number of them are compound values containing more than one field. A sructure definition provides the format for each such option. These structures are defined in OpenTptPPP.h and are described in the sections below.

When to Set Endpoint Options

By default, OT/PPP endpoints use the settings in the user's currently selected PPP and Modem configurations, which are edited, respectively, with the PPP and Modem control panels. For this reason, it is not necessary to set every OT/PPP option on an endpoint. In fact, if the user has created both a PPP and a Modem configuration, it is not necessary to set any endpoint options before initiating a connection unless your application needs to override the user's choices.

If the user has not created a PPP or a Modem configuration, you can still connect without setting options with the following exceptions: the DTE address (phone number) and the modem script. All other settings have pre-defined defaults that apply in the absence of a user configuration. If no PPP configuration has been saved, you must set the CC_OPT_DTEADDRESS option (see below). If no Modem configuration has been saved, you must set the modem script using the I_OTScript command described in the Commands section of this document.

How OT/PPP Options are Applied

The options specified in the following sections are associated with the OT/PPP interface rather than with particular client endpoints. When you set an option with your endpoint, it applies to all endpoints opened on the same PPP port.

When you set an OT/PPP option the value remains in effect until one of the following happens: the option is set again with another OTOptionManagement call on any open endpoint, the value is modified during the connection negotiation (if it is a negotiable option), or the last OT/PPP endpoint is closed and the connection is terminated for any reason.

Option settings do not permanently change the user's selected configurations or any system defined default values. There is currently no facility for programmatically altering users' configurations. Configurations must be edited manually using the PPP and Modem control panels.

If multiple client applications attempt to change the options associated with the same PPP port, inconsistent results may occur. A client application may verify the result of a previously set option by reading back the current option setting using OTOptionManagement with the T_CURRENT flag. However, there is no guarantee that the option will not be changed again by a contending client application after you check the option value but before you issue the connect command.

IPCP_OPT_GETREMOTEPROTOADDR

This is a read-only option, which returns the remote node's protocol (IP) address. A client application may read this option after its endpoint notifier function receives the *kPPPIPCPOpenedEvent* event; the address is 0 before the arrival of the *kPPPIPCPOpenedEvent* event. The format of the value of this option is the InetAddress data structure, defined in the OpenTptInternet.h file:

```
struct InetAddress
{
   OTAddressType      fAddressType;  // always AF_INET
   InetPort           fPort;         // Port number; unused for this option
   InetHost           fHost;         // Host address in net byte order
   UInt8              fUnused[8];    // Don't use these bytes!!
};
typedef struct InetAddress InetAddress;
```

IPCP_OPT_GETLOCALPROTOADDR

This is a read-only option, which returns the local node's protocol (IP) address. Different local protocol addresses may be returned before and after the arrival of the *kPPPIPCPOpenedEvent* event; if this occurs, the address obtained after the arrival of the *kPPPIPCPOpenedEvent* event should override the address obtained prior to the arrival of the event. If the local node's protocol address is not defined prior to the arrival of the *kPPPIPCPOpenedEvent* event, a 0 address is returned if the option is read. The format of the value of this option is the InetAddress data structure.

IPCP_OPT_TCPHDRCOMPRESSION

This option specifies whether the local host will attempt to negotiate the TCP/IP header-compression option with the remote node. The option value is a 32-bit Boolean long word The acceptable option values are: {kIPCPTCPHdrCompressionDisabled = TCP/IP header compression disabled, kIPCPTCPHdrCompressionEnabled = TCP/IP header compression enabled}. The default value of this option is kPPPTCPHdrCompressionEnabled - i.e., the local host will attempt to negotiate TCP/IP header compression option. Enabling of this option does not guarantee that the compression will actually take place. The local host and the remote node must agree on the setting. TCP/IP header compression can reduce over 30 bytes of TCP/IP header overhead per frame and is essential for lower-speed links. It can be used in conjunction with the PPP datalink header-compression option described below. The recommended setting for this option for high-speed links is kIPCPTCPHdrCompressionDisabled.

```
    #define kIPCPTCPHdrCompressionDisabled   0
    #define kIPCPTCPHdrCompressionEnabled    1
```

LCP_OPT_ PPPCOMPRESSION

This option specifies whether the local host will attempt to negotiate the PPP compression option with the remote node. The option value is a 32-bit long word. The acceptable option values are: kPPPCompressionDisabled = PPP datalink compression disabled, or any combination of kPPPProtoCompression and kPPPAddrCompression. The default value of this option is (kPPPProtoCompression | kPPPAddrCompression), i.e. the local host will attempt to negotiate compression of the PPP packet protocol and address fields. Enabling this option does not guarantee that the compression will actually take place. The local host and the remote node must agree on the settings. PPP protocol and address field compression will reduce up to 3 bytes of header overhead per frame. PPP datalink header compression is useful for lower-speed links and can be used in conjunction with the TCP/IP header-compression option. The recommended setting for this option for high-speed links is kPPPCompressionDisabled = 0.

```
        #define kPPPCompressionDisabled     0x00000000
        #define kPPPProtoCompression        0x00000001
        #define kPPPAddrCompression         0x00000002
```

LCP_OPT_ MRU

This option specifies the desired MRU (Maximum Receive Unit) size and the upper and lower limits of the MRU negotiation range. The local host will advertise the desired MRU size to the remote node and will accept a proposed MRU size that falls within the range specified by the upper and lower MRU limits. The MRU size is the maximum size (i.e., datalink protocol header, trailer, and padding are excluded) of an OT/PPP frame that the local host is capable of receiving. The PPPMRULimits data structure specifies the option value format for LCP_OPT_MRU. The acceptable values for the desired MRU size and the upper and lower MRU limits are from 0 to 4500. The default value for the desired MRU size is 1500. The local host will not advertise the MRU size and the default MRU size (i.e., 1500) will be assumed, if the desired MRU size is set to 0. The default upper MRU Limit is 1500. The default lower MRU limit is 0; i.e., the local host will accept any proposed MRU size as long as it is smaller than the upper MRU limit.

To determine the optimal setting for this option, you should consider both response-time and throughput requirements of the intended user applications. You should balance the desired responsiveness of interactive applications such as Telnet and the desired throughput of bandwidth-intensive data-transfer applications such as FTP. You should also consider the adverse interaction between interactive traffic and bulk data-transfer traffic. The response time of interactive applications will be poor when small interactive packets are queued after much larger data-transfer packets. Link speed must also be taken into account when determining the MRU setting. Generally speaking, MRU may be further reduced to counter the inherent sluggishness of slower-speed links. Conversely, MRU may be further increased to boost the effective throughput of slower-speed links. Most ISPs (Internet Service Providers) recommend a certain MRU setting based on various factors.

```
typedef
{
   UInt32     mruSize;  // proposed or actual
   UInt32     upperMRULimit;
   UInt32     lowerMRULimit;
}
PPPMRULimits;
```

LCP_OPT_ RCACCMAP

This option applies only to asynchronous PPP framing. It sets up the local host's receive-side Asynchronous-Control-Character-Map (ACC-Map). The option value (i.e., the ACC-Map) is a 32-bit binary mask which specifies up to 32 control characters that may have a special meaning to the local DCE (Data Circuit-terminating Equipment) for in-band control purposes. If used by the local DCE (i.e., modem, DSU/CSU, etc.), the in-band control characters must be mapped into the appropriate 2-character escape sequences by the remote node in order to ensure successful reception of such characters by the local host. Each bit position (i.e., 0 to 31) of the ACC-Map corresponds to an ASCII control character of the same value; bits are set to inform the remote node to map the corresponding control characters before transmitting them across the link to the local host. The default OT/PPP option value is kPPPAsyncMapCharsNone, i.e. no control characters are mapped into the appropriate 2-character sequences. This default setting differs from the RFC specification but agrees with most current PPP server implementations.

Since it is often unnecessary to map any characters (if it is necessary, only a few selected characters are mapped), the local host may use this negotiation option to keep the inbound traffic overhead (i.e., stuffed characters) to the minimum. The local host must be able to handle unsolicited mapped characters since the remote node may map certain control characters so that they will pass through the remote DCE. If X-On and X-Off (i.e., instead of CTS/RTS hardware handshake) are used for DCE flow control, this option value may be set to kPPPAsyncMapCharsXOnXOff. When running asynchronous PPP framing over a bit synchronous link such as ISDN, the option value should be set to kPPPAsyncMapCharsNone.

```
#define kPPPAsyncMapCharsNone       0
#define kPPPAsyncMapCharsXOnXOff    0x000a0000
#define kPPPAsyncMapCharsAll        0xffffffff
```

LCP_OPT_ TXACCMAP

This option applies only to asynchronous PPP framing. It sets up the local host's transmit-side Asynchronous-Control-Character-Map (ACC-Map). The option value (i.e., the ACC-Map) is a 32-bit binary mask which specifies up to 32 control characters that may have a special meaning to the local DCE for in-band control purposes. If used by the local DCE, the in-band control characters must be mapped into the appropriate 2-character escape sequences by the local host in order to ensure successful transmission of all characters through the local DCE. Each bit position (i.e., 0 to 31) of the ACC-Map corresponds to an ASCII control character of the same value; bits are set to indicate which control characters are to be mapped by the local host before transmitting them across the link to the remote host. In addition to the local outbound character mapping requirement imposed by the local DCE, the local host must also map control characters according to the remote node's receive-side ACC-Map before transmitting them across the link to the remote node; the requirement imposed by the remote node is handled dynamically by the protocol and is transparent to the user. The default OT/PPP option value is kPPPAsyncMapCharsNone, i.e. no control characters are mapped into the appropriate 2-character sequences. This default setting differs from the RFC specification but agrees with most current PPP server implementations.

Since it is often unnecessary to map any characters (if it is necessary, only a few selected characters are mapped), the local host may use this option to keep the outbound traffic overhead (i.e., stuffed characters) to the minimum.   If X-On and X-Off (i.e., instead of CTS/RTS hardware handshake) are used for DCE flow control, the option value may be set to kPPPAsyncMapCharsXOnXOff. When running asynchronous PPP framing over a bit synchronous link such as ISDN, the option value should be set to kPPPAsyncMapCharsNone.

SEC_OPT_ OUTAUTHENTICATION

This option specifies the outbound (local-to-remote) authentication method to be used by the local PPP interface. Two authentication protocols are supported: Password Authentication Protocol (PAP) and Challenge-Handshake Authentication Protocol (CHAP). CHAP is preferred and more secured because it uses the password to encrypt a challenge string before sending the resulting encrypted response packet back to the remote node. Unlike CHAP, PAP sends a password string in a clear text format that makes PAP vulnerable to security attacks.

The option value is an unsigned 32-bit long word. The acceptable option values are: kPPPNoOutAuthentication= none, kPPPCHAPOrPAPOutAuthentication = either CHAP or PAP. The default option value is kPPPCHAPOrPAPOutAuthentication. Authentication is negotiated across the link and the ID and password values set by the SEC_OPT_ID option and the SEC_OPT_PASSWORD option are used. If these have not been set, the values from the current PPP configuration are used.

The SEC_OPT_ OUTAUTHENTICATION option should be set to kPPPNoOutAuthentication if the remote node does not require any PPP authentication methods (e.g., guest login).

```
#define kPPPNoOutAuthentication          0
#define kPPPCHAPOrPAPOutAuthentication   1
```

SEC_OPT_ ID

This option specifies the name used for outbound authentication. The ID value is a variable-length byte array containing up to 256 bytes. The option value must be passed as ASCII characters with no preceding length byte or trailing NULL character. The character array must begin immediately after the last field of the TOptionHeader structure, which is the first byte of the "value" field in the TOption structure. The user name length is determined by the length field of the TOptionHeader structure.

SEC_OPT_ PASSWORD

This option specifies the password to be used for outbound authentication. The password value is a variable-length byte array containing up to 256 bytes. The option value must be passed as ASCII characters with no preceding length byte or trailing NULL character. The character array must begin immediately after the last field of the TOptionHeader structure, which is the first byte of the "value" field in the TOption structure. The password length is determined by the length field of the TOptionHeader structure.

CC_OPT_ REMINDERTIMER

This option sets the connection reminder timer. Refer to the OPT_ALERTENABLE section for information on how to enable connection reminders. If the connection reminder dialog is enabled, it will appear when the connection reminder timer expires. If the connection reminder dialog remains unacknowledged by a user, the connection will be disconnected. The option value is an unsigned 32-bit long word. The acceptable option values are: kCCReminderTimerDisabled = connection reminder timer disabled; T_INFINITE (0xffffffff) = unlimited time; non-zero = enabled (timer value in milliseconds). The default option value is kCCReminderTimerDisabled.

```
#define kCCReminderTimerDisabled   0
```

CC_OPT_ IPIDLETIMER

This option sets the IP connection idle timer . If TCP/IP remains idle longer than the time specified by this option, the PPP connection will be torn down. The option value is an unsigned 32-bit long word. The acceptable option values are: kCCIPIdleTimerDisabled = IP connection idle timer disabled; T_INFINITE (0xffffffff) = unlimited time; non-zero = enabled (idle timer value in milliseconds). The default option value is 600,000 milliseconds (10 minutes).

```
#define kCCIPIdleTimerDisabled    0
```

CC_OPT_ DTEADDRESSTYPE

This option specifies the format of the remote Data Terminal Equipment (DTE) address to be set by the CC_OPT_DTEADDRESS option. The remote DTE address format is specific to the physical media used. It uses E.164 telephone number string format if the physical-media type is modem, DSU/CSU, or ISDN; it uses X.121 address format if the physical media type is X.25. The default option value is kPhoneAddress. The option value format is a 32-bit unsigned integer. The option setting must be consistent with the DTE address set by the CC_OPT_DTEADDRESS option.

```
#define kE164Address                1
#define kPhoneAddress               1
#define kCompoundPhoneAddress       2
#define kX121Address                3
```

CC_OPT_ DTEADDRESS

This option specifies the remote DTE address to be used when making a call. The DTE Address value is a variable-length character array containing up to 128 characters. The option value must be passed as ASCII characters with no preceding length byte or trailing NULL character. The character array must begin immediately after the last field of the TOptionHeader structure, which is the first byte of the "value" field in the TOption structure. The address length is determined by the length field of the TOptionHeader structure.

Before issuing an I_OTConnect command, a DTE address must be specified either programmatically by executing a CC_OPT_DTEADDRESS OTOptionManagement call or manually by using the PPP control panel.

CC_OPT_ CALLINFO

This option specifies the call info to be used when making a call. This optional call info may be required when an intermediate access network (i.e., X.25 PAD, X.25 packet switching, ISDN circuit/packet switching, etc.) is used to reach the remote peer. The call info value is a variable-length byte array containing up to 256 characters. CC_OPT_CALLINFO is not used in OT/PPP Version1.0.

CC_OPT_ GETMISCINFO

This is a read-only option which returns miscellaneous connection information, including connection status, elapsed connection time, remaining connection time, number of data bytes transmitted, and number of data bytes received. A client application may read this option after its endpoint notifier function receives the *kPPPConnectionCompleteEvent* notification; the option values are zeroed before the arrival of this event. The option value format is given by the CCMiscInfo structure. All times are expressed in milliseconds.

```
#define kPPPConnectionStatusIdle            1
#define kPPPConnectionStatusConnecting      2
#define kPPPConnectionStatusConnected       3
#define kPPPConnectionStatusDisconnecting   4

typedef struct
{
   UInt32     connectionStatus;
   UInt32     connectionTimeElapsed;
   UInt32     connectionTimeRemaining;
   UInt32     bytesTransmitted;
   UInt32     bytesReceived;
   UInt32     reserved;
}
CCMiscInfo;
```

PPP_OPT_ GETCURRENTSTATE

This is a read-only option which returns the current state of the PPP protocol. The option value is a 32-bit unsigned long word. A client application may read this option anytime after the endpoint is opened. The acceptable option values are listed below. Refer to RFC 1661 for details on what each state means.

```
#define kPPPStateInitial     1
#define kPPPStateClosed      2
#define kPPPStateClosing     3
#define kPPPStateOpening     4
#define kPPPStateOpened      5
```

OPT_ ALERTENABLE

OPT_ALERTENABLE is a generic Open Transport option which can be used to enables and disable OT/PPP connection reminders and dialogs. The option value is specified as a 32-bit binary field. If a bit is set or cleared, its corresponding dialog or reminder is enabled or disabled. The acceptable option values are listed below. The default option value is (kPPPConnectionFlashingIconFlag | kPPPOutPasswordDialogsFlag).

```
#define kPPPConnectionStatusDialogsFlag    0x00000001
#define kPPPConnectionRemindersFlag        0x00000002
#define kPPPConnectionFlashingIconFlag     0x00000004
#define kPPPOutPasswordDialogsFlag         0x00000008
#define kPPPAllAlertsDisabledFlag          0x00000000
#define kPPPAllAlertsEnabledFlag           0x0f
```

If the kPPPConnectionStatusDialogsFlag bit is set, OT/PPP will present a modal status window during the connect and disconnect phases. If the kPPPConnectionRemindersFlag bit is set, connection reminder dialogs will appear when the connection timer expires. If the kPPPConnectionFlashingIconFlag bit is set, the OT/PPP icon will flash continuously over the Apple menu while the connection is up. If the kPPPOutPasswordDialogsFlag bit is set, password and CCL ASK dialogs may be presented automatically by OT/PPP if user input is required. If this bit is not set and password or ASK dialog input is required to complete a connection, the connection may fail.

Events

By default, all events specified below are disabled. To receive these events, client applications must activate them by issuing the I_OTGetMiscellaneousEvent IOCTL command. The I_OTGetMiscellaneousEvent command tells the OT/PPP endpoint and stacks that the client is interested in receiving PPP events. Here is the syntax of the I_OTGetMiscellaneousEvent call:

```
OTResult result = OTIoctl(ep, I_OTGetMiscellaneousEvents, 1);
```

Passing 1 in the data argument enables PPP events; passing 0 disables them. Unless stated otherwise, the cookie and result parameters to OT/PPP notifier functions have no meaning for the following asynchronous notification events.

kPPPLowerLayerUpEvent

This event indicates that the lower-layer interface has gone up. The lower-layer interface refers to the serial interface hardware (i.e., RS232, RS422/449, V.35, X.21, etc.) that sits below the PPP link layer. The interface is up when the required hardware signals are present and the required handshakes are successful.

kPPPLowerLayerDownEvent

This event indicates that the lower-layer interface has gone down. The lower-layer interface refers to the serial interface hardware (i.e., RS232, RS422/449, V.35, X.21, etc.) that sits below the PPP link layer. The interface is down if the required hardware signals are not present, the required handshakes are not successful, or the local host has disabled the interface.

kPPPDCEInitStartedEvent

This event indicates that the local DCE (i.e., modem, DSU/CSU, ISDN, etc.) is being initialized. If the local DCE is a modem, kPPPDCEInitStartedEvent indicates that the modem initialization command has been sent to the local modem.

kPPPDCEInitFinishedEvent

This event indicates that the local DCE (i.e., modem, DSU/CSU, ISDN, etc.) initialization has completed. The `cookie` parameter passed to the endpoint's notifier contains the result code for this operation. The `result` parameter always contains `kOTNoError`.

kPPPDCECallStartedEvent

This event indicates that the local DCE (i.e., modem, DSU/CSU, ISDN, etc.) is initiating a call. If the local DCE is a modem, this indicates that the modem dialing is in progress.

kPPPDCECallFinishedEvent

This event indicates the completion of a call attempt by the local DCE. The `cookie` parameter passed to the endpoint's notifier function contains the result code for the call attempt. The `result` parameter always contains `kOTNoError`.

kPPPLCPUpEvent

This event indicates that the Link Control Protocol (LCP) has gone into the Opened state. This happens when LCP has successfully negotiated its options with its connection peer.

kPPPLCPDownEvent

This event indicates that LCP has gone out of the Opened state. This event may be caused by an administrative action (i.e., an intentional disconnect) or by catastrophic or temporary link-error conditions. In the case of temporary link-error conditions, automatic recovery may be performed; i.e. the LCP will transition back to the Opened state when the error conditions are removed.

kPPPAuthenticationStartedEvent

This event indicates that the authentication exchange has been started.

kPPPAuthenticationFinishedEvent

This event indicates that an authentication exchange has completed. The `cookie` parameter passed to the endpoint's notifier contains the result code for the authentication exchange. The `result` parameter always contains `kOTNoError`.

kPPPIPCPUpEvent

This event indicates that the IPCP has gone into the Opened state and is fully operational. This happens when IPCP successfully negotiates its options with its connection peer. TCP/IP data traffic is enabled over the OT/PPP link when this event is generated.

kPPPIPCPDownEvent

This event indicates that the IPCP has gone out of the Opened state. This event may be caused by an administrative action (i.e., an intentional disconnect) or by catastrophic or temporary link-error conditions. In the case of temporary link-error conditions, automatic recovery may be performed; i.e. the IPCP will transition back to the Opened state when the error conditions are removed.

kPPPConnectCompleteEvent

This event indicates the completion of the OT/PPP connection process. When kPPPConnectCompleteEvent is received, the endpoint is either fully connected, or ready for another connection attempt because the previous one did not succeed. The endpoint notifier must examine the result code in the cookie parameter to determine whether a connection attempt succeeded or failed.

Note: the I_OTConnect command always executes asynchronously on OT/PPP endpoints, even if the IOCTL is sent while the endpoint is in synchronous mode. The completion of the IOCTL call simply means the command was started successfully (or could not be started if there was error). The endpoint user must handle kPPPConnectCompleteEvent in the notifier to know when the command is complete.

kPPPDisconnectCompleteEvent

This event indicates the completion of the OT/PPP disconnect process. When kPPPDisconnectCompleteEvent is received, the endpoint is either fully disconnected, or is ready for another disconnect attempt because the previous one did not succeed. The endpoint notifier must examine the result code in the cookie parameter to determine whether a disconnection attempt succeeded or failed.

Note: the I_OTDisconnect command always executes asynchronously on OT/PPP endpoints, even if the IOCTL is sent while the endpoint is in synchronous mode. The completion of the IOCTL call simply means the command was started successfully (or could not be started if there was error). The endpoint user must handle kPPPDisconnectCompleteEvent in the notifier to know when the command is complete.

kPPPDisconnectEvent

This event indicates that an unrequested disconnection occurred on the OT/PPP endpoint. This can happen for a number of reasons, including the expiration of a login account time limit, a forced disconnection by the server administrator, or loss of contact with the server. The endpoint notifier may examine the result code in the cookie parameter to determine the cause of the disconnect (if it is known).

kPPPSetScriptCompleteEvent

This event indicates that a previous attempt to set an endpoint's modem or connect script has completed. The endpoint notifier may examine the result code in the cookie parameter to determine whether the script change succeeded or not. Like I_OTConnect and I_OTDisconnect, I_OTScript always executes asynchronously, even if the IOCTL command is sent while the endpoint is in synchronous mode. The completion of the IOCTL call simply means the command was started successfully (or could not be started if there was an error). The endpoint user must handle kPPPSetScriptCompleteEvent in the notifier to know when the command is complete.

Sample Code

The sections below present samples of how to open and use OT/PPP control endpoints. They do not demonstrate the use of all the available options and are not meant to be a complete OT/PPP application reference.

The basic sequence of events when to follow when using OT/PPP endpoints is:

- open an the endpoint
- install a notifier function
- set the endpoint operation mode (i.e., asynchronous or synchronous)
- activate PPP-specific notification events with I_OTGetMiscellaneousEvents
- set any required XTI options, modem scripts or connect scripts
- issue the I_OTConnect command

After the I_OTConnect command is issued, your notifier will begin receiving connection events. At any time during or after the connection establishment phase, you can issue the I_OTDisconnect command to terminate the connection. After your notifier receives the kPPPConnectCompleteEvent event, you can use OTOptionManagement calls to retrieve the current value of any of the PPP options that can change during negotiation or are updated periodically while a connection is active (e.g. CC_OPT_GETMISCINFO).

Opening OT/PPP Endpoints

The code snippet below shows how to use OTOpenEndpoint to open an OT/PPP control endpoint. Asynchronous opens can also be done with the OTAsyncOpenEndpoint call.

```
#include <OpenTransport.h>
#include <OpenTptPPP.h>

TendpointInfo  epInfo;
OSStatus       err;
EndpointRef    ep;

//
// Open a synchronous OT/PPP control endpoint
//
ep = OTOpenEndpoint(OTCreateConfiguration(kPPPControlName), 0, &epInfo, &err));
```

OT/PPP clients must install a notifier function in order to monitor endpoint events. Because the connection and disconnection processes continue after the synchronous completion of I_OTConnect and I_OTDisconnect commands, you must monitor events to know when these processes are complete.

```
err = OTInstallNotifier(ep, MyNotifierFunc, &MyOptionalContext);
```

An endpoint client can remove a notifier function using the OTRemoveNotifier call and re-install a new one using the OTInstallNotifier call.

A client application may change the operation mode of an OT/PPP control endpoint by using the *SetSynchronous* or *SetAsynchronous* call as follows:

```
        SetAsynchronous(ep);
        SetSynchronous(ep);
```

Enabling OT/PPP Events

By default, Open Transport endpoint notifiers support a standard set of events. The additional events defined by Open Transport/PPP are not automatically sent to endpoints. You can enable these events with the follow IOCTL command:

```
        err = OTIoctl(ep, I_OTGetMiscellaneousEvent, 1);
```

OT/PPP events can be disabled with the same command: pass 0 instead of 1 as the IOCTL data.

Starting a Connection

If you don't need to override the current PPP configuration (selected in the PPP control panel), your endpoint now ready to make a connection as soon as the open call completes.

```
        err = OTIoctl(ep, I_OTConnect, NULL);
```

The data parameter for the I_OTConnect command must be NULL. If the result value indicates success (kOTNoError), the connection attempt has been started. The I_OTConnect command always completes immediately, whether the OT/PPP endpoint is in synchronous or asynchronous mode. However, the connection process (setting up the modem, dialing the phone number, negotiating PPP options, etc.) always completes asynchronously. Completion is signaled with the kPPPConnectCompleteEvent event. You must have a notifier installed to get all the PPP events that happen during the connect. You also have to issue the I_OTGetMiscellaneousEvents IOCTL to enable the flow of PPP-specific events on the endpoint.

Monitoring Events

You use the same kind of notifier function for OT/PPP control endpoints as for other types of Open Transport endpoints. The code fragment below shows a notifier which handles a generic endpoint event (T_OPTMGMTCOMPLETE) and one of the OT/PPP event (kPPPConnectCompleteEvent). For a complete list and description of all OT/PPP events, refer to the Events section of this document. Remember that PPP-specific events are not enabled by default when you open an OT/PPP endpoint. You must enable them with I_OTGetMiscellaneousEvents IOCTL command. See the "Enabling Events" section for details on how to use this command.

```
        pascal void
        MyPPPEndpointNotifier (void *context, OTEventCode code, OTResult result, void *cookie)
        {
           switch (code)
           {
              case T_OPTMGMTCOMPLETE:
                 TOptMgmt*  temp = (TOptMgmt *) cookie;
                 TOption*   option = (TOption *) temp->opt.buf;
```

```
            //
            // Note: the cookie may contain a TOptMgmt with more than
            // one trailing TOption, depending on the original request.
            // Use the "len" field of TOptMgmt and the OT macros for
            // for handling options to parse the TOptMgmt buffer. This
            // snippet just checks the first option in the list.
            //
            if (option->status != T_SUCCESS)
               AlertOptionFailed(option);
            break;

         case kPPPConnectCompleteEvent:
            //
            // Note: result code for PPP-specific events is in the
            // "cookie" field, not the "result" field!
            //

            OTResult connectResult = (OTResult) cookie;

            if (connectResult != kOTNoError)
            {
               DebugStr("\p kPPPConnectCompleteEvent with error!");
               gPPPConnected = false;
               // Handle connection failure...
            }
            else
            {
               gPPPConnected = true;
               // Handle connection success...
            }
            break;

         default:
            DebugStr("\p Unknown event code!");
            break;
      }
   }
```

Note that most OT/PPP events do not return a result code in the `result` parameter of the notifier: this parameter normally contains the `kOTNoError` value. Result codes for OT/PPP-specific events are passed in the `cookie` parameter.

If you open your OT/PPP synchronously using OTOpenEndpoint, you can install your notifier at any time after the open call completes. The following snippet installs the MyPPPEndpointNotifier function with a NULL context parameter:

```
   err = OTInstallNotifier(ep, MyPPPEndpointNotifier, (void *) NULL);
```

It is not necessary to include mixed-mode glue for Open Transport notifier functions.

Setting Option Values

Your application can temporarily override OT/PPP configuration settings by making OTOptionManagement calls with the T_NEGOTIATE option flag. The effect of the new settings will last until either the options are overridden again or the last endpoint of the connection is closed and the connection becomes inactive (i.e. disconnected). The code fragment below demonstrates setting a single option value, the Maximum-Receive-Unit for LCP.

```
UInt8           buf[sizeof(TOptionHeader) + sizeof(PPPMRULimits)];
TOption*        option;
PPPMRULimits*   pppMRU;

cmd.opt.buf      = buf;
cmd.opt.len      = sizeof(TOptionHeader) + sizeof(PPPMRULimits);
cmd.opt.maxlen   = sizeof(buf);
cmd.flags        = T_NEGOTIATE;

option           = (TOption *) buf;
option->len      = sizeof(TOptionHeader) + sizeof(PPPMRULimits);
option->level    = COM_PPP;
option->name     = LCP_OPT_MRU;
option->status   = 0;

pppMRU                  = (PPPMRULimits *) &option->value[0];
pppMRU->desiredMruSize  = 500;
pppMRU->upperMruLimit   = 1500;
pppMRU->lowerMruLimit   = 250;

err = OTOptionManagement(ep, &cmd, &cmd);
```

Note that the TOptionHeader "status" field must be set to 0 before making the T_NEGOTIATE call. Note also that the value for this option contains the PPPMRULimits structure. The value data always begins immediately after the "status" field of TOptionHeader. A common mistake when using Open Transport option management is to place the value data after a TOption structure: TOption includes the first sizeof(UInt32) bytes of the value.

Setting the Modem Script

The modem script selection set by the user in the Modem control panel can be temporarily overridden on OT/PPP endpoints using the I_OTScript command. Refer to the "Commands" section of this document for more details on the I_OTScript command. The following code snippet shows how to issue the command. Note that the data parameter to the IOCTL call must point to an OTScriptInfo structure. The fTheScript member of OTScriptInfo must point to the script contents, not to an FSSpec structure or other type of file descriptor. You must read the contents into a buffer from the file or other source yourself before issuing the I_OTScript call.

```
SInt32        scriptLen = 0;
OTScriptInfo  info;
UInt8*        data;

//
// Open file here, read in contents, set scriptLen to length
// of data read from script file...
//
```

```
info.fScriptLength    = fileLen;
info.fScriptType      = kPPPScriptTypeModem;
info.fTheScript       = (void *) data;

OTSetAsynchronous(ep);
err = OTIoctl(ep, I_OTScript, &info);
```

The completion of the I_OTScript command is signaled by the kPPPSetScriptCompleteEvent. You must handle this event in your endpoint notifier to detect whether setting a script succeeded or failed.

Retrieving Option Values

You can retrieve current and default option values with the OTOptionManagement call and the T_CURRENT and T_DEFAULT flags. Default options represent the values used by an OT/PPP endpoint if no overriding value has been set with the T_NEGOTIATE flag by any endpoint on the given OT/PPP port. Default values correspond to settings in the user's current PPP configuration, which is selected and edited in the PPP control panel, or the current Modem configuration from the Modem control panel. In some cases (e.g. LCP MRU), a setting can not be directly edited with either the PPP or the Modem control panel. In such cases, option management is the only way, at this time, to override the default value.

The following snippet demonstrates how to retrieve the current value of the CC_OPT_MISCINFO option. Note that the value is actually a structure containing a number of informative fields, and that this option has no "default" value per se. It is only useful when a connection is in progress.

```
TOptMgmt   cmd;
TOption*   option;
UInt8      buf[kBigEnough];

cmd.opt.buf      = buf;
cmd.opt.len      = sizeof(TOptionHeader);
cmd.opt.maxlen   = sizeof buf;
cmd.flags        = T_CURRENT;

option           = (TOption *) buf;
option->level    = COM_PPP;
option->name     = CC_OPT_GETMISCINFO;
option->status   = 0;
option->len      = sizeof(TOptionHeader);

OTSetSynchronous(ep);
err = OTOptionManagement(ep, &cmd, &cmd);

option = (TOption *) cmd.opt.buf;

if (option->status == T_SUCCESS)
{
   CCMiscInfo *info = (CCMiscInfo *) &option->value[0];
   //
   // Are we connected?
   //
   if (info->connectionStatus == kPPPConnectionStatusConnected)
   {
      // Update time-elapsed display, for example...
   }
}
```

Disconnecting OT/PPP

The I_OTDisconnect command can be issued at any time on an OT/PPP endpoint to disconnect an established connection or to cancel a connection attempt that is in progress. I_OTDisconnect does not require a data parameter, and it always executes asynchronously. You must handle the kPPPDisconnectComplete event in your notifier to detect completion of the command.

```
        err = OTIoctl(ep, I_OTDisconnect, NULL);
```

Closing OT/PPP Endpoints

After setting up an OT/PPP connection, your application can keep its endpoint open to monitor the link status or issue an I_OTDisconnect. It can also close the endpoint. Closing the endpoint does not automatically disconnect OT/PPP, even if your endpoint is the last one opened for the given OT/PPP port. The I_OTDisconnect command must be issued on an open endpoint, or a disconnect event must occur, for the connection to be torn down. OT/PPP endpoints are closed with the same OT call used for other types of endpoints:

```
    err = OTCloseProvider(ep);
```

After closing your OT/PPP endpoint, you can open another one at a later time to issue more commands or examine the endpoint's options.

Index