Alcatel·Lucent

VitalQIP® DNS/DHCP & IP Management Software

API Toolkit | Release 7.3PR2

User's Guide

190-409-033R7.3PR2
Issue 1
November 2012

Contents

About this document

Purpose

Welcome to the API Toolkit, a valuable add-on to the base VitalQIP® product. The API Toolkit is a developer's toolkit comprised of the VitalQIP Application Program Interface (API) routines and the Lucent Dynamic Host Configuration Protocol (DHCP) server API routines. These API routines allow for the extension and integration of VitalQIP into other critical applications such as asset tracing, other network management applications, and billing applications. The API Toolkit is purchased separately and must be configured in the VitalQIP license key.

Refer to this preface for the audience, organization, and typographical conventions used in this manual. The preface also describes the package contents, how to order additional manuals, and how to obtain technical support.

Reason for reissue

The VitalQIP API Toolkit was certified for use with VitalQIP 7.3PR2, Lucent DHCP 5.6, and Lucent DHCP 6.0 servers.

Intended audience

This manual is intended for API Toolkit users who plan to manage and administer an IP network address infrastructure. The reader is expected to understand basic networking concepts and have a working knowledge of the operating system that the API Toolkit is running on.

Two types of groups interact with the API Toolkit:

The API Toolkit administrators—The Information Technology (IT) professionals who install, configure, and administer the API Toolkit product.

- The API Toolkit users—The IT professionals who use the API Toolkit.

How to use this document

This manual is organized as follows:

Chapter 1 – VitalQIP API

This chapter provides an overview of the VitalQIP API, information about the relationship between API routines and VitalQIP commands, and a description of the VitalQIP API functions.

Chapter 2 – Lucent DHCP API

This chapter provides an overview of the **Lucent DHCP** API, including information about integration points, shared libraries and DLLs, and routine policies and calls. It also describes the **Lucent DHCP** API functions.

Conventions used

This guide uses the following typographical conventions:

Appearance	Description
italicized text	• File and directory names • Emphasized information • Titles of publications • A value that the user supplies
graphical user interface text or key name	• Text that is displayed in a graphical user interface or in a hardware label • The name of a key on the keyboard
`input text`	Command names and text that the user types or selects as input to a system
`output text`	Text that a system displays or prints

Related information

The following documents describe the VitalQIP product:

- *VitalQIP Administrator Reference Manual* (part number: 190-409-042R7.3PR2)

  This guide describes planning and configuring your network, information about the VitalQIP interface, advanced DNS and DHCP configurations, and troubleshooting.
- *VitalQIP Installation Guide* (part number: 190-409-043R7.3PR2)

  This guide describes how to install the VitalQIP product.
- *VitalQIP Command Line Interface User's Guide* (part number: 190-409-044R7.3PR2)

  This guide discusses and describes how to use the VitalQIP Command Line Interface.
- *VitalQIP Web Client User's Guide* (part number: 190-409-079R7.3PR2)

  This guide describes how to use the VitalQIP web client.

Product Training Support

Alcatel-Lucent University offers cost-effective educational programs that support the VitalQIP product. Our offerings also include courses on the underlying technology for the VitalQIP products (for example, DNS and DHCP). Our classes blend presentation, discussion, and hands-on exercises to reinforce learning. Students acquire in-depth knowledge and gain expertise by practicing with our products in a controlled, instructor-facilitated setting. If you have any questions, please contact us at 1-888-LUCENT8, Option 2, Option 2.

Technical support

If you need assistance with VitalQIP, you can contact the Technical Assistance Center for your region. Contact information is provided in the following table.

Phone	Email
1. Go to http://alcatel-lucent.com/support/supportredirect.html. 2. Select your country.	support@alcatel-lucent.com

How to order

To order Alcatel-Lucent documents, contact your local sales representative or use the Online Customer Support Site (OLCS) web site (http://support.lucent.com).

How to comment

To comment on this document, go to the Online Comment Form (**http://www.lucent-info.com/comments/**) or e-mail your comments to the Comments Hotline (**comments@alcatel-lucent.com**).

1 VitalQIP API

Overview

Purpose

This chapter specifies the API for VitalQIP version 5.0 or later.

Contents

This chapter covers these topics.

Introduction to the VitalQIP API

Overview

The API provides a C or C++ interface to VitalQIP functions. It is intended to serve those applications that must invoke VitalQIP functions but cannot or should not use the existing command line utilities.

Each C API returns an integer indicating success or failure. If the return code is less than zero, then an error occurred. The value of the return code indicates the general error condition. In addition, the API provides access to the VitalQIP internal error codes, enabling more detailed error analysis.

The C++ APIs handle errors by using exceptions. Calls into the C++ APIs must be enclosed in a try/catch block configured to catch objects of type *APIException*. You can then retrieve the error code and text of the error using the *APIException* object. For details, refer to “APIException” (p. 1-165).

Some of the procedural calls in the API Toolkit have been replaced with objects that contain methods to perform the same functions. This allows you to have more flexibility as you develop new applications that utilize the VitalQIP API Toolkit.

The following calls have been replaced with the following objects:

Call	Replaced with
qip_getadminprof	APIAdmin object
qip_getadminlst	APIAdmin object
qip_freeadminprof	APIAdmin object
qip_connect	APILogin object
qip_disconnect	APILogin object

The following VitalQIP API object references are available as part of the API Toolkit:

Object	Usage
APIAdmin	Used to get and set administrator profiles and to retrieve lists of administrators.
APILogin	Used to connect and disconnect from the database.
APIException	Exception class. When an API object generates an exception this specifies the type.
APIStringList	Utility class to handle lists of strings.

Object	Usage
`APINameValue`	Utility class to handle a name/value pair.
`APINameValueList`	Utility class to handle a list of name/value pairs.
`APIManagedOrg`	Used to get and set administrator permissions to specific organizations.
`APIMRItem`	Used to manage elements of the managed list contains within an **`APIManagedOrg`** object.
`APINamingPolicy`	Used to manage naming policies for organizations.
`APIObjectClass`	Used to manage object classes.

qipapi.h file description

The *qipapi.h* file contains the function prototypes for all C API routines

In addition there are now several other header files that may need to be included in a program that uses the API. These include:

- *APIAdmin.h*
- *APILogin.h*
- *APIDomain.h*
- *APIManagedOrg.h*
- *APIMRItem.h*
- *APINamingPolicy.h*
- *APIObjectClass.h*
- *APIOrg.h*
- *APIZone.h*

Each of these header files contains the class definition of a C++ Object that is accessible via the API.

Memory management

Some of the API routines return dynamically allocated structures and lists. For each routine that returns dynamically allocated structures and lists, there is another API call to free that memory.

All the C++ Objects in the API may allocate memory when the object is created and will release the memory when the object is destroyed.

Note: If the memory is freed directly instead of through the routine call, unpredictable behavior may occur.

API usage skeleton

The following is a skeleton program that uses the API Toolkit:

```
#include <stdio.h>
#include <qipapi.h>
#include "APILogin.h"

int main(int argc, char *argv[])
{
  char *user = "Thelma";
  char *pswd = "Louise";
  char *dbname = "ORCL";
  char *loginserver = "127.0.0.01";
  char *orgname = "VitalQIP Organization";

  try
  {
    // Creation of an APILogin object will instantiate a
  connection //to the database.

      APILogin *apiLogin = new APILogin(loginserver, dbname,
  user, pswd, orgname);

      // Invoke API routines here
// Deletion of the APILogin object will close the connection
     //to the database.


      delete apiLogin;
  }
  catch(APIException &ex)
  {
    printf("Error %d %s", ex.GetErrorCode(), ex.GetText());
    return ex.GetErrorCode();
  }
  catch (...) { // catch the login exception
       printf("Failed to login\n");
       exit(1);
  }
  return 0;
}
```

API/command line interface (CLI) relationships

Table 1-1 shows the relationship of routines to commands in VitalQIP. If you are familiar with VitalQIP's commands, you might want to know which routines can be used in place of certain commands. For more information regarding commands, refer to the *VitalQIP Command Line Interface User's Guide*.

Note: Only commands with corresponding routines are included in this table.

Table 1-1 Relationship of routines to commands

CLI	API	Notes
enterdomain	qip_setdomainprof	Only mandatory fields are supported.
enterdnssvr	qip_setdnsserverprof	
enterospf	qip_setospfprof	Only mandatory fields are supported.
enternetwork	qip_setnetworkprof	
entersubnet	qip_setsubnetprof	
entersimpleobj	qip_setobjectprof	
enterorganization	qip_setorgprof	
enterusergrp	qip_setusergroupprof	
enteruser	qip_setuserprof	
qip-active	qip_active	
qip-bootgen	qip_bootpgen	
qip-check	qip_check	
qip-checkobjname	qip_checkobjectname	
qip-clear	qip_clear	
qip-del	qip_del	
qip-deleteobject	qip_del	
qip-delnmdnsserver	qip_delnmdnsprof	
qip-dhcpgen	qip_dhcpgen	
qip-dhcpsync	qip_dhcpsync	
qip-dnsgen	qip_dnsgen	
qip-dnsupdate	qip_dnsupdate	
qip-getapplst	qip_getappllst	

CLI	API	Notes
qip-getcontactlst	qip_getcontactlst	
qip-getdomainext	qip_getdomainext	
qip-getdomainprof	qip_getdomainprof	
qip-getdomnlst	qip_getdomainlst	
qip-getfreesubnetlst	qip_getfreesubnetlst	
qip-gethublst	qip_gethublst	
qip-getipaddr	qip_getipaddr	
qip-getloclst	qip_getlocationlst	
qip-getnetlst	qip_getnetworklst	
qip-getnmdnsserver	qip_getnmdnsprof	
qip-getobjectlst	qip_getobjectlst	
qip-getobjectprof	qip_getobjectprof	
qip-getobjname	qip_getobjectname	
qip-getobjectrr	qip_getobjectresourcelst	
qip-getorganizationlst	qip_getorglst	
qip-getospflst	qip_getospflst	
qip-getospfprof	qip_getospfprof	
qip-getprimdnssvrlst	qip_getprimdnsserverlst	
qip-getrtrlst	qip_getrouterlst	
qip-getsnorglst	qip_getsnorglst	
qip-getsnorgprof	qip_getsnorgprof	
qip-getsubnetlst	qip_getsubnetlst	
qip-getsubnetprof	qip_getsubnetprof	
qip-getudflst	qip_getudflst	
qip-getuser	qip_getuserprof	
qip-globalmacpool	qip_globalmacpool	
qip-mcancel	qip_movecancel	
qip-move	qip_move	
qip-search	qip_search	

CLI	API	Notes
qip-setcontact	qip_setcontactprof	
qip-setdomainext	qip_setdomainext	
qip-setdomainprof	qip_setdomainprof	
qip-setlocation	qip_setlocationprof	
qip-setnmdnsserver	qip_setnmdnsprof	
qip-setobject	qip_setobjectprof	
qip-setobjectrr	qip_setobjectresourcelst	
qip-setospfprof	qip_setospfprof	
qip-setsnorgprof	qip_setsnorgprof	
qip-setsubnet	qip_setsubnetprof	
qip-setudf	qip_setudf	
qip-ungetipaddr	qip_ungetipaddr	
qip-unlock	qip_unlock	

VitalQIP API function reference

Overview

This section provides detailed descriptions of the VitalQIP API functions.

qip_enablelog

The **`qip_enablelog`** routine turns on logging for error messages. Additional text provides more information when an error occurs. The messages are copied to the file as specified in **pszFile** parameter.

Prototype

```
int qip_enablelog(char *pszFile)
```

Parameters

`pszFile`	The directory and file names where the error messages will be copied. *STDOUT* or *STDERR* can also be used as a parameter.

Return value

`QIP_OK`	Process succeeded.
`QIP_ERR_PARMS`	Invalid parameters. The organization already exists. The organization name or description is too long (limited to 30 and 255 characters respectively).

Remarks

None.

qip_disablelog

The **`qip_disablelog`** routine disables logging for error messages.

Prototype

```
int qip_disablelog(void);
```

Parameters

None.

Return value

`QIP_OK`	Process succeeded.

Remarks

None.

qip_errno

The **`qip_errno`** routine returns extended error information from the last routine call. This is an internal error number that can be supplied to VitalQIP for additional diagnostics.

Prototype

```
int qip_errno(void);
```

Parameters

None.

Return value

An integer containing the internal error code from the routine operation.

Remarks

The values of these error codes often correlate to the error codes in the **`error_msg`** table.

qip_error_message_text

The **`qip_error_message_text`** routine returns an error message providing details about the most recent routine error.

Prototype

```
char *qip_error_message_text(void);
```

Parameters

None.

Return value

String containing details about the most recent routine error.

Remarks

The pointer returned is a pointer to a statically allocated buffer, which is overwritten each time a routine error occurs.

qip_alloc

The **`qip_alloc`** routine allocates memory from the same heap as the API shared object or DLL.

Prototype

```
void *qip_alloc(unsigned int size);
```

Parameters

`size`	Size of the memory block, in bytes, to allocate.

Return value

A pointer to the allocated memory or NULL on failure.

Remarks

This routine is useful for allocating memory for list elements that will later be passed to the routines.

qip_free

The **qip_free** routine frees memory allocated by the **qip_alloc** routine.

Prototype

```
void qip_free(void *ptr);
```

Parameters

ptr	Pointer to the memory to be freed.

Return value

None.

Remarks

This routine is useful for freeing memory allocated by the **qip_alloc** routine.

qip_active

The **`qip_active`** routine retrieves a list of leases based on lease type, or if the type is QIP_ACTIVE_DELETE, deletes the specified lease.

Prototype

```
int qip_active(char *szDHCPServer, char *szAddress,
                  char chLeaseType, QIP_ACTIVE_LST
                  **ppActiveLst);
```

Parameters

`szDHCPServer`	String containing the name of the DHCP server.
`szAddress`	Object or subnet address or NULL for all.
`chLeaseType`	One of QIP_ACTIVE_ACTIVE, QIP_ACTIVE_EXPIRED, QIP_ACTIVE_ALL, or QIP_ACTIVE_DELETE.
`ppActiveLst`	Pointer to the anchor for the returned active list. The anchor will point to a list of QIP_ACTIVE_LST structures or will be NULL. Refer to Table 1-2 for the values of the fields in the QIP_ACTIVE_LST structure

Table 1-2 QIP_ACTIVE_LST structure fields

Field	Value
name	Object name.
ipaddr	Object IP address.
domain_name	Object domain.
mac_addr	Object Media Access Control (MAC) address.
expire	Yes or No.
dhcpserver	DHCP server IP address.
leasegranted	Date and time of lease grant.
leaseexpired	Date and time of lease expiry.
lasttrans	Date and time of last transaction.
next	Pointer to the next element in the list; NULL indicates the end of list.

Return value

`QIP_OK`	Completed without error. If an empty list is returned, **`ActiveLst`** is NULL.
`QIP_ERR_PARMS`	No value specified for the DHCP server.
`QIP_ERR_DB`	Database error encountered during processing. Refer to **`qip_errno`** for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeactivelst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeactivelst

The **`qip_freeactivelst`** routine frees the memory allocated in the **`qip_active`**() routine call.

Prototype

```
int qip_freeactivelst(QIP_ACTIVE_LST *pActiveLst);
```

Parameters

`pActiveLst`	Pointer to the first element of a QIP_ACTIVE_LST allocated by the API.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_active`** routine. Freeing the memory directly might produce unpredictable results.

qip_bootpgen

The **qip_bootpgen** routine generates the *Bootp* configuration file.

Prototype

```
int qip_bootpgen(char *szBootpServer, char *szFileName);
```

Parameters

szBootpServer	String containing the name of the Bootp server.
szFileName	File name of the configuration file.

Return value

QIP_OK	Completed without error.
QIP_ERR_PARMS	Could not open output file.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.

Remarks

The connected user must be the super user or owner of the file.

qip_check

The **`qip_check`** routine checks the current date with the date of all reserved addresses. If the reserved addresses have expired, then the status of the object is set to **unused**.

Prototype

```
int qip_check();
```

Parameters

None.

Return value

`QIP_OK`	Success without error.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.

Remarks

None.

qip_checkobjectname

The **qip_checkobjectname** routine returns a list of all addresses with the same hostname or alias name that is passed.

Prototype

```
int qip_checkobjectname (char *szName, QIP_ADDRESS_LST
                                   **ppAddressList);
```

Parameters

szName	String containing the name to check.
ppAddressList	Pointer to the anchor for the returned Address list. The anchor must be a pointer to a QIP_ADDRESS_LST structure. Refer to Table 1-3 for the values of the fields in the QIP_ADDRESS_LST structure.

Table 1-3 QIP_ADDRESS_LST structure fields

Field	Value
ipaddr	IP address.
next	Pointer to the next QIP_ADDRESS_LST element. NULL indicates end of list.

Return value

`QIP_OK`	Success without error. If an empty list is returned, **AddressList** is NULL.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **qip_errno** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that *must* be freed by a call to **qip_freeaddresslst**. Freeing the memory directly might produce unpredictable results.

qip_clear

The **`qip_clear`** routine clears entries in the audit table before the date specified.

Prototype

```
int qip_clear(char *szDate);
```

Parameters

`szDate`	String containing the date/time stamp that will be used to clear entries within the audit tables. All audit entries that occurred before this date will be cleared. The date/time must be entered in the following format **`mmddyyyy`**.

Return value

`QIP_OK`	Success without error.
`QIP_ERR_PARMS`	Date not specified.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.

Remarks

You ***must*** enter the date in the format that is defined for **`szDate`** in the “Parameters” section.

qip_del

The **`qip_del`** routine deletes the specified object, subnet, or domain from the VitalQIP database.

Prototype

```
int qip_del(char *szAddress, char *szName, int iType, iMulti,
               char *szApplName);
```

Parameters

`szAddress`	String containing the IP address of the item to be deleted.
`szName`	String containing the name of the item to be deleted.
`iType`	Type of item. Supported types are QIP_TYPE_OBJECT, QIP_TYPE_SUBNET, and QIP_TYPE_DOMAIN.
`iMulti`	Delete multiple items that match the specified name or address, if non-zero. An error is generated if zero and multiple items are to be deleted.
`szApplName`	Associated application name (or the empty string).

Return value

`QIP_OK`	The item was deleted from the database.
`QIP_ERR_PARMS`	Parameter error. Check that both the IP address and hostname are not NULL or empty. If specifying the IP address, ensure it is a valid “dotted quad” address. Also, this return value occurs if multiple items match the specified name or address and **`iMulti`** is 0.
`QIP_ERR_DB`	A database error occurred during the delete operation.

Remarks

Generally, only one of either **`szAddress`** or **`szName`** need be specified. If they are both specified, the IP address takes precedence. However, **`szName`** must be specified for types where IP address is inappropriate (for example, QIP_TYPE_DOMAIN).

qip_dhcpgen

The **qip_dhcpgen** routine generates and sends DHCP configuration files to the DHCP server that is passed.

Prototype

```
int qip_dhcpgen(char *szDHCPServer,char *szDirectory);
```

Parameters

szDHCPServer	String containing the fully qualified domain name of the DHCP server that configuration files should be generated.
szDirectory	Directory where the configuration files will be placed. If the directory is NULL, the default directory defined within VitalQIP will be used.

Return value

QIP_OK	Function completed successfully without error.
QIP_ERR_DB	Database error encountered. Refer to the **qip_errno** routine for more details.

Remarks

None.

qip_dhcpsync

The **qip_dhcpsync** routine synchronizes VitalQIP with true client-suggested names and their IP address bindings that are located in the DHCP server's Active Lease database. Refer to the **qip-dhcpsync** CLI command in the *VitalQIP Command Line Interface User's Guide* for more information.

Prototype

```
int qip_dhcpsync(char *szDHCPServer, char *szSubnetAddress);
```

Parameters

szDHCPServer	String containing the fully qualified domain name of the DHCP server that synchronization will occur.
szSubnetAddress	String containing a specific subnet that will be synchronized, or set this value to NULL to synchronize all subnets.

Return value

QIP_OK	Function completed successfully without error.
QIP_ERR_DB	Database error encountered. Refer to the **qip_errno** routine for more details.

Remarks

None.

qip_dnsgen

The **qip_dnsgen** routine creates the configuration and data files for the DNS server specified.

Prototype

```
int qip_dnsgen(char *szDNSServer, char *szDirectory,
                    int iLocalFlag, int iZoneFlag);
```

Parameters

szDNSServer	String containing the fully qualified domain name of the DNS server that configuration files should be generated.
szDirectory	Directory where the configuration files will be placed. If the directory is NULL, the default directory defined within VitalQIP will be used.
iLocalFlag	Flag specifying one of the following constants: QIP_DB_DNSUPDATE_SERVER and QIP_DB_DNSUPDATE_LOCAL. Both are defined in the *qipapi.h* file.
iZoneFlag	If non-zero, only push zone files that have changed.

Return value

QIP_OK	Function completed successfully without error.
QIP_ERR_DB	Database error encountered. Refer to the **qip_errno** routine for more details.
QIP_ERR_MEMORY	Memory allocation error during the routine call.

Remarks

None.

qip_dnsupdate

The **`qip_dnsupdate`** routine creates the configuration and data files for the DNS server specified. The serial number in the Start of Authority (SOA) record will be increased and the server will be notified to reread its configuration files.

Prototype

```
int qip_dnsupdate(char *szDNSServer, char *szDirectory,
                    int iLocalFlag, int iZoneFlag);
```

Parameters

`szDNSServer`	String containing the fully qualified domain name of the DNS server that configuration files should be generated.
`szDirectory`	Directory where the configuration files will be placed. If the directory is NULL, the default directory defined within VitalQIP will be used.
`iLocalFlag`	Flag specifying one of the following constants: QIP_DB_DNSUPDATE_SERVER and QIP_DB_DNSUPDATE_LOCAL. Both are defined in the *qipapi.h* file.
`iZoneFlag`	If non-zero, only push zone files that have changed.

Return value

`QIP_OK`	Function completed without error.
`QIP_ERR_DB`	Database error encountered. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

None.

qip_globalmacpool

The **qip_globalmacpool** routine adds or deletes a specified MAC address. Alternatively, this routine returns a list of MAC addresses from the MAC address pool for a particular DHCP server, depending on the action specified in the **iAction** parameter.

Prototype

```
int qip_globalmacpool(char *szDHCPServer, char *szMACAddr,
     int iAction, int iHardwareType, int ex_flag
     QIP_MACADDR_LST **ppMACAddrLst);
```

Parameters

szDHCPServer	String containing the name of the DHCP server.
szMACAddr	MAC address, which must be specified if the action is QIP_MACPOOL_ADD or QIP_MACPOOL_DEL.
iAction	One of QIP_MACPOOL_ADD, QIP_MACPOOL_DEL, or QIP_MACPOOL_QUERY
iHardwareType	One of the QIP_HW defines.
ex_flag	Includes the MAC address if it is zero. If the non-zero value is used, the MAC address is included in the excluded pool. This parameter can also be used as a field for the QIP_MACADDR_LST structure.
ppMACAddrLst	Pointer to the anchor for the returned MAC address list in the case of the QIP_MACPOOL_QUERY action. The anchor will point to a list of QIP_MACADDR_LST structures or will be NULL. Refer to Table 1-4 for the values of the fields in the QIP_MACADDR_LST structure.

Table 1-4 QIP_MACADDR_LST structure fields

Field	Value
mac_addr	MAC address
hardware_type	One of the QIP_HW defines
next	Pointer to the next element in the list; NULL indicates end of list.

Return value

`QIP_OK`	Completed without error. If an empty list is returned, **`subnetLst`** is NULL.
`QIP_ERR_PARMS`	No value specified for the DHCP server, or a bad action specified, or no MAC address specified on an add or delete, or a bad MAC address or hardware type specified, or the MAC address not found on a delete.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freemacaddrlst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freemacaddrlst

The **`qip_freemacaddrlst`** routine frees the memory allocated in the **`qip_globalmacpool()`** routine call.

Prototype

```
int qip_freemacaddrlst(QIP_MACADDR_LST *pMACAddrLst);
```

Parameters

`pMACAddrLst`	Pointer to the first element of a QIP_MACADDR_LST allocated by the API.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_globalmacpool`** routine. Freeing the memory directly might produce unpredictable results.

qip_move

The **qip_move** routine moves objects from one subnet to another on demand or prescheduled for a future date.

Prototype

```
int qip_move(char *szFromAddress, char *szToAddress,
               int iFromType, int iToType,
               char *szScheduleDate, QIP_MOVE_LST **ppMoveList)
```

Parameters

szFromAddress	The object's IP address that is to be moved.
szToAddress	The IP address or subnet address to which objects will be moved.
iFromType	Either QIP_TYPE_OBJECT or QIP_TYPE_SUBNET.
iToType	Either QIP_TYPE_OBJECT or QIP_TYPE_SUBNET.
szScheduleDate	If schedule date is specified, the object will be moved on this date/time. To move the object immediately, set this field to NULL. Use the date/time format of **mm/dd/yyyy [hh:mm:ss]**. The time element is optional.
pMoveList	Pointer to the anchor for the returned Move list. The anchor will be a pointer to a QIP_MOVE_LST structure. Refer to Table 1-6 for a list of the fields in the supported QIP_MOVE_LST structure. If the move is a scheduled move, the QIP_MOVE_LST contains all the **to_addr**. If not, the list contains all the **from_addr** and **to_addr**.

Table 1-5 QIP_MOVE_LST structure fields

Field	Value
from_addr	The address from which an object is to be moved
to_addr	The address to which an object is to be moved
next	Pointer to the next element

Return value

`QIP_OK`	Function completed successfully without error. If an empty list is returned, **`pMoveLst`** is NULL.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freemovelst`** routine. Freeing the memory directly might produce unpredictable results.

qip_movecancel

The **`qip_movecancel`** routine cancels a previously scheduled move.

Prototype

```
int qip_movecancel(char *szAddress, int iType,
                   char *szStartDate, char *szEndDate);
```

Parameters

`szAddress`	IP address of the object or subnet, which you want to cancel the move.
`iType`	Either QIP_TYPE_OBJECT or QIP_TYPE_SUBNET.
`szStartDate`	The date to begin the cancellation from using the date format **`mmddyyyyhhmm`**.
`szEndDate`	The date to end the cancellation (until this date) using the date format **`mmddyyyy hhmm`**.

Return value

`QIP_OK`	Function completed successfully without error.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.

Remarks

Moves are scheduled using the **`qip_move`** routine.

qip_freemovelst

The **`qip_freemovelst`** routine frees the memory allocated in the **`qip_move()`** routine call.

Prototype

```
int qip_freemovelst(QIP_MOVE_LST *pMoveList);
```

Parameters

`pMoveList`	Pointer to the first element of a QIP_MOVE_LST allocated by the routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_move`** routine. Freeing the memory directly might produce unpredictable results.

qip_search

The **`qip_search`** routine searches the VitalQIP database for an object matching the supplied name and type.

Prototype

```
int qip_search (char *szSearchString, int iType,
                             QIP_SEARCH_LST **ppSearchLst);
```

Parameters

`szSearchString`	String containing a search pattern. The routine compares the pattern against names in the database. The comparison is case insensitive. In addition, the pattern can contain “glob” style wildcard characters. A **?** matches a single character and an ***** matches multiple characters. For example, **A*** matches a name of any length starting with the letter A. **A?** matches any two-letter name starting with A.
`iType`	Integer specifying the type of items to search. Only items matching this type are returned. Refer to Table 1-6 for a list of the fields in the supported QIP_TYPE_* defines.
`ppSearchLst`	The pointer to the anchor for the returned list. The list contains all of the items found matching the search criteria. The pointer must have the type QIP_SEARCH_LST *. Refer to Table 1-7 for a description of the elements of the QIP_SEARCH_LST:

Table 1-6 QIP_TYPE_* defines fields

Field	Value
QIP_TYPE_ALL	Match any type
QIP_TYPE_OBJECT	Match objects only
QIP_TYPE_SUBNET	Match subnets only
QIP_TYPE_SUBNETORG	Match subnet organizations only
QIP_TYPE_DOMAIN	Match domains only
QIP_TYPE_NETWORK	Match network names only
QIP_TYPE_OSPF	Match Open Shortest Path First (OSPF) areas only
QIP_TYPE_RESOURCE_REC	Match resource records
QIP_TYPE_ROUTERGRP	Match router groups only
QIP_TYPE_MAC_ADDRESS	Match MAC addresses only

Field	Value
QIP_TYPE_DECNET	Match DECnet only
QIP_TYPE_USERFIELD	Match user fields only

Table 1-7 QIP_SEARCH_LST elements

Element	Value
name	Item name
ipaddr	Item IP address
objtype	Item object type, using QIP_TYPE_* #defines
next	Pointer to the next QIP_SEARCH_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	The search completed without error. An empty list is *not* considered an error.
`QIP_ERR_MEMORY`	Memory allocation error occurred during processing.
`QIP_ERR_DB`	A database error occurred during the search operation.

Remarks

- An empty list is ***not*** considered an error.
- The returned list ***must*** be freed by a call to the **`qip_freesearchlst`** return.
- Ensure the returned **`SearchLst`** anchor is not NULL before referencing it.
- Remember to pass the address of the anchor pointer to the search routine. For example:

```
/* Find all domains in the database */
   int qiprc;
   QIP_SEARCH_LST *srchlst, *srchwrk;
   if ( (qiprc = qip_search("*", QIP_TYPE_DOMAIN, &srchlst)) < 0)
   /* bail out */
   else {
   for (srchwrk = srchlst, srchwrk; srchwrk=srchwrk->next) {
     printf("Found %s (%s)\n", srchwrk->name, srchwrk->ipaddr)
   }
   qip_freesearchlst(srchlst);
   }
```

qip_freesearchlst

The **qip_freesearchlst** routine frees the memory allocated in the **qip_search()** routine call.

Prototype

```
int qip_freesearchlst(QIP_SEARCH_LST *pSearchLst);
```

Parameters

pSearchLst	The QIP_SEARCH_LST anchor is set by the **qip_search** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_search** routine. Freeing the memory directly might produce unpredictable results.

qip_unlock

The **qip_unlock** routine clears all selected addresses. It also resets the object's status to unused.

Prototype

```
int qip_unlock(char *szNameOrAddr);
```

Parameters

szNameOrAddr	String containing the IP address or the object name for which the Selected status will be cleared. To clear all Selected objects within the database, pass the text "corp".

Return value

QIP_OK	Function completed successfully without error.
QIP_ERR_DB	Database error encountered. Refer to the **qip_errno** routine for more details.

Remarks

None.

qip_getappllst

The **`qip_getappllst`** routine retrieves the entire list of existing applications from the VitalQIP database.

Prototype

```
int qip_getappllst(QIP_APPL_LST **pApplList);
```

Parameters

`pApplList`	Pointer to the anchor for the returned application list. The anchor must be a pointer to a QIP_APPL_LST structure. Refer to Table 1-8 for the values of the fields in the QIP_APPL_LST structure.

Table 1-8 QIP_APPL_LST structure fields

Field	Value
name	Application name
next	Pointer to the next QIP_APPL_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Completed without error. If an empty list is returned, **`pApplList`** is NULL.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeappllst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeappllst

The **`qip_freeappllst`** routine frees the memory allocated in the **`qip_getappllst()`** routine call.

Prototype

```
int qip_freeappllst(QIP_APPL_LST *pApplLst);
```

Parameters

`pApplLst`	The QIP_APPL_LST anchor set by the **`qip_getappllst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getappllst`** routine. Freeing the memory directly might produce unpredictable results.

qip_getcontactlst

The **qip_getcontactlst** routine retrieves a list of contact records that match the specified first and/or last name. If neither is specified, ***all*** contact records are returned.

Prototype

```
int qip_getcontactlst(char *szLastName, char *szFirstName,
                             QIP_CONTACT_LST **ppPersonLst);
```

Parameters

szLastName	Contact's last name for which to search (optional).
szFirstName	Contact's first name for which to search (optional).
ppPersonLst	Pointer to the anchor for the returned contact list. The anchor points to a list of QIP_CONTACT_LST structures, or will be NULL. Refer to Table 1-9 for the values of the fields in the QIP_CONTACT_LST structure.

Table 1-9 QIP_CONTACT_LST structure fields

Field	Value
last_name	Last name
first_name	First name
email_addr	E-mail address
phone_num	Phone number
pager_num	Pager number
next	Pointer to the next element in the list; NULL indicates the end of list.

Return value

QIP_OK	Completed without error. If an empty list is returned, **PersonLst** is NULL.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.
QIP_ERR_MEMORY	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **qip_freecontactlst** routine. Freeing the memory directly might produce unpredictable results.

qip_freecontactlst

The **qip_freecontactlst** routine frees the memory allocated in the **qip_getcontactlst()** routine call.

Prototype

```
int qip_freecontactlst(QIP_CONTACT_LST *pPersonLst);
```

Parameters

pPersonLst	Pointer to the first element of a QIP_CONTACT_LST allocated by the routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getcontactlst** routine. Freeing the memory directly might produce unpredictable results.

qip_setcontactprof

The **`qip_setcontactprof`** routine creates or modifies the specified contact record, based on first and last name as specified in **`pPersonProf`**.

Prototype

```
int qip_setcontactprof(QIP_CONTACT_LST *pPersonProf);
```

Parameters

`pPersonProf`	Pointer to the contact record to add or update. **Note:** If a linked list of more than one record is passed, only the one directly pointed to by **`pPersonProf`** is considered. Refer to the **`qip_getcontactlst`** routine for a description of the QIP_CONTACT_LST structure.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_PARMS`	Last name was not specified or an e-mail address was specified but does not contain an @ sign.

Remarks

None.

qip_getdnsserverprof

The **`qip_getdnsserverprof`** routine retrieves the profile for the specified DNS server. The profile is an extensive data structure containing detailed information about the server.

Prototype

```
int qip_getdnsserverprof(char *szDNSServerName,
                                  QIP_DNS_SERVER_PROF
                                  **ppDNSServerProf);
```

Parametersp

`szDNSServerName`	String containing the fully qualified DNS server name for which the profile is to be retrieved.
`ppDNSServerProf`	Pointer to be filled in with the address of the server profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-10 for the values of the fields in the DNS Server Profile.

Table 1-10 DNS Server Profile fields

Field	Value
server_name	A fully-qualified DNS server name.
server_type	A server type. The values are: - BIND-8.X - BIND-9 - LUCENT DNS - MICROSOFT-NT 4.0 - LOCAL HOST - WINDOWS 2000 DNS - WINDOWS 2000 DC - BOOTP
domain_dir	A string that determines the location of the DNS data files specified with a fully-qualified path name. You ***must*** provide a value for this field. On Windows-based systems, this includes the drive letter (for example, *c:\qip\named*).

Field	Value
dns_files	A numeric value between 0 and 127, which represents seven bits. Only the first five bits are used: - 1st bit - *db.127.0.0* - 2nd bit - *db.cache* - 3rd bit - *db.root* - 4th bit - DNS boot file - 5th bit - create out-of-zone glue records
push_lock	A Boolean value that enables or disables push lock.
server_IP	A numeric value that determines the IP address of the server.
pre_extension	A string value or "NULL" that defines the directive(s) to appear at the beginning of the *named.boot* file.
corp_extension	A string value or "NULL" defines the directives, such as options or additional zones, to be used or managed by this server. Options should be included in an options {…} block. These additional options are covered in *DNS and BIND, Fourth Edition*.
cache_file_extension	A string value or "NULL". This parameter allows you to enter additional entries, which appear at the beginning of the *db.cache* file. Use standard *db.cache* file format.
remote_server_proxy	Numeric values that create a list of IP addresses for remote proxy servers. When a firewall prevents a client from pushing to a remote server, enter IP address(es) to the remote proxy server that has access to the remote server through the firewall.
update_hr	If the schedule update is interval, this value is the hour of the day.
update_date	If the schedule update is interval, this value is the interval day.
update_day_hr[6]	If the schedule update is set to time of day, the value is in hours. A maximum of six time intervals for each day can be specified.
update_day_min[6]	If the schedule update is set to time of day, the value is the number of minutes for the hour(s). A maximum of six time intervals for each day can be specified.

Field	Value
fully_managed	A Boolean value that determines is a managed or non-managed server. If the field is set to False, it is assumed that this DNS server is managing zones that VitalQIP does not know. If there is an existing *named.conf* on the system, the VitalQIP-managed zones are merged into this file to create a new *named.conf*. If there is no *named.conf* on the system, VitalQIP creates one. A zero value means a server is not managed, and a non-zero value means the server is managed.
disable_recursion	A Boolean value that determines whether the server does recursive lookups. Only applicable to Windows 2000 DNS.
round_robin	A Boolean value that determines whether the server round robins multiple A records. Only applicable to Windows 2000 DNS.
scavenge_interval	A numeric value (in hours) that specifies how often a DNS server is enabled for scavenging to remove stale records. Zero indicates that scavenging is not set. Only applicable to Windows 2000 DNS.
secure_DNS_update	A Boolean value the enables or disables secure DNS updates. Only applicable to Lucent DNS.
override_realm	A string value that determines the name of the DNS server's Kerberos Realm. It is used when the **secure_dns_update** field is set to a False.
gss_interop_flags	A Boolean value that enables or disables support of legacy software. This field should be left at True since it is designed to permit support of legacy software. It is used only if the **secure_dns_update** field is set to a True value.
gss_max_context	A numeric value that determines the maximum number of security contexts, which the server can have active at a time. The default is 5,000. It is used only if the **secure_dns_update** field is set to True.
zone_mail	A string value for a zone's e-mail address.

Field	Value
boot_method	The values are "file" or "directory". If you are booting from file, all zones are file-based. If you are booting from a directory, all zones are directory-based. When booting from Active Directory, any parameters not specified in the Active Directory are taken from the Registry. When booting from disk, any parameters not specified in the boot file are taken from the Registry. If the **boot_method** field is set to "directory", the **fully_managed** field must be set to True. Only applicable to Windows 2000 DNS.
rndc_key	A string value of the key to be placed in the *named.conf* file, which allows the **rndc** tool to restart the DNS server. Only for BIND-9.
rndc_path	A string value for the path to the **rndc** executable. Only applicable to BIND-9.

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	DNS server not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

Use the **`qip_freednsserverprof()`** routine to dispose of the returned profile.

qip_freednsserverprof

The **`qip_freednsserverprof`** routine frees the memory allocated in the **`qip_getdnsserverprof()`** routine call.

Prototype

```
int qip_freednsserverprof(QIP_DNS_SERVER_PROF
                                *pDNSServerProf);
```

Parameters

`pDNSServerProf`	The QIP_DNS_SERVER_PROF structure allocated by the **`qip_getdnsserverprof`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getdnsserverprof`** routine. Freeing the memory directly might produce unpredictable results.

qip_setdnsserverprof

The **qip_setdnsserverprof** routine creates or modifies the specified DNS server record based on what is specified in **pDNSServerProf**.

Prototype

```
int qip_setdnsserverprof(QIP_DNS_SERVER_PROF
 *pDNSServerProf);
```

Parameters

pDNSServerProf	Pointer to the server record to add/update. Refer to the **qip_getdnsserverprof** routine for a description of the QIP_DNS_SERVER_PROF structure.

Return value

QIP_OK	Completed without error.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.
QIP_ERR_PARMS	Server name or other parameters in the server profile are incorrectly specified. The **qip_error_message_text** routine can provide details on the problematic parameter.

qip_getdomainlst

The **qip_getdomainlst** routine retrieves the entire domain list, or the associated domain(s) for the particular subnet address that is provided.

Prototype

```
int qip_getdomainlst(char *szSubnetAddress, QIP_DOMAIN_LST
**ppDomainList);
```

Parameters

szSubnetAddress	String containing the IP address of the desired subnet. If NULL, then all domains within the VitalQIP database will be returned.
ppDomainList	Pointer to the anchor for the returned domain list. The anchor must be a pointer to a QIP_DOMAIN_LST structure. Refer to Table 1-11 for the values of the fields in the QIP_DOMAIN_LST structure.

Table 1-11 QIP_DOMAIN_LST structure fields

Field	Value
name	Domain name
next	Pointer to the next QIP_DOMAIN_LST element. NULL indicates end of list.

Return value

QIP_OK	Completed without error. If an empty list is returned, **pDomainList** is NULL.
QIP_ERR_DB	Database error encountered during search. Refer to the **qip_errno** routine for more details.
QIP_ERR_MEMORY	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **qip_freedomainlst** routine. Freeing the memory directly might produce unpredictable results.

qip_freedomainlst

The **`qip_freedomainlst`** routine frees the memory allocated in the **`qip_getdomainlst()`** routine call.

Prototype

```
int qip_freedomainlst(QIP_DOMAIN_LST *pDomainLst);
```

Parameters

`pDomainLst`	The QIP_DOMAIN_LST anchor set by the **`qip_getdomainlst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_getdomainprof

The **qip_getdomainprof** routine retrieves the profile for the specified domain name. The profile is an extensive data structure containing detailed information about the domain.

Prototype

```
int qip_getdomainprof(char *szDomain, QIP_DOMAIN_PROF
**ppDomainProf);
```

Parametersp

szDomain	String containing the domain name.
ppDomainProf	Pointer to be filled in with the address of the domain profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-12 for the values of the fields in the Domain Profile.

Table 1-12 Domain Profile fields

Field	Value
domain_name	Domain name
zone_opts	Zone Options structure. Refer to "qip_getzoneopt" (p. 1-134) for a description of the QIP_ZONE_OPTIONS_REC structure.
dns_server_lst	A list of DNS server list structures. Refer to Table 1-13 for a description of the **QIP_DNS_SERVER_LST** structures.

Table 1-13 QIP_DNS_SERVER_LST structures

Structure	Value
server_name	Server name
primary_name	Primary server name (if server is not primary)
server_type	P (primary), S (secondary), or U (unmanaged)
next	Pointer to the next QIP_DNS_SERVER_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	Domain not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

Use the **`qip_freedomainprof()`** routine to dispose of the returned profile.

qip_freedomainprof

The **qip_freedominprof** routine frees the memory allocated in the **qip_getdomainprof()** routine.

Prototype

```
int qip_freedomainprof(QIP_DOMAIN_PROF *pDomainProf);
```

Parameters

pDomainProf	The QIP_DOMAIN_PROF structure allocated by the **qip_getdomainprof** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getdomainprof** routine. Freeing the memory directly might produce unpredictable results.

qip_setdomainprof

The **`qip_setdomainprof`** routine creates or modifies the specified domain based on what is specified in **`pDomainProf`**.

Prototype

```
int qip_setdomainprof(QIP_DOMAIN_PROF *pDomainProf);
```

Parameters

`pDomainProf`	Pointer to the domain record to add/update. Refer to the **`qip_getdomainprof`** routine for a description of the QIP_DOMAIN_PROF structure.

Return value

`QIP_OK`	Completed without error
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_PARMS`	Domain name or other parameters in the Domain Profile are incorrectly specified. The **`qip_error_message_text`** routine can provide details on the problematic parameter.

Remarks

None.

qip_getdomainext

The **`qip_getdomainext`** routine retrieves the extension string for the specified domain name.

Prototype

```
int qip_getdomainext(char *szDomain, QIP_DOMAIN_EXT
 **ppDomainExt);
```

Parameters

`szDomain`	String containing the domain name.
`ppDomainExt`	Pointer to be filled in with the address of the domain extension record. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-14 for the fields in the domain extension.

Table 1-14 Domain extension fields

Field	Value
prefix_ext_str	Prefix extension string
postfix_ext_str	Postfix extension string

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	Domain not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

- Use the **`qip_freedomainext()`** routine to dispose of the returned profile.
- Use of the **`qip_getdomainprof`** routine is preferred.

qip_freedomainext

The **qip_freedomainext** routine frees the memory allocated in the **qip_getdomainext()** routine call.

Prototype

```
int qip_freedomainext(QIP_DOMAIN_EXT *pDomainExt);
```

Parameters

pDomainExt	The QIP_DOMAIN_EXT structure allocated by the **qip_getdomainext** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getdomainext** routine. Freeing the memory directly might produce unpredictable results.

qip_setdomainext

The **qip_setdomainext** routine modifies the domain extension for the specified domain based on what is specified in **pDomainExt**.

Prototype

```
int qip_setdomainext(char *szDomain, QIP_DOMAIN_EXT
*pDomainExt);
```

Parameters

szDomain	String containing the domain name.
pDomainExt	Pointer to extension record to use to update the domain. Refer to the **qip_getdomainext** routine for a description of the QIP_DOMAIN_EXT structure.

Return value

QIP_OK	Completed without error
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.
QIP_ERR_PARMS	Domain name not specified.

Remarks

None.

qip_getfreesubnetlst

The **`qip_getfreesubnetlst`** routine retrieves a list of free subnets (subnets not yet defined), or all available existing subnets (subnets defined, but unused) contained in the supplied network and subnet mask.

Prototype

```
int qip_getfreesubnetlst(char *szNetworkAddress,
                         char *szSubnetMask, int iExistant,
                         QIP_FREE_SUBNET_LST **ppFreeSubnetLst);
```

Parameters

`szNetworkAddress`	String containing the network address in dotted decimal notation to search for free subnets.
`szSubnetMask`	String containing the subnet mask in dotted decimal notation that will be used in conjunction with the network address to search for free subnets.
`iExistant`	If set to 0, it gets free subnets. If set to non-zero, it gets all defined, unused subnets.
`ppFreeSubnetList`	Pointer to the anchor for the returned subnet list. The anchor must be a pointer to a QIP_FREE_SUBNET_LST structure. Refer to Table 1-15 for the values of the fields in the QIP_FREE_SUBNET_LST structure.

Table 1-15 QIP_FREE_SUBNET_LST structure fields

Field	Value
name	Subnet name
ipaddr	Subnet IP address
next	Pointer to the next QIP_FREE_SUBNET_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Search without error. If an empty list is returned, **`FreeSubnetList`** is NULL.
`QIP_ERR_PARMS`	Bad network or netmask.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to **`qip_freefreesubnetlst`**. Freeing the memory directly might produce unpredictable results.

qip_freefreesubnetlst

The **`qip_freefreesubnetlst`** routine frees the memory allocated in the **`qip_getfreesubnetlst()`** routine.

Prototype

```
int qip_freefreesubnetlst(QIP_FREE_SUBNET_LST
                                *pFreeSubnetList);
```

Parameters

`pFreeSubnetList`	The QIP_FREE_SUBNET_LST anchor set by the **`qip_getfreesubnetlst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_gethublst

The **`qip_gethublst`** routine fetches the hubs from the supplied subnet.

Prototype

```
int qip_gethublst(char *szSubnetAddress, QIP_HUB_LST
                        **ppHubList);
```

Parameters

`szSubnetAddress`	String containing the name of the subnet to search.
`ppHubList`	The pointer to the anchor for the returned list. The list contains the hubs found in the supplied subnet. The pointer must have the type QIP_HUB_LST. Refer to Table 1-16 for a description of the elements of QIP_HUB_LST.

Table 1-16 QIP_HUB_LST elements

Field	Value
name	Hub name
ipaddr	Hub IP address
next	Pointer to the next QIP_HUB_LST element. NULL indicates end of list.

Return value

`QIP_OK`	The search completed without error. An empty list is *not* considered an error.
`QIP_ERR_NOTFOUND`	Unknown subnet.
`QIP_ERR_PARMS`	The supplied subnet address is invalid.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	A database error occurred during the search operation.

Remarks

The returned list ***must*** be freed by a call to the **`qip_freehublst`** routine.

qip_freehublst

The **`qip_freehublst`** routine frees the memory allocated in the **`qip_gethublst()`** routine.

Prototype

```
int qip_freehublst(QIP_HUB_LST *pHubList);
```

Parameters

`pHubList`	The QIP_HUB_LST anchor set by the **`qip_gethublst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_getipaddr

The **qip_getipaddr** routine retrieves the next free IP address and marks the address as selected.

Prototype

```
int qip_getipaddr(char *szDomain, char *szSubnetAddress,
                        char *szObjectAddress);
```

Parameters

szDomain	String containing the fully qualified domain name from which you want to retrieve an address.
szSubnetAddress	String containing the subnet address from which you want to retrieve an address.
szObjectAddress	The next free IP object address. Make sure it is a buffer large enough to hold an IP address (QIP_IPADDR_LEN); this routine does not allocate memory.

Return value

QIP_OK	Success without error.
QIP_ERR_DB	Database error encountered. Refer to the **qip_errno** routine for more details.
QIP_ERR_PARMS	Domain or subnet not (or not properly) specified.

Remarks

If you decide not to use the returned object address, you must use the **qip_ungetipaddr** routine to reset the object address back to unused. Note that the Schedule Service (**qipd**) cleans up these entries automatically, as well as every day.

qip_ungetipaddr

The **qip_ungetipaddr** routine is used to reset a selected address back to the unused state. Use this routine to clean up addresses that are marked as selected by the **qip_getipaddr** routine.

Prototype

```
int qip_ungetipaddr(char *szAddress);
```

Parameters

szAddress	The IP address of the selected object that you want to returned an unused status.

Return value

QIP_OK	Success without error.
QIP_ERR_DB	Database error encountered during search. Refer to ***qip_errno*** for more details.
QIP_ERR_PARMS	Unknown IP address.

Remarks

Used in conjunction with the **qip_getipaddr** routine.

qip_getlocationlst

The **`qip_getlocationlst`** routine retrieves a list of location records that match fields specified in the **`pLocationSearch`** profile. If this parameter is NULL, all location records are returned

Prototype

```
int qip_getlocationlst(QIP_LOCATION_LST *pLocationSearch,
                       QIP_LOCATION_LST **ppLocationList);
```

Parameters

`pLocationSearch`	Profile specifying field values to match in the location database. Any or all of the fields may be specified to restrict the search. This parameter is optional; if NULL is specified, all records are returned.
`ppLocationList`	Pointer to the anchor for the returned location list. The anchor will point to a list of QIP_LOCATION_LST structures, or will be NULL. Refer to Table 1-17 for the values of the fields in the QIP_LOCATION_LST structure.

Table 1-17 QIP_LOCATION_LST structure fields

Field	Value
loc_id	Location ID
street1	Street1
street2	Street2
city	City
state	State
zip	Zip or postal code
country	Country
next	Pointer to the next element in the list; NULL indicates the end of list

Return value

`QIP_OK`	Completed without error. An empty list can be returned, in which case **`LocationList`** will be NULL.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to **`qip_freelocationlst`**. Freeing the memory directly might produce unpredictable results.

qip_freelocationlst

The **`qipfreelocationlst`** routine frees the memory allocated in the **`qip_getlocationlst()`** routine call.

Prototype

```
int qip_freelocationlst(QIP_LOCATION_LST *pLocationList);
```

Parameters

`pLocationList`	Pointer to the first element of a QIP_LOCATION_LST allocated by the routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_setlocationprof

The **`qip_setlocationprof`** routine creates or modifies the specified location record, based on whether or not the **`loc_id`** is specified (non-zero). If the **`loc_id`** is specified, it modifies the record if it can find it; otherwise it creates it.

Prototype

```
int qip_setlocationprof(QIP_LOCATION_LST *pLocationProf);
```

Parameters

`pLocationProf`	Pointer to the location record to add/update. **Note:** If a linked list of more than one record is passed, only the one directly pointed to by **`pLocationProf`** is considered. Refer to the **`qip_getlocationlst`** routine for a description of the QIP_LOCATION_LST structure.

Return value

`QIP_OK`	Completed without error
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.

Remarks

None.

qip_getnetworklst

The **`qip_getnetworklst`** routine retrieves the entire network list from the VitalQIP database.

Prototype

```
int qip_getnetworklst(QIP_NETWORK_LST **ppNetworkList);
```

Parameters

`ppNetworkList`	Pointer to the anchor for the returned network list. The anchor must be a pointer to a QIP_NETWORK_LST structure. Refer to Table 1-18 for the values of the fields in the QIP_NETWORK_LST structure.

Table 1-18 QIP_NETWORK_LST structure fields

Field	Value
ipaddr	Network address
next	Pointer to the next QIP_NETWORK_LST element. NULL indicates end of list.

Return value

`QIP_OK`	Completed without error. If an empty list is returned, **`pDomainList`** is NULL.
`QIP_ERR_DB`	Database error encountered during search. Refer to **`qip_errno`** for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freenetworklst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freenetworklst

The **`qip_freenetworklst`** routine frees the memory allocated in the **`qip_getnetworklst()`** routine.

Prototype

```
int qip_freenetworklst(QIP_NETWORK_LST *pNetworkLst);
```

Parameters

`pNetworkLst`	The QIP_NETWORK_LST anchor set by the **`qip_getnetworklst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_setnetworkprof

The **`qip_setnetworkprof`** routine creates or modifies the specified network record based on its existence.

Prototype

```
int qip_setnetworkprof(QIP_NETWORK_PROF *pNetworkProf);
```

Parameters

`pNetworkProf`	Profile specifying fields to modify or add in the database. Refer to Table 1-19 for the values of the fields in the QIP_NETWORK_PROF structure.

Table 1-19 QIP_NETWORK_PROF structure fields

Field	Value
network_addr	Network address
network_name	Network name
cidr	Y or N
mask_length	Netmask length
warning_percent	Warning percent
warning_type	Visual, Email or Both
email_addr	E-mail address (for new networks only)
dns_server	Reverse zone server (optional)

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_PARMS`	IP address is malformed, mask_length does not represent a valid subnet mask, mask_length represents a classless mask and Classless Inter-Domain Routing (CIDR) is **N**, warning percent or type is out of range, or e-mail address is not valid and this is a new network.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

E-mail address is not changed if the network is being modified (as opposed to being created). The reverse DNS server is optional.

qip_getnmdnsprof

The **`qip_getnmdnsprof`** routine retrieves the profile for the specified non-managed DNS server. The profile is an extensive data structure containing detailed information about the server.

Prototype

```
int qip_getnmdnsprof(char *szServerName,
                        char *szServerAddress, QIP_NMDNS_PROF
                        **ppNmDNSProf);
```

Parameters

`szServerName`	String containing the primary DNS server name.
`szServerAddress`	String containing the primary DNS server IP address.
`ppNmDNSProf`	Pointer to be filled in with the address of the server profile. The API routine allocates the memory for the structure and fills in the fields. Refer to Table 1-20 for the values of the fields in the NmDNS server profile.

Table 1-20 NmDNS server profile fields

Field	Value
server_name	Fully qualified DNS server name
nm_server_name	Non-managed server name
nm_server_ipaddr	Non-managed server IP address
zone_lst	List of zones (of type QIP_NAME_LST)
network_lst	List of networks (of type QIP_ADDRESS_LST)

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	Server name or IP address not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

None.

qip_freenmdnsprof

The **`qip_freenmdnsprof`** routine frees the memory allocated in the **`qip_getnmdnsprof()`** routine call.

Prototype

```
int qip_freenmdnsprof(QIP_NMDNS_PROF *pNmDNSProf);
```

Parameters

`pNmDNSProf`	The QIP_NMDNS_PROF structure allocated by the **`qip_getnmdnsprof`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getnmdnsprof`** routine. Freeing the memory directly might produce unpredictable results.

qip_setnmdnsprof

The **`qip_setnmdnsprof`** routine creates or modifies the specified non-managed DNS server record based on what is specified in **`pNmDNSProf`**.

Prototype

```
int qip_setnmdnsprof(QIP_NMDNS_PROF *pNmDNSProf);
```

Parameters

`pNmDNSProf`	Pointer to the server record to add or update. Refer to the **`qip_getnmdnsprof`** routine for a description of the QIP_NMDNS_PROF structure.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_PARMS`	Server name, IP address, or other parameters in the server profile are incorrectly specified. The **`qip_error_message_text`** routine can provide details on the problematic parameter.

Remarks

None.

qip_delnmdnsprof

The **`qip_delnmdnsprof`** routine deletes the specified non-managed DNS server.

Prototype

```
int qip_delnmdnsprof(char *szServerName,
char *szServerAddress);
```

Parameters

`szServerName`	String containing the primary DNS server name.
`szServerAddress`	String containing the primary DNS server IP address.

Return value

`QIP_OK`	Profile found and deleted.
`QIP_ERR_PARMS`	Server name or IP address not specified.
`QIP_ERR_DB`	Database error during processing.

Remarks

None.

qip_getobjectname

The **`qip_getobjectname`** routine retrieves a new object name based on the specified object class.

Prototype

```
int qip_getobjectname(char *szObjectClass,
                             char *szObjectName);
```

Parameters

`szObjectClass`	String containing the object class, when a new name is requested. (For example, “Workstation”.)
`szObjectName`	The returned object name. Make sure it is a buffer large enough to hold a string of size QIP_NAME_LEN; this routine does not allocate memory.

Return value

`QIP_OK`	Success without error.
`QIP_ERR_DB`	Database error encountered. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_PARMS`	Object Class not (or not properly) specified.

Remarks

None.

qip_getobjectlst

The **`qip_getobjectlst`** routine fetches the objects from the supplied subnet. The caller can have all objects within the subnet returned, or have only used objects returned.

Prototype

```
int qip_getobjectlst(int iObjStatus, char *szSubnetAddress,
                           QIP_OBJECT_LST **ppObjectList);
```

Parameters

`iObjStatus`	Integer indicating return of all objects or just used ones. Use the QIP_USED_OBJECTS or QIP_ALL_OBJECTS #define as appropriate.
`szSubnetAddress`	String containing the name of the subnet to search.
`ppObjectList`	The pointer to the anchor for the returned list. The list contains the objects found in the supplied subnet. The pointer must have the type QIP_OBJECT_LST. Refer to Table 1-21 for a description of the elements of QIP_OBJECT_LST.

Table 1-21 QIP_OBJECT_LST elements

Element	Value
ipaddr	Object IP address
name	Object name

Element	Value
class_desc	One of the following object classes: • Workstation • X-terminal • PC • Printer • Server • Wiring_HUB • Router • Bridge • Terminal_Server • Switch • Legacy_System • Gateway • Test_Equipment • Undefined • Others • External • User-defined

Element	Value
object_type	One of the following object types: • Static • Reserved • M-BOOTP • M-DHCP • A-BOOTP • A-DHCP • D-DHCP • Dynamic • Planned_Used • Static(SM*) • Reserved(SM) • M-BOOTP(SM) • M-DHCP(SM) • A-BOOTP(SM) • A-DHCP(SM) • D-DHCP(SM • Dynamic(SM) • Selected • Unused • Static(GAP) * SM indicates that the object has “scheduled move” status.
domain_name	Domain name of this object
next	Pointer to the next QIP_OBJECT_LST element. NULL indicates end of list.

Return value

`QIP_OK`	The search completed without error. An empty list is ***not*** considered an error.
`QIP_ERR_NOTFOUND`	Unknown subnet.
`QIP_ERR_PARMS`	The supplied subnet address is invalid.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	A database error occurred during the search operation.

Output examples

All Objects

The following is output when you specify QIP_ALL_OBJECTS on the **`iObjStatus`** parameter

```
"10.58.208.5"   "dhcp-client-0003" "PC"     "D-DHCP" "seg4.qa.quadritek.com"
"10.58.208.4"   "dhcp-client-0002" "PC"     "D-DHCP" "seg4.qa.quadritek.com"
"10.58.208.3"   ""      ""      "Unused"        ""
"10.58.208.2"   "dhcp-client-0001" "PC"     "D-DHCP" "seg4.qa.quadritek.com"
"10.58.208.1"   ""      ""      "Unused"        ""
```

Used Objects

The following is output when you specify QIP_USED_OBJECTS on the **`iObjStatus`** parameter.

```
------ Domain: seg5.qa.quadritek.com     Object Name: wsp000087WSS ---------

Description: Workstation
Subnet Address: 10.200.100.0 Mask 255.255.255.0
Location Information:

Contact Information:
        First Name:
        Last Name:
        E-mail:
        Phone:
        Pager:
DHCP Server:    DHCP Template:
DNS Servers:
Time Servers:
Routers:
        Name:  (10.200.100.1)
MX Hosts:
Forwarding Hosts:
Aliases:
```

```
Hub Slots:
```

Remarks

- An empty list is ***not*** considered an error.
- The returned list ***must*** be freed by a call to the **`qip_freeobjectlst`** routine.

qip_freeobjectlst

The **qip_freeobjectlst** routine frees the memory allocated in the **qip_getobjectlst()** routine.

Prototype

```
int qip_freeobjectlst(QIP_OBJECT_LST *pObjectList);
```

Parameters

pObjectList	The QIP_OBJECT_LST anchor set by the **qip_getobjectlst** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getobjectlst** routine. Freeing the memory directly might produce unpredictable results.

qip_getobjectprof

The **`qip_getobjectprof`** routine retrieves the profile for the specified object. The profile is an extensive data structure containing detailed information about the object.

Prototype

```
int qip_getobjectprof(char *szObjectAddress, QIP_OBJECT_PROF
                                  **ppObjectProf);
```

Parameters

`szObjectAddress`	String containing the IP address of the desired object.
`ppObjectProf`	Pointer to be filled in with the address of the object's profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-22 for the values of the fields in the Object Profile:

Table 1-22 Object profile fields

Field	Value
domain_name	Domain in which the object resides
obj_name	Object name
obj_desc	Object description
ipaddr	Object IP address
obj_class_des	Object class description (for example, Xterminal)
appl_name	Application name
mac_addr	Object's MAC address
exp_date	Lease expiry date
bootp_type	Dynamic configuration type. Refer to QIP_DYNCFG_* #defines in the *qipapi.h* file for details.
subnet_addr	Subnet IP address where the object resides
subnet_mask	Subnet mask where the object resides
subnet_name	Subnet name where the object resides
dual_prot_code	Dual protocol code. Indicates if the object supports dual protocol stacks. Refer to QIP_DUAL_* defines in *qipapi.h.*
decnet_area	DECnet area. Set only if dual_prot_code indicates DECnet as the second protocol.

Field	Value
decnet_addr	DECnet address. Set only if dual_prot_code indicates DECnet as the second protocol.
hub_name	Hub to which object is wired
slot_name	Slot for hub_name
port_num	Port for slot_name
floor	Location information
room_id	
street	
city	
state	
zip	
country	
object_tag	Asset information
manufacturer	
model_type	
serial_no	
asset_no	
host_id	
purchase_date	
contact_lname	Last name
contact_fname	First name
contact_eaddr	E-mail address
contact_phone	Phone number
contact_pager	Pager number
objsvr_lst	Object DNS server list pointer
objrtr_lst	Object router list pointer
objts_lst	Object time server list pointer
alias_lst	Object alias list pointer
mx_host_lst	Object mail exchange (MX) host list pointer

Field	Value
forward_lst	Object forwarding host list pointer
hub_slot_lst	Object hub slot list pointer
ftp_svr_name	Object's default Trivial File Transfer Protocol (TFTP) server
hardware_type	Networking hardware. Refer to #defines for values.
dhcp_server	Object's DHCP server hostname
dhcp_template	DHCP template name
bootfile_name	*BOOTP* file name
rtr_grp_name	Router group name
tftp_flag	1=object is TFTP server, 0=not a TFTP server
time_flag	1=object is time server, 0=not a time server
ns_type	1=object to be registered with name servers
ns_update_flags	NS update flags
ttl_time	Time to Life (TTL) time
lease_time	0=unlimited, >0=expiration time
vendor_class	Vendor class
netbios_domain	NetBIOS domain
netbios_name	NetBIOS name

Return value

`QIP_OK`	Object Profile found.
`QIP_ERR_PARMS`	Invalid (for example, poorly formed) object address.
`QIP_ERR_NOTFOUND`	Unknown object.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

- Use the **`qip_freeobjectprof()`** routine to dispose of the returned profile.
- The **`obj_class_des`** field is chosen from the name policy table in the database. While it is a string, the valid values are restricted to those in the table.
- The Object Profile contains list anchors for various servers and aliases associated with the object. Refer to the *qipapi.h* file for the structure definitions.

qip_freeobjectprof

The **qip_freeobjectprof** routine frees the memory allocated in the **qip_getobjectprof()** routine.

Prototype

```
int qip_freeobjectprof(QIP_OBJECT_PROF *pObjectProf);
```

Parameters

pObjectProf	The pointer to a QIP_OBJECT_PROF structure returned by the **qip_getobjectprof** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getobjectprof** routine. Freeing the memory directly might produce unpredictable results.

qip_setobjectprof

The **qip_setobjectprof** routine sets the profile for the specified object.

Prototype

```
int qip_setobjectprof(QIP_OBJECT_PROF *pObjectProf);
```

Parameters

pObjectProf	A pointer to a QIP_OBJECT_PROF structure. Typically, set by the **qip_getobjectprof** routine.

Return value

QIP_OK	Object Profile updated.
QIP_ERR_PARMS	Invalid or poorly formed elements in the structure. Check the various IP addresses and masks for validity.
QIP_ERR_MEMORY	Memory allocation error during processing.
QIP_ERR_DB	Database error occurred during update. Check that the domain name, application name, and DHCP parameters are valid and in the database.

Remarks

To change any fields in an Object Profile, the calling routine should first retrieve the existing parameters via the **qip_getobjectprof** routine, modify the desired field, and then update the profile via the **qip_setobjectprof** routine. Refer to the **qip_setsubnetprof** routine for an example.

qip_getobjectresourcelst

The **qip_getobjectresourcelst** routine retrieves the Resource Record list for the specified object.

Prototype

```
int qip_getobjectresourcelst(char *szObjectAddress,
                             QIP_RESOURCE_LST **ppResourceLst);
```

Parameters

szObjectAddress	Object IP address for which resource records are retrieved.
ppResourceLst	Pointer to the anchor for the returned resource record list. The anchor must be a pointer to a QIP_RESOURCE_LST structure. Refer to Table 1-23 for the values of the fields in the QIP_RESOURCE_LST structure.

Table 1-23 QIP_RESOURCE_LST structure fields

Field	Value
rr_owner	Owner
rr_type	Type
rr_class	Class
rr_text	Free text
rr_min_ttl	Minimum TTL
rr_apply_to_zone	Apply to **Zone** flag
rr_change_flag	Change flag to indicate whether to add or delete this element from the **qip_setobjectresourcelst** routine. Refer to the QIP_CHANGE_FLAG_ #defines.
rr_id	ID
next	Pointer to the next QIP_RESOURCE_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_PARMS`	Bad object IP address.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeobjectresourcelst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeobjectresourcelst

The **qip_freeobjectresourcelst** routine frees the memory allocated in the **qip_getobjectresourcelst()** routine.

Prototype

```
int qip_freeobjectresourcelst(QIP_RESOURCE_LST
                                  *pResourceLst);
```

Parameters

pResourceLst	The QIP_RESOURCE_LST anchor set by the **qip_getobjectresourcelst** routine.

Return value

QIP_OK	Memory freed.

Remarks

None.

qip_setobjectresourcelst

The **qip_setobjectresourcelst** routine adds and/or deletes Resource Records associated with the specified object.

Prototype

```
int qip_setobjectresourcelst(char *szObjectAddress,
   QIP_RESOURCE_LST *pResourceLst);
```

Parameters

szObjectAddress	Object's IP address to add or delete the resource records to.
pResourceLst	List of resource records to add or delete from the database for the specified object. The **rr_change_flag** of each element must be set to either QIP_CHANGE_FLAG_ADD or QIP_CHANGE_FLAG_DEL. Refer to the **qip_setobjectresourcelst** routine for a description of this structure.

Return value

QIP_OK	Completed without error.
QIP_ERR_PARMS	IP address is malformed; or an element contains invalid data.
QIP_ERR_DB	Database error encountered during processing. Refer to the "qip_errno" (p. 1-16) routine for more details.
QIP_ERR_MEMORY	Memory allocation error during the routine call.

Remarks

None.

qip_getorglst

The **qip_getorglst** routine retrieves the entire organization list from the VitalQIP database.

Prototype

```
int qip_getorglst(QIP_NAME_LST **ppOrgLst);
```

Parameters

ppOrgLst	Pointer to the anchor for the returned organization list. The anchor must be a pointer to a QIP_NAME_LST structure. Refer to Table 1-24 for the values of the fields in the QIP_NAME_LST structure.

Table 1-24 QIP_NAME_LST structure fields

Field	Value
name	Item name
next	Pointer to the next QIP_NAME_LST element. NULL indicates end of list

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **qip_errno** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **qip_freeorglst** routine. Freeing the memory directly might produce unpredictable results.

qip_freeorglst

The **`qip_freeorglst`** routine frees the memory allocated in the **`qip_getorglst()`** routine.

Prototype

```
int qip_freeorglst(QIP_NAME_LST *pOrgLst);
```

Parameters

`pOrgLst`	The QIP_ORG_LST anchor set by the **`qip_getorglst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_setorgprof

The **qip_setorgprof** routine creates or modifies the specified organization record, if it exists.

Prototype

```
int qip_setorgprof(QIP_ORG_PROF *pOrgProf);
```

Parameters

pOrgProf	Profile specifying fields to modify or add in the database. Refer to Table 1-25 for the values of the fields in the QIP_ORG_PROF structure.

Table 1-25 QIP_ORG_PROF structure fields

Field	Value
org_name	Organization name
org_desc	Organization description
max_objects	Maximum number of objects

Return value

QIP_OK	Completed without error.
QIP_ERR_PARMS	Organization name malformed.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.

Remarks

None.

qip_getospflst

The **`qip_getospflst`** routine retrieves the list of all OSPF areas defined in the VitalQIP database.

Prototype

```
int qip_getospflst(QIP_OSPF_LST **ppOSPFList);
```

Parameters

`ppOSPFList`	Pointer to the anchor for the returned OSPF list. The anchor must be a pointer to a QIP_OSPF_LST structure. Refer to Table 1-26 for the values of the fields in the QIP_OSPF_LST structure.

Table 1-26 QIP_OSPF_LST structure fields

Field	Value
network_addr	Network address
name	OSPF area name
start_addr	Starting address
end_addr	Ending address
next	Pointer to the next QIP_OSPF_LST element. NULL indicates end of list.

Return value

`QIP_OK`	Completed without error. If an empty list is returned, **`pOSPVLst`** is NULL.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeospflst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeospflst

The **qip_freeospflst** routine frees the memory allocated in the **qip_getospflst()** routine.

Prototype

```
int qip_freeospflst(QIP_OSPF_LST *pOSPFList);
```

Parameters

pOSPFList	The QIP_OSPF_LST anchor returned by the **qip_getospflst** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getospflst** routine. Freeing the memory directly might produce unpredictable results.

qip_getospfprof

The **`qip_getospfprof`** routine retrieves the profile for the specified OSPF name. The profile is an extensive data structure containing detailed information about the OSPF area.

Prototype

```
int qip_getospfprof(char *szOSPFName, QIP_OSPF_PROF
                       **ppOSPFProf);
```

Parameters

`szOSPFName`	String containing the OSPF name.
`ppOSPFProf`	Pointer to be filled in with the address of the OSPF profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-27 for the values of the fields in the OSPF Profile.

Table 1-27 OSPF Profile fields

Field	Value
ospf_range_lst	List of QIP_OSP_RANGE_LST structures. Refer to Table 1-28 for a description of these structures.
ospf_subnet_lst	List of QIP_OSPF_SUBNET_LST structures. Refer to Table 1-29 for a description of these structures.
ospf_num	OSPF number
ospf_name	OSPF name
ospf_compliant	OSPF compliant flag
warning_percent	Warning percent
warning_type	Warning type

Table 1-28 QIP_OSPF_RANGE_LST structures

Structure	Value
start_string	Range start IP address
end_strng	Range end IP address
mask_string	Dotted quad mask
next	Pointer to the next QIP_OSPF_RANGE_LST element. NULL indicates the end of list.

Table 1-29 QIP_OSPF_SUBNET_LST structures

Structure	Value
subnet_addr	Subnet IP address
subnet_name	Subnet name
subnet_mask	Dotted quad subnet mask
network_addr	Network IP address
network_name	Network name
assign_flag	Y or N
subnet_org_name	Subnet organization name
ospf_name	OSPF name
next	Pointer to the next QIP_OSPF_SUBNET_LST element. NULL indicates end of list.

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	OSPF name not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

Use the **`qip_freeospfprof()`** routine to dispose of the returned profile.

qip_freeospfprof

The **qip_freeospfprof** routine frees the memory allocated in the **qip_getospfprof()** routine.

Prototype

```
int qip_freeospfprof(QIP_OSPF_PROF *pOSPFProf);
```

Parameters

pOSPFProf	The QIP_OSPF_PROF structure allocated by the **qip_getospfprof** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getospfprof** routine. Freeing the memory directly might produce unpredictable results.

qip_setospfprof

The **`qip_setospfprof`** routine creates or modifies the specified OSPF area based on what is specified in **`pOSPFProf`**.

Prototype

```
int qip_setospfprof(QIP_OSPF_PROF *pOSPFProf);
```

Parameters

`pOSPFProf`	Pointer to the OSPF profile to add/update. Refer to the ***qip_getospfprof*** routine for a description of the QIP_OSPF_PROF structure.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_PARMS`	OSPF name or other parameters in the OSPF profile are incorrectly specified. The **`qip_error_message_text`** routine can provide details on the problematic parameter.

Remarks

None.

qip_getprimdnsserverlst

The **qip_getprimdnsserverlst** routine retrieves the list of all primary DNS servers defined for the specified domain.

Prototype

```
int qip_getprimdnsserverlst(char *szDomain, QIP_SERVER_LST
                                         **ppServerList);
```

Parameters

szDomain	Domain to search.
ppServerList	Pointer to the anchor for the returned server list. The anchor must be a pointer to a QIP_SERVER_LST structure. Refer to Table 1-30 for the values of the fields in the QIP_SERVER_LST structure.

Table 1-30 QIP_SERVER_LST structure fields

Field	Value
name	Server name
ipaddr	Server IP address
next	Pointer to the next QIP_SERVER_LST element. NULL indicates end of list.

Return value

`QIP_OK`	Completed without error. If an empty list is returned, **ppServerList** is NULL.
`QIP_ERR_PARMS`	Domain not specified.
`QIP_ERR_DB`	Database error encountered during search. Refer to **qip_errno** routine for details.
`QIP_ERR_MEMORY`	Memory allocation error during the call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **qip_freeserverlst** routine. Freeing the memory directly might produce unpredictable results.

qip_freeserverlst

The **`qip_freeserverlst`** routine frees the memory allocated in the **`qip_getprimdnsserverlst()`** routine.

Prototype

```
int qip_freeserverlst(QIP_SERVER_LST *pServerList);
```

Parameters

`pServerList`	The QIP_SERVER_LST anchor set by the **`qip_getprimdnsserverlst`** routine (or similar routine call that returns a server list).

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getprimdnsserverlst`** routine (or similar calls). Freeing the memory directly might produce unpredictable results.

qip_getrouterlst

The **`qip_getrouterlst`** routine fetches the routers from the supplied subnet.

Prototype

```
int qip_getrouterlst(char *szSubnetAddress, QIP_ROUTER_LST
                        **ppRouterList);
```

Parameters

`szSubnetAddress`	String containing the IP address of the subnet to search.
`ppRouterList`	The pointer to the anchor for the returned list. The list contains the routers found in the supplied subnet. The pointer must have the type QIP_ROUTER_LST. Refer to Table 1-31 for a description of the elements of QIP_ROUTER_LST.

Table 1-31 QIP_ROUTER_LST elements

Element	Value
name	Router name
ipaddr	Router IP address
next	Pointer to the next QIP_ROUTER_LST element. NULL indicates the end of list

Return value

`QIP_OK`	The search completed without error. An empty list is ***not*** considered an error.
`QIP_ERR_NOTFOUND`	Unknown subnet.
`QIP_ERR_PARMS`	The supplied subnet address is invalid.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	A database error occurred during the search operation.

Remarks

- An empty list is ***not*** considered an error.
- The returned list ***must*** be freed by a call to the **`qip_freerouterlst`** routine.

qip_freerouterlst

The **`qip_freerouterlst`** routine frees the memory allocated in the **`qip_getrouterlst()`** routine.

Prototype

```
int qip_freerouterlst(QIP_ROUTER_LST *pRouterLst);
```

Parameters

`pRouterLst`	The QIP_ROUTER_LST anchor set by the **`qip_getrouterlst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getrouterlst`** routine. Freeing the memory directly might produce unpredictable results.

qip_getsnorglst

The **`qip_getsnorglst`** routine fetches all the subnet organizations from the database.

Prototype

```
int qip_getsnorglst(QIP_NAME_LST **ppSnorgLst);
```

Parameters

`ppSnorgLst`	The pointer to the anchor for the returned list. The list contains the subnet organizations found in database. The pointer must have the type QIP_NAME_LST. Refer to Table 1-32 for a description of the elements of QIP_NAME_LST.

Table 1-32 QIP_NAME_LST elements

Field	Value
name	Subnet organization name
next	Pointer to the next QIP_NAME_LST element. NULL indicates end of list.

Return value

`QIP_OK`	The fetch completed without error. An empty list is ***not*** considered an error.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	A database error occurred during the search operation.

Remarks

The returned list ***must*** be freed by a call to the **`qip_freesnorglst`** routine.

qip_freesnorglst

The **qip_freesnorglst** routine frees the memory allocated in the **qip_getsnorglst()** routine.

Prototype

```
int qip_freesnorglst(QIP_NAME_LST *pSnorgLst);
```

Parameters

pSnorgLst	The QIP_NAME_LST anchor set by the **qip_getsnorglst** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getsnorglst** routine. Freeing the memory directly might produce unpredictable results.

qip_getsnorgprof

The **qip_getsnorprof** routine retrieves the subnet organization profile for the specified subnet organization name. The profile is an extensive data structure containing detailed information about the subnet organization.

Prototype

```
int qip_getsnorgprof(char *szSubnetOrgName, QIP_SNORG_PROF
                        **ppSnorgProf);
```

Parameters

szSubnetOrgName	String containing the subnet organization name.
ppSnorgProf	Pointer to be filled in with the address of the Subnet Organization Profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-33 for the values of the fields in the Subnet Organization Profile.

Table 1-33 Subnet Organization Profile fields

Field	Value
alloc_lst	List of QIP_ALLOC_LST structures. Refer to Table 1-34 for a description of these structures.
subnet_lst	List of QIP_SNORG_SUBNET_LST structures. Refer to Table 1-35 for a description of these structures.
dhcp_server	Name of the DHCP server for this subnet
dhcp_option_template	Name of the DHCP option template for this subnet
subnet_organization_name	Subnet organization name
warning_percent	Warning percent
warning_type	Warning type

Table 1-34 QIP_ALLOC_LST structures

Structure	Value
object_class	Object class string
relative_addr	Object relative address
next	Pointer to the next QIP_ALLOC_LST element. NULL indicates the end of list.

Table 1-35 QIP_SNORG_SUBNET_LST structures

Structure	Value
subnet_addr	Subnet IP address
subnet_mask	Dotted quad subnet mask
network_addr	Network IP address
next	Pointer to the next QIP_SNORG_SUBNET_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	Subnet organization name not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

Use the **`qip_freesnorgprof()`** routine to dispose of the returned profile.

qip_freesnorgprof

The **qip_freesnorgprof** routine frees the memory allocated in the **qip_getsnorgprof()** routine.

Prototype

```
int qip_freesnorgprof(QIP_SNORG_PROF *pSnorgProf);
```

Parameters

pSnorgProf	The QIP_SNORG_PROF structure allocated by the **qip_getsnorgprof** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getsnorgprof** routine. Freeing the memory directly might produce unpredictable results.

qip_setsnorgprof

The **`qip_setsnorgprof`** routine creates or modifies the specified subnet organization based on what is specified in **`pSnorgProf`**.

Prototype

```
int qip_setsnorgprof(QIP_SNORG_PROF *pSnorgProf);
```

Parameters

`pSnorgProf`	Pointer to the subnet organization profile to add or update. Refer to the **`qip_getsnorgprof`** routine for a description of the QIP_SNORG_PROF structure.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_PARMS`	Subnet organization name or other parameters in the Subnet Organization Profile are incorrectly specified. The **`qip_error_message_text`** routine can provide details on the problematic parameter.

Remarks

None.

qip_getsubnetaddr

The **`qip_getsubnetaddr`** routine retrieves the subnet address for the given address of an object.

Prototype

```
int qip_getsubnetaddr(char *szObjectAddress,
                         char *szSubnetAddress);
```

Parameters

`szObjectAddress`	The IP address of the object that you want to determine the subnet address.
`szSubnetAddress`	The returned Subnet address for the IP address that was passed. Make sure it is a buffer large enough to hold a string of size QIP_IPADDR_LEN; this routine does not allocate memory.

Return value

`QIP_OK`	Success without error.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_PARMS`	Unknown object address.

Remarks

None.

qip_getsubnetlst

The **qip_getsubnetlst** routine retrieves a list of subnets contained in the supplied entity as specified in the **iType** parameter. For example, if **iType** is QIP_TYPE_DOMAIN, it retrieves a list of subnets defined for the specified domain.

Prototype

```
int qip_getsubnetlst(int iType, char *szNameOrAddr,
                            QIP_SUBNET_LST **ppSubnetList);
```

Parameters

iType	Type of entity for which to retrieve the defined subnets. The valid types are QIP_TYPE_CORPORATION, QIP_TYPE_DOMAIN, QIP_TYPE_SUBNETORG, QIP_TYPE_NETWORK, and QIP_TYPE_OSPF.
szNameOrAddr	Name or IP address of the entity, depending on type. For example, if the type were QIP_TYPE_DOMAIN, the domain name would be specified here. This parameter is ignored for QIP_TYPE_CORPORATION (all subnets are retrieved).
ppSubnetList	Pointer to the anchor for the returned subnet list. The anchor will be a pointer to a QIP_SUBNET_LST structure. Refer to Table 1-36 for the values of the fields in the QIP_SUBNET_LST structure.

Table 1-36 QIP_SUBNET_LST structure fields

Field	Value
name	Subnet name
ipaddr	Subnet IP address
next	Pointer to the next QIP_SUBNET_LST element. NULL indicates end of list.

Return value

`QIP_OK`	Search without error. If an empty list is returned, **`SubnetList`** is NULL.
`QIP_ERR_PARMS`	The value specified for **`iType`** is invalid, or the **`szNameOrAddr`** parameter was not specified or specified incorrectly.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freesubnetlst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freesubnetlst

The **qip_freesubnetlst** routine frees the memory allocated in the **qip_getsubnetlst()** routine.

Prototype

```
int qip_freesubnetlst(QIP_SUBNET_LST *pSubnetList);
```

Parameters

pSubnetList	The QIP_SUBNET_LST anchor set by the **qip_getsubnetlst** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getsubnetlst** routine. Freeing the memory directly might produce unpredictable results.

qip_getsubnetprof

The **`qip_getsubnetprof`** routine retrieves the profile for the supplied subnet. The profile is an extensive data structure containing detailed information about the subnet.

Prototype

```
int qip_getsubnetprof(char *szAddress, QIP_SUBNET_PROF
                                 **ppSubnetProf);
```

Parameters

`szAddress`	String containing the IP address of the desired subnet.
`ppSsubnetProf`	Pointer to be filled in with the address of the subnet's profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-37 for the values of the fields in the Subnet Profile.

Table 1-37 Subnet Profile fields

Field	Value
subnet_addr	Subnet IP address
subnet_name	Subnet name
subnet_mask	Subnet mask
subnet_desc	Subnet description
network_name	Network name
interface_type	Interface type
floor	Location information
street	
city	
state	
zip	
country	
contact_lname	Last name
contact_fname	First name
contact_eaddr	E-mail address
contact_phone	Phone number

Field	Value
contact_pager	Pager number
appl_name	Subnet application name
domain_lst	Subnet's domain list
objsvr_lst	Subnet DNS server list pointer
objrtr_lst	Subnet router list pointer
objts_lst	Subnet time server list pointer
tftp_svr_name	Subnet's default TFTP server
check_use	Y or N
show_used	Y or N
hardware_type	Networking hardware. Refer to #defines for values.
dhcp_server	Subnet's DHCP server hostname
dhcp_template	DHCP template name
shared_network	The name of the shared network of which the subnet is a part.

Return value

`QIP_OK`	Subnet Profile found.
`QIP_ERR_PARMS`	Invalid (for example, poorly formed) subnet address.
`QIP_ERR_NOTFOUND`	Unknown subnet.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

- Use the **`qip_freesubnetprof()`** routine to dispose of the returned profile.
- There are three lists attached to the subnet profile. They list the subnet's DNS Servers, Time Servers, and Routers. Refer to the *qipapi.h* file for the format of these structures.

qip_freesubnetprof

The **`qip_freesubnetprof`** routine frees the memory allocated in the **`qip_getsubnetprof()`** routine.

Prototype

```
int qip_freesubnetprof(QIP_SUBNET_PROF *pSubnetProf);
```

Parameters

`pSubnetProf`	The QIP_SUBNET_PROF anchor set by the **`qip_getsubnetprof`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getsubnetprof`** routine. Freeing the memory directly might produce unpredictable results.

qip_setsubnetprof

The **`qip_setsubnetprof`** routine sets the profile for the specified subnet.

Prototype

```
int qip_setsubnetprof(QIP_SUBNET_PROF *pSubnetProf);
```

Parameters

`pSubnetProf`	A pointer to a QIP_SUBNET_PROF structure (most likely set by the **`qip_getsubnetprof`** routine).

Return value

`QIP_OK`	Subnet Profile updated.
`QIP_ERR_PARMS`	Invalid or poorly formed elements in the structure. Check the various IP addresses and masks for validity.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error occurred during update.

Remarks

To change any fields in a subnet profile, the calling routine should first retrieve the existing parameters via the **`qip_getsubnetprof`** routine, modify the desired field, and then update the profile via the **`qip_setsubnetprof`** routine. For example:

```
QIP_SUBNET_PROF *snProf;
int qiprc;
char *subnetaddr = “150.150.1.0”;

if ( (qiprc = qip_getsubnetprof(subnetaddr, &snProf)) < 0 ) {
        /* Bail out */
} else {
       /* Update description field */
       snProf->subnet_desc = “Was Boston, now Omaha”;
       if ( ( qiprc = qip_setsubnetprof(snProf) ) < 0 ) {
               fprintf(stderr, “Profile Update failed %d(%d)\n”,
                       qiprc, qip_errno());
       }
       qip_freesubnetprof(snProf);
}
```

qip_getudflst

The **`qip_getudflst`** routine retrieves the list of user defined fields (UDFs) for a particular entity type as defined in the **`iType`** parameter, and optionally retrieves the values for those fields when a particular instance of the entity type is specified.

Prototype

```
int qip_getudflst(int iType, int iFieldOrValue,
                      char *szNameOrAddr, QIP_UDF_LST
                      **ppUDFLst);
```

Parameters

`iType`	Entity type for which to retrieve the user defined fields. The valid types are QIP_TYPE_OBJECT, QIP_TYPE_SUBNET, QIP_TYPE_CONTACT (user), QIP_TYPE_PROV_POOL, and QIP_TYPE_PROV_BLOCK.
`iFieldOrValue`	Either QIP_UDF_FIELD, or QIP_UDF_VALUE, depending whether to retrieve just the field names for the specified type, or the name/value pairs for a specified entity. If the latter is specified, its name or address must be specified in the **`szNameOrAddr`** parameter.
`szNameOrAddr`	Name or address of the specific entity for which field values are desired (ignored if QIP_UDF_FIELD is specified).
`ppUDFLst`	Pointer to the anchor for the returned user defined field list. The anchor is a pointer to a QIP_UDF_LST structure. Refer to Table 1-38 for the values of the fields in the QIP_UDF_LST structure.

Table 1-38 QIP_UDF_LST structure fields

Field	Value
field	Field name
value	Field value
next	Pointer to the next QIP_UDF_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_PARMS`	**`Bad iType`** or **`iFieldOrValue`** parameter or name or address not specified when QIP_UDF_VALUE is specified.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeudflst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeudflst

The **`qip_freeudflst`** routine frees the memory allocated in the **`qip_getudflst()`** routine call.

Prototype

```
int qip_freeudflst(QIP_UDF_LST *pUDFLst);
```

Parameters

`pUDFLst`	The QIP_UDF_LST anchor set by the **`qip_getudflst`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_setudf

The **`qip_setudf`** routine sets the value of a specified user defined field for a specified entity.

Prototype

```
int qip_setudf(int iType, char *szNameOrAddr, char *szField,
                                    char *szValue);
```

Parameters

`iType`	Type of entity. Refer to the **`qip_getudflst`** routine for the supported types.
`szNameOrAddr`	Name or address of the specific entity for which the field value is to be set. For example, if the type were QIP_TYPE_OBJECT, this would be the IP address of the specific object. For users (QIP_TYPE_CONTACT), it would be the login name.
`szField`	Field name of the user defined field on the specific entity that is having its value set.
`szValue`	Actual value to set. A value of NULL or the empty string will cause deletion of the field value.

Return value

`QIP_OK`	Completed without error.
`QIP_ERR_PARMS`	Bad IP address, or bad type, or entity specified by **`szNameOrAddr`** not found.
`QIP_ERR_DB`	Database error encountered during processing. Refer to the **`qip_errno`** routine for more details.

Remarks

None.

qip_getuserprof

The **qip_getuserprof** routine retrieves the User Profile for the specified login name. The profile is an extensive data structure containing detailed information about the user.

Prototype

```
int qip_getuserprof(char *szLogin, QIP_USER_PROF
                    **ppUserProf);
```

Parameters

szLogin	String containing the login name.
ppUserProf	Pointer to be filled in with the address of the User Profile. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-39 for the values of the fields in the User Profile.

Table 1-39 User Profile fields

Field	Value
login_name	Login name
last_name	Contact information last name
first_name	Contact information first name
phone_num	Telephone number
email_addr	E-mail address
password	Password
pin	PIN
descr	Description
isp	Internet Service Provider (ISP)
authentication	Authentication
street1	Location information (1)
street2	Location information (2)
city	City
state	State
zip	Zip code
country	Country

Field	Value
activation_status	Activation status
group_lst	List of QIP_NAME_CHANGE_LST structures Refer to Table 1-40 for a description of the QIP_NAME_CHANGE_LST structures. List of groups of which user is a member is also used to add/delete groups via the **`qip_setuserprof`** routine and the **`change_flag`** member.

Table 1-40 QIP_NAME_CHANGE_LST structures

Structure	Value
name	Group name
change_flag	One of QIP_CHANGE_FLAG_
next	Pointer to the next QIP_NAME_CHANGE_LST element. NULL indicates end of list

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	Login name not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

Use the **`qip_freeuserprof()`** routine to dispose of the returned profile.

qip_freeuserprof

The **`qip_freeuserprof`** routine frees the memory allocated in the **`qip_getuserprof()`** routine call.

Prototype

```
int qip_freeuserprof(QIP_USER_PROF *pUserProf);
```

Parameters

`pUserProf`	The QIP_USER_PROF structure allocated by the **`qip_getuserprof`** routine.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getuserprof`** routine. Freeing the memory directly might produce unpredictable results.

qip_setuserprof

The **qip_setuserprof** routine creates or modifies the specified user, based on what is specified in **pUserProf**.

Prototype

```
int qip_setuserprof(QIP_USER_PROF *pUserProf);
```

Parameters

pUserProf	Pointer to the User Profile to add/update. Refer to the **qip_getuserprof** routine for a description of the QIP_USER_PROF structure. To add/delete groups via the group_lst member, set the change_flag member of each element to QIP_CHANGE_FLAG_ADD or QIP_CHANGE_FLAG_DEL as appropriate.

Return value

QIP_OK	Completed without error.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.
QIP_ERR_MEMORY	Memory allocation error during processing.
QIP_ERR_PARMS	Login name or other parameters in the profile are incorrectly specified. The **qip_error_message_text** routine can provide details on the problematic parameter.

Remarks

None.

qip_setusergroupprof

The **qip_setusergroupprof** routine creates or modifies the specified user group, based on what is specified in **pUserGroupProf**.

Prototype

```
int qip_setusergroupprof(QIP_USER_GROUP_PROF
                          *pUserGroupProf);
```

Parameters

pUserGroupProf	Pointer to the User Group Profile to add or update. Refer to Table 1-41 for a description of the User Group Profile structure. To set a contact on the group, you must specify the first and last name in the contact member.

Table 1-41 Fields in the User Group Profile structure

Field	Value
user_group_name	Group name
user_group_desc	Description
contact	Refer to the **qip_getcontactlst** routine for a description of the contact member. Use the first and last name to set a contact on a group. A contact is not created by this routine if one does not exist.

Return value

QIP_OK	Completed without error.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.
QIP_ERR_NOTFOUND	Specified contact not found.
QIP_ERR_PARMS	User group name not specified.

Remarks

None.

qip_getzoneopt

The **`qip_getzoneopt`** routine retrieves the zone options for the specified domain name.

Prototype

```
int qip_getzoneopt(char *szDomain, QIP_ZONE_OPTIONS_REC
                         **ppZoneOptions);
```

Parameters

`szDomain`	String containing the domain name.
`ppZoneOptions`	Pointer to be filled in with the address of the zone options record. The routine allocates the memory for the structure and fills in the fields. Refer to Table 1-42 for a description of the fields and values in the **`QIP_ZONE_OPTIONS_REC`** structure.

Table 1-42 QIP_ZONE_OPTIONS_REC structure fields

Field	Value
refresh_time	Domain refresh time (in seconds)
expire_time	Domain expire time (in seconds)
retry_time	Domain retry time (in seconds)
min_time	Domain Default TTL (in seconds)
neg_cache_ttl	Negative Cache TTL (in seconds)
w2000_zone_age_no_refresh	Windows 2000 DNS options – no-refresh-interval (in hours)
w2000_zone_age_refresh	Windows 2000 DNS Options – refresh-interval (in hours)
post_extension	Postfix extension (free text)
prefix_extension	Prefix extension (free text)
zone_mail	Zone email address
bind8_zone_allow_query	Any, None, localhost, localnets, Use List, Use Server Value
bind8_zone_allow_xfer	Any, None, localhost, localnets, Use List, Use Server Value
bind8_zone_allow_update	Any, None, localhost, loocalnets, Use List, Use Server Value
bind8_zone_check_name	Warn, Fail, Ignore, Use Server Value
bind8_zone_notify	No, Yes, Use Server Value
bind8_zone_block	Free text
bind9_zone_allow_query	Any, None, localhost, localnets, Use List, Use Server Value

Field	Value
bind9_zone_allow_notify	Any, None, localhost, localnets, Use List, Use Server Value
bind9_zone_allow_xfer	Any, None, localhost, localnets, Use List, Use Server Value)
bind9_zone_allow_update	Any, None, localhost, localnets, Use List, Use Server Value
bind9_zone_notify	No, Yes, Explicit, Use Server Value
bind9_zone_block	Free text
lucent3_zone_edup	True, False
lucent3_zone_allow_query	Any, None, localhost, localnets, Use List, Use Server Value
lucent3_zone_allow_xfer	Any, None, localhost, localnets, Use List, Use Server Value
lucent3_zone_allow_update	Any, None, localhost, localnets, Use List, Use Server Value)
lucent3_zone_allow_notify	Not used
lucent3_zone_check_name	Warn, Fail, Ignore, Use Server Value
lucent3_zone_notify	No, Yes, Use Server Value
lucent3_zone_block	Free text
lucent4_zone_edup	True, False
lucent4_zone_allow_notify	Any, None, localhost, localnets, Use List, Use Server Value
lucent4_zone_allow_query	Any, None, localhost, localnets, Use List, Use Server Value
lucent4_zone_allow_xfer	Any, None, localhost, localnets, Use List, Use Server Value
lucent4_zone_allow_update	Any, None, localhost, localnets, Use List, Use Server Value
lucent4_zone_notify	No, Yes, Explicit, Use Server Value
lucent4_zone_block	Free text
w2000_zone_aging	True, False, Use Server Value
w2000_zone_allow_xfer	Any, None, Name Servers Only, Use List, Use Server Value
w2000_zone_allow_update	Yes, No, Use Server Value
w2000_zone_notify	Yes, No, Use Server Value
w2000_zone_option	Free text

Return value

`QIP_OK`	Profile found.
`QIP_ERR_PARMS`	Domain not specified.
`QIP_ERR_MEMORY`	Memory allocation error during processing.
`QIP_ERR_DB`	Database error during processing.

Remarks

Use the **`qip_freezoneopt()`** routine to dispose of the returned profile.

qip_freezoneopt

The **qip_freezoneopt** routine frees the memory allocated in the **qip_getzoneopt()** routine call.

Prototype

```
int qip_freezoneopt(QIP_ZONE_OPTIONS_REC *pZoneOptions);
```

Parameters

pZoneOptions	The QIP_ZONE_OPTIONS_REC structure allocated by the **qip_getzoneopt** routine.

Return value

QIP_OK	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **qip_getzoneopt** routine. Freeing the memory directly might produce unpredictable results.

qip_setzoneopt

The **qip_setzoneopt** routine modifies the zone options for the specified domain based on what is specified in **pZoneOptions**.

Prototype

```
int qip_setzoneopt(char *szDomain, QIP_ZONE_OPTIONS_REC
                           *pZoneOptions);
```

Parameters

szDomain	String containing the domain name.
pZoneOptions	Pointer to zone options record to use to update the domain. Refer to "qip_getzoneopt" (p. 1-134) for a description of the QIP_ZONE_OPTIONS_REC structure.

Return value

QIP_OK	Completed without error.
QIP_ERR_DB	Database error encountered during processing. Refer to the **qip_errno** routine for more details.
QIP_ERR_PARMS	Domain name not specified or error in the zone options.

Remarks

None.

qip_freeaddresslst

The **`qip_freeaddresslst`** routine frees the memory allocated by any API call that creates a QIP_ADDRESS_LST.

Prototype

```
int qip_freeaddresslst(QIP_ADDRESS_LST *pAddressList);
```

Parameters

`pAddressList`	The QIP_ADDRESS_LST created by another routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_freenamechangelst

The **`qip_freenamechangelst`** routine frees the memory allocated by any routine call that creates a QIP_NAME_CHANGE_LST.

Prototype

```
int qip_freenamechangelst(QIP_NAME_CHANGE_LST
                                  *pNameChangeList);
```

Parameters

`pNameChangeList`	The QIP_NAME_CHANGE_LST created by another routine.

Return value

`QIP_OK`	Memory freed.

Remarks

None.

qip_getuserlst

The **`qip_getuserlst`** routine retrieves a list of users, which are associated with objects in the supplied subnet as specified in the **`szIpAddr`** parameter.

Prototype

```
int qip_getuserlst(char *pszIpAddr, QIP_USER_LST
                             **ppUserList, int* records)
```

Parameters

`pszIpAddr`	IP address of the subnet you want to retrieve the users.
`ppUserList`	Pointer to the anchor for the returned user list. The anchor will be a pointer to a QIP_USER_LST structure. Refer to Table 1-43 for the values of the fields in the QIP_USER_LST structure.
`records`	Pointer to the number of user records retrieved.

Table 1-43 QIP_USER_LST structure fields

Field	Value
login_name	Login name of the user
last_name	Last name of the user
first_name	First name of the user
objIpLst*	Pointer to the list of object IP addresses that this user is associated with that is contained in the structure QIP_ADDRESS_LST
next	Pointer to the next QIP_USER_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Search without error. An empty list can be returned, in which case User List will be NULL. The **Records** field will be 0.
`QIP_DB_NOT_FOUND`	The value specified for subnet is not found in the data server.
`QIP_INVALIDIP`	The value specified for subnet is not a valid IP format.
`QIP_ERR_PARMS`	Invalid parameter passed during the call to routine.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeuserlst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeUserLst

The **`qip_freeUserLst`** routine frees the memory allocated in the **`qip_getuseriplst()`** and **`qip_getuserlst()`** routines.

Prototype

```
void qip_freeUserLst(QIP_USER_LST *pUserList);
```

Parameters

`pUserList`	The QIP_USER_LST anchor set by the **`qip_getuseriplst`** and the **`qip_getuserlst`** routines.

Return value

None.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getuseriplst`** and the **`qip_getuserlst`** routines. Freeing the memory directly might produce unpredictable results.

qip_getuseriplst

The **`qip_getuseriplst`** routine retrieves a list of object IP addresses, which are associated with the user in the supplied **`login_name`** parameter or the **`last_name`** and **`first_name`** parameters.

Prototype

```
int qip_getuseriplst(char *login_name, char *last_name,
                     char *first_name, QIP_USER_LST **ppIpList)
```

Parameters

`login_name`	The login name of the user you want to retrieve the object IP addresses.
`last_name`	The last name of the user you want to retrieve the object IP addresses (not required if login_name is supplied).
`first_name`	The optional first name of the user you want to retrieve the object IP addresses (must be used with the last name, but not required if login_name is supplied).
`ppIpLst`	The anchor for the returned IP list. The anchor is a pointer to a QIP_USER_LST structure. Refer to Table 1-44 describes the fields and values in the QIP_USER_LST structure.

Table 1-44 QIP_USER_LST structure fields

Field	Value
login_name	Login name of the user
last_name	Last name of the user
first_name	First name of the user
objIpLst*	Pointer to the list of object IP addresses that this user is associated with that is contained in the structure QIP_ADDRESS_LST
next	Pointer to the next QIP_USER_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Search without error. If an empty list is returned, **`UserList`** is NULL**.**
`QIP_DB_NOT_FOUND`	The user specified for **`login_name`** or **`last_name`** a **`first_name`** is not found in the data server.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeuserlst`** routine. Freeing the memory directly might produce unpredictable results.

qip_getusedobjlst

The **`qip_getusedobjlst`** routine retrieves a list of IP addresses within a specified subnet that are designated as used. The subnet is specified in the **`pszIpAddr`** parameter.

Prototype

int qip_getusedobjlst(char *pszIpAddr, QIP_IP_LST **ppIpLst)

Parameters

`pszIpAddr`	The subnet address from which you want to retrieve the used object IP address.
`ppIpLst`	The anchor for the returned IP list. The anchor will be a pointer to a QIP_IP_LST structure. Refer to Table 1-45 for the values of the fields in the QIP_IP_LST structure.

Table 1-45 QIP_IP_LST structure fields

Field	Value
objIp	The IP address of an object
ipAttrib	The attribute of the IP address. In this routine, it is the subnet address that the object is on.
next	Pointer to the next QIP_IP_LST element. NULL indicates the end of list.

Return value

`QIP_OK`	Search without error. An empty list can be returned, in which **`UserList`** will be NULL.
`QIP_DB_NOT_FOUND`	The value specified for subnet is not found in the data server.
`QIP_INVALIDIP`	The value specified for subnet is not a valid IP format
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.
`QIP_ERR_PARMS`	Invalid parameter passed during the call to routine.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeIpLst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeIpLst

The **`qip_freeIpLst`** routine frees the memory allocated in the **`qip_getusedobjlst()`** routine.

Prototype

```
void qip_freeIpLst(QIP_IP_LST *ppIpList);
```

Parameters

`ppIpList`	The QIP_IP_LST anchor set by the **`qip_getusedobjlst`** routine.

Return value

None.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getusedobjlst`** routine. Freeing the memory directly might produce unpredictable results.

qip_getaddrrangelst

The **`qip_getaddrrangelst`** routine retrieves a list of defined address ranges on a given network.

Prototype

```
qip_getaddrrangelst(char *pszAddrType, QIP_ADDRRANGE_LST
                               **ppList)
```

Parameters

`pszAddrType`	The address range type defined within subnet or network.
`ppList`	Pointer to the anchor for the returned address range list. The anchor will point to QIP_ADDRRANGE_LST structure. Refer to Table 1-46 for the values of the fields in the QIP_ADDRRANGE_LST structure.

Table 1-46 QIP_ADDRRANGE_LST structure fields

Field	Value
Network or subnet IP address	The IP address of the network or subnet in which the address/objects reside
First address	First IP address in the address range
Last address	Last IP address in the address range
Next*	Pointer to the next record

Return value

`QIP_OK`	Search without error. If an empty list is returned, **`UserList`** is NULL.
`QIP_DB_NOT_FOUND`	The value specified for subnet is not found in the data server.
`QIP_INVALIDIP`	The value specified for subnet is not a valid IP format.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine call.
`QIP_ERR_PARMS`	Invalid parameter passed during the routine call.

Remarks

This routine returns memory that ***must*** be freed by a call to the **`qip_freeaddrrangeLst`** routine. Freeing the memory directly might produce unpredictable results.

qip_freeaddrrangelst

The **`qip_freeaddrrangelst`** routine frees the memory allocated in the **`qip_getaddrrangelst`**() routine.

Prototype

```
int qip_freeaddrangelst(QIP_ADDRRANGE_LST *paddrrange);
```

Parameters

`paddrrange`	The QIP_ADDRRANGE_LST anchor set by **`qip_getaddrrangelst`**.

Return value

`QIP_OK`	Memory freed.

Remarks

This routine ***must*** be called to free memory returned by the **`qip_getaddrrangelst`** routine. Freeing the memory directly might produce unpredictable results.

qip_gettemplate

The **`qip_gettemplate`** routine retrieves information of the DHCP Template of the supplied **`pszTemplateName`** parameter.

Prototype

```
int qip_gettemplate(char *TemplateName,
                     char **&dataArray,  long& lines)
```

Parameters

`TemplateName`	The name of the DHCP Template you to retrieve the information.
`dataArray`	The address of the char array where the data will be placed.
`lines`	The number of rows in the data array.

Return value

`QIP_OK`	Search without error.
`QIP_DB_NOT_FOUND`	The user specified for template name is not found in the data server.
`QIP_ERR_DB`	Database error encountered during search. Refer to the **`qip_errno`** routine for more details.
`QIP_ERR_MEMORY`	Memory allocation error during the routine.

Remarks

None.

VitalQIP API Object Reference

Purpose

This section describes the object-oriented calls in the VitalQIP API Toolkit.

The following objects are available as part of the API Toolkit and are documented in this section:

- APIAdmin: Used to get and set administrator profiles and to retrieve lists of administrators.
- APILogin: Used to connect and disconnect from the database.
- APIException: Exception class. When an API object generates an exception this specifies the type.
- APIStringList: Utility class to handle lists of strings.
- APINameValue: Utility class to handle a name/value pair.
- APINameValueList; Utility class to handle a list of name/value pairs.
- APIManagedOrg: Used to get and set administrator permissions to specific organizations.
- APIMRItem: Used to manage elements of the managed list contains within an APIManagedOrg object.
- APINamingPolicy: Used to manage naming policies for organizations.
- APIObjectClass: Used to manage object classes.

APIAdmin

Purpose

This class of object gets and sets administrator profiles and retrieves lists of administrators.

Class definition

```
class APIAdmin
```

Private members

```
QAPIAdmin*m_Admin;// pointer to internal QIP admin object
QAPIContact *m_Contact;// pointer to internal QIP contact object
QSIString*m_XML;// string buffer for XML output
```

Private methods

```
  QAPIAdmin *GetQAPIObject();// for QIP internal use only
  void SetQAPIObject(QAPIAdmin *obj);// for QIP internal use only

  // determine if the contact already exists.  Throw an exception if the
  contact
  // exists and we are attempting to create or if the contact does not exist
  and
  // we are not attempting to create
  void CheckContact(bool new_contact);
```

Public methods

```
  // default constructor - allocates memory for m_Admin and m_Contact objects
APIAdmin ();
// destructor - frees memoroy to m_Admin and m_Contact objects
  virtual ~APIAdmin ();

  // get methods
  const char * GetName();
  const char * GetType();
  const char * GetFirstName() const;
  const char * GetLastName() const;
  const char * GetEmail() const;
  const char * GetTelephone() const;
  const char * GetPager() const;
  const char * GetBusUnit() const;
  const char * GetPrtr() const;
```

```
   // set methods
   void SetName(const char *name);
   void SetType(const APILogin& login, const char *name);
   void SetPassword(const char *pwd);
   void SetFirstName(const char *name);
   void SetLastName(const char *name);
   void SetEmail(const char *email);
   void SetTelephone(const char *phone);
   void SetPager(const char *pager);
   void SetBusUnit(const char *bus_unit);
   void SetPrtr(const char *prtr);

   // retrieve a list of the default system privileges.  Each member of list
   may also have
// a listof children.
   APINameValue *GetSystemPrivs(const APILogin &login);
   // set the value of a specific privilege.  The privilege list for the
   administrator
   // will be traversed to find the privilege specified by the name and the
   value
// will be set.
   void SetSystemPrivilege(const APILogin& login, const char *name, const
   char *value);

   // add a managed organization to the administrator based on the
   organization name
   void AddManagedOrg(const APILogin& login, const char *org_name);
   // add a managed organization to the administrator based on the
   APIManagedOrg object
   void AddManagedOrg(const APILogin& login, APIManagedOrg *org);
   // remove the managed organization from the adminstrator
   void DeleteManagedOrg(const APILogin& login, const char *org_name);
   // retrieve a list of managed organizations assigned to the administrator
   APIManagedOrg *GetManagedOrgs(const APILogin &login);

   // standard database functions to add, modify, get, and delete the admin
   // For the Add method the new_contact parameter specifies whether the
   contact
// information should be used to create a new contact within QIP
   void Add(const APILogin& login, bool new_contact);
   void Load(const APILogin& login);
   void Modify(const APILogin& login);
   void Delete(const APILogin& login);

   // retrieve a list of administrators
// must call DestroyList passing original pointer returned to release
memory
static APIAdmin *LoadList(const APILogin& login);
static void DestroyList(APIAdmin *top);
// returns character string which contains the admin profile in an XML
format
const char *GetXML(const APILogin& login);
```

Friend classes

```
friend class APIManagedOrg;
```

Example

```
// construct the login object….this will connect to the database
APILogin *myLogin = new APILogin("127.0.0.1", "QIPSYBASE",
"qipman", "qipman", "VitalQIP Organization");
// construct the object
APIAdmin *myAdmin = new APIAdmin;
// set the administrator's name
myAdmin ->SetName("my admins name");
// load the administrator from the database
myAdmin ->Load(*myLogin);
// use get and set methods to retrieve or modify the properties
of the administrator

// save the administrator to the database
myAdmin ->Modify(*myLogin);
// delete allocated memory
delete myAdmin;
// destroy the login object….this will disconnect from the
database
delete myLogin;
```

APIManagedOrg

Purpose

This class gets and sets administrator permissions to specific organiztions.

Class definition

`class APIManagedOrg`

Private members

```
QAPIManagedOrganization*m_Org;// pointer to internal QIP
   organization object
QSIString*m_XML;// string buffer for XML output
```

Private methods

```
QAPIManagedOrganization *GetQAPIObject();// for QIP internal
use only
void SetQAPIObject(QAPIManagedOrganization *obj);// for QIP
internal use only
```

Public methods

```
   // default constructor - allocates memory for m_Org
APIManagedOrg ();

// destructor - frees memory to m_Org
virtual ~APIManagedOrg ();

   // get methods
   const char * GetName();
   const char * GetManagedType();
   bool GetReadOnly();

   // set methods
   void SetName(const char *name);
   void SetManagedType(const char *type);
   void SetReadOnly(bool readonly);

   // retrieve a list of the default system privileges.  Each member of list
   may also have
// a list of children.
   APINameValue *GetSystemPrivs(const APILogin &login);
// set the value of a specific privilege for a specific adminstrator.  The
// privilege list for the administrator will be
// traversed to find the privilege specified by the name and the value will
   be set.
```

```
void SetSystemPrivilege(const APILogin& login, APIAdmin *admin, const char
   *name,
const char *value);

  // retrieve a list of items in the managed list (as APIMRItem objects)
  APIMRItem *_GetManagedRange(const APILogin &login);
  // add a new item to the managed list
  void AddManagedItem(const APILogin& login, APIMRItem *item);

  // retrieve a list of role names in the managed list
  APIStringList* GetRoles(const APILogin& login);
  // add a new role to the managed list
  void AddRole(const APILogin& login, APIAdmin *admin, const char
   *role_name);

  // set the QIP Organization that this managed organization is based on
  void SetBaseOrg(APIOrg *obj);

  // load and/or modify the managed organization.  Managed organizations are
   not deleted.
  // They can be removed from a specific administrator via the APIAdmin
   object
  void Load(const APILogin& login, APIAdmin *admin);
  void Modify(const APILogin& logni, APIAdmin *admin);

  // returns character string which contains the admin profile in an XML
   format
  const char *GetXML(const APILogin& login);
```

Friend class

```
friend class APIAdmin;
```

Example

```
  // this example adds an organization to an administrator's
  managed list of
// organizations
// construct the login object….this will connect to the database
  APILogin *myLogin = new APILogin("127.0.0.1", "QIPSYBASE",
   "qipman", "qipman", "VitalQIP Organization");
  // construct an admin object
  APIAdmin *myAdmin = new APIAdmin;
  // set the administrator's name
  myAdmin ->SetName("my admins name");
  // load the administrator from the database
  myAdmin ->Load(*myLogin);
```

```
// create an org object, set the name of the org object
APIOrg *tmp_org = new APIOrg;
tmp_org->SetName("VitalQIP Organization");
// set this to be the active organization
myLogin->SetOrganization("VitalQIP Organization");
// load the organization
tmp_org->Load(myLogin);
// construct the managed organization object
APIManagedOrg *man_org = new APIManagedOrg;
// set the base organization of the managed organization object
m_ManOrg->SetBaseOrg(tmp_org);
// add the managed organization to the administrator
m_Admin->AddManagedOrg(*myLogin, man_org);
// save the administrator to the database
myAdmin ->Modify(*myLogin);
// destroy allocated memory
delete myAdmin;
delete tmp_org;
delete man_org;
// destroy the login object….this will disconnect from the
 database
delete myLogin;
```

APIMRItem

Purpose

This class manages elements of the managed list contained within an APIManagedOrg object.

Class definition

```
class APIMRItem
```

Private members

```
QAPIMRItem*m_Item;// pointer to internal QIP managed range object
```

Private methods

```
  QAPIMRItem *GetQAPIObject();// for QIP internal use only
  void SetQAPIObject(QAPIMRItem *obj);// for QIP internal use
  only
```

Public methods

```
  // default constructor - allocates memory for m_Item
APIMRItem ();

// destructor - frees memory to m_Org
virtual ~APIMRItem ();

  // get methods
  const char *GetName();
  const char *GetStartAddress();
const char *GetEndAddress();
const char *GetType();
bool GetReadOnly();

  // set methods
  void SetName(const char *name);
  void SetStartAddress(const char *addr);
  void SetEndAddress(const char *addr);
  void SetType(const char *type);
  void SetServerType(const char *type);
void SetReadOnly(bool readonly);
```

Friend class

```
  friend class APIManagedOrg;
```

APIOrg

Purpose

This class connects and disconnects from other databases.

Note: This is not a fully functional organization class. The only components of this class that have been implemented are those required to support the APIManagedOrganization

Class definition

```
class APIOrg
```

Private members

```
QAPIOrganization*m_Org;// pointer to internal QIP managed range
   object
```

Private methods

```
   QAPIOrganization *GetQAPIObject();// for QIP internal use only
   void SetQAPIObject(QAPIOrganization *obj);// for QIP internal
   use only
```

Public methods

```
   // default constructor - allocates memory for m_Org
APIOrg ();

// destructor - frees memory to m_Org
virtual ~APIOrg ();

   // set methods
void SetName(const char *name);

void Load(const APILogin& login);
```

Friend class

```
   friend class APIManagedOrg;
```

APINamingPolicy

Purpose

This class manages naming policies for organizations.

Class definition

```
class APINamingPolicy
```

Private members

```
  QAPINamingPolicy *m_NamingPolicy;// pointer to internal QIP
   object
```

Private methods

```
  QAPINamingPolicy *GetQAPIObject();// for QIP internal use only
void SetQAPIObject();// for QIP internal use only
```

Public methods

```
//default constructor - allocates memory for m_ NamingPolicy
   object
APINamingPolicy();

// destructor - free memory allocates to m_NamingPolicy object
~APINamingPolicy()

// Set methods
void SetObjectClass(const char *obj_class);
void SetPrefix(const char *prefix)
void SetSuffix(const char *suffix)
void SetLength(int length)
void SetUseDefault(bool def);

// Get methods
const char *GetObjectClass();
const char *GetPrefix();
const char *GetSuffix();
int GetLength();
bool GetUseDefault();

// Database methods
//
// modify the naming policy
void Modify(const APILogin& login);

// load the naming policy from the database.
```

```
void Load(const APILogin& login);

// load a list of object classes from the database
static APINamingPolicy *LoadList(const APILogin& login);
```

APIObjectClass

Purpose

This class supports object classes within the API Toolkit.

Class definition

`class APIObjectClass`

Private members

```
  QAPIObjectClass *m_ObjClass;// pointer to internal QIP object
QSIString*m_XML;// string buffer for XML output
```

Private methods

```
  QAPIObjectClass *GetQAPIObject();// for QIP internal use only
void SetQAPIObject();// for QIP internal use only
```

Public methods

```
//default constructor - allocates memory for m_ObjClass object
APIObjectClass();

// destructor - free memory allocates to m_ObjClass object
~APIObjectClass();

// Set methods
void SetName(const char *obj_class_name);
void SetDeviceType(const APILogin& login, const char
   *device_type);
void SetPrefix(const char *prefix);
void SetSuffix(const char *suffix);
void SetLength(int length);

// Get methods
const char *GetName();
const char *GetDeviceType();
const char *GetPrefix();
const char *GetSuffix();
const char *GetLength();

// Database methods
//
// add the new object class to the database
void Add(const APILogin& login);
// modify the object class
void Modify(const APILogin& login);
```

```
// delete the object class from the database.
void Delete(const APILogin& login);

// load the object class from the database.
void Load(const APILogin& login);

// load a list of object classes from the database
static APIObjectClass *LoadList(const APILogin& login);

  // returns objects classes is an XML format
  const char * GetXML(const APILogin& login);
  const char * GetXMLList(const APILogin &login);
```

APILogin

Purpose

This class connects to the database when an object is created and disconnects from the database when the object is destroyed.

Class definition

```
Class APILogin
```

Private members

```
  QAPILogin* m_Login;// pointer to internal QIP object
```

Public methods

```
  // constructors
  APILogin(const char* loginServer,
        const char* dbServer,
        const char *uname,
        const char *pword,
        const char* szOrg);

  // destructor
  virtual ~APILogin();

  // get pointer to internal QIP object
  const QAPILogin& GetLogin() const;

  // set the active organization
void SetOrganization(const char *org);

  // return the name of the logged on administrator
  const char *GetName();
```

APIException

Purpose

Exception class. When an API object generates an exception this specifies the type.

Class definition

`Class APIException`

Private members

```
QSIException *exception;// pointer to internal QIP object
```

Public methods

```
  // constructor
  APIException(QSIException *ex);
  //destructor
    virtual ~APIException();
  // returns error code
  int GetErrorCode();
  // return text associated with the error
  const char *GetText();
};
```

APIStringList

Purpose

Utility class for managing lists of strings.

Class definition

```
Class APIStringList
```

Private members

```
    QSIStringList *list;// pointer to internal QIP object
```

Public methods

```
  // constructor
    APIStringList();
  // destructor
    ~APIStringList();
  // return the item in the list of the 'i' index (starting from
  0)
  const char *operator[](int i) const;
  // return number of objects in the list
    int Count() const;
  // add an item to the list
   void AddItem(const char *ptr);
};
```

APIBase

Purpose

Base object to all API objects. This object should never be created directly by a program. It is for internal API use.

Class definition

```
Class APIBase
```

Private members

```
  APIBase *prev;// pointer to previous object
  APIBase *next;// pointer to next object
```

Public methods

```
  // Constructor
  APIBase();
  // Destructor
  virtual ~APIBase(){};

  // Set previous object
  void SetPrev(APIBase *obj);
  // Set next object
  void SetNext(APIBase *obj);

  // return previous object
  APIBase *Prev();
  // return next object
  APIBase *Next();

  // static method to destroy any list of API objects
  static void DestroyList(APIBase *top);
};
```

APINameValue

Purpose

Utility class for managing name/value pairs.

Class definition

`Class APINameValue`

Private members

```
QSIString *name;// name
QSIString *value;// value
APINameValue *children;// list of APINameValue that are
 'children' of this
         // object.  Used when specific properties of an
         // API object are dependant on the value of another
         // property
```

Public method

```
  // constructor
  APINameValue();
  //destructor
  ~APINameValue();

  // get methods
  const char *GetName();
  const char *GetValue();
  APINameValue *GetChildren();

  // set methods
  void SetName(const char *);
  void SetValue(const char *);

  // used by QIP to convert QIP object lists to APINameValues
  static APINameValue* ConvertParmList(const APILogin& login,
   QAPIList *parm_list);
  static APINameValue* ConvertSvrList(const APILogin& login,
   QAPIList *dns_list);
};
```

APINameValueList

Purpose

Utility class for managing lists of name/value pairs.

Class definition

`Class APINameValueList`

Private members

```
  APINameValue *first;// first item in a list
APINameValue *tmp;// internal use
```

Public method

```
// constructor
APINameValueList();

// add an item to the list
void Append(APINameValue* api_obj);
// get the first item in the list
APINameValue* GetFirst();
```

2 Lucent DHCP API

Overview

Purpose

This chapter describes the Lucent DHCP API that controls or audits the behavior of the Lucent DHCP server.

This information presents the following topics.

Introduction to the Lucent DHCP API

The Lucent DHCP API controls or audits the behavior of the Lucent DHCP server. The Lucent DHCP server is controlled or audited by using protocol messages, which provide integration points for attaching code to any major point within the Lucent DHCP server. Lucent DHCP server operations can then be customized without disturbing the server core. In order to implement this attached code, customers or OEM vendors must develop, build, and install the callout library (either a DLL or a shared object) as defined in Table 2-1, "Platforms associated with the Lucent DHCP shared library/DLL" (p. 2-6).

Customers and OEM vendors of the Lucent DHCP server can use the integration points to extend the capability of the service. These integration points provide features, such as Virtual Local Area Network (VLAN) support, user authentication integration, and auditing capabilities.

The Lucent DHCP API also offers the following capabilities:

- Notification of an external process whenever key events are executed within the DHCP protocol
- Request and receive authentication information from an external source
- Request and receive VLAN configuration information, such as what subnet address is offered to the client
- Use user-defined naming schemas for auditing and tracking clients.

Alcatel-Lucent ***strongly*** recommends you familiarize yourself with Request for Comments (RFC) 2131 and RFC 2132 before using the Lucent DHCP API.

Note: When using Lucent DHCP 5.5 or Lucent DHCP 5.6, multiple API callout library usage is supported through the use of the callout policies described in "Multiple callout library feature" (p. 2-12). This allows the interoperability of VitalQIP add-on products such as LDRM, Registration Manager, and custom API callouts.

When using Lucent DHCP 5.4, the DHCP API Toolkit cannot be used with the Registration Manager on the same server. The DHCP API will overwrite the Registration Manager shared object or DLL file. However, they can be used together when the DHCP API Toolkit and Registration Manager reside on separate servers.

Integration points

Protocol messages are used as integration points, which can change the behavior of the Lucent DHCP server. Code attached to the integration points determines what information is recorded, the address offered to a client, what subnet the address is offered from, and if a particular address is offered from a subnet. The following is a list of protocol messages that are used as integration points:

- After a DHCP-DISCOVER message is received from the DHCP Client and before the offer is sent, the client-provided hostname, domain name, requested class, user class, vendor-specific information, and lease time option values, as well as the address of the incoming packet record structure (see "Structure definitions" (p. 2-9)) are sent to the API. The API returns an authorization flag (authorized or not authorized) to the DHCP Server and optionally can change any of the client-provided option values that were sent to the API. The API can also provide a list of suggested subnets, from which the server can attempt to provide an address. Each address in the list must be 15 characters in length, separated by a single space or comma, and in the following dotted decimal format: xxx.xxx.xxx.xxx.
- After a DHCP-REQUEST message is received from the DHCP Client, the client-provided hostname, domain name, requested class, user class, vendor-specific information, and lease time option values, as well as the address of the incoming packet record structure (see "Structure definitions" (p. 2-9)) are sent to the API.
- After a DHCP-OFFER message is sent from the Lucent DHCP server to the DHCP Client, the outgoing packet is sent to the API in the parsed format of the packet record structure. This structure contains the MAC address, as well as the DHCP options.
- After a DHCP-ACK message is sent from the Lucent DHCP server to the DHCP Client, the outgoing packet is sent to the API in the parsed format of the packet record structure. This structure contains the MAC address, as well as the DHCP options.
- After a DHCP-NACK message is sent from the Lucent DHCP server to the DHCP Client, the outgoing packet is sent to the API in the parsed format of the packet record structure. This structure contains the MAC address, as well as the DHCP options.
- After a BOOTP-REQUEST message is received from the Bootp Client, the same parameters are sent to and returned from the API, as described for the DHCP-DISCOVER message.
- After a BOOTP-REPLY message is sent from the Lucent DHCP server to the Bootp client, the outgoing packet is sent to the API in the parsed format off the packet record structure. This structure contains the MAC address assigned to the client.
- After a DHCP-RELEASE is received from the DHCP client and processed by the server, the client's IP address, MAC address, hostname, domain name, and option 82 data (if available) are sent to the API.

Lucent DHCP 5.6 only

- After a DHCP-LEASEQUERY message is received by the DHCP server and the giaddr is confirmed to be non-zero, as required by RFC 4388, the address of the incoming packet record structure (see "Structure definitions" (p. 2-9)) is sent to the API. The API returns an authorization flag to the DHCP Server allowing for the specification of continued processing, when the returned value is true. When the returned authorization flag value is false, further processing of the leasequery message is precluded.
- After a DHCP-LEASEACTIVE message is sent by the DHCP to the address provided in the giaddr field of the DHCP-LEASEQUERY message, the outgoing packet is sent to the API in the parsed format of the packet record structure. This structure contains the queried IP address in the ciAddr field, the MAC address in the chAddr field, the lease time remaining (option 51) in the option array, and if requested in the lease query message and if available, the vendor class (option 60), relay agent information (option 82) and client last transaction time (option 91) options. If applicable, the associated IP list (option 92) is provided (see**RFC 4388**).
- After a DHCP-LEASEUNASSIGNED message is sent by the DHCP to the address provided in the giaddr field of the DHCP-LEASEQUERY message, the outgoing packet is sent to the API in the parsed format of the packet record structure. This structure contains the queried IP address in the ciAddr field.
- After a DHCP-LEASEUNKNOWN message is sent by the DHCP to the address provided in the giaddr field of the DHCP-LEASEQUERY message, the outgoing packet is sent to the API in the parsed format of the packet record structure.

Lucent DHCP shared library/DLL

Lucent DHCP 5.4 Server only

At startup, the Lucent DHCP server determines if the Lucent DHCP API DLL or shared object is present. The Lucent DHCP API DLL or shared library must be present for the Lucent DHCP server to make the calls to the integration points.

Install the custom library that you develop and build, as described in Table 2-1.

Table 2-1 Platforms associated with the Lucent DHCP shared library/DLL

Platform	Library Name	Directory
Microsoft Windows	**`Qdhcp_api.dll`**	*%QIPHOME%\lib*
Linux	**`libQdhcp_api.so`**	*$QIPHOME/usr/lib*
Solaris	**`libQdhcp_api.so`**	*$QIPHOME/usr/lib*

The Lucent DHCP API only works with the Lucent DHCP server.

Routine policies

Lucent DHCP 5.4 Server only

Most called routines are policy driven. The routine policies are defined in the Lucent DHCP policy file (*dhcpd.pcy*) which is managed throught Lucent VitalQIP. For information on Lucent VitalQIP, refer to the *VitalQIP Administrator's Guide* for more information. If a policy is on, the server calls out to the library for that particular interface. If the policy is off, or not present in the policy file, a callout attempt is not made by the server. Each routine that has an associated policy is listed in Table 2-2.

Table 2-2 Policies with associated routines

Policy	Routine	Data type	Default values
DiscoverAPICallout	discoverAPI	Boolean	0 (False)
OfferAPICallout	offerAPI	Boolean	0 (False)
RequestAPICallout	requestAPI	Boolean	0 (False)
Request2APICallout	request2API	Boolean	0 (False)
AckAPICallout	ackAPI	Boolean	0 (False)
NackAPICallout	nackAPI	Boolean	0 (False)
BootpRequestAPICallout	bootpRequestAPI	Boolean	0 (False)
BootpReplyAPICallout	bootpReplyAPI	Boolean	0 (False)
DepletedSubnetAPICallout	depletedSubnetAPI	Boolean	0 (False)
IgnoredPacketAPICallout	ignoredPacketAPI	Boolean	0 (False)
PacketReceiptAPICallout	packetReceiptAPI	Boolean	0 (False)
ReleaseAPICallout	releaseAPI	Boolean	0 (False)
UseAPICallout2	Must be set to **1** (on) for use with this API.	Boolean	0 (False)

Note: The **UseAPICallout2** policy in the Lucent DHCP server's *dchpd.pcy* file ***must*** be set in order to make use of these new routine prototypes. If this policy is not set, the server will accept the previous version of the documented routine.

The policy associated with the routine must be enabled for the routine to be called. If the routine policy is off or not present in the *dhcpd.pcy* file, no routine call is made.

All routine calls originating from the Lucent DHCP server are blocked. Blocked calls wait for a response from the routine call before processing continues. In order for routines originating from the Lucent DHCP not to be blocked, the process must be forked and/or a thread provided. Control can then be returned to the Lucent DHCP server.

Structure definitions

The DHCP packet format is defined in **RFC 2131**. It is recommended that the routine callouts have access to the entire packet as received and sent by the DHCP server. To simplify the routine's processing of the raw packet, the server passes a structure that represents all fields of the packet. This is similar to the internal representation of the packet within the DHCP server after it has been received and parsed. A pointer to this structure is passed to the routine callout. The structure and its contents are read-only. An example follows.

```
struct optionRecord {
  int fromClient;/* flag - client sent option (1) or API callout
   created option (0) */
  intlen;
  unsigned char *data;
};
typedef struct optionRecord optionRecType;


struct packetRecord {
  BYTEop;
  BYTEhType;
  BYTEhLen;
  BYTEhops;
  longxid;/* 4 bytes */
  shortsecs;/* 2 bytes */
  shortflags;/* 2 bytes */
  BYTEciAddr[CIADDR_SIZE];
  BYTEyiAddr[YIADDR_SIZE];
  BYTEsiAddr[SIADDR_SIZE];
  BYTEgiAddr[GIADDR_SIZE];
  BYTEchAddr[CHADDR_SIZE];
  charsName[SNAME_SIZE];
  charbootFile[BOOTFILE_SIZE];
  optionRecTypeoptionArray[MAX_OPTIONS];
};
typedef struct packetRecord packetRecType;
```

Refer to Table 2-3 for a description of the **`optionRecord`** structure and its members.

Table 2-3 optionRecord structure fields

Field	Value
fromClient	Flag used to indicate source of option data (set/used only by DHCP server).

len	The length of the option in bytes
data	A pointer to the actual option data

optionRecord

The **`optionRecord`** structure represents a single DHCP option as specified in RFC 2132, or any subsequently published RFC containing the specfications for additional options.

packetRecord

Refer to Table 2-4 for a description of the **`packetRecord`** structure and its members.

Table 2-4 packetRecord structure fields

Field	Value
op	The **op** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the message op code/message type (1=request, 2=reply).
htype	The **htype** field of the Bootp/DHCP packet header. Defined in RFC 2131 as the hardware address type (for example, 1=10mb Ethernet).
hlen	The **hlen** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the hardware address length (for example, 6=10mb Ethernet).
hops	The **hops** field of the Bootp/DHCP packet header. Defined by RFC 2131 to be used by the relay agents.
xid	The **xid** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the transaction ID as chosen by the client.
secs	The **secs** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the client provided seconds elapsed since client began address aquisition or renewal process.
flags	The **flags** field of the Bootp/DHCP packet header.
ciaddr	The **ciaddr** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the client's IP address, which is non-zero if the client is in a bound, renew, or rebinding state and can respond to Address Resolution Protocol (ARP) requests.
yiaddr	The **yiaddr** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the IP address assigned by the DHCP server to the client.
siaddr	The **siaddr** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the IP address of the next server to use in the bootstrap process.
giaddr	The **giaddr** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the IP address of the relay agent.

Field	Value
chaddr	The **chaddr** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the client's hardware address.
sname	The **sname** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the optional server host name as a null terminated string.
bootfile	The **file** field of the Bootp/DHCP packet header. Defined by RFC 2131 as the bootfile name as a null terminated string.

The detailed Lucent DHCP API routines are discussed in "Lucent DHCP API routines" (p. 2-15).

Multiple callout library feature

Overview

The Lucent DHCP 5.5 and DHCP 5.6 Server support up to nine different callout libraries in a single instance of the server executable.

Multiple callout libraries

The Lucent DHCP 5.5 and DHCP 5.6 Server can load and make callouts to multiple, site-specific API callout libraries. The order in which the callouts are made to the various libraries at a particular callout point is configurable.

The callout policies have a numerical suffix that identifies for which of the multiple callout libraries the policy is intended. The numerical suffix must be a single digit since the maximum number of callout libraries the server supports is nine. The policy tags (without the numerical suffix) and default values are shown in the following table.

If a policy is ON (1), the callout in the associated library is made. If a policy is OFF (0), or not present, the callout attempt is not made by the server.

> **Note:** All API callout policies are set in the **Additional Policies** section of the DHCP server profile. For more information, refer to “Adding additional Lucent DHCP policies”, in chapter 25 of the *Administrator Reference Manual*.

Table 2-5 Multiple callout library policies

Callout policy	Default value
`APICalloutLibraryName`	null
`APIPacketReceiptCallout`	0
`APIOfferCallout`	0
`APIDiscoverCallout`	0
`APIRequestCallout`	0
`APIRequestResponseCallout`	0
`APIAckCallout`	0
`APINackCallout`	0
`APIBootpRequestCallout`	0
`APIBootpReplyCallout`	0
APIReleaseCallout	0
APILeaseQueryCallout *(DHCP 5.6 Server only)*	0

Callout policy	Default value
APILeaseActiveCallout *(DHCP 5.6 Server only)*	0
APILeaseUnassignedCallout *(DHCP 5.6 Server only)*	0
APILeaseUnknownCallout *(DHCP 5.6 Server only)*	0

Library calling sequence policy

To allow for a variable order of executing the callouts associated with a particular library, each callout has a library calling sequence policy associated with it. The library calling sequence policy values are a comma- or space-delimited list of the callout library suffix number. These policies identify the order in which the callout APIs for a particular callout point are made, and are as shown below. The default, however, is to call the APIs in the numerical order as defined by the API library naming policy (**`APICalloutLibraryName`**).

Table 2-6 Multiple callout library calling sequence policies

Order	Callout library
1	**`APIPacketReceiptOrder`**
2	**`APIOfferOrder`**
3	**`APIDiscoverOrder`**
4	**`APIRequestOrder`**
5	**`APIRequestResponseOrder`**
6	**`APIAckOrderAPINackOrder`**
7	**`APIBootpRequestOrder`**
8	**`APIBootpReplyOrder`**
9	**`APIReleaseOrder`**
10	**`APIDepletedSubnetOrder`**
11	**`APIIgnoredPacketOrder`**
12	**`APILeaseQueryOrder`** *(DHCP 5.6 Server only)*
13	**`APILeaseActiveOrder`** *(DHCP 5.6 Server only)*
14	**`APILeaseUnassignedOrder`** *(DHCP 5.6 Server only)*
15	**`APILeaseUnknownOrder`** *(DHCP 5.6 Server only)*

Multiple callout library example

```
APICalloutLibraryName1=libapi1
APIPacketReceiptCallout1=1
APIDiscoverCallout1=1
APIRequestCallout1=1
APICalloutLibraryName2=libapisample
APIPacketReceiptCallout2=1
APIDiscoverCallout2=1
APIRequestCallout2=1
APIPacketReceiptOrder=1,2
APIDiscoverOrder=2,1
APIRequestOrder=2,1
```

Lucent DHCP API routines

discoverAPI

The Lucent DHCP server calls the **discoverAPI** routine when a DHCP-DISCOVER message has been received from the DHCP client. The call is made after the incoming packet has been verified as a valid DHCP packet, and the fields of the packet have been parsed. The intention of this callout is to allow the interface to control the Lucent DHCP server's behavior while processing the client's Discover packet.

This callout can modify one or more of the available parameters in order to override or supplement those values that came from the client. In this way, the Lucent DHCP server processes the packet as if these fields came from the client. For example, the client might send a host name of “Bob”, but this “discoverAPI” callout can change the host name to “client-123”, thereby affecting the name that is updated in VitalQIP and DDNS.

> **Note:** Refer to the individual parameters for a description of how each can affect the Lucent DHCP server processing.

In addition, the interface may also provide a suggested subnet from which the server should offer the address. If a suggested subnet is provided, the server attempts to *only* offer an address from the suggested subnet.

> **Note:** If this routine callout specifies a suggested subnet, the programmer ***must*** also code similar logic into the **requestAPI** routine to return the same suggested subnet. This is required due to the nature of the DHCP protocol, as described here.

Prototype

```
int discoverAPI  (  const packetRecType             *packetRec,
                    char                            *hostName,
                    char                            *domainName,
                   char                         *requestedIpAddr,
                    unsigned char                   *clientFQDN,
char                        *bootfile,
char                        *tftpServer,
unsigned char               *vendorClass,
unsigned char               *userClass,
unsigned char               *vendorSpecific,
long                        *leaseTime,
                    int                             *authorize,
                   char                         *suggestedsubnet,
                  int                        *enforceSubnet
   )
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, incoming DHCPDISCOVER packet. Refer to the description in "packetRecord" (p. 2-10).
`hostName`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client hostname. This parameter corresponds to the **Host Name** option (12) of the DHCP packet, as described in section 3.14 of RFC 2132. The value of this parameter is usually the same as that in the optionArray[12] element in the **`packetRec`** parameter. However, it may be different if the hostname has acted upon the DHCP server as instructed by the **`ClientHostNameProcessing`** policy (refer to the *Administrator Reference Manual* for information on this Lucent DHCP policy). If this parameter is modified by the interface, the Lucent DHCP server uses this as the client hostname for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parameter is 63 bytes.
`DomainName`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client hostname. This parameter corresponds to the **DomainName** option (15) of the DHCP packet as described in section 3.17 of RFC 2132. The value of this parameter is usually the same as that in the optionArray[12] element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this as the domain name for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parameter is 190 bytes.
`requestedIpAddr`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the IP address requested by the client, if available. The IP address is formatted in dotted-decimal notation. This parameter corresponds to the **Requested IP Address** option (50) of the DHCP packet, as described in section 9.1 of RFC 2132. If this parameter is modified by the interface, the Lucent DHCP server *attempts* to offer the client this address. The address supplied ***must*** be a valid configured *dynamic* address within the server's configuration for the subnet specified in the **`suggestSubnetAddr`** argument (if specified). If the value of these addresses conflicts with the server's configuration or with each other, this parameter is ignored. The maximum length for this parameter is 16 bytes.

`clientFQDN`	READ/WRITE. A pointer to an unsigned character buffer representation of the client's dynamic DNS information. This parameter corresponds to the **Client FQDN** option (81) of the DHCP packet, as described in the Internet Draft, *draft-ietf-dhc-fqdn-option-00.txt* **Note:** The FQDN option draft was not published as an RFC and was officially deleted as an Internet draft since it was not revised prior to its expiration date. This draft may become active again, depending on the actions of its authors, but the Lucent DHCP API and the Lucent DHCP server currently implement the version of the draft listed above, which is the most widely implemented version among DHCP clients. The value of this parameter is the same as that in the optionArray[81] element in the **`packetRec`** parameter. If this parameter is modified by the interface, the resulting changes must conform to the specifications outlined in the draft noted above. The Lucent DHCP server attempts to use this value in further processing with respect to the dynamic DNS updates that this option attempts to determine. In other words, the server treats the contents of this parameter, as returned by the interface, as if the data had come from the DHCP client itself. The maximum length for this parameter is 255 bytes.
`bootfile`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client's **bootfile** field. The string buffer must not exceed 128 bytes in length. This parameter corresponds to the **file** field of the DHCP packet header, as described in section 2 of RFC 2131. If this parameter is modified by the interface, the Lucent DHCP server uses this for the contents of the **file** field on the DHCP Offer to be sent to the client in response to this Discover.
`tftpServer`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the address of the TFTP server. The address must be formatted in standard dotted-decimal notation. This parameter corresponds to the **siaddr** field of the DHCP packet header, as described in section 2 of RFC 2131. If this parameter is filled in by the interface, the Lucent DHCP server converts this string to the 4-octet representation of the IP address. This value is then used for the contents of the **siaddr** field on the DHCP Offer to be sent to the client in response to this Discover. **Note:** For the Discover callout, this parameter is zero, as defined by RFC 2131 to be zero in all client-side messages. Maximum length for this parameter is 16 bytes.

`vendorClass`	READ/WRITE. A pointer to an unsigned character buffer representation of the client's vendor class. This parameter corresponds to the **Vendor Class Identifier** option (60) of the DHCP packet, as described in section 9.13 of RFC 2132. The value of this parameter is the same as that in **`optionArray[60]`** element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself. Maximum length for this parameter is 255 bytes.
`userClass`	READ/WRITE. A pointer to an unsigned character buffer representation of the client's user class. This parameter corresponds to the **User Class Identifier** option (77) of the DHCP packet as described in Internet Draft *draft.ietf.dhc-userclass-02.txt*. **Note:** The *userclass* draft has been published as RFC 3004, which allows multiple userclass values in the option data. Since the majority of DHCP clients serviced by the Lucent DHCP server, including Microsoft clients, conform to the original single-value format specified in the draft noted above, the Lucent DHCP API and the Lucent DHCP Server currently support the single value format. The value of this parameter is the same as that in **`optionArray[77]`** element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself. Maximum length for this parameter is 255 bytes.
`vendorSpecific`	READ/WRITE. A pointer to an unsigned character buffer representation of the vendor-specific data. This parameter corresponds to the **Vendor Specific Information** option (43) of the DHCP packet as described in section 8.4 of RFC 2132. The value of this parameter is the same as that in **`optionArray[43]`** element in the **`packetRec`** parameter. If this parameter is filled in by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself. Maximum length for this parameter is 255 bytes.

`leaseTime`	READ/WRITE. An unsigned character buffer representation of the lease time. This parameter corresponds to the **DHCP Lease Time** option (52) of the DHCP packet, as described in section 9.2 of RFC 2132. This parameter has a maximum length of 4 bytes. The value is specified in seconds and stored in network byte order. The value of this parameter will be the same as that in **`optionArray[52]`** element in the **`packetRec`** parameter. If this parameter is filled in by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself.
`authorize`	WRITE ONLY. A pointer to an integer. This parameter is modified by the interface to provide a Boolean value to authorize the server to offer an address/lease to the client. A value of zero instructs the server to ignore the client request. Therefore, it will ***not*** be sending a DHCPOFFER in response to the client request. A non-zero value instructs the server to continue with processing the request.
`suggestedSubnet`	WRITE ONLY. A pointer to a character buffer that can hold a zero-terminated ASCII string representation of the suggested subnet address. This parameter may be modified by the interface to identify a subnet from which the server should offer an address. If filled in by the interface, this parameter must be formatted in standard dotted-decimal notation. The size of the character buffer supplied by the server is fixed at 16 bytes. This address must be a valid address in one of the subnet directives in the *dhcpd.conf* file. By default, if no such subnet is found, the server reverts to standard processing of the request in an attempt to offer an address to the client. If no addresses are available within the suggested subnet, the server does not offer an address to the client.
`enforceSubnet`	This parameter is not used by the server logic. However, the API interface must include it.

Return values

The return code from the API is not used by the server logic. The value returned in the **`authorize`** parameter (described above) is used to determine if the client message is processed by the Lucent DHCP server.

Remarks

- If **`hostName`** is modified by the **`discoverAPI`** routine, the Lucent DHCP server uses **`hostName`** as the client hostname for updates to VitalQIP and DDNS.
- If the **`suggestedSubnet`** is specified, the **`requestAPI`** routine must also return the same **`suggestedSubnet`**.

- By default, if no subnet is found containing **`suggestedSubnet`**, the Lucent DHCP server reverts to standard processing of the request for offering an address to the client.
- No address is offered to a client if addresses are not available in the **`suggestedSubnet`**.

offerAPI

The Lucent DHCP server calls the **`offerAPI`** routine after sending a DHCPOFFER to the client. This call is made directly after a message is successfully sent and verified. The intention of this callout is to allow the interface to record any information that may be necessary.

Prototype

```
void offerAPI  (  const packetRecType *packetRec   )
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing DHCPOFFER packet. Refer to the description in "packetRecord" (p. 2-10).

Return values

None.

Remarks

Additional information about DHCP Options, which may be present in the buffer for **`dhcpOptions`**, are described in RFC 2132.

requestAPI

The Lucent DHCP server calls this interface after receiving a DHCPREQUEST from the client. This call is made directly after the incoming packet has been verified as a valid DHCP packet, and the fields of the packet have been parsed. The intention of this callout is to allow the interface to authorize the server to lease an address to the client and modify specific parameters. It can also suggest the subnet from which the client's leased address should come. Refer to the description for the **`discoverAPI`** routine.

Prototype

```
int requestAPI  (  const packetRecType *packetRec,
                    char *hostName,
                    char *domainName,
                    char *requestedIpAddr,
                    unsigned char *clientFQDN,
char     *bootfile,
char     *tftpServer,
unsigned char*vendorClass,
unsigned char*userClass,
unsigned char*vendorSpecific,
long     *leaseTime,
                    int *authorize,
                    char *suggestedSubnet,
             int *enforceSubnet  )
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, incoming DHCPREQUEST packet. Refer to the description in "packetRecord" (p. 2-10).
`hostName`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client host name. This parameter corresponds to the **Host Name** option (12) of the DHCP packet as described in section 3.14 of RFC 2132. The value of this parameter is usually the same as that in the **`optionArray[12]`** element in the **`packetRec`** parameter. However, it may be different if the hostname was acted upon by the DHCP server as instructed by the ClientHostNameProcessing policy (refer to the *Administrator Reference Manual* for information on this Lucent DHCP policy). If this parameter is modified by the interface, the Lucent DHCP server uses this as the client hostname for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parameter is 63 bytes.

`DomainName`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client hostname. This parameter corresponds to the **DomainName** option (15) of the DHCP packet as described in section 3.17 of RFC 2132. The value of this parameter is usually the same as that in the **`optionArray[12]`** element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this as the domain name for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parameter is 190 bytes.
`requestedIpAddr`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the IP address requested by the client, if available. The IP address is formatted in dotted-decimal notation. This parameter corresponds to the **Requested IP Address** option (50) of the DHCP packet as described in section 9.1 of RFC 2132. If this parameter is modified by the interface, the Lucent DHCP server ***attempts*** to offer the client this address. The address supplied ***must*** be a valid configured ***dynamic*** address within the server's configuration for the subnet specified in the **`suggestSubnetAddr`** argument (if specified). If the value of these addresses conflicts with the server's configuration or with each other, this parameter is ignored. The maximum length for this parameter is 16 bytes.

`clientFQDN`	READ/WRITE. A pointer to an unsigned character buffer representation of the client's dynamic DNS information. This parameter corresponds to the **Client FQDN** option (81) of the DHCP packet as described in the Internet Draft, *draft-ietf-dhc-fqdn-option-00.txt*. **Note:** The FQDN option draft was not published as an RFC and was officially deleted as an Internet draft since it was not revised prior to its expiration date. This draft may become active again, depending on the actions of its authors, but the Lucent DHCP API and the Lucent DHCP Server currently implement the versioin of the draft listed above, which is the most widely implemented version among DHCP clients. The value of this parameter is the same as that in the **`optionArray[81]`** element in the **`packetRec`** parameter. If this parameter is modified by the interface, the resulting changes must conform to the specifications outlined in the draft noted above. The Lucent DHCP server attempts to use this value in further processing with respect to the dynamic DNS updates that this option attempts to determine. In other words, the server treats the contents of this parameter, as returned by the interface, as if the data had come from the DHCP client itself. The maximum length for this parameter is 255 bytes.
`bootfile`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client's **bootfile** field. The string buffer must not exceed 128 bytes in length. This parameter corresponds to the **file** field of the DHCP packet header, as described in section 2 of RFC 2131. If this parameter is modified by the interface, the Lucent DHCP server uses this for the contents of the **file** field on the DHCP Offer to be sent to the client in response to this Discover.
`tftpServer`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the address of the TFTP server. The address must be formatted in standard dotted-decimal notation. This parameter corresponds to the **siaddr** field of the DHCP packet header, as described in section 2 of RFC 2131. If this parameter is filled in by the interface, the Lucent DHCP server converts this string to the 4-octet representation of the IP address. This value is then used for the contents of the **siaddr** field on the DHCP Offer to be sent to the client in response to this Discover. **Note:** For the Discover callout, this parameter is zero, as it is defined by RFC 2131 to be zero in all client-side messages. Maximum length for this parameter is 16 bytes.

`vendorClass`	READ/WRITE. A pointer to an unsigned character buffer representation of the client's vendor class. This parameter corresponds to the **Vendor Class Identifier** option (60) of the DHCP packet as described in section 9.13 of RFC 2132. The value of this parameter is the same as that in **`optionArray[60]`** element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself. Maximum length for this parameter is 255 bytes.
`userClass`	READ/WRITE. A pointer to an unsigned character buffer representation of the client's user class. This parameter corresponds to the **User Class Identifier** option (77) of the DHCP packet as described in Internet Draft *draft.ietf.dhc-userclass-02.txt*. **Note:** The *userclass* draft has been published as RFC 3004, which allows multiple userclass values in the option data. Since the majority of DHCP clients serviced by the Lucent DHCP server, including Microsoft clients, conform to the original single-value format specified in the draft noted above, the Lucent DHCP API and the Lucent DHCP Server currently support the single value format. The value of this parameter is the same as that in optionArray[77] element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself. Maximum length for this parameter is 255 bytes.
`vendorSpecific`	READ/WRITE. A pointer to an unsigned character buffer representation of the vendor-specific data. This parameter corresponds to the **Vendor Specific Information** option (43) of the DHCP packet, as described in section 8.4 of RFC 2132. The value of this parameter is the same as that in optionArray[43] element in the **`packetRec`** parameter. If this parameter is filled in by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself. Maximum length for this parameter is 255 bytes.

`leaseTime`	READ/WRITE. An unsigned character buffer representation of the lease time. This parameter corresponds to the **DHCP Lease Time** option (51) of the DHCP packet, as described in section 9.2 of RFC 2132. This parameter has a maximum length of 4 bytes. The value is specified in seconds and stored in network byte order. The value of this parameter is the same as that in optionArray[51] element in the **`packetRec`** parameter. If this parameter is filled in by the interface, the Lucent DHCP server uses this value for further processing, as if this data had come directly from the client itself.
`authorize`	WRITE ONLY. A pointer to an integer. This parameter is modified by the interface to provide a Boolean value to authorize the server to offer an address/lease to the client. A value of zero instructs the server to ignore the client request, and therefore does ***not*** send a DHCPOFFER in response to the client request. A non-zero value instructs the server to continue with processing the request.
`suggestedSubnet`	WRITE ONLY. A pointer to a character buffer that can hold a zero-terminated ASCII string representation of the suggested subnet address. This parameter may be modified by the interface to identify a subnet from which the server should offer an address. If filled in by the interface, this parameter must be formatted in standard dotted-decimal notation. The size of the character buffer supplied by the server is fixed at 16 bytes. This address must be a valid address in one of the subnet directives in the *dhcpd.conf* file. By default, if no such subnet is found, the server reverts to standard processing of the request, in an attempt to offer an address to the client. If no addresses are available within the suggested subnet, the server does not offer an address to the client.
`enforceSubnet`	This parameter is not used by the server logic. However, the API interface must include it.

Return values

non-zero	Indicates a failure within the routine. The DHCP Client's request is processed as if the routine had not been called.
0	Indicates success. An address is offered to the DHCP Client based upon the values of the **`authorize`** and **`suggestSubnet`** parameters.

Remarks

- If the **`requestAPI`** routine modifies **`hostName`**, the Lucent DHCP server uses **`hostName`** as the client hostname for updates to VitalQIP and DDNS.

- If the **`suggestedSubnet`** is specified, it must correspond to the **`suggestedSubnet`** in the **`discoverAPI`** routine.
- By default, if no subnet is found containing **`suggestededSubnet`**, the server reverts to standard processing of the request trying to offer an address to the client.
- No address is offered to a client if addresses are not available in the **`suggestedSubnet`**.

request2API

The Lucent DHCP server calls the **`request2API`** routine after receiving a DHCPREQUEST from the client and after processing the request to the point where an "ACK" is to be sent. At this point, the server determines if the client is in the "selecting" state or if the client has been issued the address. The intention of this callout is to allow the interface to inform the server as to whether it should send an "ACK" or a "NAK" in response to the client request. A value of zero instructs the server to send a negative acknowledgment (NAK), whereas a non-zero value instructs the server to send a positive acknowledgment (ACK).

Prototype

```
int request2API  (  const packetRecType *packetRec,
  int  renew,
  int* acknowledge)
```

Parameters

`packetRec`	READ ONLY. A pointer to the structurre containing the parsed, incoming DHCPREQUEST packet. Refer to the description in "packetRecord" (p. 2-10).
`renew`	READ ONLY. An integer indicating whether the client is renewing a previously allocated lease or not. A non-zero value (for example, 1) indicates that this client has already been issued the address denoted in **ciAddr** of the **`packetRec`** structure and is in the Init-Reboot, Renewal, or Rebinding state. A zero value indicates that the client is in the Selecting state, and therefore is about to bind to this address for the first time.
`acknowledge`	WRITE ONLY. A pointer to an integer. This parameter is modified by the interface to provide a Boolean value to inform the server that it should send an "ACK" or a "NAK" in response to the client's request.

Return values

The return code from the API is not used by the server logic. The value returned in the acknowledge parameter described above determines if the server sends an "ACK" or a "NAK."

ackAPI

The Lucent DHCP server calls the **`ackAPI`** routine after sending a DHCPACK to the client. This call is made directly after a successful send of the message has been verified. The intention of this callout is to allow the interface to record any information that may be necessary, or to notify other processes of the event.

Prototype

```
void ackAPI  (  const packetRecType *packetRec)
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing DHCPACK packet. Refer to the description in "packetRecord" (p. 2-10).

Return values

None.

Remarks

Additional information about DHCP Options that may be present in the buffer for **`dhcpOptions`** are described in RFC 2132.

nackAPI

The Lucent DHCP server calls the **`nackAPI`** routine after sending a DHCPNACK to the client. This call is made directly after a successful send of the message has been verified. The intention of this callout is to allow the interface to record any information that may be necessary or to notify other processes of the event.

Prototype

```
void nackAPI  (  const packetRecType *packetRec)
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing DHCPNAK packet. Refer to the description in "packetRecord" (p. 2-10).

Return values

None.

Remarks

None.

bootpRequestAPI

The Lucent DHCP server calls the **`bootpRequestAPI`** routine when a BootpRequest message has been received from the client. This call is directly made after the incoming packet has been verified as a valid Bootp packet, and the fields of the packet have been parsed. The intention of this callout is to allow the interface to control the Lucent DHCP server's behavior while processing the client's Bootp request packet. This callout can modify one or more of the available parameters in order to override or supplement those values that came from the client. In this way, the Lucent DHCP server processes the packet as if these fields came from the client.

Note: Refer to the individual parameters for a description of how each can affect the Lucent DHCP server processing.

In addition, the interface may also provide a suggested subnet from which the server should offer the address.

Prototype

```
int bootpRequestAPI (  const packetRecType            *packetRec,
                    char                             *hostName,
                    char                             *domainName,
                   char                          *requestedIpAddr,
                    unsigned char                    *clientFQDN,
char                            *bootfile,
char                            *tftpServer,
unsigned char                   *vendorClass,
unsigned char                   *userClass,
unsigned char                   *vendorSpecific,
long                            *leaseTime,
                    int                              *authorize,
                   char                          *suggestedsubnet,
                  int                        *enforceSubnet
   )
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, incoming BOOPTREQUEST packet. Refer to the description in "packetRecord" (p. 2-10).
`hostName`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client host name. This parameter corresponds to the **Host Name** option (12) of the DHCP packet as described in section 3.14 of RFC 2132. The value of this parameter is usually the same as that in the **`optionArray[12]`** element in the **`packetRec`** parameter. However, it may be different if the hostname was acted upon by the DHCP server as instructed by the ClientHostNameProcessing policy (refer to the *Administrator Reference Manual* for information on this Lucent DHCP policy). If this parameter is modified by the interface, the Lucent DHCP server uses this as the client hostname for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parameter is 63 bytes.
`DomainName`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client domain name. This parameter corresponds to the **DomainName** option (15) of the DHCP packet as described in section 3.17 of RFC 2132. The value of this parameter is usually the same as that in the **`optionArray[12]`** element in the **`packetRec`** parameter. If this parameter is modified by the interface, the Lucent DHCP server uses this as the domain name for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parameter is 60 bytes.
`requestedIPAddr`	This parameter is not used by the server logic for a bootp request. However, it must be in the API parameters to conform to the standard interface.
`clientFQDN`	This parameter is not used by the server logic for a bootp request. However, it must be in the API parameters to conform to the standard interface.
`bootfile`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the client's **bootfile** field. The string buffer must not exceed 128 bytes in length. This parameter corresponds to the **file** field of the DHCP packet header, as described in section 2 of RFC 2131. If this parameter is modified by the interface, then the Lucent DHCP server will use this for the contents of the **file** field on the DHCP Offer to be sent to the client in response to this Discover.

`tftpServer`	READ/WRITE. A pointer to a zero-terminated ASCII character string representation of the address of the TFTP server. The address must be formatted in standard dotted-decimal notation. This parameter corresponds to the **siaddr** field of the DHCP packet header, as described in section 2 of RFC 2131. If this parameter is filled in by the interface, then the Lucent DHCP server will convert this string to the 4-octet representation of the IP address. This value will then be used for the contents of the **siaddr** field on the BootpReply to be sent to the client in response to this BootpRequest. Maximum length for this parameter is 16 bytes.
`vendorClass`	This parameter is not used by the server logic for a bootp request. However, it must be in the API parameters to conform to the standard interface.
`userClass`	This parameter is not used by the server logic for a bootp request. However, it must be in the API parameters to conform to the standard interface.
`vendorSpecific`	This parameter is not used by the server logic for a bootp request. However, it must be in the API parameters to conform to the standard interface.
`leaseTime`	This parameter is not used by the server logic for a bootp request. However, it must be in the API parameters to conform to the standard interface.
`authorize`	WRITE ONLY. A pointer to an integer. This parameter is modified by the interface to provide a Boolean value to authorize the server to offer an address/lease to the client. A value of zero instructs the server to ignore the client request, and therefore it does ***not*** send a BootpReply in response to the client request. A non-zero value instructs the server to continue with processing the request.
`suggestedSubnet`	WRITE ONLY. A pointer to a character buffer that can hold a zero-terminated ASCII string representation of the suggested subnet address. This parameter may be modified by the interface to identify a subnet from which the server should offer an address. If filled in by the interface, this parameter must be formatted in standard dotted-decimal notation. The size of the character buffer supplied by the server is fixed at 16 bytes. This address must be a valid address in one of the subnet directives in the *dhcpd.conf* file. By default, if no such subnet is found, the server reverts to standard processing of the request in an attempt to offer an address to the client. If no addresses are available within the suggested subnet, the server does not offer an address to the client.

`enforceSubnet`	This parameter is not used by the server logic. However, the API interface must include it.

Return values

The return code from the API is not used by the server logic. The value returned in the **`authorize`** parameter (described above) is used to determine if the client message is processed by the Lucent DHCP server.

Remarks

- By default, if no subnet is found containing **`suggestedSubnet`**, the server reverts to standard processing of the request by offering an address to the client.
- No address is offered to a client if addresses are not available in the **`suggestedSubnet`**.

bootpReplyAPI

The Lucent DHCP server calls the **`bootpReplyAPI`** routine after sending a BootpReply to the client. This call is made directly after a successful send of the message has been verified. The intention of this callout is to allow the interface to record any information that may be necessary, or to notify other processes of the event.

Prototype

```
void bootpReplyAPI  ( const packetRecType *packetRec)
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing BOOTPREPLY packet. Refer to the description in "packetRecord" (p. 2-10).

Return values

None.

Remarks

Additional information about DHCP Options that may be present in the buffer for **`dhcpOptions`** are described in RFC 2132.

depletedSubnetAPI

The Lucent DHCP server calls the **`depletedSubnetAPI`** routine when it is unable to offer an address to a client due to address depletion in a subnet. After the client's subnet has been determined, either via the normal protocol or from the **`discoverAPI`** routine, the server attempts to offer a lease to the client using an available address from the subnet. If no addresses are available in the subnet, then the server will not send a DHCPOFFER and a call to the **`depletedSubnetAPI`** routine is made. The intention of this callout is to allow the interface to record any information that may be necessary, and/or take any actions that are deemed appropriate.

Prototype

```
void depletedAPI  ( const packetRecType *packetRec,
         const char *depletedSubnetAddr)
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, incoming DHCPDISCOVER packet. Refer to the description in "packetRecord" (p. 2-10).
`depletedSubnetAddr`	READ ONLY. A pointer to a zero-terminated ASCII character string representation of the address of the depleted subnet.

Return values

None.

Remarks

None.

ignoredPacketAPI

The Lucent DHCP server calls the **`ignoredPacketAPI`** routine when it is unable to determine the subnet on which the client request originated, and therefore cannot proceed with processing of the packet. This routine can be called for DHCPDISCOVER, DHCPREQUEST, DHCPRELEASE, DHCPDECLINE, DHCPINFORM, and BOOTREQUEST type messages. The intention of this callout is to allow the interface to record any information that may be necessary or to notify other processes of the event.

Prototype

```
void  ignoredPacketAPI (const packetRecType *packetRec)
```

Parameters

`PacketRec`	READ ONLY. A pointer to the structure containing the parsed, incoming packet. Refer to the description in "packetRecord" (p. 2-10).

Return values

None.

Remarks

None.

packetReceiptAPI

The Lucent DHCP server calls the **`packetReceiptAPI`** routine immediately after a packet is received and parsed. The intention of this callout is to allow the interface to specifically modify the hardware address (hwAddr) of the client as well as the hostname, for every packet received, regardless of type.

> **Note:** When the **`PacketReceiptAPI`** callout is utilized to generate a MAC address (chaddr) to track the DHCP lease in VitalQIP, according to this generated MAC, you must add the following policy to the "Additional Policies" section of the DHCP server profile. This ensures that the DHCPOFFER and DHCPACK messages are properly sent to the device(s) that initiated the DHCP message exchange:

`RestoreOriginalChaddr=1`

Prototype

```
int  packetReceiptAPI (const packetRecType *packetRec,
char *hostName,
unsigned char *hwAddr)
```

Parameters

`PacketRec`	READ ONLY. A pointer to the structure containing the parsed, incoming packet. Refer to the description in "packetRecord" (p. 2-10).
`hostName`	WRITE ONLY. A pointer to a zero-terminated ASCII character string representation of the client hostname. If this parameter is modified by the interface, the Lucent DHCP Server uses this as the client hostname for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager. The maximum length for this parmater is 63 bytes.
`hwAddr`	WRITE ONLY. A pointer to a six byte long unsigned character array intended to contain the modified hardware address of the client's interface. If this parameter is modified by the interface, the Lucent DHCP Server uses this as the client hardware address for updates to its lease database and for updates to VitalQIP, DDNS, and Audit Manager.

Return Values

0	Indicates a failure within the routine, and the packet was not processed as intended. The packet will be discarded with no further processing.
Non-zero	Indicates success. The packet will continue to be processed according to the message type.

Remarks

None.

releaseAPI

The Lucent DHCP server calls the **`releaseAPI`** routine after receiving and processing a release packet from a DHCP client. The intent of this callout is to allow the interface to record any information that is required, or to inform another process of the event.

Prototype

```
void releaseAPI (unsigned char *ipAddr,
unsigned char *hwAddr,
char *hostName,
char *domainName,
unsigned char *option82)
```

Parameters

`ipAddr`	READ ONLY. Pointer to a four byte unsigned character array containing the IP address of client that sent DHCPRELEASE.
`hwAddr`	READ ONLY. Pointer to a six byte unsigned character array containing the hardware (MAC) address of the client.
`hostName`	READ ONLY. A pointer to a zero-terminated ASCII character string representation of the client hostname.
`domainName`	READ ONLY. A pointer to a zero-terminated ASCII character string representation of the client hostname.
`option82`	READ ONLY. Pointer to unsigned character array containing the option 82 data that was received with the discover message for this client. (byte 1 = opcode, byte 2 = length of option data, byte 3+ = option data.)

Return Values

None.

Remarks

None.

leaseQueryAPI

Lucent DHCP 5.6 Server only

The Lucent DHCP 5.6 server calls the **`leaseQueryAPI`** routine when a DHCP-LEASEQUERY message has been received by the DHCP server. This call is made after the incoming packet has been verified, including that the giaddr is confirmed to be non-zero, as required by RFC 4388. The intention of this callout is to allow the interface to control whether or not the Lucent DHCP server processes the lease query request, through the use of the authorize parameter.

Prototype

```
void  leaseQueryAPI (const packetRecType   *packetRec,
                         int *authorize);
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, incoming DHCPLEASEQUERY packet. Refer to the description in "packetRecord" (p. 2-10). as well as **RFC 4388** for the leasequery unique fields.
`Authorize`	WRITE ONLY. A pointer to an integer. This parameter is modifies by the interface to provide a Boolean value to authorize the DHCP server to continue processing the query. A value of zero instructs the server to ignore the query. A non-zero value instructs the server to continue with processing the query.

Return values

None.

Remarks

Additional information about the DHCP-LEASEQUERY message and the applicable options are described in **RFC 4388**.

leaseActiveAPI

Lucent DHCP 5.6 Server only

The Lucent DHCP 5.6 server calls the **`leaseActiveAPI`** routine after a DHCP-LEASEACTIVE message has been sent by the DHCP server in response to a DHCP-LEASEQUERY message that requested information regarding a lease that is active and managed by the DHCP server. This call is made after the outgoing packet has been sent. The intention of this callout is to allow the interface to audit and potentially log the event and the contents of the packet.

Prototype

```
void  leaseActiveAPI (const packetRecType     *packetRec);
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing DHCPLEASEACTIVE packet. Refer to the description in "packetRecord" (p. 2-10). as well as **RFC 4388** for the leasequery unique fields.

Return values

None.

Remarks

Additional information about the DHCP-LEASEACTIVE message and the applicable options are described in RFC 4388.

leaseUnassignedAPI

Lucent DHCP 5.6 Server only

The Lucent DHCP 5.6 server calls the **`leaseUnassignedAPI`** routine after a DHCP-LEASEUNASSIGNED message has been sent by the DHCP server in response to a DHCP-LEASEQUERY message that requested information regarding a lease that is not active but managed by the DHCP server. This call is made after the outgoing packet has been sent. The intention of this callout is to allow the interface to audit and potentially log the event and the contents of the packet.

Prototype

```
void  leaseUnassignedAPI (const packetRecType*packetRec);
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing DHCPLEASEUNASSIGNED packet. Refer to the description in “packetRecord” (p. 2-10). as well as **RFC 4388** for the leasequery unique fields.

Return values

None.

Remarks

Additional information about the DHCP-LEASEUNASSIGNED message and the applicable options are described in RFC 4388.

leaseUnknownAPI

Lucent DHCP 5.6 Server only

The Lucent DHCP 5.6 server calls the **`leaseUnknownAPI`** routine after a DHCP-LEASEUNKNOWN message has been sent by the DHCP server in response to a DHCP-LEASEQUERY message that requested information regarding a lease that is not managed by the DHCP server. This call is made after the outgoing packet has been sent. The intention of this callout is to allow the interface to audit and potentially log the event and the contents of the packet.

Prototype

```
void  leaseUnknownAPI (const packetRecType*packetRec);
```

Parameters

`packetRec`	READ ONLY. A pointer to the structure containing the parsed, outgoing DHCPLEASEUNKNOWN packet. Refer to the description in "packetRecord" (p. 2-10). as well as **RFC 4388** for the leasequery unique fields.

Return values

None.

Remarks

Additional information about the DHCP-LEASEUNKNOWN message and the applicable options are described in RFC 4388.

initAPI

This **`initApi`** routine is called when the Lucent DHCP server is started or restarted. This routine is used to initialize any data structures that are used by the API module. Typically, initializing data includes reading a policy file, initializing debug logs, and allocating static memory to be used by the other, implemented routines in the API callout module. The **`initApi`** routine must exist in the Lucent DHCP API, even if it only returns without processing being performed.

Prototype

```
int initAPI(void)
```

Parameters

None.

Return values

`0`	Indicates failure to initialize the API callout module. The returned value indicates a fatal condition in which the DHCP server is shut down.
`1`	Indicates successful initialization of the API callout module.

finalAPI

This **`finalAPI`** routine is called when the Lucent DHCP server is stopped or restarted. This routine is used to clean up any data structures that may be used by the API module. Cleanup includes closing and releasing file descriptors, and freeing any statically allocated memory. This **`finalAPI`** routine must exist in the Lucent DHCP API, even if it returns without processing being performed.

Prototype

```
void finalAPI(void)
```

Parameters

None.

Return values

None.

threadStopAPI

The **threadStopAPI** routine is called for each thread when the Lucent DHCP server is stopped or restarted. This routine is used to clean up any thread local data that is created within the API on a per thread basis. Typically, cleanup includes closing and releasing file descriptors, and freeing any thread local storage that has been allocated. This routine must exist in the Lucent DHCP API, even if it returns without processing being performed.

Prototype

```
int threadStopAPI(void)
```

Parameters

None.

Return values

None.

Index