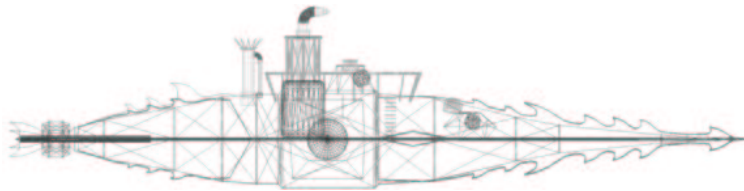




# Operation of Nshell and HTTP server



## "The Nautilus Project"

### *PCS*

Revision 0.5L

<b>Authors:</b>	Jens Altmann	<b>Division:</b>	Personal Connectivity Solutions	
	Sören Schneider		<b>Project:</b>	Nautilus
	Frank Schücke		<b>Document name:</b>	operation.pdf
			<b>Last changed:</b>	August 30, 2003
<b>Distribution:</b>	AMD WLAN project team, contractors and customers			
<b>Preface:</b>	This specification provides information about the usage of development tools delivered with beta releases of the Embedded Driver for AMD WLAN. These tools are intended to be used by development teams to enable them to use/configure the Embedded Driver.			

© 2002 Advanced Micro Devices, Inc. All Rights Reserved.

Advanced Micro Devices, Inc. ("AMD") reserves the right to make changes in its products without notice in order to improve design or performance characteristics.

The information in this publication is believed to be accurate at the time of publication, but AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication or the information contained herein, and reserves the right to make changes at any time, without notice. AMD disclaims responsibility for any consequences resulting from the use of the information included in this publication.

This publication neither states nor implies any representations or warranties of any kind, including but not limited to, any implied warranty of merchantability or fitness for a particular purpose. AMD products are not authorized for use as critical components in life support devices or systems without AMD's written approval. AMD assumes no liability whatsoever for claims associated with the sale or use (including the use of engineering samples) of AMD products except as provided in AMD's Terms and Conditions of Sale for such products.

#### Trademarks:

AMD, the AMD logo, AMD Athlon, AMD Duron and combinations thereof, 3DNow!, AMD EasyNow!, and AMD PowerNow! are trademarks of Advanced Micro Devices, Inc. AMD-K6 and AMD-K6-III are registered trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

# Contents

<b>1</b>	<b>Operation</b>	<b>5</b>
1.1	General . . . . .	5
1.2	Linux . . . . .	5
1.2.1	Parts of the release . . . . .	5
1.2.2	Install and load the device driver . . . . .	6
1.2.2.1	Installation on the target machine . . . . .	6
1.2.2.2	Loading the device driver . . . . .	6
1.2.3	Connect to the network interface . . . . .	7
1.3	Windows CE 4.1 . . . . .	8
1.3.1	Parts of the release . . . . .	8
1.3.2	Install and load the device driver . . . . .	8
1.4	Configure and use the Nautilus driver . . . . .	8
1.4.1	Configuration with Windows CE 4.1 Zero Config . . . . .	8
1.4.2	Configuration from Nshell . . . . .	9
1.4.2.1	Initialize the HW and setup the drivers MAC address . . . . .	10
1.4.2.2	Setup WEP . . . . .	11
1.4.2.3	Scan the wireless media for available access points . . . . .	12
1.4.2.4	Join to one of the found BSS . . . . .	13
1.4.2.5	Authenticate at the access point . . . . .	14
1.4.2.6	Associate to the BSS . . . . .	15
1.4.2.7	Example session . . . . .	15
1.4.3	Configuration via Web Interface . . . . .	19
1.4.3.1	Introduction . . . . .	19
1.4.3.2	Reset . . . . .	20
1.4.3.3	WLAN Configuration . . . . .	20
1.4.3.4	Searching for Networks . . . . .	23
1.4.3.5	Connecting to an existing Network . . . . .	26
1.4.3.6	Starting an Independent BSS . . . . .	28
1.5	Build own driver and applications . . . . .	30

1.5.1	Linux . . . . .	30
1.5.1.1	General . . . . .	30
1.5.1.2	Un-packing of the source distribution . . . . .	30
1.5.1.3	Set-up the environment . . . . .	31
1.5.1.4	Configure Nautilus . . . . .	32
1.5.1.5	Compile the driver and applications . . . . .	34
1.5.2	Windows CE . . . . .	35
1.5.2.1	Create a WinCe Platform . . . . .	35
1.5.2.2	Platform build process . . . . .	37

# Revision History

## Document Version

The version of all documents in this project is indicated in its version number as follows:

<major version>.<minor version><document state>

Where *major version*, the first digit is incremented and the last digit is reset to “0” when the document has been reviewed. *minor version*, the last digit is incremented, when changes have been incorporated in the document but have not been reviewed.

## Document State

The status of this document is indicated by the state field on the first page. This can take any of the following values:

E	<i>Editing</i> , the document is in preparation
R	<i>Review</i> , the document ready for review
A	<i>Approved</i> , the document has passed review and is updated
L	<i>Live</i> , the document has passed review, is updated based on the respective comments and will continuously be updated during the development process

The category 'Live' indicates that the document is a working document, and will be updated from time to time. It is correct at the time of writing.

## Document Revisions

0.1E	Document started
0.1R	First preliminary revision for review
0.2L	Updated for first release
0.3L	Added instructions how to build Linux and WinCE driver
0.4L	Internal release of Nautilus
0.5L	Preparation for release

# 1 Operation

## 1.1 General

The following description gives an overview how to load and operate the Nautilus WLAN driver.

## 1.2 Linux

### 1.2.1 Parts of the release

The following components are part of the release:

- AMD WLAN driver
  - `Nautilus.o` Device driver for AMD WLAN
- AMD WLAN applications
  - `HttpServer` Web server interface for driver configuration
  - `Nshell` Command line interface with MLME syntax for driver configuration
  - `Nset` Command line interface to set/change the drivers debug and trace level during operation (AMD internal).
  - `UTECClient` Universal Test System (UTE) client for remote configuration and testing (AMD internal test environment).

Main applications to control the driver are `HttpServer` and `Nshell`. The `Nset` and `UTECClient` application are used for AMD internal debugging purposes. The MLME interface syntax of `Nshell` is more flexible than configuring the driver using `HttpServer` but requires more knowledge about the parameters used and their allowed values. The sequence of following actions are important for the correct operation of the driver.

1. Install and load the device driver
2. Connect the driver to the network interface
3. Configure, join, authenticate and associate to the Wireless LAN

## 1.2.2 Install and load the device driver

### 1.2.2.1 Installation on the target machine

- Login as super user on the machine where the driver shall be installed and started (it is assumed the hardware is already plugged in)
- Open a terminal connected to the target machine

### 1.2.2.2 Loading the device driver

1. Copy the device driver and applications into a directory on the machine to be used (Nautilus.o, Nshell and HttpServer). When the device driver was already running on your system:
  - Check whether the device driver is already running on your system. The command `lsmod` gives an overview about the loaded modules on the system.
  - If the device driver is loaded, remove the older driver using: `rmmod Nautilus`. Before removing the device driver, make sure, there's no network interface connected to driver, i.e. close an existing network interface before removing the device driver. Assuming `eth1` is the network interface used by the device driver the command would be: `ifconfig down eth1`.
2. To install the new Nautilus driver change to the directory the driver was copied to and use: `insmod Nautilus.o` to load the device driver module. The driver creates a network interface using the first unused interface number. For example: If the system is equipped with one other network card using device `eth0`, the Nautilus device driver will use `eth1` for its interface created.
3. For Nshell IOCTL calls of all applications a device node is required. To create it, issue:

```
mknod /dev/nautilus0 c `grep nautilus /proc/devices | cut -d " " -f 1` 0
ln -s /dev/nautilus0 /dev/nautilus
```

You need to do this only once for each system in use. A shell script generating

the device entries above can be found in the `Linux/Tools` directory of the source distribution.

### 1.2.3 Connect to the network interface

For easy switching between Nshell and System-shell it's recommended to open a second terminal for communication with the operating system and a third terminal to watch the system log where trace messages of the device driver are directed to.

A continuous system log (usually `/var/log/messages`) can be shown by watching the tail of this file. Issue the following command:

```
tail -f /var/log/messages
```

Connect the network interface with Nautilus and configure the interface IP address. The address `192.168.1.1` is used here as example. Use the system shell for giving the command:

```
ifconfig eth1 192.168.1.1
```

In the example above, `eth1` stands for the network interface created, when the device driver was loaded. This is depending on your system. The name, especially the number, may be different.

The correct connection of the interface can be verified by displaying the network interface configuration. Use the following command:

```
ifconfig eth1
```

This will print you an overview about all connected network interfaces e.g:

```
eth1      Link encap:Ethernet  HWaddr 00:01:02:03:04:05
2          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::201:2ff:fe03:405/10 Scope:Link
4          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
6          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
8          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

## 1.3 Windows CE 4.1

### 1.3.1 Parts of the release

The following components are part of the release:

- Complete Platform to boot a complete system
  - `nk.bin` Complete platform binary to boot an WinCE system including driver and applications.
- Driver and Applications to be loaded on an existing platform
  - `Am1772.dll` Windows CE 4.1 device driver for AMD WLAN
  - `Am1772.reg` Windows CE 4.1 registry entries for AMD WLAN
  - `HttpServer.exe` Web server interface for driver configuration
  - `Nshell.exe` Command line interface with MLME syntax for driver configuration

For debug versions of driver and applications, the respective `*.map` and `*.pdb` symbol files are included in the binary distribution.

### 1.3.2 Install and load the device driver

Install the Complete Platform distribution using Microsoft Platform Builder or directly on a boot media. Please refer to the development board documentation for details. To install the Driver and Applications distribution, load the files to your system. Refer to your systems documentation on how to load a device driver. Start AMD WLAN applications from a Command Prompt.

## 1.4 Configure and use the Nautilus driver

### 1.4.1 Configuration with Windows CE 4.1 Zero Config

When the Nautilus AMD WLAN driver is loaded either during system startup or manually and the driver has been configured to use Windows CE 4.1 Zero Config (this has been done for the binary distribution), the system starts Zero Config and displays the configuration screen. The system automatically searches for available networks and

displays them. Select the network to connect to and set the respective parameters according to the network settings.

The following restrictions apply to the current state of `Zero Config` implementation:

- The IP address can be set only with DHCP
- DHCP may not work if a lot of debug output is issued. Reduce the debug output during the address negotiation phase to avoid DHCP to time out.
- Depending on the amount of debug output you may need to increase the beacon period. RECOMMENDATION: Use 500 ms beacon period.
- The system may hang when disconnecting from a network.

### 1.4.2 Configuration from Nshell

Nshell can be started with issuing `Nshell.exe` (Windows CE 4.1) or `./Nshell` (Linux). To exit from Nshell use the command `exit` or hit `Ctrl-c`.

To operate the system (Natilus device driver and AMD WLAN hardware) as INFRA-BSS station requires a 5 step setup. All steps are covered by MLME commands to be issued using Nshell. The sequence of these steps is important:

1. Initialize the hardware and setup the drivers MAC address
2. Setup WEP and WEP-keys
3. Scan the wireless media for available access points
4. Join to one of the found BSS
5. Authenticate at the access point
6. Associate to the BSS

#### 1.4.2.1 Initialize the HW and setup the drivers MAC address

During hardware initialization the MAC hardware address of the device used will be needed. To obtain the MAC hardware address from the driver, use the MLME\_GET.request command.

Syntax:

```
MLME MLME_GET.request(mibattribute)
```

- mibattribute = MIB attribute requested

Example:

```
MLME MLME_GET.request(dot11StationID)
```

A possible result might be:

```
MLME MLME_GET.confirm(SUCCESS, dot11StationID, 00:01:02:03:04:05)
```

Resetting the driver is the first step to be done. Use the MLME\_Reset.request command.

Syntax:

```
MLME MLME_RESET.request(macaddress, setdefaultmib)
```

- macaddress = six byte ethernet address
- setdefaultmib = (TRUE or FALSE) directs the driver to retain the current MIB settings or to reinitialize with default values

Example:

```
MLME MLME_RESET.request(00:01:02:03:04:05, FALSE)
```

### 1.4.2.2 Setup WEP

Before using WEP the respective WEP keys need to be set. in the WEP Default Key Table. For each default key to be set the appropriate index (1...4) needs to be set before setting the key itself. The default key is the key which is currently used for encryption. The key length may either be zero, 5 or 13 bytes. For setting the default key index and default key, the MLME\_SET.request command is used:

Syntax:

```
MLME MLME_SET.request(mibattribute, mibattributevalue)
```

- mibattribute = MIB attribute to be set
- mibattributevalue = MIB attribute value to be set

Example (setting default WEP key number 1):

```
MLME MLME_SET.request(dot11WEPDefaultKeyIndex, 1)
MLME MLME_SET.request(dot11WEPDefaultKeyValue, 12:34:56:78:9a)
```

Repeat this setting for all WEP keys needed (1..4). After setting the WEP key, the key used for transmission and reception of data. This is done by selecting the WEP key index (0..3) using the MLME\_SET.request command:

Example (setting WEP key index 0):

```
MLME MLME_SET.request(dot11WEPDefaultKeyID, 0)
```

**Note: Please note that the WEP key index starts with 0 (0..3) and the WEP key numbers start with 1 (1..4).**

Last step in setting up WEP is to enable the feature. Whether WEP is available for shared key authentication or to transmit data frames depends on the MIB attribute dot11PrivacyInvoked which must be set either to TRUE or to FALSE.

Example:

```
MLME MLME_SET.request(dot11PrivacyInvoked, TRUE)
```

### 1.4.2.3 Scan the wireless media for available access points

Scan will be done via the command `MLME MLME_SCAN.request`. Scan is not required to join to a network but it retrieves all necessary parameters for a successful network join.

Syntax:

```
MLME MLME_SCAN.request(bsstype, bssid, ssid, scantype, probedelay, channels,
    mintime, maxtime)
```

- `bsstype` = type of the BSS you want to scan (INFRASTRUCTURE, INDEPENDENT, ANY\_BSS)
- `bssid` = six byte ethernet address. This may be the broadcast address to scan for all visible BSS
- `ssid` = a string of up to 32 characters or an empty string if you want to scan for all visible BSS
- `scantype` = ACTIVE or PASSIVE. An active scan sends a probe request frame. A passive scan does only analyze received beacons
- `probedelay` = defines the delay in  $\mu$ s before transmitting a probe request during active scan
- `channels` = list of up to 8 channels to scan
- `mintime` = least number of time units (1TU = 1024 $\mu$ s) to stay on a channel even if a beacon or probe request has been received
- `maxtime` = upper limit of time units to stay on a channel

Example:

```
MLME MLME_SCAN.request( INFRASTRUCTURE, FF:FF:FF:FF:FF:FF,
    "", PASSIVE, 10, [1,3,4], 100, 1200)
```

A possible result might be:

```
MLME MLME_SCAN.confirm(((12:34:56:78:9A:BC, "test", INFRASTRUCTURE, 1000, 3,
    0xFFFFFFFF00000080, 0x000000000000D4F2F, (1), (0, 0, 0, 0), (0), 0x0001,
    [2, 4]))),SUCCESS)
```

The result of the `MLME_SCAN.request` is a set of all found BBS descriptions that match the requested parameters.

#### 1.4.2.4 Join to one of the found BSS

Join is the process of synchronizing the MAC hardware to a wireless LAN. This requires some configuration parameters which will be delivered with the `MLME_JOIN.request` command. This document describes only the parameters that are necessary to join an INFRA-BSS.

Syntax:

```
MLME MLME\_JOIN.request((bssid, ssid, bsstype, beaconperiod, dtimperiod,
    timestamp, localtime, phypset, cfpset, ibsspset, capainfo, bssbasicrateset),
    joinfailuretimeout, probedelay, operationalrateset)
```

- bssid = six byte ethernet address of the access point to join
- ssid = the SSID of the BSS to join
- bsstype = the type of the BSS (INFRASTRUCTURE, INDEPENDENT)
- beaconperiod = the beacon period in TUs of the BSS to join
- dtimperiod = the DTIM period in number of beacon intervals of the BSS to join
- timestamp = set always to 0x0000000000000000
- localtime = set always to 0x0000000000000000
- phypset = the RF channel of the BSS (1...14)
- cfpset = not used, set to zero (0, 0, 0, 0)
- ibsspset = only for IBSS, set to zero (0)
- capainfo = capability information, set to 0 for Infrastructure STA.
- bssbasicrateset = set of rates in number of 500kBits/s which must be supported by all stations in the BSS
- joinfailuretimeout = number of beacon intervals until the join is either successful or will be canceled
- probedelay = defines the delay in  $\mu$ s prior sending a probe request at active scan
- operationalrateset = set of rates in number of 500kBits/s which are supported by this station. Must contain at least all basic rates!

Example:

```
MLME MLME_JOIN.request((12:34:56:78:9A:BC, "test", INFRASTRUCTURE, 1000, 3,
0x0000000000000000, 0x0000000000000000, (1), (0, 0, 0, 0), (0), 0, [2, 4]),
2, 10, [2, 4])
```

There are three possible results for a Join command:

- **SUCCESS** the BSS has been successfully joined, you may proceed with Authentication
- **TIMEOUT** the BSS could not be joined. The reason might be wrong parameters or no reception of beacons
- **INVALID\_PARAMETERS** some of the parameters in the commandline don't match or are invalid .

#### 1.4.2.5 Authenticate at the access point

The current version supports OpenSystem authentication only. However, using WEP for TX, RX is possible. How to turn on WEP is explained above.

Syntax:

```
MLME MLME_AUTHENTICATE.request(peerstaaddr , authtype , timeout)
```

- **peerstaaddr** = six byte ethernet MAC address of the access point
- **authtype** = set to OPEN\_SYSTEM or SHARED\_KEY
- **timeout** = limit of time in TU after which the request will be terminated.

Example:

```
MLME MLME_AUTHENTICATE.request(12:34:56:78:9A:BC, OPEN_SYSTEM, 2000)
```

Result:

```
MLME MLME_AUTHENTICATE.confirm(12:34:56:78:9A:BC, OPEN_SYSTEM, SUCCESS)
```

### 1.4.2.6 Associate to the BSS

Association is the last step to setup a working connection via Wireless LAN. The current version of the device driver does not provide extensive sanity checks for capability information.

Syntax:

```
MLME MLME_ASSOCIATE.request(peerstaaddr, associatefailuretimeout, capainfo,
    listeninterval)
```

- peerstaaddr = six byte ethernet MAC address of the access point
- capainfo = set as used in the Join command
- associatefailuretimeout = limit of time in TU after which the request will be terminated.
- listeninterval = number of beacon intervals that may pass before the STA awakens and listens for the next beacon

Example:

```
MLME MLME_ASSOCIATE.request(12:34:56:78:9a:bc, 1000, 0, 10)
```

Result:

```
MLME MLME_ASSOCIATE.confirm(SUCCESS)
```

### 1.4.2.7 Example session

This section shows a log from an example session using the Nautilus device driver together with Nshell.

```
elsa:~/sample # lsmod
2 Module              Size Used by    Not tainted
  ipv6                123424 -1  (autoclean)
4  isa-pnp             27816  0  (unused)
  joydev              5728   0  (unused)
6  evdev               3904   0  (unused)
  input               3072   0  [joydev evdev]
8  usb-uhci            20996   0  (unused)
  usbcore             55136   1  [usb-uhci]
10 pcnet32             12896   1
```

```

12 mii                      1040    0 [pcnet32]
reiserfs                  158816    1
elsa:~/sample # insmod Nautilus.o
14 elsa:~/sample # lsmod
Module                    Size  Used by    Not tainted
16 Nautilus                181024    0 (unused)
   ipv6                   123424   -1 (autoclean)
18 isa-pnp                  27816    0 (unused)
   joydev                  5728    0 (unused)
20 evdev                    3904    0 (unused)
   input                   3072    0 [joydev evdev]
22 usb-uhci                 20996    0 (unused)
   usbcore                 55136    1 [usb-uhci]
24 pcnet32                  12896    1
   mii                     1040    0 [pcnet32]
26 reiserfs                158816    1

28 ...

30 elsa:~/sample # mknod /dev/nautilus0 c 'grep nautilus /proc/devices | \
    cut -d " " -f 1' 0
32 elsa:~/sample # ln -s /dev/nautilus0 /dev/nautilus

34 ...

36
elsa:~/sample # ./Nshell
38 APP_IF.enable(MLME_IND | UMIT_IND, SUCCESS)
nsh> exit
40 elsa:~/sample #

42 ...

44 elsa:~/sample # ifconfig eth1 192.168.68.1
elsa:~/sample # ifconfig
46 eth0      Link encap:Ethernet  HWaddr 00:30:84:6A:A8:39
            inet addr:172.20.36.105  Bcast:172.20.255.255  Mask:255.255.0.0
48            inet6 addr: fe80::230:84ff:fe6a:a839/10 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
50            RX packets:10062 errors:0 dropped:0 overruns:0 frame:0
            TX packets:2115 errors:0 dropped:0 overruns:0 carrier:0
52            collisions:0 txqueuelen:100
            RX bytes:2476211 (2.3 Mb)  TX bytes:338166 (330.2 Kb)
54            Interrupt:11 Base address:0xec00

```

```

56 eth1      Link encap:Ethernet  HWaddr 00:00:1A:18:ED:D7
            inet addr:192.168.68.1  Bcast:192.168.68.255  Mask:255.255.255.0
58            inet6 addr: fe80::200:1aff:fe18:edd7/10 Scope:Link
            UP BROADCAST MULTICAST  MTU:1500  Metric:1
60            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
62            collisions:0 txqueuelen:100
            RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
64
66 lo        Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128 Scope:Host
68            UP LOOPBACK RUNNING  MTU:16436  Metric:1
            RX packets:28 errors:0 dropped:0 overruns:0 frame:0
70            TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
72            RX bytes:2120 (2.0 Kb)  TX bytes:2120 (2.0 Kb)

74 elsa:~/sample #
76 ...

78 elsa:~/sample # ./Nshell
APP_IF.enable(MLME_IND | UMIT_IND, SUCCESS)
80 nsh> MLME MLME_GET.request(dot11StationID)
nsh> MLME MLME_GET.confirm(SUCCESS, dot11StationID, 00:00:1A:18:ED:D7)
82 nsh> MLME MLME_RESET.request(00:00:1A:18:ED:D7, FALSE)
nsh> MLME MLME_RESET.confirm(SUCCESS)
84 ...

86 nsh> MLME MLME_SET.request(dot11PrivacyInvoked,TRUE)
88 nsh> MLME MLME_SET.confirm(SUCCESS, dot11PrivacyInvoked)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyID, 0)
90 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyID)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyIndex,1)
92 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyIndex)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyValue,"12:34:56:78:90")
94 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyValue)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyIndex,2)
96 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyIndex)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyValue,"")
98 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyValue)

```

```

nsh> MLME MLME_SET.request(dot11WEPDefaultKeyIndex,3)
100 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyIndex)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyValue,"")
102 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyValue)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyIndex,4)
104 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyIndex)
nsh> MLME MLME_SET.request(dot11WEPDefaultKeyValue,"")
106 nsh> MLME MLME_SET.confirm(SUCCESS, dot11WEPDefaultKeyValue)

108 ...

110 nsh> MLME MLME_SCAN.request(INFRASTRUCTURE, FF:FF:FF:FF:FF:FF, "", PASSIVE, 10, \
    [1,3,4], 100, 1200)
112 nsh> MLME MLME_SCAN.confirm(((00:40:96:54:17:31, "amd wlan", INFRASTRUCTURE, 100, \
    2, 0xFFFFFFFF00000080, 0x00000000ACC12333, (7), (0, 0, 0, 0), (0), 0x0031, \
114    [2, 4, 11, 22]), (12:34:56:78:9A:BC, "nautilus_infra_test", INFRASTRUCTURE, \
    1000, 2, 0xFFFFFFFF00000080, 0x00000000AC8E90F4, (1), (0, 0, 0, 0), (0), \
116    0x0001, [2, 4, 11, 22])), SUCCESS)

118 ...

120 nsh> MLME MLME_JOIN.request((12:34:56:78:9A:BC, "nautilus_infra_test", \
    INFRASTRUCTURE, 1000, 2, 0x0000000000000000, 0x0000000000000000, (1), \
122    (0, 0, 0, 0), (0), 0x0000, [2, 4, 11, 22]), 2, 10, [2, 4, 11, 22])
nsh> MLME MLME_JOIN.confirm(SUCCESS)
124
...
126
nsh> MLME MLME_AUTHENTICATE.request(12:34:56:78:9A:BC, OPEN_SYSTEM, 2000)
128 nsh> MLME MLME_AUTHENTICATE.confirm(12:34:56:78:9A:BC, OPEN_SYSTEM, SUCCESS)

130 ...

132 nsh> MLME MLME_ASSOCIATE.request(12:34:56:78:9A:BC, 1000, 0, 10)
nsh> MLME MLME_ASSOCIATE.confirm(SUCCESS)

```

## 1.4.3 Configuration via Web Interface

### 1.4.3.1 Introduction

The WLAN System can be controlled via a web user interface, which connects to a HTTP server which controls the driver.

To start the http server issue the following command:

```
./HttpServer
```

Please note that a handle for the Nshell IOCTL has to be created first as described previously (mknod command).

To access the user interface a web browser needs to point to the system where the WLAN driver is running.

```
http://localhost:10080/
```

whereas "localhost" is the name of the WLAN system where the HTTP server is running. The initial screen shown in Figure 1.1 will appear.

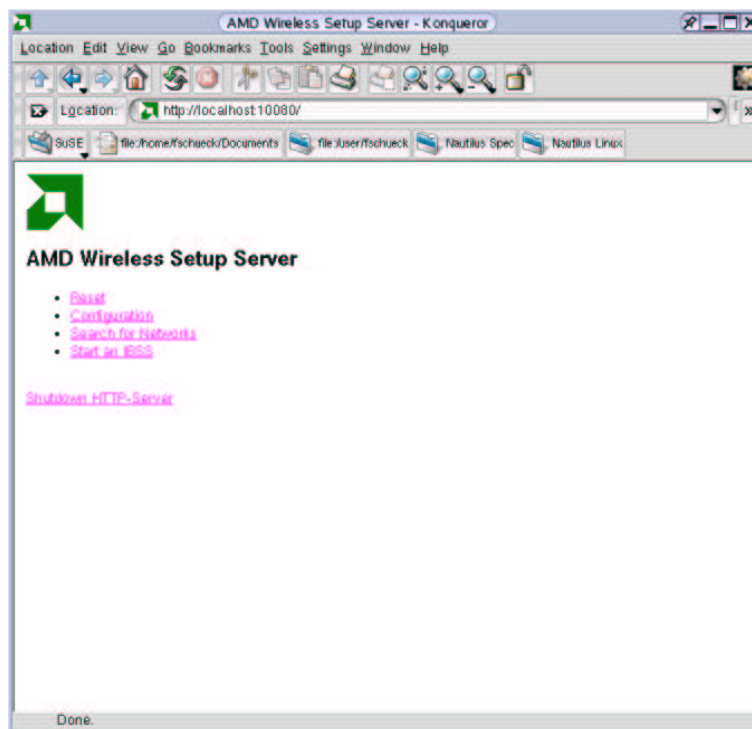


Figure 1.1: Initial Screen

As it can be seen a variety of features are provided by the web interface. The sequence of actions is important during operation.

### 1.4.3.2 Reset

Once the http server is started the WLAN system is reset. This can be done again at any time by selecting the "Reset" link.

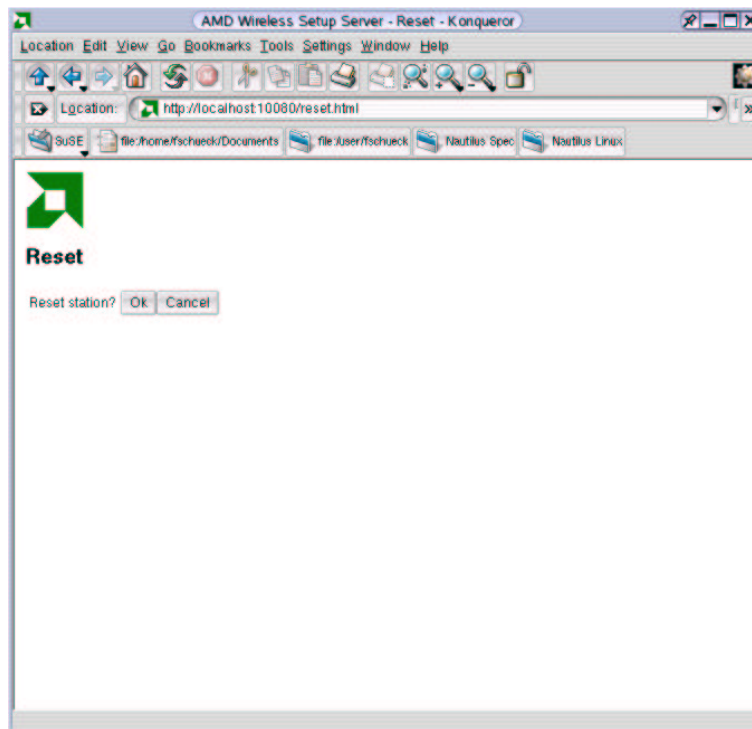


Figure 1.2: Reset

After pressing the OK button (Figure 1.2) the driver will reset itself. Please note that this leads to a disconnection from any network the system is connected to at this point of time. Additionally this will issue the reload of the standard configuration values from the driver (see next section).

### 1.4.3.3 WLAN Configuration

After selecting the "Configuration" link the dialog shown in Figure 1.3 is available. The values shown are the default values from the driver after reset.

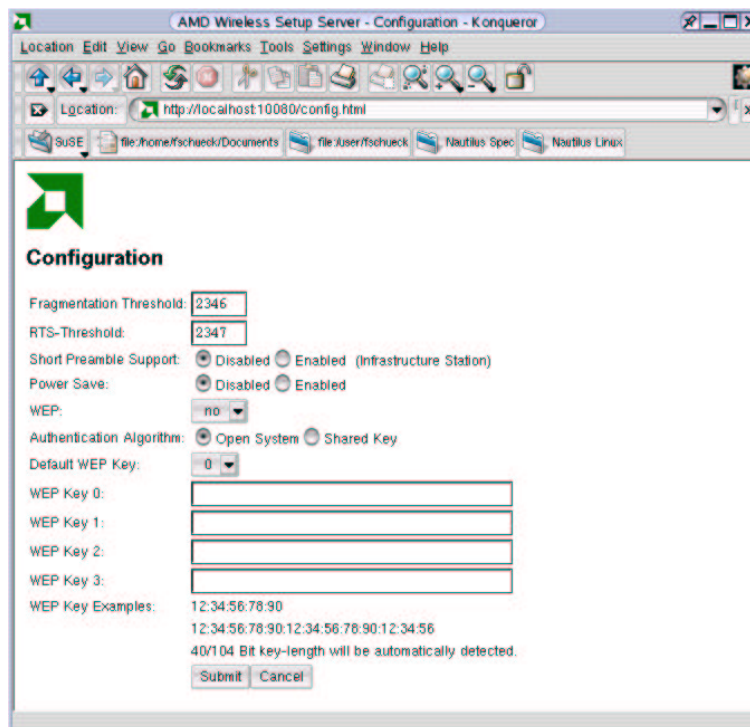


Figure 1.3: Configuration

Fragmentation- and RTS-Threshold can be set manually. Currently no range check on the input values is performed, thus it is important to enter meaningful parameters.

In case security shall be enabled (either for starting a new IBSS or for connecting to an existing network which requires WEP), the correct security settings need to be set prior to any further action. This includes switching WEP on/off, selecting the appropriate key length, or any of the predefined keys. At the moment 4 keys for both 40 bit and 104 bit WEP are available (Figure 1.4). These keys are presented on the screen and cannot be changed. The same WEP keys need to be applied on the peer AP/Station.

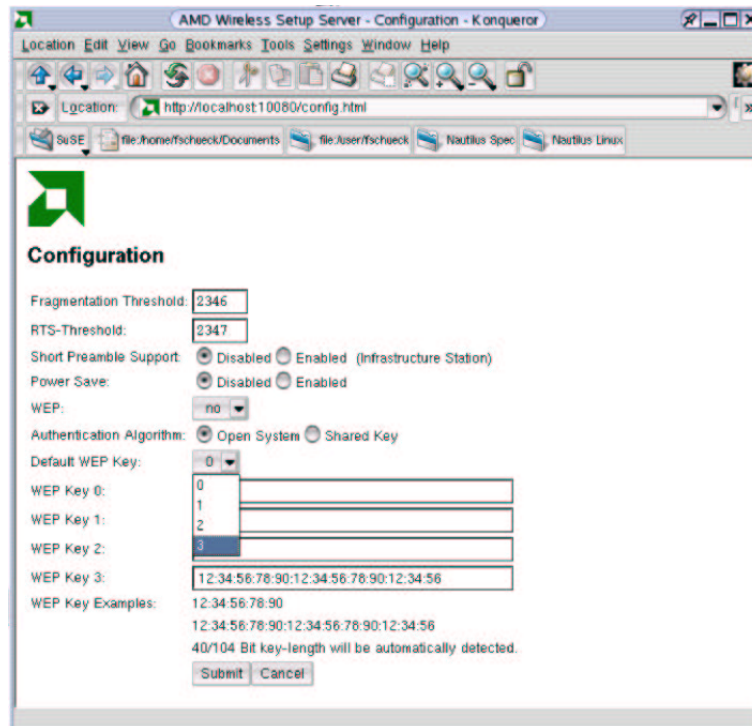


Figure 1.4: Configuration Select WEP Key

Please note that the WEP Key ID transmitted in the encrypted frame is always Key ID 0, independent of the selected WEP key. The current values are applied to the system after pressing "Submit".

#### 1.4.3.4 Searching for Networks

By selecting the "Searching for Networks" link the screen shown in Figure 1.5 becomes available.

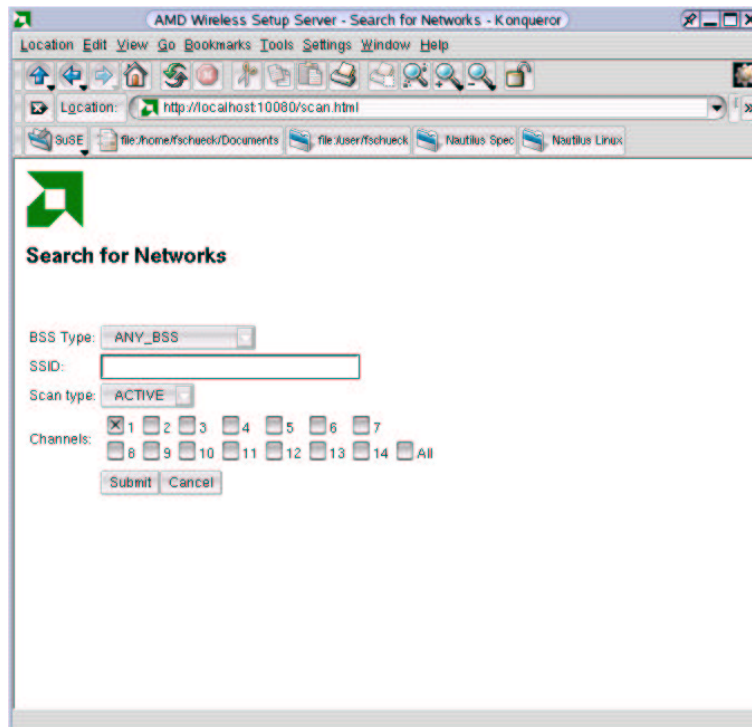


Figure 1.5: Scan Standard screen

The BSS type the system will be searching for can be defined as shown in Figure 1.6.

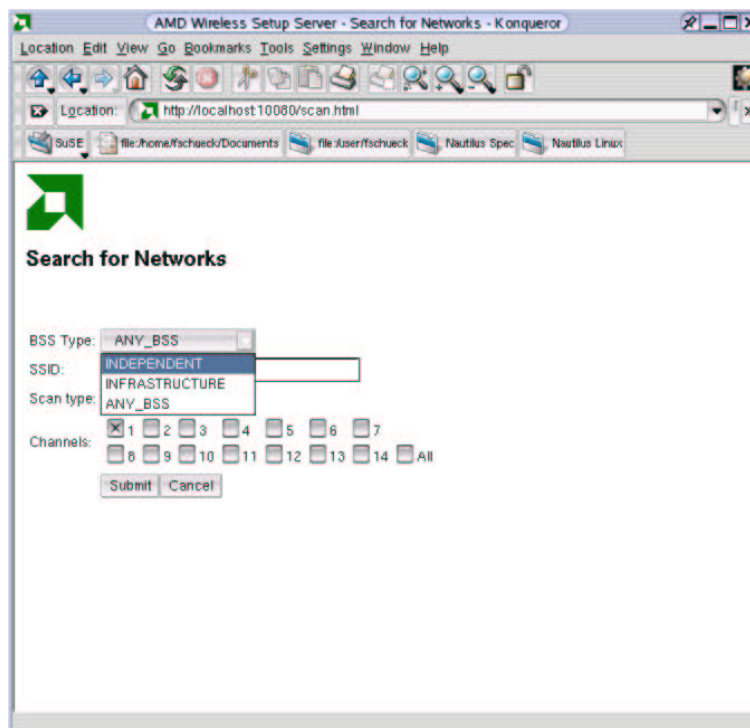


Figure 1.6: Scan specific network type

Per default the SSID is empty, so every available network will be presented. In case a specific SSID is of interest, the network name can be specified. See Figure 1.7 for details.

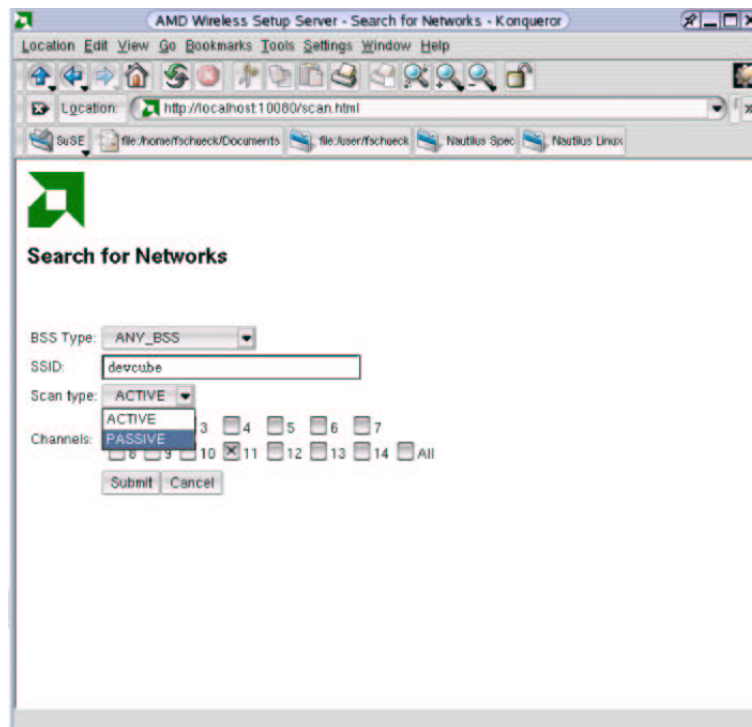


Figure 1.7: Scan Specific SSID

Scanning can be performed either active or passive. Initially only channel 1 is selected, but any channel of interest can be applied. By selecting "all" scanning will be performed on all channels.

After successfully scanning the results are presented as shown in Figure 1.8.

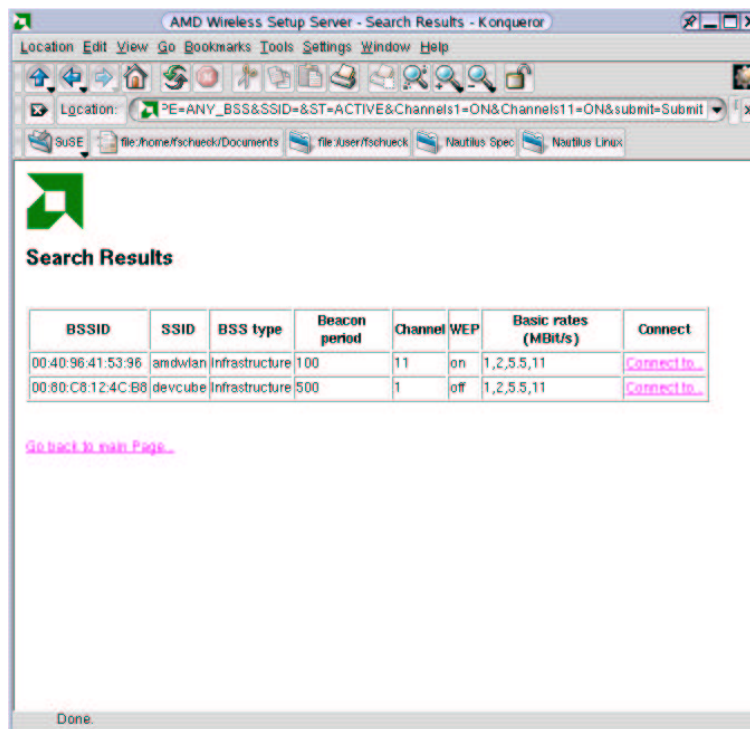


Figure 1.8: Scan Networks Results

Now the network of interest, which the system shall connect to, can be selected. Again, the correct WEP settings need to be specified prior to this step.

Note that scanning is currently not supported once the system is connected to any network or has started its own network. A new successful scan initiation requires to reset the system.

#### 1.4.3.5 Connecting to an existing Network

After a successful scan a specific network can be chosen by selecting the link in the right column of the result table. A dialog showing the network settings appears (Figure 1.9).

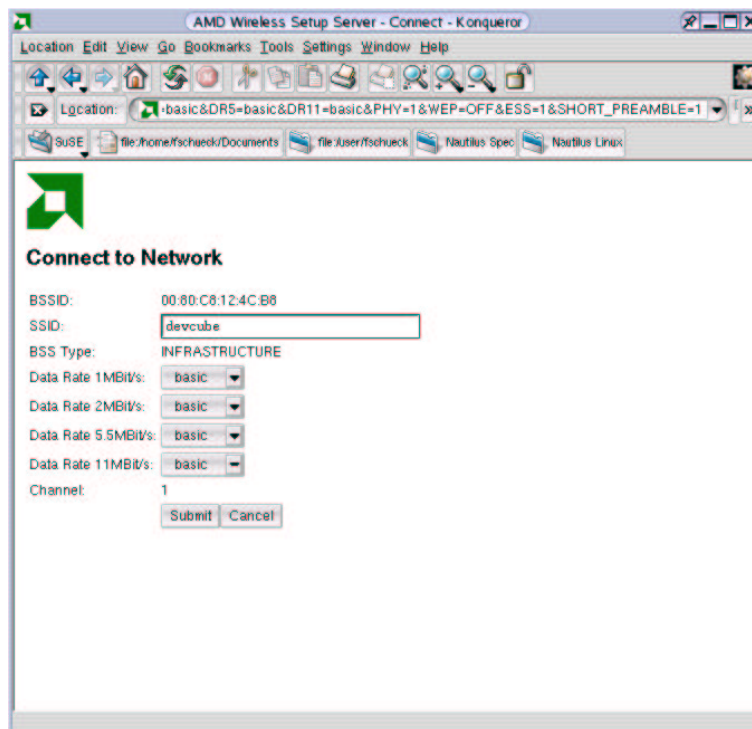


Figure 1.9: Connect to Network

The SSID and the Basic Rates of the network shall remain unchanged for successful connection. All other rates are optional (but shall not be changed to basic).

After pressing the "Submit" button the system tries to join the network. In case of an Infrastructure BSS also Open System Authentication and Association is initiated. Once this has been finished successfully, the result is presented as shown in Figure 1.10.

Now the machine is part of the WLAN network.

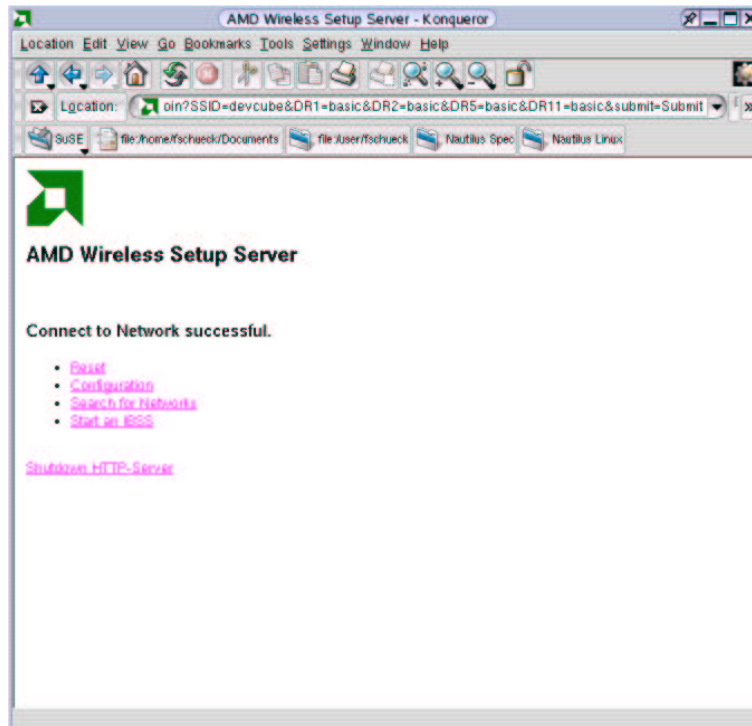


Figure 1.10: Connect Result

#### 1.4.3.6 Starting an Independent BSS

By selecting the link "Start an IBSS" from the initial screen, the dialog as shown in Figure 1.11 is active.

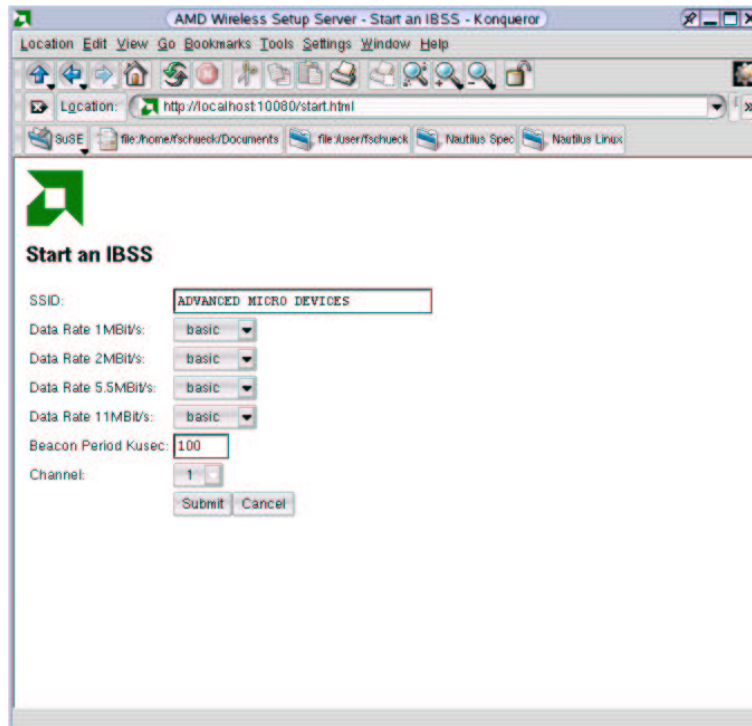


Figure 1.11: Start IBSS

The user can choose the SSID, the Basic Rates, the Beacon Period, and the channel. Refer to Figure 1.12.

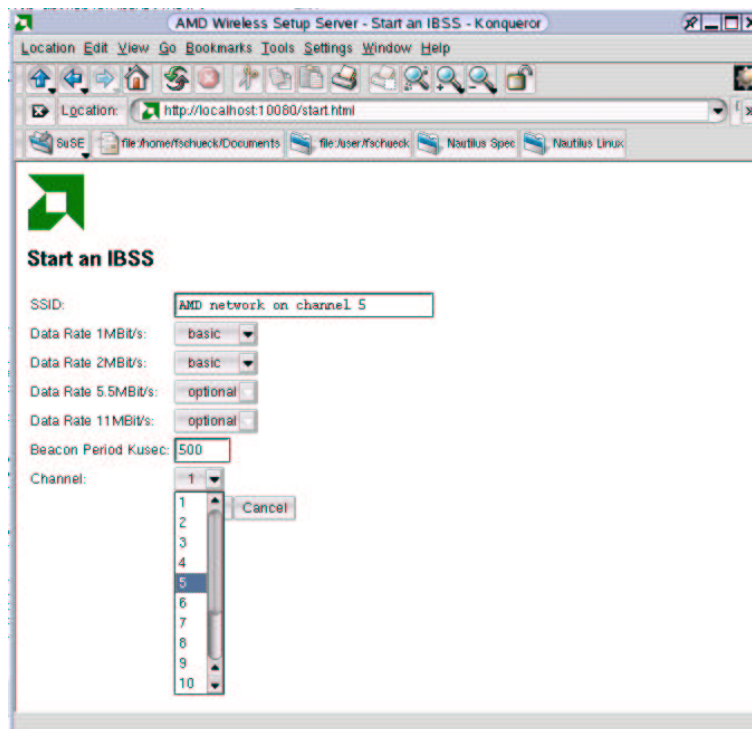


Figure 1.12: Start specific IBSS

## 1.5 Build own driver and applications

### 1.5.1 Linux

#### 1.5.1.1 General

Building the driver and applications on Linux does require a number of standard tools installed on the build machine. For cross-compiling the MIPS binaries on x86 the Hard-Hat Linux 2.0 distribution need to be installed on the build system.

Whether you need to add tools to your compile machine depends on the Linux distribution used. All tools are usually included in the main Linux distributions like RedHat or SuSE.

#### 1.5.1.2 Un-packing of the source distribution

Untar the Nautilus Linux “tgz” file:

```
$> tar -xvzf source_linux_ST6.0.tgz
```

This will generate the following directories:

- source\_linux\_ST6.0/Include/
- source\_linux\_ST6.0/Linux/
- source\_linux\_ST6.0/MacCore/
- source\_linux\_ST6.0/Tools/
- source\_linux\_ST6.0/Applications

### 1.5.1.3 Set-up the environment

Change to the Linux build directory.

```
$> cd source_linux_ST6.0/Linux
```

Do start a new bash in your command window even if your default shell is already set to use bash.

```
$> bash
```

Source the respective target platform setup script. If compiling for MIPS or XScale, the HardHat 2.0 cross compiling tool chain is expected to be installed on the compile system. The compiled Kernel sources for the MIPS or XScale target need to be installed at the location specified in the respective \*\_personal.settings file (complete MIPS or XScale file system). Settings for this location and for instance for the install target directory can be changed in \*\_personal.settings files.

```
$> source ./x86_setup.script
```

or

```
$> source ./mips_setup.script
```

or

```
$> source ./xscale_setup.script
```

#### 1.5.1.4 Configure Nautilus

Configuration for Nautilus is done using special `make` targets. The setup is done in three steps:

1. Setting up the target platform
2. Setting up the target hardware interfaces
3. Select debug or release version

The following make targets are available:

```

$> make help
2 make <target>
.. help ..... print this screen
4 .. compile ..... build this directories target
.. depend ..... rebuild dependencies
6 .. qac ..... collect code metrics and prepare summaries \
                    (runs on Solaris only)
8 .. set_platf_x86 ..... set target platform to x86 (preceeded by clean and HW \
                    interface de-selection)
10 .. set_platf_aul ..... set target platform to MIPS Aulxxx (preceeded by \
                    clean and HW interface de-selection)
12 .. set_platf_xsc ..... set target platform to Xscale (preceeded by clean \
                    and HW interface de-selection)
14 .. set_soc_acc_px2_idp ... set target platform to Xscale Accelent (preceeded by \
                    clean and HW interface de-selection)
16 .. set_soc_cog_csb226 .... set target platform to Xscale Accelent (preceeded by \
                    clean and HW interface de-selection)
18 .. set_pci_on ..... enable PCI hardware interface (preceeded by clean)
.. set_pci_off ..... disable PCI hardware interface (preceeded by clean)
20 .. set_sdio_on ..... enable SDIO hardware interface (preceeded by clean)
.. set_sdio_off ..... disable SDIO hardware interface (preceeded by clean)
22 .. set_cf_on ..... enable CF hardware interface (preceeded by clean)
.. set_cf_off ..... disable CF hardware interface (preceeded by clean)
24 .. set_sram_on ..... enable SRAM hardware interface (preceeded by clean)
.. set_sram_off ..... disable SRAM hardware interface (preceeded by clean)
26 .. all ..... build this module and all sub modules
.. clean ..... clean this module and all sub modules regarding all \
28                    aspects
.. set_debug ..... turn on debug mode for compilation \
30                    (preceeded by clean)
.. set_release ..... turn off debug mode for compilation \
32                    (preceeded by clean)
.. debug ..... turn on debug mode; rebuild this module and all sub \
34                    modules
.. release ..... turn off debug mode; rebuild this module and all sub \
36                    modules

```

For building on x86 systems use the following configuration:

```
$> make set_platf_x86 set_pci_on set_release
```

or

```
$> make set_platf_x86 set_pci_on set_debug
```

For MIPS Pb1500 use:

```
$> make set_platf_aui set_pci_on set_release
```

or

```
$> make set_platf_aui set_pci_on set_debug
```

### 1.5.1.5 Compile the driver and applications

Compile the Nautilus AMD WLAN driver and applications:

```
$> make
```

Install the driver and applications:

```
$> make install
```

Driver and applications will be copied to the target directories specified in the `*_personal.settings` file.

## 1.5.2 Windows CE

### 1.5.2.1 Create a WinCe Platform

Install Platform Builder:

Install Microsoft Platform Builder 4.1 on your development system.

For x86 CE PC use the BSP provided with the platform builder and apply the patches delivered with Nautilus x86 BSP-patches.

For Au1x00 Platform, use the BSP provided with is Nautilus release and update your installed BSP. Apply all patches as described in the README.txt delivered with the BSP-update.

Install the Nautilus source package:

This installation procedure succeeds only if the Platform Builder is installed on hard drive "C:" in the default directory "C:\WINCE410".

1. Unpack the zip file with the sources to a directory on your local hard drive.
2. Start "setup.bat". This will copy the files into the Platform Builders directory structure.

Platform Builder Integration:

1. Uninstall previous driver versions from Platform Builder. If no previous version is installed continue with "New Driver Installation" else uninstall the old version before continuing.
  - a) Open every platform which is using the Am1772 driver and remove the Am1772 feature including all Am1772 applications from you platform. After this refresh the platform features (Menu: "Platform"->"Refresh Features"). Save and close all platforms (Menu: "File"->"Save Workspace")
  - b) Remove the old Am1772 feature form the Platform Builder Catalog. Select the option "Manage Catalog Features..." in the Platformbuilder's "File" Menu, select the Am1772 entry from the list and press the "Remove" button. Now press the "Refresh" button to update the Catalog window.
  - c) Close the dialog box.

## 2. Import the .cec file in the Platform Builder

- a) Select the option “Manage Catalog Features...” in the Platformbuilder’s “File” Menu and press the “Import..” Button and navigate to the Am1772.cec file. After you closed this dialog you will find the Am1772 group in the “Device Drivers” section. You will find the Am1772.cec file in the directory:  
“C:\WINCE410\PUBLIC\COMMON\OAK\DRIVERS\NETCARD\Am1772\WinCE\Build\”
- b) Close the dialog box.

## 3. Enable Windows CE Zero Config Support

- a) Zero Config can be enabled or disabled by setting an Platform Builder environment variable named “OID\_ENABLE\_ZERO\_CONFIG”. If you set this variable to “1” the Zero Config support is enabled. Use the “Settings...” dialogue from the “Platform” menu to set it.

Configure the platform:

1. Open the WinCE Platform Builder
2. Start the “New Platform Wizard” by selecting “New Platform...” from the “File”-Menu
3. Step 2: Select the BSP: “CEPC: X86” and/or “PB1500”
4. Step 3: Select the available configuration: “Internet Appliance” and enter a platform name
5. Step 4: Select the configuration variant of your choice, e.g. “Internet Appliance with Applications”
6. Step 5: Select the platform features of your choice, e.g use the default settings
7. Step 6: Select the network features of your choice, select at least all LAN features
8. Step 7: Finish the wizard
9. After finishing the wizard you can add more features by drag and drop from the catalog to the features window
10. Select “Settings...” from the “Platform”-Menu
11. First select “All Configurations” in the “Settings For:” list box

12. Select the “Environment”-Tab and create a new variable by pressing the “New...” button
13. Set the variable “IMGRAM64” to 1. This allows Windows CE to use 64 MB of RAM.
14. To check your platform do Platform/Resolve Features. There should no “Unresolved Features” be shown. In case of the Pb1500 BSP the USB-Function stays unresolved - choose “Netchip NET2890 USB Function (serial interface)” to be used for your platform.

### 1.5.2.2 Platform build process

1. Select the target configuration (release or debug) for your appropriate BSP.
2. Start the platform build process, menu “Build” select “Build Platform” <sup>1</sup>

---

<sup>1</sup>While building the platform it may appear requests to update components of the PlatformBuilder e.g. the .NET compact framework. Follow the steps for downloading and installation. It's recommended to boot the computer afterwards and to create a new platform!



## List of Tables



## List of Figures

1.1	Initial Screen . . . . .	19
1.2	Reset . . . . .	20
1.3	Configuration . . . . .	21
1.4	Configuration Select WEP Key . . . . .	22
1.5	Scan Standard screen . . . . .	23
1.6	Scan specific network type . . . . .	24
1.7	Scan Specific SSID . . . . .	25
1.8	Scan Networks Results . . . . .	26
1.9	Connect to Network . . . . .	27
1.10	Connect Result . . . . .	28
1.11	Start IBSS . . . . .	29
1.12	Start specific IBSS . . . . .	30