

**Adding DHCP options to NET+OS (V6.0 – V6.3)
in an ACE environment**

Introduction

Purpose

This document describes the steps necessary for adding additional DHCP options to an outgoing DHCP request message packet. This might be used for requesting option 12 (host name) or option 81 (fully qualified domain name – FQDN).

Scope

This document describes the necessary steps for adding DHCP options to an outgoing request packet. This document is not meant to be a tutorial on DHCP, ACE (address configuration executive), FQDN, NET+OS or any other technology besides adding DHCP options to a DHCP message.

The information in this document is applicable to NET+OS V6.0 – V6.3. That is, versions of NET+OS that run with the fusion stack and ACE. Thus versions of NET+OS produced after V6.3 (versions that run the treck stack and use IAM) are incompatible with the information contained herein.

Audience

The content of this document is intended for developers with a good knowledge of NET+OS internals, especially NET+OS internals associated with ACE. Users of this document should also have knowledge of DHCP, TCP/IP networking and c data structures.

Glossary

ACE – address configuration executive
bootp – bootstrap protocol
DHCP – dynamic host control protocol
FQDN – fully qualified domain name
IAM – Internet Address Manager
NET+OS – An embedded operating system produced by Digi
tftp – trivial file transport protocol

Adding DHCP options to NET+OS (V6.0 – V6.3)

What is bootp?

bootp is short-hand notation for the bootstrap protocol. It is a UDP-based protocol that was first used for giving diskless workstations access to IP configuration information such as IP address, subnet mask and gateway. It can also be used, in conjunction with the tftp protocol for downloading some or all of the operating system and applications to run in a diskless workstation or network-connected device. bootp is based on RFC 951.

What is dhcp?

BOOTP has more or less been replaced by DHCP. DHCP has all the functionality of bootp plus some. DHCP, like bootp can be used for obtaining IP address configuration information. Think of it as the functional successor to bootp. It is based on RFC 2131.

Adding DHCP options to a DHCP packet

DHCP is made up of 4 operations as follows:

- Discover – client requests configuration information via broadcast
- Offer – server sends a potential reserved IP address via unicast
- Request – Client accepts or rejects the server’s offer via broadcast
- Acknowledgement – server acknowledges the request and supplies requested options et al.

Format of a DHCP packet

The following picture represents the format of a DHCP packet:

0	7 8	15 16	23 24	31
Op code (1 byte)	H/W type (1 byte)	H/W Len (1 byte)	Hops (1 byte)	
←-----Transaction	ID (4 bytes)-----	-----	-----	-----→
←-----Seconds-----	--elapsed--(2 bytes)--→	←-----flags----	--(2 bytes)-----	-----→
←-----	Client IP addr-----	(4 bytes)-----	-----	-----→
←-----	Your IP addr	(4 bytes)-----	-----	-----→
←-----	Server IP addr	(4 bytes)-----	-----	-----→
←-----	Gateway IP addr	(4 bytes)-----	-----	-----→
←-----	Client H/W addr	(16 bytes)-----	-----	-----
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----→
←-----	Boot file name	(128 Bytes)-----	-----	-----
-----	-----	-----	-----	-----→
←-----	DHCP options	Variable length-----	-----	-----→
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----

Adding DHCP options to a DHCP packet

Theory of Operation

In function `customizeStartAce`, a buffer large enough for holding both an existing DHCP request packet and additional options is allocated. Function `dhcpConfigToProtoInfo` zeros out the buffer, copies the existing DHCP config packet into the new buffer. The additional options are added to the end of the buffer (after the existing DHCP data). The size field of the DHCP packet is adjusted to include the additional options. Control is then given back to existing ACE code that will ensure that the packet gets out.

Files that must be modified/enhanced

To enable the adding of options to an outgoing DHCP packet, two files need to be modified, namely `aceparams.c` and `acecallbacks.c`. These two files can be found in the following directory:

```
<your_netos_directory>/src/bsp/platforms/<your platform>
```

After modifying these files, you must rebuild your bsp. After modifying your bsp, you must rebuild your application. Rebuilding your bsp, also rebuilds your file `rom.bin`, the bootloader contained in the platforms directory. Thus if you are running your application out of flash, you'll need to reflash the bootloader into your flash device. Then you'll need to reflash your board with your application.

Changes required to files

The following section describes the actual changes required to the aforementioned files for enabling additional DHCP options.

aceparams.c

In the static constant `dhcp_desired_params`, add FQDN (all upper case) if it is not already there.

acecallbacks.c

In this example my device is setting a FQDN name. This definition code is added to `acecallbacks.c` just after the typedef struct called `interfaceAddrInfo`.

Create a FQDN typedef

```
typedef struct _FQDN_STRUCT
{
    unsigned char theCode;           // option code, must be 81
    unsigned char theLen;           // length of remaining bytes
    unsigned char theFlags;        // one byte of flags
    unsigned char theRCode1;       // MBZ
    unsigned char theRCode2;       // MBZ
    char theFQNDData[FQDN_MAX_LEN]; // octet string of FQDN data
} bsp_fqdn_struct;
```

You'll need a new buffer big enough to hold the existing DHCP packet PLUS the new option

```
char * theNewDHCPConfigBuffer = NULL; // will replace existing buffer and be big
// enough to hold options
```

Adding DHCP options to a DHCP packet

Next you'll want to fill in an FQDN structure with the requisite data

```
static bsp_fqdn_struct theFQDNDataBuffer =
    {81, // option 81
      130, // 130 bytes after length
      0x01, // N = 0, E = 1, O = 0, S = 1
      0x0, // MBZ
      0x0, // MBZ
      {'w','a','l','t','h','a','m','.',
       'd','i','g','i','.',
       'c','o','m'
      }
    };
```

Adding DHCP options to a DHCP packet

The bulk of the changes are isolated in function `dhcpConfigToProtoInfo`. I have included a modified `dhcpConfigToProtoInfo` function here. The changed code is further isolated with a `#ifdef JZW_DDNS` and a `#endif`

```
static BOOLEAN dhcpConfigToProtoInfo(aceProtocolInfo *proto_info, configAceDhcpInfo
*dhcp_config)
{
    BOOLEAN result = FALSE;

#ifdef JZW_DDNS
    char * theBeginningPointer = NULL; // keep track of the beginning of the buffer
    int  sizeOfBuffer = (sizeof(bsp_fqdn_struct) + sizeof(configAceDhcpInfo));
#endif

    if (dhcp_config->isConfigValid && dhcp_config->isEnabled)
    {
        memcpy(&proto_info->proto_start_info, &dhcp_config->startInfo, sizeof proto_info-
>proto_start_info);
        // set things up for the new option
#ifdef JZW_DDNS
        // the new way
        memset(theNewDHCPConfigBuffer, 0, sizeOfBuffer);
        theBeginningPointer = theNewDHCPConfigBuffer;
        proto_info->proto_specific_info = theNewDHCPConfigBuffer;
        //get the existing dhcp option data
        memcpy((char *)theNewDHCPConfigBuffer, (char *)dhcp_config, (sizeof(configAceDhcpInfo)));
        // get ourselves to where the new option will begin
        theBeginningPointer = theBeginningPointer + sizeof(configAceDhcpInfo);
        // move in the new stuff
        memcpy(theBeginningPointer, &theFQDNDataBuffer, sizeof(bsp_fqdn_struct));
#else
        // the original way
        proto_info->proto_specific_info = dhcp_config;
#endif
#ifdef JZW_DDNS
        // set up for new size
        proto_info->proto_specific_info_size = sizeOfBuffer;
#else
        // set up for old size
        proto_info->proto_specific_info_size = sizeof *dhcp_config;
#endif
        proto_info->proto_event_info_size = ace_dhcp_event_info_size;
        proto_info->start_protocol_fn = ace_start_dhcp;
        proto_info->stop_protocol_fn = ace_stop_dhcp;
        proto_info->get_addr_fn = ace_get_dhcp_addr_info;
        proto_info->release_addr_fn = ace_release_dhcp_addr;
        if (dhcp_config->arp_reply_timeout >= 0)
            proto_info->detect_addr_conflict = FALSE;
        else
            proto_info->detect_addr_conflict = TRUE;
        result = TRUE;
    }
    return result;
}
```

Adding DHCP options to a DHCP packet

The buffer for holding the new DHCP configuration message should be allocated in function `customizeStartAce`. An example of a modified `customizeStartAce` function is included herein. As before the updated code is isolated with `#ifdef JZW_DDNS` and `#endif`.

```
int customizeStartAce(void)
{
    static aceProtocolInfo protoInfo[CONFIG_ACE_MAX_INTERFACES][ACE_MAX_PROTOCOLS];
    static aceConfigInfo aceConfig;
    int result, faceIndex, protoCount = 0, ifCount = 0;

#ifdef JZW_DDNS
    int size1 = (sizeof(bsp_fqdn_struct) + sizeof(configAceDhcpInfo));
#endif
    customizeAceCreateLock();

#ifdef JZW_DDNS
    // allocate space for new DHCP options
    if(theNewDHCPConfigBuffer == NULL)
    {
        theNewDHCPConfigBuffer = (char *)malloc(size1);
        if(theNewDHCPConfigBuffer == NULL) // malloc failed
        {
            printf("customizeStartAce - ace params buffer malloc failed\n");
            return -1;
        }
        // if ok, zero out the buffer
        memset(theNewDHCPConfigBuffer, 0, size1);
    }
#endif

    result = customizeAceGetConfig(&aceConfig);

    if (result != BP_SUCCESS)
    {
        customizeErrorHandler(ERROR_ACE_FAILURE, ERROR_SUBCODE_BAD_ACE_CONFIG);
    }
    result = aceInitialize();

#ifdef BSP_ENABLE_ADDR_CONFLICT_DETECTION == TRUE
    aceRegisterAddrConflictCallback(customizeAceAddrConflict);
    NARegisterAddrConflictCallback(NAAddrConflict);
#endif

    ip_configured = IP_STATE_RESTARTING;

    getPresentIpConfig(&default_ip_config);

    for (faceIndex = 0; (faceIndex < CONFIG_ACE_MAX_INTERFACES) && (result == 0); faceIndex++)
    {
        aceConfigInterfaceInfo *nif = &aceConfig.ace_interface_info[faceIndex];

        protoCount = aceInterfaceConfigToProtoTable(nif, &protoInfo[ifCount][0]);

        if (protoCount > 0)
        {
```

Adding DHCP options to a DHCP packet

```
char *pname = malloc(strlen(nif->ifname) + 1);
if (pname == NULL)
{
    result = ENOMEM;
    break;
}
memcpy(pname, nif->ifname, strlen(nif->ifname) + 1);

customizeAceLock();

if_addr_info[interface_count].ifname = pname;
interface_count++;

ip_add_static_route(nif->ifname, 0, 0, 0, 1, ADD_ROUTE);

result = NAIconfigBringDeviceUp(nif->ifname);
if (result == 0)
{
    result = aceStart(nif->ifname, &protoInfo[ifCount][0], protoCount,
                    customizeAceHaveAddress,
                    customizeAceHaveAnotherAddress,
                    customizeAceLostAddress);
}
customizeAceUnlock();

ifCount++;
}
}
return result;
}
```


Adding DHCP options to a DHCP packet

As stated earlier, you must now rebuild your bsp. Following the rebuilding of your bsp, you must rebuild your application. If you are running from flash, you need to update both the bootloader (rom.bin in your platforms directory) and your application, on your flash device before testing.

Source Files: [v6x_add_options.zip](#)