



# Aemulus PXI/PXIe Instrument Software Library

Application Notes

*Version 1.1, 03-2013*

## Table of Contents

---

TABLE OF CONTENTS .....	1
SECTION 1: CORE AND WRAPPER SOFTWARE LIBRARIES .....	2
SECTION 2: RESOURCE PLANNING .....	3
2.1 INTRODUCTION .....	3
2.2 RESOURCE MAPPING.....	4
SECTION 3: RESOURCE PLANNER .....	5
3.1 TERMINOLOGY .....	5
3.2 USER INTERFACE .....	6
3.3 CREATING A DUT <i>AMAP</i> FILE .....	8
SECTION 4: PLATFORM RESOURCE EDITOR .....	10
4.1 PLATFORM <i>AMAP</i> FILE .....	10
4.2 USER INTERFACE .....	12
4.3 CREATING A PLATFORM <i>AMAP</i> FILE .....	13
SECTION 5: REVISION HISTORY .....	14

## Section 1: Core and Wrapper Software Libraries

---

There are 3 types of software libraries available for Aemulus PXI/PXIe instruments:

1. C++ library
2. .NET library
3. .NET wrapper library

Both C++ and .NET libraries are **core** libraries where they contain the low-level APIs to access the PXI/PXIe instruments.

On the other hand, .NET **wrapper** library is created based on the initiative to ease test development for users who are already very familiar with API library interface used by AMB1340c SMU products.

The operations of wrapper and core libraries are essentially the same. Each API in wrapper library takes the same function name as that of AMB1340c's, but the underlying functions are still from the core library. For example, *DriveVoltage* in the wrapper library consists of following core library functions:

ConfigureOutputFunction (DVCI mode)  
ConfigureVoltageLevel

Refer to AM400e .NET Wrapper Library Programming Manual for details.

Wrapper library must be used hand in hand with **Resource Planner**. Together with wrapper library, Resource Planner allows users to have just one set of test program capable of running on both PXI/PXIe and AMB1340c tester platforms, without having to change the test program.

Examples of C++ and its equivalent .NET libraries, as well as wrapper library are

C++ Core Library	.NET Core Library	.NET Wrapper Library
AemDCPwr.dll	Aemulus.Hardware.SMU.PXI.dll	Aemulus.Hardware.SMU.dll
AM430e.dll	-	-
AM471e.dll	-	-

Table 1: Examples of Various Libraries

## Section 2: Resource Planning

---

### 2.1 Introduction

In the wrapper library, it is NOT necessary to specify the instrument handle that points to test resource ID/address as well as the resource channel number. Instead, a user-defined pin name is required.

For example, a typical *DriveVoltage* function prototype is

*DriveVoltage (Instrument handle, resource channel number, voltage to be driven)*

For wrapper library, the same function is

*DriveVoltage (pin alias, voltage to be driven)*

The first argument of the API is a user-defined name, which is normally created to help explain the actual operation in a more interactive way. The pin alias normally takes the device pin name as stated in the datasheet, such as "Vcc", "Vmode", etc.

In the following example, it is very clear that 5V is sourced to Vcc pin of the device-under-test.

*DriveVoltage ("Vcc", 5.0)*

## 2.2 Resource Mapping

The actual resource used by the test program is specified in a file with file extension “*amap*”, which is normally known as the *amap* file. This file is generated by **Resource Planner**, and it contains the mapping information between pin alias in test program and the appointed tester resources, and

Users can generate as many *amap* files as possible. Test program will then need import the desired *amap* file during test. Normally one *amap* file is generated for one DUT (device-under-test).

Advantages:

1. Resource Planner avoids the need for users to remember the specific test resource during test program development.

For a high pin-count device, it is somewhat difficult to manage test development, as users need to specify all the resource ID/address in the test program. If there are changes in resource ID/address, depending on how users structure the test program, changes in test program could be disastrous.

By using Resource Planner, users only need to quote the pin name while performing any test operation. This will also help troubleshooting and debugging especially when there are too many test resources in the system.

2. Test program can maintain even if changing between AMB1340c or PXI/PXIe tester platform. Only *amap* file change is required. This eases platform conversion process.

## Section 3: Resource Planner

---

### 3.1 Terminology

*Test Head* refers to test handler. *Test Head* number starts from 0.

*Test Site* refers to physical location of testing site for a device. *Test Site* number starts from 0.

*Hardware Profile* refers to tester platform, which contains specific test resource configuration.

*DUT amap file* is unique and specific to a **device-under-test** (DUT). Each DUT will have its own *amap* file that contains pin information of the DUT. Resource Planner is used to create and edit user *amap* files.

*Platform amap file* is unique and specific to a **tester configuration**. This file contains information about tester configuration.

3.2 User Interface

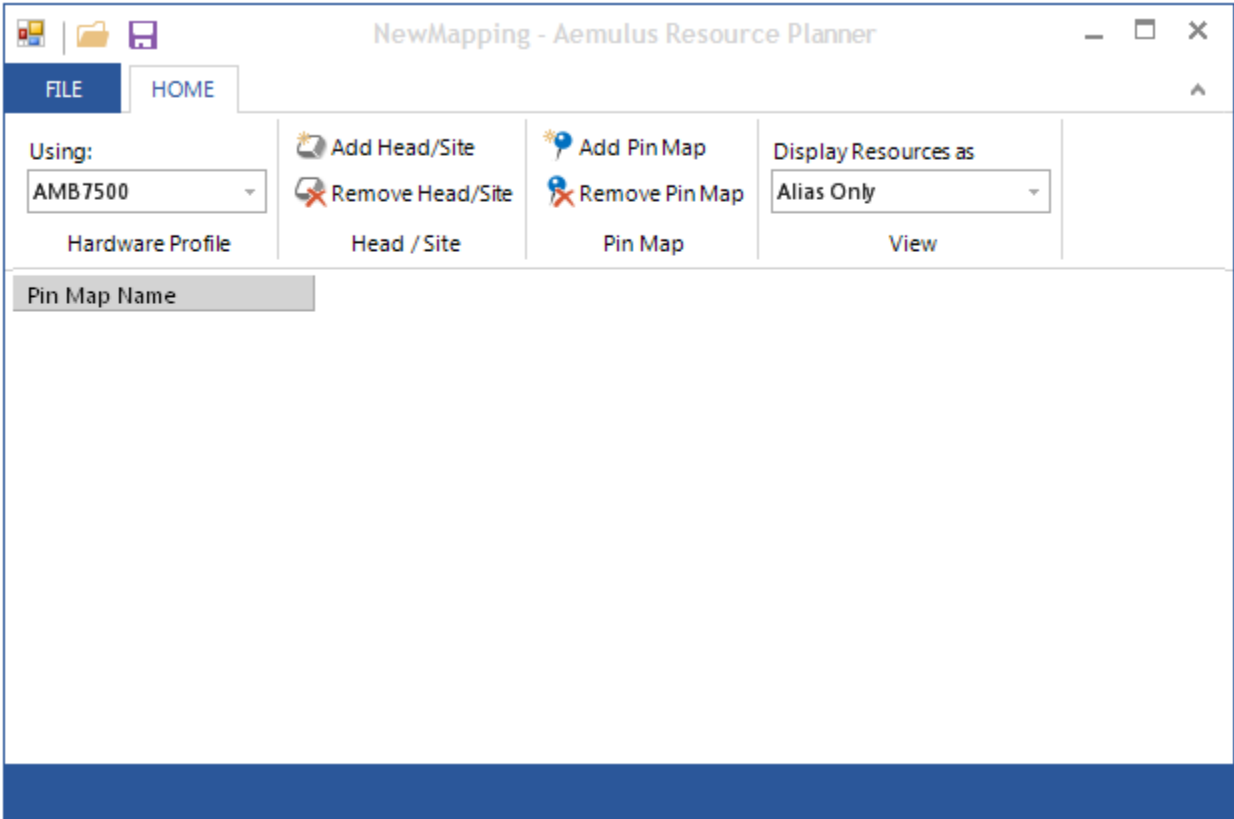


Figure 1: ResourcePlanner.exe

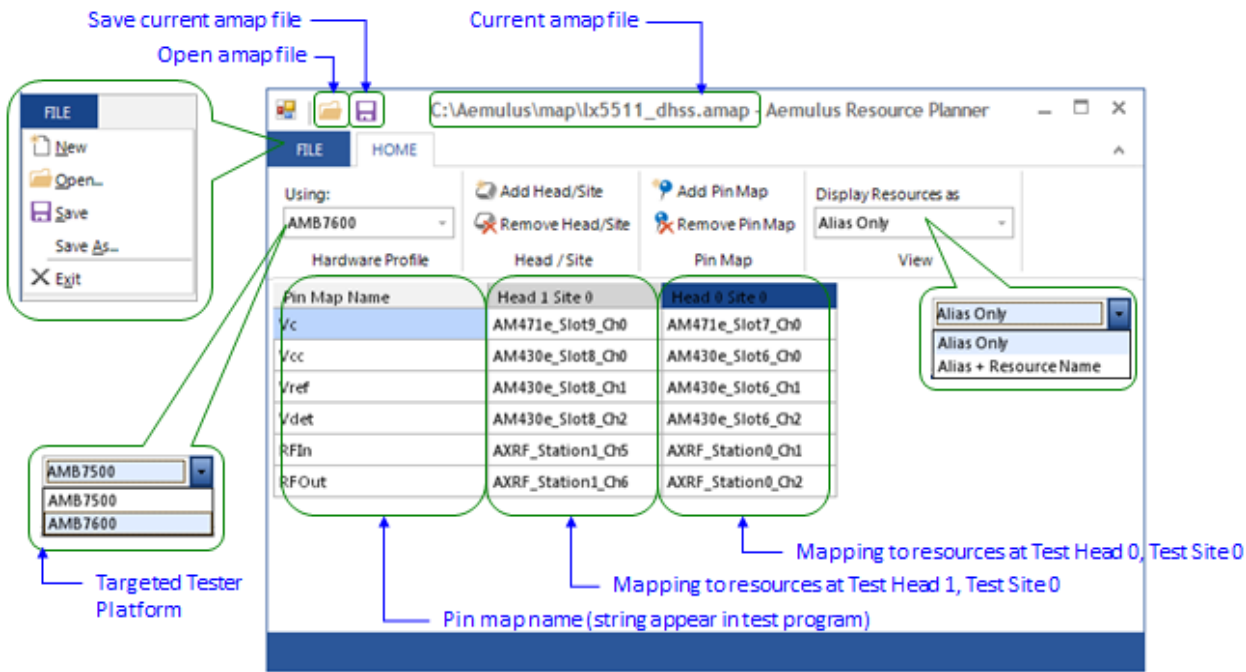


Figure 2: Example of *amap* file

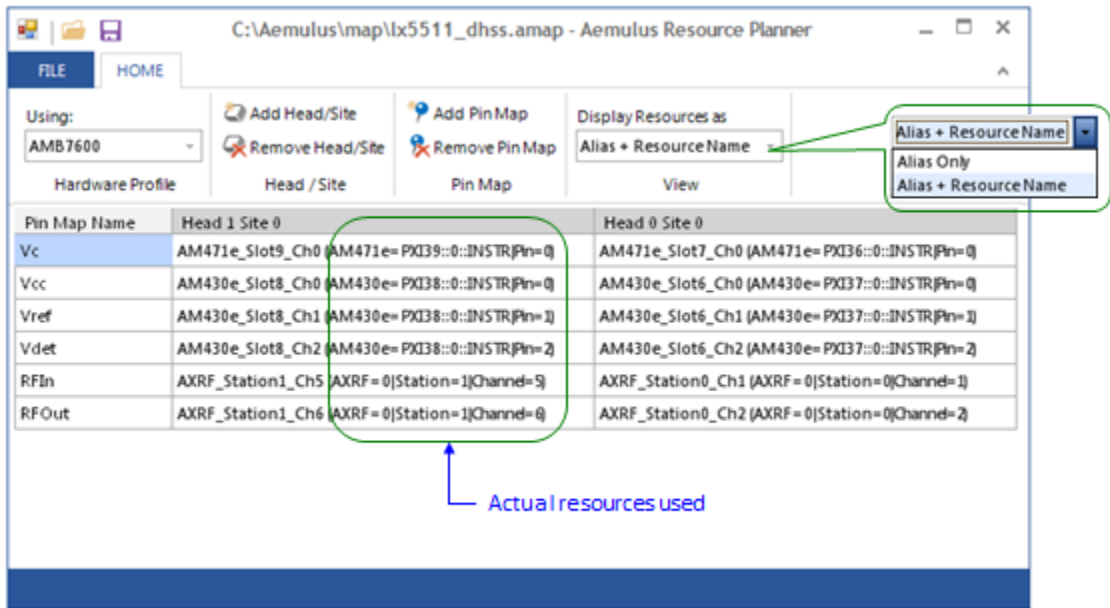


Figure 3: Actual Resource Used by Each Pin



### 3.3 Creating a DUT *amap* File

1. Launch ResourcePlanner.exe.
2. Click on *File*, select *New* to create a new mapping file.
3. Choose desired hardware profile.  
When changing hardware profile, the following message will pop up

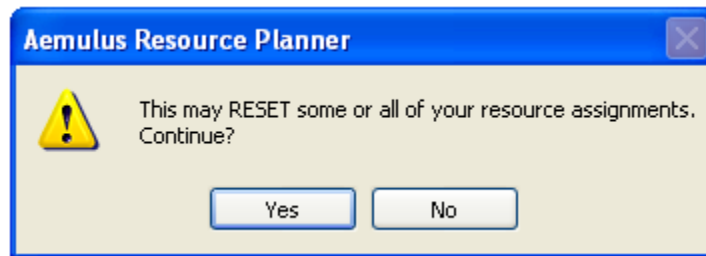


Figure 4: Changing Hardware Profile

Click *Yes* to continue.

4. Click on *Add Pin Map*, enter a pin name that will be used in test program later.

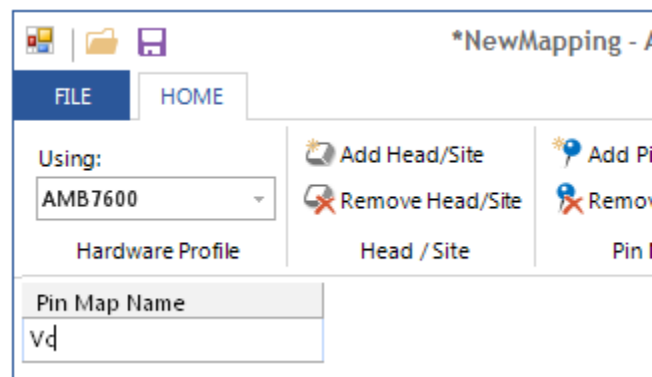


Figure 5: Add New Pin

- 5. Click on *Add Head/Site*, choose desired test head and test site number, and then click OK.

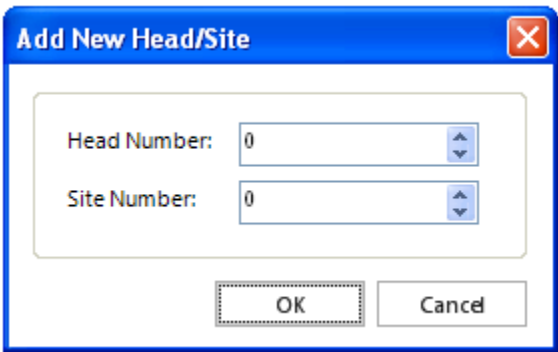


Figure 6: Add New Head/Site

- 6. For the new pin name created, select the desired test resource from a drop-down list.

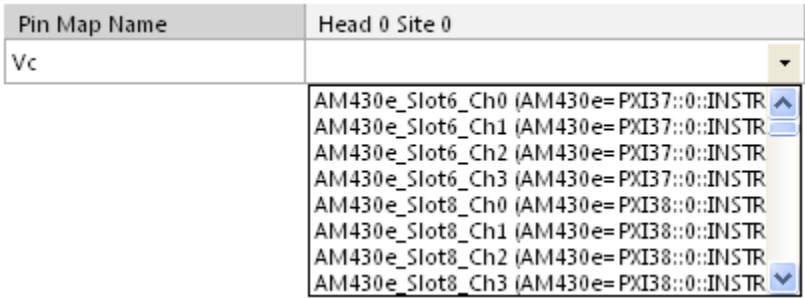


Figure 7: Select Test Resource

The test resources are specific to the hardware profile selected in step (3).

- 7. Repeat steps (4)-(6) to create new pin map names and perform corresponding test resource mapping.
- 8. Click *Save* to save the *amap* file with a user-defined file name.

## Section 4: Platform Resource Editor

### 4.1 Platform *amap* File

Referring to Figure 2, the targeted tester platform actually refers to configuration stored in platform *amap* file. One tester platform refers to one platform *amap* file.

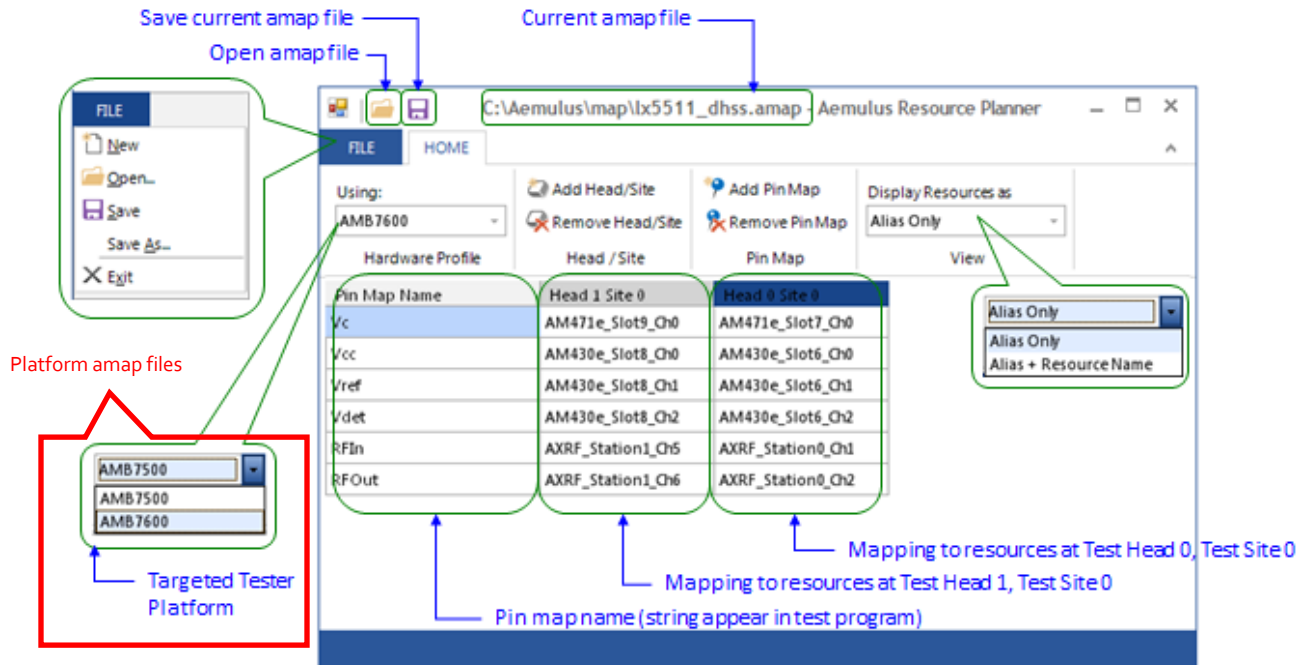


Figure 8: Target Platform

Platform *amap* file contains following tester configuration information:

1. Number of different PXI/PXIe modules in the system
2. Number of accessible channels of each test resource
3. ID or address of each test resource

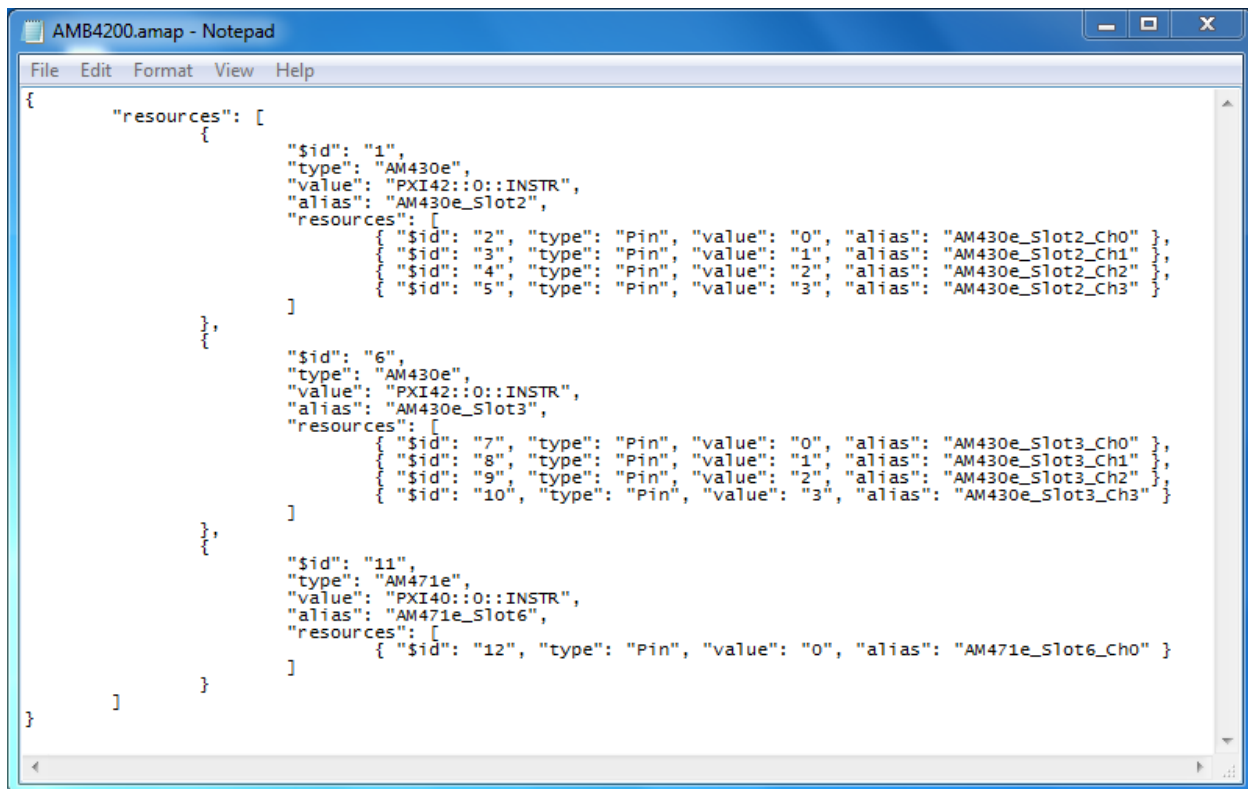
When tester configuration changes, e.g. add/remove/change test resources, or PXI/PXIe address change, there is **NO** need to change test program and DUT *amap* file. Only platform *amap* file needs to be updated.

By default, platform *amap* files are placed at following directory:

Windows	Directory
7	C:\ProgramData\Aemulus\map
XP	C:\Documents and Settings\All Users\Application Data\Aemulus\map

Table 2: Platform *amap* File Installation

During driver installation of 400e modules, a sample of platform *amap* file (AMB4200.amap) is provided, as shown below.



```

{
  "resources": [
    {
      "$id": "1",
      "type": "AM430e",
      "value": "PXI42::0::INSTR",
      "alias": "AM430e_Slot2",
      "resources": [
        { "$id": "2", "type": "Pin", "value": "0", "alias": "AM430e_Slot2_Ch0" },
        { "$id": "3", "type": "Pin", "value": "1", "alias": "AM430e_Slot2_Ch1" },
        { "$id": "4", "type": "Pin", "value": "2", "alias": "AM430e_Slot2_Ch2" },
        { "$id": "5", "type": "Pin", "value": "3", "alias": "AM430e_Slot2_Ch3" }
      ]
    },
    {
      "$id": "6",
      "type": "AM430e",
      "value": "PXI42::0::INSTR",
      "alias": "AM430e_Slot3",
      "resources": [
        { "$id": "7", "type": "Pin", "value": "0", "alias": "AM430e_Slot3_Ch0" },
        { "$id": "8", "type": "Pin", "value": "1", "alias": "AM430e_Slot3_Ch1" },
        { "$id": "9", "type": "Pin", "value": "2", "alias": "AM430e_Slot3_Ch2" },
        { "$id": "10", "type": "Pin", "value": "3", "alias": "AM430e_Slot3_Ch3" }
      ]
    },
    {
      "$id": "11",
      "type": "AM471e",
      "value": "PXI40::0::INSTR",
      "alias": "AM471e_Slot6",
      "resources": [
        { "$id": "12", "type": "Pin", "value": "0", "alias": "AM471e_Slot6_Ch0" }
      ]
    }
  ]
}

```

Figure 9: Platform *amap* File

4.2 User Interface

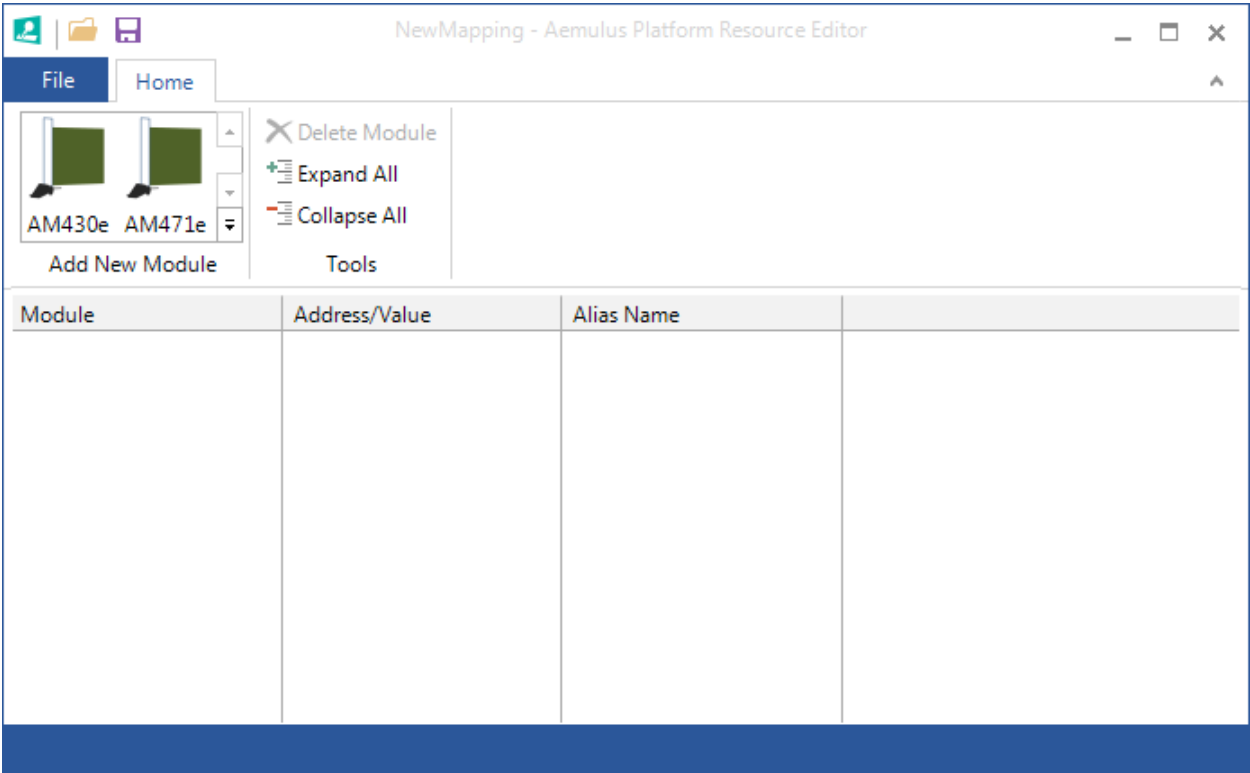


Figure 10: Platform Resource Editor

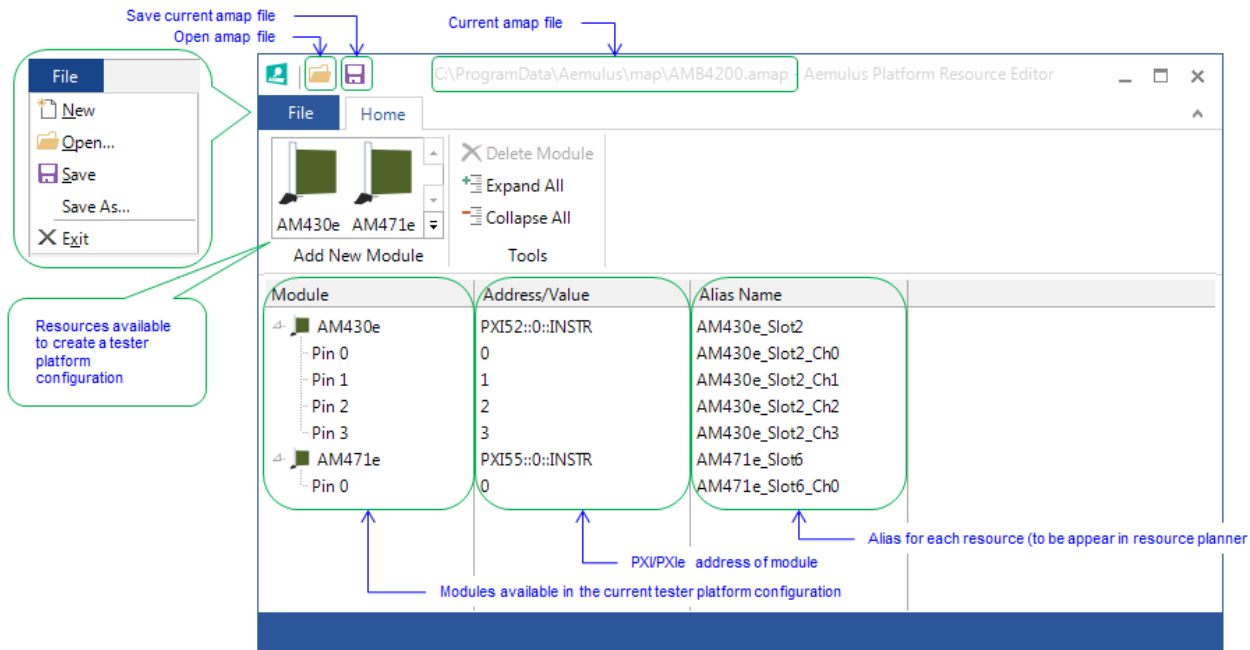


Figure 11: Example of Platform amap File

### 4.3 Creating a Platform *amap* File

1. Launch PlatformResourceEditor.exe.
2. Add a new module from the available resources.

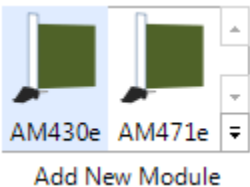


Figure 12: Add New Module

3. Edit the PXI/PXIe address.

Module	Address/Value	Alias Name
AM430e	PXI52::0::INSTR	
Pin 0	0	AM430e_0_Ch0
Pin 1	1	AM430e_0_Ch1
Pin 2	2	AM430e_0_Ch2
Pin 3	3	AM430e_0_Ch3

Figure 13: Edit Address

4. Edit the alias of the resource.

Module	Address/Value	Alias Name
AM430e	PXI0::0::INSTR	AM430e_0
Pin 0	0	AM430e_0_Ch0
Pin 1	1	AM430e_0_Ch1
Pin 2	2	AM430e_0_Ch2
Pin 3	3	AM430e_0_Ch3

Figure 14: Edit Alias

5. Repeat steps (2)-(4) to add more resources into the tester configuration.
6. Click *Save* to save the platform *amap* file with a user-defined file name.

**Section 5: Revision History**

---

1.0	FEB 2013	INITIAL RELEASE
1.1	MAR 2013	ADDED STEPS TO CREATE/EDIT PLATFORM RESOURCE AMAP FILE
1.2	MAY 2013	ADDED PLATFORM RESOURCE EDITOR