

BeIA Device Validation

This document is confidential and may not be distributed without the permission of Be Inc.

copyright © 2000 Be Incorporated

BelA Device Validation

Table of Contents

Device Validation Overview	3
1 Creating a Validation Image.	3
2 The Validate Application	3
The Validate Application	5
1 Running Validate	5
1.1 Validation Settings	5
2 Validate in Action.	5
2.1 Starting the Tests	6
2.2 Completion	7
3 Validation Success.	7
3.1 Validation Failure	7
Validation Settings	9
1 Setting the Test Order	9
2 Fine-Tuning Tests	9
3 Adding a Test	9
4 Validation Operation Control.	10
Validation Test Specifications	11
1 Keyboard	11
2 Mouse	11
3 External Speaker	12
4 Internal Speakers	12
5 Modem	13
6 Ethernet	14
7 File Read	14
8 File Write	15

BeIA Device Validation

Device Validation Overview

Device validation is a series of checks and tests that ensures that each tested device is suitable for shipment to an end user. Be performs software validation before providing a BeIA image to the vendor; the rest of the tests must be performed in the factory on the individual devices themselves. This chapter describes how to create a “validation image”, how to run the validation tests, and gives step-by-step operating instructions for the validation tests.

The rest of this overview describes the validation process as it lies in the larger context of the BeIA development cycle, and introduces the principal validation programs and files. The other sections of this chapter are:

- *The Validate Application* describes **Validate**, the application that performs the validation tests.
- *Validation Settings* describes the **validation settings file** that you can use to fine-tune the tests in the **Validate** suite.
- *Validation Test Specifications* provides operating instructions for the **Validate** tests.

1 Creating a Validation Image

When a BeIA image is sent to the vendor, the image’s **boot mode**, as explained in the *Boot Mode* document, is set to `tools`. While in `tools` mode, the vendor can customize the user interface, develop new browser plug-ins, and make other modifications to the BeIA software stack.

At the end of the customization cycle—i.e. when the vendor is satisfied with the customization—the vendor loads the image onto a target device and runs the **Make_Release_Image** program (this is also explained in *Boot Mode*). **Make_Release_Image** prepares the device’s software for duplication (it creates a “release image”), and puts the device into `validate` boot mode; this is the mode that we’re concerned with for the purposes of validation.

After running **Make_Release_Image**, the vendor duplicates the release image and loads a copy onto each shippable device. These devices will also be in `validate` boot mode. The first time a `validate` device is booted, the validation process is automatically started.

2 The Validate Application

Validate is a BeIA application that comprises a series of software and middleware tests that ensure that the BeIA image was properly installed. It also tests some hardware components (the mouse, keyboard, speaker, etc.).

On a device in `validate` boot mode, **Validate** runs automatically when the device is booted. For test debugging purposes, the application can also be launched manually when the device is in `tools` boot mode: After booting the device, type `alt+q` to dismiss the browser, and then double-click the **Validate** icon on the desktop.

Validate requires a human operator to monitor its progress and assess the success or failure of the tests. When the app has finished running, the operator can choose to “prepare” the device for shipment (given that all the test were successful). **Validate** is described in greater detail in *The Validate Application* section of this chapter.

BeIA Device Validation

The Validate Application

Description: Device validation application.

Executable: /boot/test/validate

The **Validate** application runs a series of tests in which the BeIA operating system checks to ensure that it has been properly installed, and that the device hardware—the mouse, keyboard, speakers, and so on—is functioning correctly. The suite of tests must be human-operated, takes a few minutes to run, and is intended to be executed on every shipped unit. The validation suite is designed to be run in the factory, just before shipping.

1 Running Validate

Normally, **Validate** is launched automatically when the device is booted into `validate` boot mode.

If the device is in `tools` mode, **Validate** can be launched manually by exiting the browser (through **alt+q**) and then double-clicking the **Validate** icon that you'll see on the desktop.

1.1 Validation Settings

The suite of tests that **Validate** runs is defined by the **validation settings file**, `/boot/test/validate.ini`. This human-readable file lists the tests that are run, gives the order of the tests, provides parameter settings for individual tests, and lets you set certain operating parameters of **Validate** itself. The file can be edited by the vendor to fine-tune the existing test suite, to add more tests, and to more thoroughly automate **Validate**'s operation. See *BeIA Device Validation: Validation Settings* for details.

2 Validate in Action

When you launch **Validate**, the app displays the **Run Tests** window. As shown below, the window lists the tests that it knows about (as set in the `validate.ini` file), and displays some system information.

Note: The Run Tests window shown here is for illustration purposes. The window's actual configuration—the number of tests and their order—may differ for each vendor.

Test	Run	Pass	Fail	Ignore
Keyboard Key Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mouse Click Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
External Speaker Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Internal Speaker Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Modem Connection Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ethernet Connection Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
File Read Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
File Write Test	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Version Information

OS: Product - Version (Build date)

Machine ID:

BIOS: - ()

MAC Address: <not available>

(Test Location: /boot/test - settings = validate.ini)

Test Prepare Quit

The **Run** radio button selection for the tests indicates that the tests are prepared to run. The operator can skip a test by setting its selection to **Ignore**, or can “pre-approve” or “pre-fail” a test by selecting **Pass** or **Fail**.

2.1 Starting the Tests

To start the tests, the operator presses **Test**. **Quit** will abort the application. The **Prepare** button only becomes enabled when the tests have run to successful completion.

As the tests start, the **Run Tests** window is removed from the screen. It reappears at the end of the tests to provide a tally of test scores (as explained in “2.2 Completion”).

Each test display at least one test-specific window. Some windows ask the operator to perform actions, such as typing a key or clicking the mouse, while others simply report the status of a “hands free” test.

As with the **Run Tests** window, the individual test windows provide a **Fail** button.

At any time, the operator can press **alt+q** to abort the current test and go back to the **Run Tests** window.

2.2 Completion

When **Validate** completes, it re-displays the **Run Tests** window with the radio buttons set to **Pass** or **Fail** for each test. A machine is considered to have successfully passed the validation suite only if *all* the tests have passed. If any one test failed, the entire validation is considered to have failed. The next two sections describe what to do when validation succeeds and when it fails.

3 Validation Success

If **Validate** completes all the test successfully, the operator then presses one of the three buttons at the bottom of the **Run Tests** window:

- Pressing **Prepare** will prepare the machine for shipment to the end user. The **Prepare** button is enabled only if the validation was successful. **Once you turn off the machine after a successful preparation, do not turn it on again.** The next person to turn on the machine after a successful validation should be the end user, as the machine will ask for basic configuration information so it can set itself up for use.
- If the operator presses **Quit**, the machine will not be prepared for shipment. When the machine is rebooted, the validation tests will start again.
- **Test** reruns the validation tests again immediately.

3.1 Validation Failure

If one (or more) of the tests fails—either naturally or because the operator “pre-failed” it or pressed **Fail** in the test window—a message is displayed saying that the unit being tested has not passed the tests, and will not be internally configured for shipment to the user. **If this happens, you must not ship the machine until the problem has been fixed.**

The operator can rerun the entire **Validate** suite by rebooting the machine (after fixing the problem, ostensibly), or an individual failed test can be rerun by setting its radio button back to **Run** and then clicking **Test**.

BeIA Device Validation

Validation Settings

Description: Settings that you use to fine-tune the Validate application.

Declared in: /boot/test/validate.ini

The validation settings file, `validate.ini`, contains statements that set the order in which the validation tests are run, fine-tune individual tests, and let you add new test scripts. The file is located in `/boot/test/`.

The file is read and used by the **Validate** application; for information on **Validate**, see *The Validate Application*.

By default, most statements in `validate.ini` are commented out. To “turn on” a statement, you must remove the comment character (“#”) at the beginning of the line.

1 Setting the Test Order

The order of the tests can be specified in the `testorder` statement in `validate.ini`. The statement lists the names of the tests separated by semicolons (no whitespace!). For example:

```
testorder=keyboard;external_sound;sound;modem;ethernet
```

If the `testorder` statement is used, tests that aren’t included in the list are ignored. Individual tests can also be ignored by setting the name of the test to `ignore` in `validate.ini`. For example:

```
keyboard=ignore
```

2 Fine-Tuning Tests

Many of the settings in `validate.ini` are used to initialize the parameters of specific tests. For example, the `external_sound.filename` statement declares the sound file that’s used in the external speaker test.

The individual settings statements are described in the **Options:** section of each of the tests, as listed in *Validation Test Specifications*.

3 Adding a Test

You can add a test to the validation suite by adding a new test name to the `testorder` statement, and then declaring the location of the test. For example, here we add a “`check_for_files`” test (a shell script):

```
testorder=keyboard;external_sound;sound;modem;ethernet;check_for_files
...
check_for_files=/boot/test/check_for_files.sh
```

The location of the test must be a full pathname.

The test can be an executable or a shell script. Shell scripts must start with the line:

```
#!/bin/sh
```

The test must return 0 (for pass) or 1 (for failure).

Note: To return a value from a shell script, use “exit <value>”.

4 Validation Operation Control

The `validate` statements in the `validate.ini` file provides control over certain parameters of the **Validate** application:

Statement	Meaning
<code>validate.oktimeout=microsecs</code>	When Validate runs to successful completion, an acceptance notice is displayed at the end of the tests. Set this statement if you want the acceptance notice to go away automatically after n microseconds.
<code>validate.autoprepare=bool</code>	Set this statement (bool=1) if you want the device to be automatically prepared for shipping after it passes Validate .
<code>validate.endalert=bool</code>	Set this statement (bool=1) if you want to skip the “Are you sure?” alert panel that’s displayed when the Validate operator clicks the Quit or Prepare buttons (and that’s also displayed if <code>validate.autoprepare</code> is set to 1).

BeIA Device Validation Validation Test Specifications

Description: Specifications of the BeIA device validation tests.

The following sections describe the BeIA device validation tests that are run by the **Validate** app. Each test description includes preparations instructions, settings options (as set in `/boot/test/validate.ini`), and the pass/fail criteria for the test.

The tests are listed in the default test order. See *Validation Settings* for instructions on changing the order of the tests, and adding and removing tests.

1 Keyboard

Description: A window displays a message that asks the operator to press a specific key. The operator has three chances to press the correct key.

Options: • `keyboard.keys=key` specifies the key that will be requested. A random key is chosen in the absence of this option.

Pass: The correct key is pressed within three attempts.

Fail: Operator clicks **Fail**, or the wrong key is pressed in all three attempts.

2 Mouse

Description: A window displays two buttons: **Click Me** and **Fail**.

Pass: The operator clicks **Click Me**.

Fail: The operator clicks **Fail**.

3 External Speaker

Preparation: External speakers must be attached to the device.

Description: A sound is played. A window displays a message that asks the operator if the sound can be heard. After the operator responds, a message tells the operator that the speakers can be removed.

Options:

- `external_sound.filename=filename` specifies the sound file that's played. `filename` can be either an absolute pathname to the file or relative to **validate's** directory (`/boot/test`). In the absence of a specified sound file, a sine tone is generated.

Pass: The operator clicks **Pass**.

Fail: The operator clicks **Fail**.

4 Internal Speakers

Preparation: The operator must be positioned as close to the (horizontal) center of the machine as possible.

Description: A sound is played in the left speaker, the right speaker, and then both speakers together. A window displays a message that asks the operator if the sounds were heard, and from the proper speakers.

Options:

- `sound.left.filename=filename` specifies the sound file that's played in the left speaker.
- `sound.right.filename=filename` specifies the sound file that's played in the right speaker.
- `sound.phase.filename=filename` specifies the sound file that's played in both speakers together.

`filename` can be either an absolute pathname to the file or relative to **validate's** directory (`/boot/test`). In the absence of a sound file specification, a sine tone is generated.

Pass: The operator clicks **Pass**.

Fail: The operator clicks **Fail**.

Notes: The “both speakers together” test ensures that the speakers are in-phase. If the operator is close to dead center, out-of-phase speakers will cancel the sound.

5 Modem

Description: A window is opened that reports the status of the test and lets the operator abort (**Fail**). The test then follows this script (without operator input):

1. Open the modem device and set some attributes (described below).
2. Write 'ATZ' to the modem and wait for an 'ATZ' response.
3. Write the `modem.init` option string, if specified, and wait for an 'OK' response.
4. Write 'ATDT `modem.number`' and wait for the last three digits of the phone number in response.
5. Wait for a 'CONNECT' response (and report the connection speed in the window).
6. Write '+++ ' and wait for an 'OK' response.
7. Write 'ATH' and wait for an 'OK' response.
8. Close the modem.

Three attempts are made at forming a connection.

- Attributes:**
- Disable input SW flow control
 - Disable output SW flow control
 - Don't allow any character to restart input
 - Disable input parity checking
 - Disable canonical input
 - Disable echo
 - Disable signals
 - Disable CTS/RTS
 - 57,600 baud
 - 8 bit, even parity, 1 stop bit
 - Local line

- Options:**
- `modem.init=string` is an optional modem initialization string.
 - `modem.number=number` is the telephone number to dial in the 'ATDT' command.
 - `modem.device=device` is the pathname of the modem device. For example, `/dev/ports/tri_modem`.

Pass: A modem connection is successfully formed within three attempts.

Fail: A connection isn't formed within three attempts, or the operator clicks **Fail**.

6 Ethernet

- Preparation:**
- A test server (`ethernet.server_address`) must be running the **testsrv** program on port 6909. Note that this must be a remote server that's set up specifically for the ethernetvalidation test—you can't run **testsrv** on the device that you're testing. The source code for the **testsrv** program, which may need to be recompiled for your server, is provided by Be.
 - Make sure you have enough 10.x.x.x DHCP leases to serve all the clients that you're testing. You'll need to hold the addresses for at least an hour.

Description: The test follows this script without operator input:

1. Cache the local network configuration info.
2. Open a TCP connection to `ethernet.server_address` on port 6909.
3. Send and receive variously sized blocks of data.
4. Compare local test data with the data received from the server.
5. Restore the cached network configuration info.

Three attempts are made at forming a connection to the test server.

A window reports the status of the test and lets the operator abort (**Fail**) while the test is in progress.

Options: `ethernet.server_address=x.x.x.x` is the IP address of the test server that's running **testsrv**.

Pass: The connection is formed within three attempts, and the results of the data comparisons are acceptable.

Fail: The connection can't be formed, the data doesn't compare properly, or the operator clicks **Fail**.

7 File Read

Preparation: To pass the File Read test, you must first run the **Make_Release_Image** program. Running **Make_Release_Image** creates a "read these files" list which is used during this test. The list comprises every file in the file system.

Description: Without operator input, each file in the file-read list is opened, some number of bytes are read from the file, the data is verified, and the file is closed.

A window reports the status of the test and lets the operator abort (**Fail**), or skip the test without recording success or failure (**Skip**).

Pass: All files on the list are (sequentially) opened and their contents are confirmed as valid.

Fail: One or more files on the list are missing, one or more files are corrupt (can't be read, or can't contain the expected data), or the operator clicks **Fail**.

Notes: The File Read test fails if run from `tools` or user boot modes.

Due to the length of the file-read list, this test can take a very long time.

If the test finds a file in the file system that isn't on the list, it displays a warning; these "extra" files don't cause the test to fail.

8 File Write

Description: Without operator input, a series of files is created, written to, read from (to verify the write), closed, and deleted. A window reports the status of the test and lets the operator abort (**Fail**), or skip the test without recording success or failure (**Skip**).

Pass: All files are successfully created, written to, verified, and deleted.

Fail: One of the files can't be created or written to, its contents isn't successfully verified, it can't be deleted, or the operator clicks **Fail**.

Notes: The File Write test takes a fraction of the time of the File Read test.

