



**AutoIP**

# **User Guide**

**Express Logic, Inc.**

858.613.6640  
Toll Free 888.THREADX  
FAX 858.521.4259

[www.expresslogic.com](http://www.expresslogic.com)

**©2002-2016 by Express Logic, Inc.**

All rights reserved. This document and the associated NetX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden. Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of NetX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

**Trademarks**

NetX, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc. ThreadX is a registered trademark of Express Logic, Inc.

All other product and company names are trademarks or registered trademarks of their respective holders.

**Warranty Limitations**

Express Logic, Inc. makes no warranty of any kind that the NetX products will meet the USER's requirements, or will operate in the manner specified by the USER, or that the operation of the NetX products will operate uninterrupted or error free, or that any defects that may exist in the NetX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the NetX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty, and licensee may not rely on any such information or advice.

Part Number: 000-1054

Revision 5.10

# Contents

---

Chapter 1 Introduction to AutoIP .....	4
AutoIP Requirements .....	4
AutoIP Constraints .....	4
AutoIP Protocol Implementation .....	4
AutoIP Address Change .....	5
AutoIP RFCs .....	6
Chapter 2 Installation and Use of AutoIP .....	7
Product Distribution .....	7
AutoIP Installation .....	7
Using AutoIP .....	7
Small Example System .....	8
Configuration Options.....	11
Chapter 3 Description of AutoIP Services .....	13
nx_auto_ip_create.....	14
nx_auto_ip_delete .....	16
nx_auto_ip_get_address .....	17
nx_auto_ip_set_interface .....	18
nx_auto_ip_start.....	19
nx_auto_ip_stop .....	21

# Chapter 1

## Introduction to AutoIP

The AutoIP Protocol is a protocol designed for dynamically configuring IPv4 addresses on a local network. AutoIP is a simple protocol that utilizes ARP capabilities to perform its automatic IP address assignment function. AutoIP allocates addresses in the range of 169.254.1.0 through 169.254.254.255.

## AutoIP Requirements

In order to function properly, the NetX AutoIP package requires that a NetX IP instance has already been created. In addition, ARP must be enabled on that same IP instance. The NetX AutoIP package has no further requirements.

## AutoIP Constraints

The NetX AutoIP protocol implements the requirements of the RFC3927 standard. However, there are the following constraints:

1. If NetX DHCP is used, the DHCP thread must be created with a higher priority than both the NetX IP instance thread and the AutoIP thread.
2. NetX AutoIP does not provide a mechanism for old IP addresses to continue being used.
3. When the IP address changes, the application is responsible for tearing down any existing TCP connections and re-establishing them on the new IP address.

## AutoIP Protocol Implementation

The NetX AutoIP protocol first selects a random address within the AutoIP IPv4 address range of 169.254.1.0 through 169.254.254.255. Alternatively, the application may force a starting IP address by providing

it to the ***nx\_auto\_ip\_start*** function. This is useful in situations where an AutoIP address was successfully used in a prior run.

Once an AutoIP address is selected, NetX AutoIP sends out a series of ARP probes for the selected address. An ARP probe consists of an ARP request message with the sender address set to 0.0.0.0 and the target address set to the desired AutoIP address. A series of these ARP probes are sent (the actual number is determined by the define `NX_AUTO_IP_PROBE_NUM`). If another network node responds to this probe or sends an identical probe for the same address, a new AutoIP address is randomly selected within the AutoIP IPv4 address range and the probe processing repeats.

If `NX_AUTO_IP_PROBE_NUM` probes are sent without any responses, NetX AutoIP issues a series of ARP announcements for the selected address. An ARP announcement consists of an ARP request message with both the sender and target address in the ARP message set to the selected AutoIP address. A series of ARP announcement messages are sent, corresponding to the define `NX_AUTO_IP_ANNOUNCE_NUM`. If another network node responds to an announce message or sends an identical announcement for the same address, a new AutoIP address is randomly selected within the AutoIP IPv4 address range and the probe processing starts over.

When the probe and announcement completes without any detected conflicts, the selected AutoIP address is considered valid and the associated IP instance is setup with this address.

## AutoIP Address Change

As mentioned before, NetX AutoIP changes the IP instance address after successful probe and announcement processing. Monitoring for this case is not terribly important. However, it is possible to have the AutoIP address change in the future. Potential causes include future AutoIP address conflicts as well as DHCP address resolution. In order to process these potential situations properly, the application should use the following NetX API to alert it of any and all IP address changes:

```
nx_ip_address_change_notify(NX_IP *ip_ptr,  
    VOID (*ip_address_change_notify)(NX_IP *,VOID*),  
    VOID *additional_info);
```

The processing in the supplied *ip\_address\_change\_notify* function must either restart the NetX AutoIP processor or disable it if DHCP has subsequently resolved the IP address. Please refer to the *Small Example System* section for sample processing.

## **AutoIP RFCs**

NetX AutoIP is compliant with RFC3927 and related RFCs.

## Chapter 2

# Installation and Use of AutoIP

This chapter contains a description of various issues related to installation, setup, and usage of the NetX AutoIP component.

## Product Distribution

NetX AutoIP is shipped on a single CD-ROM compatible disk. The package includes three source files, one include file, and a PDF file that contains this document, as follows:

<b><code>nx_auto_ip.h</code></b>	Header file for NetX AutoIP
<b><code>nx_auto_ip.c</code></b>	C Source file for NetX AutoIP
<b><code>demo_netx_auto_ip.c</code></b>	C Source file for NetX AutoIP Demo
<b><code>nx_auto_ip.pdf</code></b>	PDF description of NetX AutoIP

## AutoIP Installation

In order to use NetX AutoIP, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory "*\threadx\arm7\green*" then the *nx\_auto\_ip.h*, *nx\_auto\_ip.c*, and *demo\_netx\_auto\_ip.c* files should be copied into this directory.

## Using AutoIP

Using NetX AutoIP is easy. Basically, the application code must include *nx\_auto\_ip.h* after it includes *tx\_api.h* and *nx\_api.h*, in order to use ThreadX and NetX. Once *nx\_auto\_ip.h* is included, the application code is then able to make the AutoIP function calls specified later in this guide. The application must also include *nx\_auto\_ip.c* in the build process. These files must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX AutoIP.

Note also that since AutoIP utilizes NetX ARP services, ARP must be enabled with the *nx\_arp\_enable* call prior to using AutoIP.

## Small Example System

An example of how easy it is to use NetX AutoIP is described in Figure 1.1, which appears below. In this example, the AutoIP include file *nx\_auto\_ip.h* is brought in at line 002. Next, the NetX AutoIP instance is created in “*tx\_application\_define*” at line 090. Note that the NetX AutoIP control block “*auto\_ip\_0*” was defined previously as a global variable at line 015. After successful creation, an NetX AutoIP is started at line 098. The IP address change callback function processing starts at line 105, which is used to handle subsequent conflicts or possible DHCP address resolution.

Note the example below assumes the host device is a single-homed device. For a multihomed device, the host application can use the NetX AutoIP service *nx\_auto\_ip\_interface\_set* to specify a secondary network interface to probe for an IP address. See the **NetX User Guide** for more details on setting up multihomed applications. Note further that the host application should use the NetX API *nx\_status\_ip\_interface\_check* to verify AutoIP has obtained an IP address.

```

000 #include "tx_api.h"
001 #include "nx_api.h"
002 #include "nx_auto_ip.h"
003
004 #define      DEMO_STACK_SIZE      4096
005
006 /* Define the ThreadX and NetX object control blocks... */
007
008 TX_THREAD      thread_0;
009 NX_PACKET_POOL pool_0;
010 NX_IP          ip_0;
011
012
013 /* Define the AUTO IP structures for the IP instance. */
014
015 NX_AUTO_IP      auto_ip_0;
016
017
018 /* Define the counters used in the demo application... */
019
020 ULONG          thread_0_counter;
021 ULONG          address_changes;
022 ULONG          error_counter;
023
024
025 /* Define thread prototypes. */
026
027 void thread_0_entry(ULONG thread_input);
028 void ip_address_changed(NX_IP *ip_ptr, VOID *auto_ip_address);
029 void _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
030
031
032 /* Define main entry point. */
033
034 int main()
035 {
036
037     /* Enter the ThreadX kernel. */
038     tx_kernel_enter();
039 }
040
041
042 /* Define what the initial system looks like. */
043

```



```

044 void    tx_application_define(void *first_unused_memory)
045 {
046
047     CHAR    *pointer;
048     UINT     status;
049
050
051     /* Setup the working pointer. */
052     pointer = (CHAR *) first_unused_memory;
053
054     /* Create the main thread. */
055     tx_thread_create(&thread_0, "thread 0", thread_0_entry, 0,
056                     pointer, DEMO_STACK_SIZE,
057                     16, 16, 1, TX_AUTO_START);
058
059     pointer = pointer + DEMO_STACK_SIZE;
060
061     /* Initialize the NetX system. */
062     nx_system_initialize();
063
064     /* Create a packet pool. */
065     status = nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", 128,
066                                   pointer, 4096);
067     pointer = pointer + 4096;
068
069     if (status)
070         error_counter++;
071
072     /* Create an IP instance. */
073     status = nx_ip_create(&ip_0, "NetX IP Instance 0", IP_ADDRESS(0, 0, 0, 0),
074                          0xFFFFFFFFUL, &pool_0, _nx_ram_network_driver,
075                          pointer, 4096, 1);
076     pointer = pointer + 4096;
077
078     if (status)
079         error_counter++;
080
081     /* Enable ARP and supply ARP cache memory for IP Instance 0. */
082     status = nx_arp_enable(&ip_0, (void *) pointer, 1024);
083     pointer = pointer + 1024;
084
085     /* Check ARP enable status. */
086     if (status)
087         error_counter++;
088
089     /* Create the AutoIP instance for IP Instance 0. */
090     status = nx_auto_ip_create(&auto_ip_0, "AutoIP 0", &ip_0, pointer, 4096, 1);
091     pointer = pointer + 4096;
092
093     /* Check AutoIP create status. */
094     if (status)
095         error_counter++;
096
097     /* Start AutoIP instances. */
098     status = nx_auto_ip_start(&auto_ip_0, 0 /*IP_ADDRESS(169,254,254,255)*/);
099
100     /* Check AutoIP start status. */
101     if (status)
102         error_counter++;
103
104     /* Register an IP address change function for IP Instance 0. */
105     status = nx_ip_address_change_notify(&ip_0, ip_address_changed,
106                                          (void *) &auto_ip_0);
107
108     /* Check IP address change notify status. */
109     if (status)
110         error_counter++;
111 }
112
113
114 /* Define the test thread. */
115
116 void    thread_0_entry(ULONG thread_input)
117 {
118
119     UINT     status;
120     ULONG     actual_status;
121
122
123     /* Wait for IP address to be resolved. */
124     do

```

```

125     {
126         /* Call IP status check routine. */
127         status = nx_ip_status_check(&ip_0, NX_IP_ADDRESS_RESOLVED,
128                                     &actual_status, 10000);
129     } while (status != NX_SUCCESS);
130
131     /* Since the IP address is resolved at this point, the application
132     can now fully utilize NetX! */
133
134     while(1)
135     {
136         /* Increment thread 0's counter. */
137         thread_0_counter++;
138
139         /* Sleep... */
140         tx_thread_sleep(10);
141     }
142 }
143
144 void ip_address_changed(NX_IP *ip_ptr, VOID *auto_ip_address)
145 {
146     ULONG ip_address;
147     ULONG network_mask;
148     NX_AUTO_IP *auto_ip_ptr;
149
150     /* Setup pointer to auto IP instance. */
151     auto_ip_ptr = (NX_AUTO_IP *) auto_ip_address;
152
153     /* Pickup the current IP address. */
154     nx_ip_address_get(ip_ptr, &ip_address, &network_mask);
155
156     /* Determine if the IP address has changed back to zero. If so,
157     make sure the AutoIP instance is started. */
158     if (ip_address == 0)
159     {
160         /* Get the last AutoIP address for this node. */
161         nx_auto_ip_get_address(auto_ip_ptr, &ip_address);
162
163         /* Start this AutoIP instance. */
164         nx_auto_ip_start(auto_ip_ptr, ip_address);
165     }
166
167     /* Determine if IP address has transitioned to a non local IP address. */
168     else if ((ip_address & 0xFFFFF000UL) != IP_ADDRESS(169, 254, 0, 0))
169     {
170         /* Stop the AutoIP processing. */
171         nx_auto_ip_stop(auto_ip_ptr);
172     }
173
174     /* Increment a counter. */
175     address_changes++;
176 }

```

Figure 1.1 Example of AutoIP use with NetX

# Configuration Options

There are several configuration options for building NetX AutoIP. Following is a list of all options, where each is described in detail:

Define	Meaning
<b>NX_DISABLE_ERROR_CHECKING</b>	Defined, this option removes the basic AutoIP error checking. It is typically used after the application has been debugged.
<b>NX_AUTO_IP_PROBE_WAIT</b>	The number of seconds to wait before sending first probe. By default, this value is defined as 1.
<b>NX_AUTO_IP_PROBE_NUM</b>	The number of ARP probes to send. By default, this value is defined as 3.
<b>NX_AUTO_IP_PROBE_MIN</b>	The minimum number of seconds to wait between sending probes. By default, this value is defined as 1.
<b>NX_AUTO_IP_PROBE_MAX</b>	The maximum number of seconds to wait between sending probes. By default, this value is defined as 2.
<b>NX_AUTO_IP_MAX_CONFLICTS</b>	The number of AutoIP conflicts before increasing processing delays. By default, this value is defined as 10.
<b>NX_AUTO_IP_RATE_LIMIT_INTERVAL</b>	The number of seconds to extend the wait period when the total number of conflicts is exceeded. By default, this value is defined as 60.
<b>NX_AUTO_IP_ANNOUNCE_WAIT</b>	The number of seconds to wait before sending announcement. By default, this value is defined as 2.

<b>NX_AUTO_IP_ANNOUNCE_NUM</b>	The number of ARP announces to send. By default, this value is defined as 2.
<b>NX_AUTO_IP_ANNOUNCE_INTERVAL</b>	The number of seconds to wait between sending announces. By default, this value is defined as 2.
<b>NX_AUTO_IP_DEFEND_INTERVAL</b>	The number of seconds to wait between defense announces. By default, this value is defined as 10.

## Chapter 3

# Description of AutoIP Services

This chapter contains a description of all NetX AutoIP services (listed below) in alphabetic order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the **NX\_DISABLE\_ERROR\_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

`nx_auto_ip_create`  
*Create AutoIP instance*

`nx_auto_ip_delete`  
*Delete AutoIP instance*

`nx_auto_ip_get_address`  
*Get current AutoIP address*

`nx_auto_ip_set_interface`  
*Set IP interface needing an AutoIP address*

`nx_auto_ip_start`  
*Start AutoIP processing*

`nx_auto_ip_stop`  
*Stop AutoIP processing*

## **nx\_auto\_ip\_create**

Create AutoIP instance

### **Prototype**

```
UINT nx_auto_ip_create(NX_AUTO_IP *auto_ip_ptr, CHAR *name,
                      NX_IP *ip_ptr, VOID *stack_ptr, ULONG stack_size,
                      UINT priority);
```

### **Description**

This service creates an AutoIP instance on the specified IP instance.

### **Input Parameters**

<b>auto_ip_ptr</b>	Pointer to AutoIP control block.
<b>name</b>	Name of AutoIP instance.
<b>ip_ptr</b>	Pointer to IP instance.
<b>stack_ptr</b>	Pointer to AutoIP thread stack area.
<b>stack_size</b>	Size of the AutoIP thread stack area.
<b>priority</b>	Priority of the AutoIP thread. <i>Note that if DHCP is used, the DHCP thread must have a higher priority than the IP instance thread and the AutoIP thread.</i>

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful AutoIP create.
<b>NX_AUTO_IP_ERROR</b>	(0xA00)	AutoIP create error.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid AutoIP, ip_ptr, or stack pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Initialization, Threads

## Example

```
/* Create the AutoIP instance "auto_ip_0" on "ip_0". */
status = nx_auto_ip_create(&auto_ip_0, "AutoIP 0", &ip_0, pointer, 4096, 1);

/* If status is NX_SUCCESS an AutoIP instance was successfully
   created. */
```

## See Also

`nx_auto_ip_delete`, `nx_auto_ip_set_interface`, `nx_auto_ip_get_address`,  
`nx_auto_ip_start`, `nx_auto_ip_stop`

## **nx\_auto\_ip\_delete**

Delete AutoIP instance

### **Prototype**

```
UINT nx_auto_ip_delete(NX_AUTO_IP *auto_ip_ptr);
```

### **Description**

This service deletes a previously created AutoIP instance on the specified IP instance.

### **Input Parameters**

**auto\_ip\_ptr**            Pointer to AutoIP control block.

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful AutoIP delete.
<b>NX_AUTO_IP_ERROR</b>	(0xA00)	AutoIP delete error.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid AutoIP pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Threads

### **Example**

```
/* Delete the AutoIP instance "auto_ip_0." */
status = nx_auto_ip_delete(&auto_ip_0);

/* If status is NX_SUCCESS an AutoIP instance was successfully
   deleted. */
```

### **See Also**

nx\_auto\_ip\_create, nx\_auto\_ip\_set\_interface, nx\_auto\_ip\_get\_address,  
nx\_auto\_ip\_start, nx\_auto\_ip\_stop



# **nx\_auto\_ip\_get\_address**

Get current AutoIP address

## **Prototype**

```
UINT nx_auto_ip_get_address(NX_AUTO_IP *auto_ip_ptr,
                           ULONG *local_ip_address);
```

## **Description**

This service retrieves the currently setup AutoIP address. If there isn't one, an IP address of 0.0.0.0 is returned.

## **Input Parameters**

<b>auto_ip_ptr</b>	Pointer to AutoIP control block.
<b>local_ip_address</b>	Destination for return IP address.

## **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful AutoIP address get.
<b>NX_AUTO_IP_NO_LOCAL</b>	(0xA01)	No valid AutoIP address.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid AutoIP pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

## **Allowed From**

Initialization, Timers, Threads, ISRs

## **Example**

```
ULONG local_address;

/* Get the AutoIP address resolved by the instance "auto_ip_0." */
status = nx_auto_ip_get_address(&auto_ip_0, &local_address);

/* If status is NX_SUCCESS the local IP address is in "local_address." */
```

## **See Also**

nx\_auto\_ip\_create, nx\_auto\_ip\_set\_interface, nx\_auto\_ip\_delete,  
nx\_auto\_ip\_start, nx\_auto\_ip\_stop

## **nx\_auto\_ip\_set\_interface**

Set network interface for AutoIP

### **Prototype**

```
UINT nx_auto_ip_set_interface(NX_AUTO_IP *auto_ip_ptr,
                             UINT interface_index);
```

### **Description**

This service sets the index for the network interface AutoIP will probe for a network IP address. The default is zero (the primary network interface). Only applicable for multihomed devices.

### **Input Parameters**

<b>auto_ip_ptr</b>	Pointer to AutoIP control block.
<b>interface_index</b>	Interface to probe IP address for

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful AutoIP interface set
<b>NX_AUTO_IP_BAD_INTERFACE_INDEX</b>	(0xA02)	Invalid network interface
<b>NX_PTR_ERROR</b>	(0x16)	Invalid AutoIP pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Initialization, Timers, Threads, ISRs

### **Example**

```
ULONG interface_index;

/* Set the network interface on which AutoIP probes for host address. */
status = nx_auto_ip_set_interface(&auto_ip_0, interface_index);

/* If status is NX_SUCCESS the network interface is valid and set in the AutoIP
control block auto_ip_0. */
```

### **See Also**

nx\_auto\_ip\_create, nx\_auto\_ip\_get\_address, nx\_auto\_ip\_delete,  
nx\_auto\_ip\_start, nx\_auto\_ip\_stop

## **nx\_auto\_ip\_start**

Start AutoIP processing

### **Prototype**

```
UINT nx_auto_ip_start(NX_AUTO_IP *auto_ip_ptr,
                     ULONG starting_local_address);
```

### **Description**

This service starts the AutoIP protocol on a previously created AutoIP instance.

### **Input Parameters**

**auto\_ip\_ptr**            Pointer to AutoIP control block.

**starting\_local\_address**

Optional AutoIP starting address. A value of IP\_ADDRESS(0,0,0,0) specifies that a random AutoIP address should be derived. Otherwise, if a valid AutoIP address is specified, NetX AutoIP attempts to assign that address.

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful AutoIP start.
<b>NX_AUTO_IP_ERROR</b>	(0xA00)	AutoIP start error.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid AutoIP pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Initialization, Threads

### **Example**

```
/* Start the AutoIP instance "auto_ip_0." */
status = nx_auto_ip_start(&auto_ip_0, IP_ADDRESS(0,0,0,0));

/* If status is NX_SUCCESS an AutoIP instance was successfully
   started. */
```

## See Also

`nx_auto_ip_create`, `nx_auto_ip_set_interface`, `nx_auto_ip_delete`,  
`nx_auto_ip_get_address`, `nx_auto_ip_stop`

## **nx\_auto\_ip\_stop**

Stop AutoIP processing

### **Prototype**

```
UINT nx_auto_ip_stop(NX_AUTO_IP *auto_ip_ptr);
```

### **Description**

This service stops the AutoIP protocol on a previously created and started AutoIP instance. This service is typically used when the IP address is changed via DHCP or manually to a non-AutoIP address.

### **Input Parameters**

**auto\_ip\_ptr**            Pointer to AutoIP control block.

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful AutoIP stop.
<b>NX_AUTO_IP_ERROR</b>	(0xA00)	AutoIP stop error.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid AutoIP pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Threads

### **Example**

```
/* Stop the AutoIP instance "auto_ip_0." */
status = nx_auto_ip_stop(&auto_ip_0);

/* If status is NX_SUCCESS an AutoIP instance was successfully
   stopped. */
```

### **See Also**

nx\_auto\_ip\_create, nx\_auto\_ip\_set\_interface, nx\_auto\_ip\_delete,  
nx\_auto\_ip\_get\_address, nx\_auto\_ip\_start