Adding DHCP options to NET+OS (V7.X) in an IAM 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, IAM (Internet Address Manager), 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 V7.X. That is, versions of NET+OS that run with treck stack and IAM. Thus versions of NET+OS produced before V7.0 (versions that run the fusion stack and use ACE) 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 IAM. 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 (V7.X)

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BOOTP has more or less been replaced by DHCP.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.

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----------	----------→

Theory of Operation

In version V7.X, the supported method is to create a completely new options buffer and override the existing DHCP options buffer using the IAM function call customizeIamSetDhcpConfig(). This function call is described in the API reference guide under the AIM heading. We recommend making the call to this function PRIOR TO starting up either treck or TCP/IP. Thus upon the first DHCP request, the new options buffer will be used. We found that the easiest way to get the existing DHCP options buffer is to boot your device in DHCP mode and look at the request using an Ethernet analyzer. Then use the analyzer data to create the buffer. Alternatively, the source code included with this paper, includes an example showing how to do this. You could copy that software, should it suffice for your usage model.

In the code you will notice that a buffer is allocated for holding the options data and this is passed to the customizeIamSetDhcpConfig() call. Since this is only called once, at system startup, this will not cause a memory leak.

File(s) that must be modified/enhanced

To enable the adding of options to an outgoing DHCP packet only one file needs to be modified, namely bsproot.c. This file can be found in the following directory:

<your_netos_directory>/src/bsp/common

After modifying this file, 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 file for enabling additional DHCP options.

bsproot.c

In the includes section, I include the file namacro.h. this allows me to use the macro asizeof.

Before the code starts I define the following to hold the data for the DHCP options:

```
unsigned char newDhcpOptionsTable[] =
{55,14,1,3,6,12,15,28,42,40,38,12,37,39,19,26,81,24,1,0,0,'d',
'o','m','a','i','n','_','n','a','m','e','.','d','i','g','i','.',
'c','o','m','.'
};
```

This contains the standard DHCP options that Digi normally uses plus it includes FQDN with an FQDN name of domain_name.digi.com.

The added variables are as follows:

```
char * theDHCPBuffer;
int theLength = 0;
NaIamDhcpParams_t * theDhcpConfigBuffer;
int callStatus = 0;
```


The actual code for changing the DHCP options is the following:

```
/* get number of entries of DHCP options*/
        theLength = asizeof(newDhcpOptionsTable);
/* allocate space for the temp buffer */
theDHCPBuffer = (char *)malloc(sizeof(NaIamDhcpParams_t));
        if(theDHCPBuffer == NULL)
        {
           /* unable to malloc space */
   netosFatalError ("Unable to allocate space for DHCP options", 1, 1);
        }
        theDhcpConfigBuffer = (NaIamDhcpParams_t *)theDHCPBuffer;

theDhcpConfigBuffer->isEnabled = TRUE;
        /* copy options from local array into buffer */

        memcpy(theDhcpConfigBuffer->optionsBuffer, newDhcpOptionsTable, theLength);
        theDhcpConfigBuffer->optionsLength = theLength;
        callStatus= customizeIamSetDhcpConfig("eth0", theDhcpConfigBuffer);
        if(callStatus != BP_SUCCESS)
        {
   netosFatalError ("Setting of new DHCP options failed", 1, 1);
        }
```

Caveats

You must not include any of the following options in your DHCP options buffer. Doing so will cause the treck stack to reject your options list and no DHCP requests will be sent out from your device:

Option Number	Option Name
52	Option Overload
53	DHCP Message Type
58	Renewal (T1) Time Value
59	Rebinding (T2) Time Value
255	End Option

Final Steps

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: v7x_add_options.zip