Open Transport/TCP Developer Note

Revision 1.1b14
1/18/96

Introduction

Open Transport TCP/IP is a replacement for MacTCP (currently at version 2.0.6). It is designed for use within the Open Transport framework.

Open Transport provides the standard XTI interface (t_open(), t_bind(), etc.) as well as preferred C and C++ versions which provide a superset of XTI. We recommend that developers consider using the preferred versions of the interface in their default asynchronous mode because they are more efficient on the Macintosh.

Installation

Open Transport/TCP is installed by the Open Transport installer. Using Easy Install, Open Transport/TCP and MacTCP backwards compatibility are both installed. Open Transport /TCP and MacTCP are mutually exclusive. The Open Transport/TCP installer will remove the MacTCP control panel if it exists, so if you want to save it, make sure to move out of the system's control panels folder before using the Open Transport installer to install TCP.

Using Custom Install, Open Transport/TCP can be installed without backwards compatibility. This can save some memory if you will not be using any applications which need the old MacTCP interface. However, even without backward compatibility, MacTCP and Open Transport/TCP cannot be run at the same time.

Use the TCP/IP application in the control panel's folder to configure Open Transport/TCP. This is detailed in a following section of this document.

Getting started with the Open Transport/TCP API.

Open Transport/TCP is programmed using the standard XTI interface. The same network API is used for AppleTalk, TCP, and other networking protocols. Developers should be familiar with the documentation from X/Open on XTI as well as the overall Open Transport documentation before proceeding with the Open Transport/TCP API. Please pay special attention to appendix B of the XTI manual, "Intern Protocol-specific Information". All of the information listed within Appendix B is valid for Open Transport/TCP. This document will generally not reiterate information covered within that documentation (e.g. usage of IP options).

To use Open Transport TCP/IP, include "OpenTptInternet.h" within your source, and link your program with either "OpenTransport.o" and "OpenTptInet.o" (for far model) or "OpenTransport.n.o" and "OpenTptInet.n.o" (for near model).

Opening TCP, UDP, and RawIP Endpoints

TCP , UDP and RawIP are opened by doing an OTAsyncOpenEndpoint() or OTOpenEndpoint() and passing in a configuration for TCP or UDP. See the sample programs for examples. Currently supported protocols are TCP, UDP, and RawIP. Constants for each protocol are defined in OpenTptInternet.h. Use kTCPName for TCP, kUDPName for UDP, and kRawIPName for RawIP. Do not open the DNR directly; use either the Internet Services interface or the Mapper interface instead.

Using RawIP

RawIP behaves for the most part identically to UDP, as a connectionless datagram interface, but there are a few unique caveats.

First, by default the RawIP interface will send ICMP packets (protocol 1). To change the protocol number being sent, use the option XTI_PROTOTYPE, defined in OpenTransport.h. The option is one long word which includes the protocol number to be used by the RawIP endpoint.

Second, when using RawIP, all RawIP endpoints bound to a specific protocol will receive a copy of any inbound packets destined for that protocol. For example, if several ping programs are using ICMP, each will receive a copy of all inbound ICMP echo datagrams.

Finally, the data returned will include the full IP header, which is 20 bytes in length if no options are present.

Internet Addresses

Fully qualified Internet addresses are 16 bytes in length and include an address type, a port number, and an IP address. The structure is laid out identically to the address structure used on most UNIX systems (e.g.. sockaddr_in). Supported address types are AF_INET (described here) and AF_DNS (described in a following section; it identifies a host by name rather than by address). Some utilities are provided (also detailed later in this document) which make address setup simpler for developers.

Internet Address Information

Information about the IP addresses of local interfaces can be obtained using the OTInetGetInterfaceInfo call which is described in the Function Descriptions section. Note that there is no active interface or IP address prior to running a TCP application unless the "Load only when needed" option is not checked within the TCP/IP control panel (by default, TCP is set to "Load only when needed"). This is because interface setup and server configuration of IP addresses is otherwise deferred until the interface is actually used for the first time.

While XTI does provide another interface to obtain address information, the t_getprotaddr call, this call doesn't behave the same way in Open Transport/TCP that it does in Open Transport/AppleTalk. Because Open Transport/TCP is designed for multihoming (*although this is not provided in the first release*) and IP is designed to allow any endpoint to use any and all IP interfaces on a system, endpoints frequently bind to an IP address of 0, which indicates the client will accept packets from any valid interface. However, a client which binds to IP address 0 and then does a t_getprotaddr call may be surprised to find an address of 0 returned. t_getprotaddr does indeed return the valid IP address which the client has bound to, but that address is 0.

Use the OTInetGetInterfaceInfo call to learn about the IP addresses of all IP interfaces currently up on a system. Then, if the client only wishes to receive packets for a single interface, bind to the IP address for that interface. After doing this, the t_getprotaddr call will indeed return the correct IP address which the client is bound to.

Domain Name Resolver (DNR)

The domain name resolver (DNR) has been implemented under Open Transport. The interface to the DNR has been provided through the InternetServices interface, described later. An additional interface to the DNR's name resolution capability has been provided through the Open Transport Mapper Library's LookupName function, described later in this document. See the *Open Transport Client Developer Note* for a description of the Mapper Library. Finally, the DNR's name resolution capability may be accessed indirectly through use of a UDP endpoint's SndUData() call, a TCP endpoint's Connect() call, or either's ResolveAddress() call.

Open Transport/TCP implements a caching stub name resolver. When the internet subsystem is loaded, the DNR is initialized with the following:

- a prioritized list of search domains. These are explicitly configured in the TCP/IP control panel, but others may be implicitly derived from the Default Domain and the Administrative Domain (see references to the Administrative Domain under the **Configuration** section).
- a prioritized list of IP addresses for name servers which can be used. These may be downloaded from a DHCP, BOOTP, or MacIP IPGATEWAY server, configured manually in the TCP/IP control panel, or be obtained from the HOSTS file.
- a (possibly empty) list of name-to-address mappings from the HOSTS file to be cached.
- a (possibly empty) list of cname-to-name mappings from the HOSTS file to be cached.

When a client of the DNR requests a name-to-address mapping, the DNR checks for a "." at the end of the name. If it exists, the name is assumed to be fully qualified (see rfcs 1034 and 1035 for an explanation of the Domain Name System). Otherwise, if the name contains at least one "." internally, it is considered to be provisionally fully qualified. Otherwise, the name is assume to be partially qualified, and the DNR will begin a search for that name in the first of the configured search domains.

For each search domain, the configured name servers are contacted in the order specified. If the name is resolved in the first search domain, that answer is returned. If an authoritative answer that the name-does-not-exist is returned, the DNR begins the search in the next configured search domain. The search continues through the domains, and if no match is found, the DNR will search the root domain if it makes sense to do so. The DNR has an overall timeout of 2 minutes after which it will abandon its search.

The DNR will make queries for any type and class via the OTInetQuery call. Simplified interfaces are provided for the most commonly made queries: name-to-address (A), address-to-name (PTR), system CPU and OS (HINFO), and mail exchange (MX) queries.

Currently, the DNR only caches name-to-address and cname-to-name mappings. Address-to-name translations are not cached, nor are they resolved by checking previously cached name-to-address translations. This is because doing so defeats some existing server load balancing schemes. In a future release address-to-name translations will be cached separately from name-to-address translations. Applications, such as web page servers, which may depend heavily on address-to-name translations may perform better if they cache the results of address-to-name translations.

The DNR does not implement negative caching - it depends on the local full service resolver to provide this facility. It always requests recursion, but it will follow up on references if recursion is not available. It does not save name server references after a query is resolved. Further queries begin anew at the configured name servers.

We recommend users of Open Transport/TCP avoid using a HOSTS file if a name server is available. Frequently HOSTS files contain many names unused by the local host. This merely wastes memory in the local cache. If a HOSTS file is used, making it as short as possible will conserve memory.

IP Multicast

Open Transport/TCP contains full support for IP Multicasting (level 2 as described in rfc 1112). IP Multicast configuration is supported through OTOptionManagement() or the IP endpoint's OptionManagement() method, both documented elsewhere.

To have IP join a multicast group, use the IP_ADD_MEMBERSHIP option, and pass in a TIPAddMulticast structure as the option value. Set the TIPAddMulticasts's multicastGroupAddress field to the group address you wish to join. Set the interfaceAddress field to the IP address of the interface on which you wish to join the group (group membership is on a per-interface basis). Use kOTAnyInetAddress for the default multicast interface's address.

Use the IP_MULTICAST_TTL to set the time-to-live value, which otherwise defaults to 1, for outbound data. Pass the desired TTL value (in seconds) as the option value. By default IP will loop back multicast datagrams to any member of the group on the sending machine.

The IP_MULTICAST_LOOP option may be use to turn off this feature (pass in a value of 0). Each of these options is demonstrated in the samples OTMulticastPitch.cp and OTMulticastCatch.cp.

MacTCP Backward Compatibility

MacTCP backward compatibility is now installed by default. If you do not want an existing MacTCP removed and replaced with Open Transport/TCP, make sure to do a custom install. The installer will remove the MacTCP control panel if one exists, so you should save a copy outside of your system folder before doing the installation.

If you are writing new applications, please use one of the Open Transport interfaces. The performance will be better than using the backward compatible API, and because the compatibility module will be loaded on demand, less memory will be used. Backward compatibility will be supported until most applications use the new interface. After that, expect it to be removed.

MDEV Backward Compatibility, SLIP and PPP

We have no plans to provide general backward compatibility below IP, at the MDEV level. However, we are supporting selected MDEVs in the first release of Open Transport TCP. Currently tested and supported MDEVs include: FreePPP 1.0.4, MacPPP 2.1.4, InterPPP 1.2.9, InterPPP II 1.0.7, MacSLIP 3.0.2, InterSLIP 1.0.1, VersaTerm SSLIP 1.1.4.

Known Problems

Check the *Client ReadMe* for a list of the currently known problems with Open Transport/TCP.

Configuration

Open Transport /TCP is supported over Ethernet, 802.3, TokenRing, and AppleTalk (as MacIP). It is also supported over serial lines when using backward compatible MDEV support.

Open Transport/TCP is configured using the TCP/IP application which is in the System's Control Panels folder. Configuration may be done manually, or via a BOOTP, DHCP, RARP, or MacIP server. The steps to follow using each of these methods are detailed below.

By default, the TCP/IP application comes up in basic mode. Advanced or administration mode may be entered via the Edit menu. These modes allow expert users additional choices as well as the ability to augment information returned from a configuration server or to fill in gaps in the returned information.

The TCP/IP application may be used at any time to reconfigure the system. However TCP will not notice the new configuration until it has unloaded from the system. In the current release, saving configuration parameters by closing the TCP/IP application forces TCP to unload immediately, breaking any existing connections. Otherwise, TCP will unload by default about 2 minutes after the last application using TCP , RawIP or UDP has gone away.

Manual Configuration

To manually setup Open Transport/TCP, follow these steps:

(1) Select the interface to use, or pick "AppleTalk (MacIP)" to run over AppleTalk on the interface selected in the AppleTalk control panel.

(2) If an Ethernet interface is selected, a check box will appear offering the use of 802.3 packet encapsulation. By default, Open Transport/TCP uses Ethernet rather than 802.3.

(3) Select "Manually" as the configuration method.

(4) Fill in the IP address in dot notation (e.g. 128.1.1.1).

(5) Fill in the search domain extensions to be used on name searches. For example, if a domain name of "apple.com" is configured, a search for "sam" would initially search for "sam.apple.com." It is not always necessary to fill in these fields when configuring from a BOOTP or DHCP server since a search domain may be returned along with the IP address.

In Advanced User mode only, you may configure an implicit domain search list . This is used to allow implicit searches. For example, if I wish to search all of the subdomains between "joesgroup.mktg.apple.com" and "apple.com", a search for the name "sam" would first look for "sam.joesgroup.mktg.apple.com", then for "sam.mktg.apple.com", then for "sam.apple.com." Implicit searching will not be done unless the Starting domain name is a subdomain of the Ending domain name (which corresponds to the domain of local administration per RFC 1535).

(6) Fill in the subnet mask in dot notation. For example, on a class B net which uses 6 bits of the host field for subnetting, the subnet mask should be entered as "255.255.252.0".

(7) Fill in the IP address(es) of one or more IP routers.

(8) Fill in the IP address(es) of one or more Domain Name Servers.

(9) If a specific Hosts file is required, select it using the Hosts file button. For details about the Hosts file, see the description which follows.

The TCP/IP application may be used at any time to reconfigure the system. However TCP will not notice the new configuration until it has unloaded from the system. In the current release, saving configuration parameters by closing the TCP/IP application forces TCP to unload immediately, breaking any existing connections. Otherwise, TCP will unload by default about 2 minutes after the last application using TCP , RawIP or UDP has gone away.

DHCP Configuration

To use a DHCP server to setup Open Transport/TCP, follow these steps:

(1) Select the interface to use, or pick "AppleTalk (MacIP)" to run over AppleTalk on the interface selected in the AppleTalk control panel.

(2) If an Ethernet interface is selected, a check box will appear offering the use of 802.3 packet encapsulation. By default, Open Transport/TCP uses Ethernet rather than 802.3.

(3) Select "Using DHCP" as the configuration method.

(4) Fill in the search domain extensions to be used on name searches. For example, if a domain name of "apple.com" is configured, a search for "sam" would initially search for "sam.apple.com." It is not always necessary to fill in these fields when configuring from a BOOTP or DHCP server since a search domain may be returned along with the IP address.

In Advanced User mode only, you may configure an implicit domain search list . This is used to allow implicit searches. For example, if I wish to search all of the subdomains between "joesgroup.mktg.apple.com" and "apple.com", a search for the name "sam" would first look for "sam.joesgroup.mktg.apple.com", then for "sam.mktg.apple.com", then for "sam.apple.com." Implicit searching will not be done unless the Starting domain name is a subdomain of the Ending domain name (which corresponds to the domain of local administration per RFC 1535).

(5) *In Advanced User mode, a subnet mask may be entered but is not required. If a value is entered, it will be used if no subnet mask is returned from the DHCP server. Otherwise, any value entered is ignored.*

(6) *In Advanced User mode, the manually entered IP addresses of routers are attached to the end of the possibly empty) list of IP routers returned by the DHCP server.*

(7) *In Advanced User mode, the manually entered IP addresses of Domain Name Servers are attached to the end of the (possibly empty) list of Name Servers returned by the DHCP server.*

(8) If a specific Hosts file is required, select it using the Hosts file button. For details about the Hosts file, see the description which follows.

DHCP Address Leases

DHCP provides a network administrator with the ability to configure a host's IP address either for an unlimited or for a limited period of time. The limited lease period is under the network administrator's control and is non-negotiable. Leases may, however, be renewed at the option of the configuring server.

Open Transport/TCP fully supports DHCP address leases. Should an interface's IP address lease expire, the interface will be closed down. However, Open Transport/TCP will automatically attempt to renew any address lease that reaches it's Renewal Interval, which defaults to half of the lease's lifetime, but may be configured to a different interval by the configuring server. Renewal will be attempted regardless of how many times the lease has already been renewed.

BOOTP Configuration

To use a BOOTP server to setup Open Transport/TCP, follow these steps:

(1) Select the interface to use, or pick "AppleTalk (MacIP)" to run over AppleTalk on the interface selected in the AppleTalk control panel.

(2) If an Ethernet interface is selected, a check box will appear offering the use of 802.3 packet encapsulation. By default, Open Transport/TCP uses Ethernet rather than 802.3.

(3) Select "Using BOOTP " as the configuration method.

(4) Fill in the search domain extensions to be used on name searches. For example, if a domain name of "apple.com" is configured, a search for "sam" would initially search for "sam.apple.com." It is not always necessary to fill in these fields when configuring from a BOOTP or DHCP server since a search domain may be returned along with the IP address.

In Advanced User mode only, you may configure an implicit domain search list . This is used to allow implicit searches. For example, if I wish to search all of the subdomains between "joesgroup.mktg.apple.com" and "apple.com", a search for the name "sam" would first look for "sam.joesgroup.mktg.apple.com", then for "sam.mktg.apple.com", then for "sam.apple.com." Implicit searching will not be done unless the Starting domain name is a subdomain of the Ending domain name (which corresponds to the domain of local administration per RFC 1535).

(5) *In Advanced User mode, a subnet mask may be entered but is not required. If a value is entered, it will be used if no subnet mask is returned from the BOOTP server. Otherwise, any value entered is ignored.*

(6) *In Advanced User mode, the manually entered IP addresses of routers are attached to the end of the (possibly empty) list of IP routers returned by the DHCP server.*

(7) *In Advanced User mode, the manually entered IP addresses of Domain Name Servers are attached to the end of the (possibly empty) list of Name Servers returned by the DHCP server.*

(8) If a specific Hosts file is required, select it using the Hosts file button. For details about the Hosts file, see the description which follows.

The TCP/IP application may be used at any time to reconfigure the system. However TCP will not notice the new configuration until it has unloaded from the system. In the current release, saving configuration parameters by closing the TCP/IP application forces TCP to unload immediately, breaking any existing connections. Otherwise, TCP will unload by default about 2 minutes after the last application using TCP , RawIP or UDP has gone away.

RARP Configuration

To use a RARP server to setup Open Transport/TCP, follow these steps:

(1) Select the interface to use, or pick "AppleTalk (MacIP)" to run over AppleTalk on the interface selected in the AppleTalk control panel.

(2) If an Ethernet interface is selected, a check box will appear offering the use of 802.3 packet encapsulation. By default, Open Transport/TCP uses Ethernet rather than 802.3.

(3) Select "Using RARP " as the configuration method.

(4) Fill in the search domain extensions to be used on name searches. For example, if a domain name of "apple.com" is configured, a search for "sam" would initially search for "sam.apple.com." It is not always necessary to fill in these fields when configuring from a BOOTP or DHCP server since a search domain may be returned along with the IP address.

In Advanced User mode only, you may configure an implicit domain search list . This is used to allow implicit searches. For example, if I wish to search all of the subdomains between "joesgroup.mktg.apple.com" and "apple.com", a search for the name "sam" would first look for "sam.joesgroup.mktg.apple.com", then for "sam.mktg.apple.com", then for "sam.apple.com." Implicit searching will not be done unless the Starting domain name is a subdomain of the Ending domain name (which corresponds to the domain of local administration per RFC 1535).

(5) Fill in the subnet mask in dot notation. For example, using a class B net which uses 6 bits of the host field used for subnetting, the subnet mask should be entered as "255.255.252.0".

(6) Fill in the IP address(es) of one or more IP routers.

(7) Fill in the IP address(es) of one or more Domain Name Servers.

(8) If a specific Hosts file is required, select it using the Hosts file button. For details about the Hosts file, see the description which follows.

The TCP/IP application may be used at any time to reconfigure the system. However TCP will not notice the new configuration until it has unloaded from the system. In the current release, saving configuration parameters by closing the TCP/IP application forces TCP to unload immediately, breaking any existing connections. Otherwise, TCP will unload by default about 2 minutes after the last application using TCP , RawIP or UDP has gone away.

MacIP Server Configuration

To use a MacIP Server to setup Open Transport/TCP, follow these steps:

(1) Select "AppleTalk (MacIP)" as the interface to use. TCP will now run over AppleTalk on the interface selected in the AppleTalk control panel.

(2) Select "Using MacIP Server" as the configuration method.

(3) Fill in the search domain extensions to be used on name searches. For example, if a domain name of "apple.com" is configured, a search for "sam" would initially search for "sam.apple.com." It is not always necessary to fill in these fields when configuring from a BOOTP or DHCP server since a search domain may be returned along with the IP address.

In Advanced User mode only, you may configure an implicit domain search list . This is used to allow implicit searches. For example, if I wish to search all of the subdomains between "joesgroup.mktg.apple.com" and "apple.com", a search for the name "sam" would first look for "sam.joesgroup.mktg.apple.com", then for "sam.mktg.apple.com", then for "sam.apple.com." Implicit searching will not be done unless the Starting domain name is a subdomain of the Ending domain name (which corresponds to the domain of local administration per RFC 1535).

(4) If a specific Hosts file is required, select it using the Hosts file button. For details about the Hosts file, see the description which follows.

Hosts File

Open Transport/TCP supports a Hosts file that may be used to supplement and/or customize the Domain Name Resolver's initial cache of information. The Hosts file is found in the System's Preferences folder. In order to support MacTCP compatibility with some applications, Open Transport will also look in the System folder if no Hosts file is found in the Preferences folder. This file is parsed when Open Transport/TCP is initialized. As in MacTCP, the supported Hosts file features follow a subset of the Domain Name System Master File Format (see RFC 1035 page 33).

Supported features include blank lines, comments (indicated by a semicolon), and data entry. Comments may begin at any location in a line; they may follow data entry on the same line. A comment extends from the semicolon to the end of the line. Data entry must follow the format:

<domain-name> <rr> [<comment>]

where <domain-name> is an absolute or Fully Qualified domain name (which, however, need not be terminated by a dot, but must contain at least one dot internally) and where

<rr> = [<ttl>] [<class>] <type> <rdata> OR [<class>] [<ttl>] <type> <rdata>

The only class currently supported is IN (Internet Domain); ttl (time to live; indicates the record's configured lifetime) is in seconds; and type can be A (host address), CNAME (canonical name of an alias), or NS (name server). If ttl is not present the entry is assumed to have an infinite lifetime; this may also be indicated by specifying a ttl of minus-one (-1).

$INCLUDE and $ORIGIN are not supported.

Examples of valid data entry lines including comments:

```
apple.com    A   130.43.2.2   ; address of host apple.com
foobar   CNAME   barfoo.apple.com   ; canonical name for the host whose local alias is "foobar"
mit.edu.   86400  NS   achilles.mit.edu  ; name server for mit.edu domain, entry has a one-week lifetime
```

Open Transport/TCP's Hosts file support is somewhat more stringent than that of MacTCP. MacTCP permitted violation of the Fully Qualified requirement for <domain-name>, and this feature was often used to avoid the necessity for entering CNAME records by associating an address directly with a non-fully qualified name. For instance, this format:

```
charlie   A   128.1.1.1
```

which was acceptable to the MacTCP DNR, is no longer permitted because of the use of domain search lists in Open Transport/TCP (charlie could potentially exist in any or all of the configured domains). If such a line exists in your hosts file, kOTBadNameError will be returned when the Hosts file is read. To accomplish the same effect, use this format instead:

```
charlie   CNAME   myhost.mydomain.edu
myhost.mydomain.edu   A   128.1.1.1
```

This associates the local alias "charlie" with the fully qualified domain name "myhost.mydomain.edu" and resolves it to the address 128.1.1.1. Use of local aliases is limited to CNAME entries; NS and A entries must use fully qualified domain names.

In general, use of the Hosts file is discouraged, as it often simply wastes memory by permanently configuring data that may only rarely be accessed. It is also highly susceptible to misuse by users who try to configure far too much information internally in order to avoid accessing DNS servers. Besides tying up memory, this practice is exactly the reason that the Domain Name System was developed in the first place - to eliminate the performance degradation caused by use of enormous hosts files.

Should a Hosts file be used, every effort should be made to keep it as small as possible and to only include entries that will be accessed frequently.

Data Structures

This section details the data structures which are unique to Open Transport TCP/IP. These structures are found in the include file OpenTptInternet.h. Typedefs commonly used are also mentioned, although their use is not necessarily unique to Open Transport TCP/IP.

Basic Typedefs

```
enum
{
      kMaxHostNameLen = 255
};
```

Type	Is usually a...
UInt8	unsigned char
UInt16	unsigned short
UInt32	unsigned long
SInt32	signed long
OTAddressType	UInt16
InetPort	UInt16
InetHost	UInt32
InetDomainName	char[kMaxHostNameLen]

Note: In all cases, the maximum valid domain name length for fully qualified domain names, kMaxHostNameLen, includes the trailing period (".") . Names not terminated with a period are limited to 254 characters.

InetAddress

An InetAddress is a "fully qualified" address. It contains information about the address type (currently AF_INET is the only supported address type), the port number, and the Host's IP address. This structure is laid out identically to the common sockaddr_in structure used in BSD UNIX.

```
struct InetAddress
{
      OTAddressType     fAddressType; /* AF_INET */
      InetPort          fPort;
      InetHost          fHost;        /* the IP address */
      UInt8             fUnused[8];   /* traditional */
};
typedef struct InetAddress InetAddress;
```

DNSAddress

A DNSAddress is a "fully qualified" address, but it is not fully resolved. It contains a name and a port number in the format "xxx[:yy]" where "xxx" is a hostname, a partially qualified domain name, a fully qualified domain name, or an Internet address in standard dot format, and "[:yy]" is and optional port number. For example: "foo.apple.com", "foo.apple.com:25", "17.202.99.99:25". Applications may use the DNSAddress directly when calling **Connect()** for a TCP endpoint, or when calling **SndUData**() for a UDP endpoint, or when calling **ResolveAddress()** with either TCP or UDP endpoints, and the name

will be automatically resolved by the DNR. Users may find this more convenient than invoking the DNR directly when they only have name-to-address resolutions to perform.

```
struct DNSAddress
{
      OTAddressType     fAddressType;      /* AF_DNS */
      InetDomainName    fName;
};
typedef struct DNSAddress DNSAddress;
```

Because domain names are limited to 255 characters, the DNSAddress type can take a string of 255 bytes or less. A string in the form "apple.com:25" where the optional ":port number" is used is not a proper domain name, but it will be accepted for convenience. However, if the combination of the domain name and the optional port number exceeds 255 bytes, the client must provide a structure using DNSAddress as a template which has sufficient space to contain the full string. The domain name itself still cannot exceed 255 bytes in any case.

DNSQueryInfo

DNSQueryInfo is used by the Domain Name Resolver to return answers to DNS queries made using the OTInetQuery Interface. Each answer returned for a particular name, query class, and query type is returned in the following structure. The resourceData array will be at least 4 bytes long, but it is usually longer. Its size is indicated by the returned resourceLen.

```
struct DNSQueryInfo
{
      UInt16            qType;
      UInt16            qClass;
      UInt32            ttl;
      InetDomainName    name;
      UInt16            responseType;
      UInt16            resourceLen;
      char              resourceData[4];
};
typedef struct DNSQueryInfo DNSQueryInfo;
```

InetHostInfo

InetHostInfo is used by the Domain Name Resolver to return basic information about a host. When filled in by the DNR in response to a StringToAddress call, it contains the fully qualified name of the host and up to ten IP addresses for that host. If only one IP address is found for a given hostname, that IP address is returned in the first slot in the addrs array, and all other addresses are set to 0. For more information, see the function description for the StringToAddress call.

```
enum
{
      kMaxHostAddrs = 10,
      kMaxHostNameLen = 255
};
struct InetHostInfo
{
      InetDomainName    name;
      InetHost          addrs[kMaxHostAddrs];
};
typedef InetHostInfo InetHostInfo;
```


InetSysInfo

InetSysInfo is used by the DNR to return detailed system information about a host's CPU type and operating system in response to a SysInfo call. See the latest "Assigned Numbers" RFC for a list of the possible CPU and OS strings. CPU and OS strings are limited to 40 characters, and will be returned as a 0 terminated C String. For more information, see the function description for the SysInfo call.

```
enum { kMaxSysStringLen = 40 };

struct InetSysInfo
{
      char  cpuType[kMaxSysStringLen+1];
      char  osType[kMaxSysStringLen+1];
};
typedef InetSysInfo InetSysInfo;
```

InetMailExchange

InetMailExchange is used by the DNR to return mail exchange and preference information in response to a MailExchange call. For more information, see the function description for the MailExchange call.

```
struct InetMailExchange
{
      UInt16            preference;
      InetDomainName    exchange;
};
typedef InetMailExchange InetMailExchange;
```

InetInterfaceInfo

InetInterfaceInfo is used by InternetServices to return Internet address information about the local host. For more information, see the function description for the OTInetGetInterfaceInfo call.

```
struct InetInterfaceInfo
{
      InetHost          fAddress;
      InetHost          fNetmask;
      InetHost          fBroadcastAddr;
      InetHost          fDefaultGatewayAddr;
      InetHost          fDNSAddr;
      UInt16            fVersion;
      UInt16            fPad;
      InetHost          fReserved[4];
      InetDomainName    fDomainName;
};
typedef InetInterfaceInfo InetInterfaceInfo;
```

TIPAddMulticast

TIPAddMulticast is the OTOptionManagement option value to join an IP Multicast address group. For more information , see the description of OTOptionManagement in the *Open Transport Client Developer's Note.*

```
struct TIPAddMulticast
{
      InetHost    multicastGroupAddress;
      InetHost    interfaceAddress;
};
typedef   TIPAddMulticast TIPAddMulticast;
```


Function Descriptions

All data structures are prefixed with Inet. Data structures are described in the previous section.

OTInitInetAddress

FUNCTION

OTInitInetAddress Initialize an InetAddress.

C INTERFACE

void OTInitInetAddress(InetAddress *inAddr, InetPort port, InetHost host);

C++ INTERFACE

None (C++ clients should use the C interface to this function).

DESCRIPTION

Parameters	Before Call	After Call
port	x	/
host	x	/
inAddr	x	x
inAddr->fAddressType	/	x
inAddr->fPort	/	x
inAddr->fHost	/	x
inAddr->fUnused	/	/

OTInitInetAddress simplifies setup of an InetAddress. Clients are not required to use this function, but may find it convenient to do so. This call cannot block and it always completes synchronously.

RESULT CODES

None

SEE ALSO

OTInitDNSAddress, OTInetStringToHost, OTInetHostToString

OTInitDNSAddress

FUNCTION

OTInitDNSAddress Initialize a DNSAddress.

C INTERFACE

void `OTInitDNSAddress`(DNSAddress *dnsAddr, char* dnsAddressString);

C++ INTERFACE

None (C++ clients should use the C interface to this function).

DESCRIPTION

Parameters	Before Call	After Call
dnsAddressString	x	/
dnsAddr	x	x
dnsAddr->fAddressType	/	x
inAddr->fName	/	x

OTInitDNSAddress simplifies setup of a DNSAddress. Clients are not required to use this function, but may find it convenient to do so. This call cannot block and it always completes synchronously.

RESULT CODES

None

SEE ALSO

OTInitInetAddress, OTInetStringToHost, OTInetHostToString

OTInetStringToHost

FUNCTION

OTInetStringToHost Converts an IP address string from either dot notation or hex notation to an InetHost.

C INTERFACE

OSErr OTInetStringToHost(char* addrString, InetHost* host);

C++ INTERFACE

None (C++ clients should use the C interface to this function).

DESCRIPTION

Parameters	Before Call	After Call
addrString	(x)	/
host	/	(x)

OTInetStringToHost converts a string denoting an IP address into an InetHost. The string must either represent an address in Internet dot notation (e.g. "12.13.14.15") or hex notation (e.g. "0x0c0d0e0f").

Clients are not required to use this function, but may find it convenient to do so. This call cannot block and it always completes synchronously. It does not invoke the Domain Name Resolver, and it should not be confused with the StringToAddress call which translates a name string to an IP address.

RESULT CODES

kNoError

kOTBadAddressErr

SEE ALSO

OTInitInetAddress, OTInetHostToString

OTInetHostToString

FUNCTION

OTInetHostToString — Converts an InetHost into a string representing that IP Address in Internet Dot notation.

C INTERFACE

```
void OTInetHostToString(InetHost host, char* stringBuf);
```

C++ INTERFACE

None (C++ clients should use the C interface to this function).

DESCRIPTION

Parameters	Before Call	After Call
host	x	/
stringBuf	x	(x)

OTInetHostToString converts an InetHost into a C string containing a dot format representation of the IP address. For example, the InetHost "0x10111213" would be converted to the string "16.17.18.19".

Clients are not required to use this function, but may find it convenient to do so. This call cannot block and it always completes synchronously. It does not invoke the Domain Name Resolver, and it should not be confused with the AddressToName call which translates an IP address into a Host name.

RESULT CODES

None

SEE ALSO

OTInitInetAddress, OTInetStringToHost

GetInterfaceInfo

FUNCTION

GetInterfaceInfo — Returns Internet address information about the local host.

C INTERFACE

```
OSErr OTInetGetInterfaceInfo(InetInterfaceInfo* info, SInt32 index);
```

C++ INTERFACE

None. C++ clients should use the C Interface.

DESCRIPTION

Parameters	Before Call	After Call
info	x	/
index	x	(x)

Takes an index into the host's array list of configured IP interfaces and, if possible, fills in the InetInterfaceInfo structure with the interface's Internet address, subnet mask, and broadcast address. The IP addresses of a default gateway and a Domain Name Server will also be returned if any are known, as will the host's Domain Name. An index of 0 (zero) returns information about the first interface. The broadcast address may not always be available, but may easily be determined from the Internet address and the subnet mask.

This call cannot block and it always completes synchronously.

Note that if Open Transport/TCP has not yet loaded, which is the case when no TCP application is running and the "TCP always loaded" option is not picked within the control panel, that no interfaces will be valid. By default, interfaces are not initialized until needed.

Also note that an application which binds to an IP address of 0 and subsequently does a getprotaddr call will get back an IP address of 0. The only mechanism for determining the IP address of running IP interfaces is through the GetInterfaceInfo call.

RESULT CODES

kOTNoError	Call completed successfully.
kOTNotFoundErr	The requested interface does not exist.
kENOMEMErr	Open Transport TCP memory depletion.

SEE ALSO

OTOpenInternetServices

FUNCTION

OTOpenInternetServices — Allows client to make use of provided Internet services, such as the Domain Name Resolver.

C INTERFACE

```
InetSvcRef  OTOpenInternetServices(OTConfiguration* otc, OTOpenFlags flags, OSErr* err);
OSErr       AsyncOTOpenInternetServices(OTConfiguration* otc, OTOpenFlags flags,
            OTNotifyProcPtr proc, void* contextPtr);
```

C++ INTERFACE

None. (C++ clients should use the C interface to this function).

DESCRIPTION

Opens the Internet Services library and returns an InetSvcRef which can be subsequently used to access other routines (currently just the DNR routines). The flags parameter is currently ignored and should be 0, and a default (DNR) OTConfiguration* will be built and used if otc is set to `kDefaultInternetServicesPath` (defined in OpenTptInternet.h). When the application completes or finishes using Internet services, calling CloseProvider frees the associated resources.

If the library is opened using the OTOpenInternetServices call, it will work by default in synchronous mode, blocking on calls until completion. If the library is opened using the AsyncOTOpenInternetServices call, it will work by default in asynchronous mode, returning immediately. In this case, if no error is returned immediately, the result of the call will be returned to the client's notifier.

For more information on the DNR, see the RFCs 1034 and 1035.

Alternately, clients may choose to use the transport independent Mapper interface to access some of the DNR's facilities. The Mapper may be used to translate a host name to an IP address, but it has no equivalent for the AddressToName, SysInfo, or MailExchange calls. For details on the Mapper, see the *Open Transport Client Developer Note*, and the function description of the LookupName call which follows in this document. Note that OTLookupName is the only Open Transport Mapper Library call supported by the DNR. Calls to OTRegisterName and OTDeleteName will return kOTNotSupportedErr.

RESULT CODES

None. Returns the InetSvcRef.

SEE ALSO

CloseProvider (*Open Transport Client Developer Note*)

Query

FUNCTION

Query — Returns information which the DNR can obtain based upon DNS queries made using the given query class and type.

C INTERFACE

```
OSErr  OTInetQuery( InetSvcRef ref, char* name, UInt16 qClass, UInt16 qType,
                    char* buf, size_t buflen, void** argv, size_t argvlen,
                    OTFlags flags);
```

C++ INTERFACE

```
OSErr  TInternetServices::Query( char* name, UInt16 qClass, UInt16 qType,
                    char* buf, size_t buflen, void** argv, size_t argvlen,
                    OTFlags flags);
```

DESCRIPTION

Parameters	Before Call	After Call
ref (C only)	x	/
name	x	/
qClass	x	/
qType	x	/
buf	x	(x)
buflen	x	/
argv	?	(?)
argvlen	?	/
flags	x	/

OTInetQuery is the most general interface to the Domain Name Resolver (DNR). Using OTInetQuery, it is possible to ask the DNR to query Domain Name Servers for information assoicated with any class and type.

Note that for several basic query types, using the simplied interfaces which follow (e.g. StringToAddress) may be easier to program. The information obtained will be the same using either call, although in some cases the simplified interfaces limit the maximum number of answers which can be returned.

OTInetQuery takes a character string (name) and two numeric identifiers (qClass and qType) and returns any information associated with that name for the give class and type which can be obtained from the configured name servers.

The flags field should be set to 0 so future enhancements to this interface will not break existing applications.

The caller passes in a buffer (buf) and its length (buflen) where one or more answers can be returned.

Argv and argvlen are optional parameters. If used, argv is the address of an empty pointer array of length argvlen. If provided, OpenTransport uses this buffer to return pointers to the location of individual answers written into buf. For example, if a query receives 3 answers, and the caller passed in an arvlen of 5, argv[0] would be a pointer to the first answer in buf, argv[1] a pointer to the second answer, argv[2] a pointer to the 3rd answer, and argv[3] would be null. Whether argv and argvlen is used or not, the answers returned in buf remain the same.

If OTInetQuery is called synchronously, it will not return until the call completes.

If OTInetQuery called asynchronously, it will return immediately; the client's completion notifier will be called with a T_DNRQUERYCOMPLETE event when the call completes. Asynchronous mode is preferred. When using asynchronous mode, the client must not touch the buf or argv structures prior to completion of the routine.

OTInetQuery works with both known and unknown query classes and types. Using the Inet query class and known query types, compressed answers are expanded prior to filling in the return buffer. Answers which are resource records of unknown class and type are copied into the return buffer unparsed. No expansion is done because it is assumed that DNS compression is not used.

Currently, only answers of type PTR and CNAME (name-to-address translations) are cached by OpenTransport. Also, OpenTransport does not currently use this cached information to resolve address-to-name translations (because this defeats some existing server load balancing schemes in operation today).

RESULT CODES

kOTNoError	Call completed successfully.
kOTNoDataErr	No data available - either timeout, or name exists but requested information does not.
kOTBadNameErr	Bad name - either name does not exist in domains examined, or bad syntax.
kENOMEMErr	Open Transport TCP memory depletion.

SEE ALSO

StringToAddress, AddressToName, SysInfo, MailExchange

StringToAddress

FUNCTION

StringToAddress — Returns information which the DNR can determine based on a string which normally contains a host name.

C INTERFACE

```
OSErr OTInetStringToAddress( InetSvcRef ref, char* string, InetHostInfo *hinfo);
```

C++ INTERFACE

```
OSErr TInternetServices::StringToAddress( char* string, InetHostInfo *hinfo);
```

DESCRIPTION

Parameters	Before Call	After Call
ref (C only)	x	/
string	x	/
hinfo	x	(x)

Uses the Domain Name Resolver (DNR) to take a character string which contains a host name, a partially qualified domain name, a fully qualified domain name, or an Internet address in dot format and, if possible, fill in the InetHostInfo structure with the host's canonical name and up to ten associated IP addresses. If less than 10 addresses are found for a given host, additional slots in the hinfo address array are filled with 0s. The caller must allocate the InetHostInfo structure prior to the call.

If OTInetStringToAddress is called synchronously, it will not return until the call completes.

If OTInetStringToAddressis called asynchronously, it will return immediately; the client's completion notifier will be called with a T_DNRSTRINGTOADDRCOMPLETE event when the call completes. Asynchronous mode is preferred. When using asynchronous mode, the client should not touch the InetHostInfo structure prior to completion of the routine. The fourth parameter passed to the client's completion notifier is a pointer to the InetHostInfo structure that was resolved; this enables client software to determine which of multiple simultaneous outstanding requests has been completed.

StringToAddress will not resolve an input character string in the form of a dot-notation Internet address to its domain name; it will simply translate the address into InetHost format. Do not use StringToAddress to resolve an address to its domain name; use instead AddressToName, described below.

RESULT CODES

kOTNoError	Call completed successfully.
kOTNoDataErr	No data available - either timeout, or name exists but requested information does not.
kOTBadNameErr	Bad name - either name does not exist in domains examined, or bad syntax.
kENOMEMErr	Open Transport TCP memory depletion.

SEE ALSO

Query, AddressToName, SysInfo, MailExchange

AddressToName

FUNCTION

AddressToName — Tries to determine information about the host associated with an IP address.

C INTERFACE

```
OSErr OTInetAddressToName( InetSvcRef ref, InetHost host, InetDomainName name);
```

C++ INTERFACE

```
OSErr TInternetServices::AddressToName( InetHost host, InetDomainName name);
```

DESCRIPTION

Parameters	Before Call	After Call
ref (C only)	x	/
host	x	/
name	x	(x)

Uses the DNR to try to determine the canonical domain name of the host associated with an IP address. The caller must allocate the InetHostInfo structure prior to the call.

If OTInetAddressToName is called synchronously, it will not return until the call completes.

If OTInetAddressToName is called asynchronously, it will return immediately; the client's completion notifier will be called with a T_DNRADDRTONAMECOMPLETE event when the call completes. Asynchronous mode is preferred. When using asynchronous mode, the client should not touch the InetHostInfo structure prior to completion of the routine. The fourth parameter passed to the client's completion notifier is a pointer to the InetHostInfo structure that was resolved; this enables client software to determine which of multiple simultaneous outstanding requests has been completed.

OTInetAddressToName does not currently support resolving addresses that are not subnetted to octet boundaries. Use OTInetQuery instead.

Do not use AddressToName to resolve a host name or domain name to its Internet address; use instead StringToAddress, described above.

Note, currently OpenTransport neither caches AddressToName translations nor resolves AddressToName queries from cached answers from previous StringToAddress translations. This is because doing so defeats some existing server load balancing schemes in operation today. In the future, OpenTransport will cache AddressToName translations, but for now applications

which depend on this feature (e.g. Web page servers) should maintain their own cache for best performance.

RESULT CODES

kOTNoError	Call completed successfully.
kOTNoDataErr	No data available - timeout.
kOTBadNameErr	Bad address - either address does not exist in domains examined, or bad syntax.
kEINVALErr	An address of invalid size was passed in.
kENOMEMErr	Open Transport TCP memory depletion.

SEE ALSO

Query, StringToAddress, SysInfo, MailExchange

SysInfo

FUNCTION

SysInfo	Returns details about the system whose name is being queried.

C INTERFACE

OSErr OTInetSysInfo(InetSvcRef ref, char* queryName, InetSysInfo* sysinfo);

C++ INTERFACE

OSErr TInternetServices:SysInfo(char* queryName, InetSysInfo* sysinfo);

DESCRIPTION

Parameters	Before Call	After Call
ref (C only)	x	/
queryName	(x)	/
sysinfo	x	(x)

Uses the DNR to attempt to find detailed information about a host's CPU and operating system. The caller passes in a name and a pointer to an InetSysInfo structure it has allocated.

If OTInetSysInfo is called synchronously, it will not return until the call completes.

If OTInetSysInfo is called asynchronously, it will return immediately; the client's completion notifier will be called with a T_DNRSYSINFOCOMPLETE event when the call completes. Asynchronous mode is preferred. When using asynchronous mode, the client should not touch the InetSysInfo structure prior to completion of the routine. The fourth parameter passed to the client's completion notifier is a pointer to the InetSysInfo structure that was resolved; this enables client software to determine which of multiple simultaneous outstanding requests has been completed.

RESULT CODES

kOTNoError	Call completed successfully.
kOTNoDataErr	No data available - either timeout, or name exists but requested information does not.
kOTBadNameErr	Bad name - either name does not exist in domains examined, or bad syntax.
kENOMEMErr	Open Transport TCP memory depletion.

SEE ALSO

Query, AddressToName, SysInfo, MailExchange

MailExchange

FUNCTION

MailExchange — Returns mail exchange and preference information about the system being queried.

C INTERFACE

OSErr OTInetMailExchange(InetSvcRef ref, char* queryName, UInt16* count, InetMailExchange* mxinfo);

C++ INTERFACE

OSErr TInternetServices::MailExchange(char* queryName, UInt16* count, InetMailExchange* mxinfo);

DESCRIPTION

Parameters	Before Call	After Call
ref (C only)	x	/
queryName	(x)	/
mxinfo	x	(x)
count	x	(x)

Uses the DNR to attempt to obtain mail exchange and preference information associated with a name. The client allocates a contiguous array of one or more InetMailExchange structures. A pointer to the number of elements in this array and a pointer to the first element are passed in. OTInetMailExchange will fill in as many of the InetMailExchange structures as it can, and will set count to the number of structures it has filled in.

If OTInetMailExchange is called synchronously, it will not return until the call completes.

If OTInetMailExchange is called asynchronously, it will return immediately; the client's completion notifier will be called with a T_DNRMAILEXCHANGECOMPLETE event when the call completes. Asynchronous mode is preferred. When using asynchronous mode, the client should not touch the InetMailExchange array prior to completion of the routine. The fourth parameter passed to the client's completion notifier is a pointer to the first element of the InetMailExchange array; this enables client software to determine which of multiple simultaneous outstanding requests has been completed.

RESULT CODES

kOTNoError	Call completed successfully.
kOTNoDataErr	No data available - either timeout, or name exists but requested information does not.
kOTBadNameErr	Bad name - either name does not exist in domains examined, or bad syntax.
kENOMEMErr	MacTCP memory depletion.

SEE ALSO

Query, StringToAddress, AddressToName, InetMailExchange

LookupName

FUNCTION

LookupName — Provides an Open Transport Mapper Library interface to the DNR's string-to-address resolution function.

C INTERFACE

OSErr OTLookupName(MapperRef ref, size_t maxcnt, TLookupRequest* request, TLookupReply* reply);

C++ INTERFACE

OSErr TMapper::LookupName(size_t maxcnt, TLookupRequest* request, TLookupReply* reply);

DESCRIPTION

Parameters	Before Call	After Call
ref (C only)	x	/
maxcnt	x	/
request->udata.maxlen	/	/
request->udata.len	x	/
request->udata.buf	(x)	/
reply->udata.maxlen	x	/
reply->udata.len	/	x
reply->udata.buf	(?)	(x)
reply->rspcount	/	x

OTLookupName is used to map a name to an InetAddress. The name is a character string whose format is xxx[:yy] where xxx is a hostname, a partially qualified domain name, a fully qualified domain name, or an Internet address in standard Internet dot format, and the optional yy is a TCP or UDP port number. OTLookupName returns an InetAddress, described earlier in this document, in reply->udata.buf. This InetAddress may be used directly in all appropriate Open Transport TCP calls requiring an InetAddress, such as **Connect()** and **SndUData()**. Should no port be entered, the returned InetAddress will contain a port number of 0 (zero). Only a single InetAddress is returned, regardless of how many addresses are known for a given name (multihomed hosts).

If the name passed to OTLookupName is in the form of a dot format Internet address, the InetAddress returned by OTLookupName will simply contain the hexadecimal values of that address plus the input port number.

While the format of a TCP/IP "name" may appear similar to that used by AppleTalk, there are distinct differences because the concepts do not map congruently from one to the other. The AppleTalk format is xxx:yy@zzz, where xxx is an object, yy is its type, and zzz is the AppleTalk zone where it's located. The Domain Name Service has no concept of "type" or

"zone", and the "objects" it deals with are primarily individual computers; hence, the object to be resolved is the name of a computer. Substituting port number for "type" results in a name/address format that conforms with standard Internet addressing practice: port 25 on foo.apple.com is represented as foo.apple.com:25, or as 17.202.99.99:25. Using this format and construction retains the AppleTalk Mapper look and feel while remaining true to Internet addressing formats.

If OTLookupName is called synchronously, it will not return until the call completes. If it is called asynchronously, it will return immediately and the client's notifier will be called when the call completes. Asynchronous mode is preferred.

Please refer to the *Open Transport Client Developer Note* for specific details on how to open a DNR Mapper endpoint and how to use it.

OTLookupName is the only Open Transport Mapper Library call supported by the DNR. Calls to OTRegisterName and OTDeleteName will return kOTNotSupportedErr.

RESULT CODES

kOTNoError	Call completed successfully.
kOTNoDataErr	No data available - either timeout, or name exists but requested information does not.
kOTBadNameErr	Bad name - either name does not exist in domains examined, or bad syntax.
kENOMEMErr	MacTCP memory depletion.
kOTNotSupportedErr	Unsupported call.

SEE ALSO

OpenMapper, CloseProvider (*Open Transport Client Developer Note*)

Appendix A: Preferences format

WARNING:

The preference format is provided for debugging purposes only.
This format is not part of the supported features of Open Transport.
It will change in a future release of Open Transport.

Should you decide to rely on any information in this appendix, be prepared to have to update your software when this format changes.

Configuration file format

The preferences file contains one or more configurations. A configuration is a set of resources that share the same resource id (≥ 128).

The preferences file also contains some global information. Resource ids < 128 are used to keep these global settings. For instance, the resource 'ccfg' #1 indicates currently active configuration.

Global information resource types

'ccfg' (id 1) indicates the currently active configuration

```
type 'ccfg'    // always id 1
{
   integer;    // Resource id of the current config (≥128)
};
```

'wpos' (id 128) postion of the main window

```
type 'wpos'    // always id 128
{
   integer;    // window's top left corner
   integer;    // global coordinates
};
```

Configuration resources types

All resources using the same resource id (≥ 128) are part of the same configuration.

'**cnam**' the name of this resource indicates the name of the configuration

```
type 'cnam'
{
      // empty resource
      // the name of the resource is used as
      // configuration name
};
```

'**cvrs**' format version of this configuration. Alaways 1 for OT 1.0/1.1

```
type 'cvrs'
{
   integer;    // always 1 for ot 1.0 and 1.1
};
```

'**dtyp**' OT device type of the selected port

```
type 'dtyp'
{
   integer;    // OT device type for the selected port
};
```

'**pwrd**' password that protects access to the Administration mode for this configuration

```
type 'pwrd'
{
   pstring;    // password - "encrypted"
               // empty string means not protected
};
```

'**port**' user readable name of the currently selected port

```
type 'port'
{
   pstring;
};
```

'**prot**' OT name of the protocol used by this configuration

```
type 'prot'
{
```



```
   string;        // "tcp" for TCP/IP, "ddp" for AppleTalk
};
```

'**ihst**' index of primary interface (always 1 in OT 1.0/1.1) and implicit domains search list definition.

```
type 'ihst'
{
   byte;       // primary interface index
               // (always 1 for OT 1.0 / 1.1)
   pstring;    // implicit search list starting point
               // (local domain name),
   pstring;    // implicit search list ending point
               // (admin domain name)
};
```

'**irte**' routing table (not used in OT 1.0 / 1.1)

```
type 'irte'
{
   integer = $$CountOf(routeList);  // number of routes
   wide array routeList
   {
      hex string[4]; // sub net mask
      hex string[4]; // address of router to use
      boolean;       // route local?
      align word;
      boolean;       // route host?
      align word;
   };
};
```

'**iitf**' list of interface configurations (only one entry for OT 1.0)

```
type 'iitf'
{
   integer = $$CountOf(interfacesList);   // number of interfaces
                                          // (1 for OT 1.0/1.1)
   wide array interfacesList
   {
      byte;          // interface active flag
      hex string[4]; // IP address
      hex string[4]; // sub net mask
      pstring;       // MacIP server zone name
                     // (if DDP is used as port)
      pstring[35];   // interface OT style name
                     // (e.g. ENET0 for built-in Ethernet)
      pstring[31];   // module name (e.g. ENETdrvr for Ethernet)
      hex string[4]; // Framinig flags for the port (use 802.3)
   };
};
```

'**idns**' list of name server addresses

```
type 'idns'
{
   integer = $$CountOf(nameServers);   // # of search domain names
   wide array searchDomains
   {
      hex string[4]; // name server address
   };
};
```

'**isdm**' list of search domain names. These domains are searched after those in the implicit search list (if any)

```
type 'isdm'
{
   integer = $$CountOf(searchDomains); // # of search domain names
   wide array searchDomains
   {
      pstring; // domain name
   };
};
```

'**stng**' indicates in the corresponding item has been locked (true = locked, false = unlocked) in the administration mode

```
type 'stng'
{
   fill word;  // usused
   boolean;    // connect via lock
   align word;
   boolean;    // AppleTalk (MacIP) configuration menu lock
   align word;
   boolean;    // AppleTalk (MacIP) zone lock
   align word;
   boolean;    // IP address lock
   align word;
   boolean;    // domain name lock
   align word;
   boolean;    // subnet mask lock
   align word;
   boolean;    // routers list lock
   align word;
   boolean;    // name servers list lock
   align word;
   boolean;    // admin domain lock
   align word;
   boolean;    // search domains list lock
   align word;
   fill word;  // usused
```



```
   boolean;    // use 802.3 lock
};
```

'**ulvl**' indicates the current user level

```
type 'ulvl'
{
   integer  kBasicMode = 1,
            kAdvancedMode,
            kAdministratioMode;
};
```

'**unld**' indicates if the stack is active or inactive and it it is unloaded when not used

```
type 'unld'
{
   integer  kActiveLoadedOnDemand = 1,
            kActiveAlwaysLoaded,
            kInactive;
};
```

Appendix B: MDEVs table format

WARNING:

The MDEV table format is provided for debugging purposes only.
This format is not part of the supported features of Open Transport.
It will change in a future release of Open Transport.

Should you decide to rely on any information in this appendix, be prepared to have to update your software when this format changes.

MDEV table format

MDEVs that are recognized by OpenTransport are described in the resource 'mdvm' id #128 located inside the "Open Tpt Internet Library" file in the "Extensions" folder.

Another copy of this table is present in the Open Transport installer script to allow the installer to preserve the selected MDEV when converting the MacTCP preferences.

This resource is a table that contains one entry per MDEV.

Each entry contains:

- the signature of the MDEV file
- the name to display in the "Connect via" popup menu
- its type (PPP, SLIP or other) used to display the appropriate "Configure" popup menu
- a unique id used internally to identify the MDEV (this should be index of the entry in the table)
- flags that indicate what the MDEV expects from MacTCP.

OpenTransport may not be able to support MDEVs that have requirements beyond those supported through the flags.

To have your MDEV recognized by Open Transport append an entry to the table and make sure the id you use is not used by any other MDEV in the table and that the flags are correctly set.


```
type 'mdvm'
{
   integer = $$CountOf(MDEVTableItem);
   array MDEVTableItem
   {
      cstring[64];      // Name to display in the "Connect Via" popup.
                        // Usually the name that was used in MacTCP

      int;              // *UNIQUE* MDEV id.
                        // Use the index of the entry in the 'mdvm' table

      literal longint;  // MDEV file creator

      integer           // MDEV type. Used to display the configuration
                        // methods in the "Configure" popup
                        kTypeOther,          // MDEV is not SLIP nor PPP.
                        kTypeSLIP,           // MDEV is SLIP
                        kTypePPP;            // MDEV is PPP.

      Boolean           // Controls loading of the MDEV
                        noLapInitAtSysBoot,  // MDEV doesn't need to be loaded
                                             // at boot time
                        needLapInitAtSysBoot;// MDEV needs to be loaded at
                                             // boot time

      Boolean;          // reserved
      Boolean;          // reserved
      Boolean;          // reserved
      align word;
   };
};
```