OPEN
Compute Project

ONIE Certification Test Procedure

Contents

List of Changes

Version	Changes	Name	Date
0.1	Initial Draft	Carlos Cardenas	2014/05/27
0.2	Review, feedback, add hardware test	Thao Nguyen	2014/06/05
0.3	Added: • Testing Environment • Manual Testing	Carlos Cardenas	2014/06/10
0.4	Incorporate RFC 2119 language, clarify port enumeration, labeling requirements	Matt Peterson	2014/06/11
0.5	Added: • Test Number references • Example dhcpd.conf file	Carlos Cardenas	2014/06/25
0.6	Added: • Clarified USB install to be optional (based on HW availability) • Clarified that "luggage tag" on device is to include additional space for end user's asset tracking system	Carlos Cardenas	2014/07/24
0.7	Corrected: • Static Install • Static Update To use onie_debugargs	Carlos Cardenas	2014/08/22
0.8	Corrected: • DHCP option for Server IP • CPU architecture name for x86_64	Carlos Cardenas	2015/02/03

License

Background

Introduction

ONIE or the Open Network Install Environment is second stage boot loader to facilitate the installation of operating systems. The original implementation was built to install network operating systems on "bare metal" or whitelabel network switches. The architecture is geared to preserve a preliminary "BIOS" or initial system ROM to bring up the system (such as UBoot for PPC or SeaBIOS for x86); while ONIE functions exclusively to discover network, fetch an OS image, and execute an unattended installer. This document serves as foundation for vendors or suppliers who wish to certify their ONIE implementation under the support of the Open Compute Project.

The defined specification follows RFC 2119 language for the terms "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY".

ONIE Specification

Prerequisites

The ONIE firmware has modest hardware requirements that are required before ONIE can be used as intended in this specification.

Board EEPROM Information Format

Each ONIE system MUST include an EEPROM which contains various system parameters assigned by the manufacturer. This EEPROM includes information such as the MAC address(es) allocated to the system, the serial number, the date of manufacturer, manufacturer name etc. The name of the EEPROM format specified here is TlvInfo, because the information contained in the EEPROM is found in TLVs, or **T**ype **L**ength and **V**alue fields.

Definition of the TlvInfo EEPROM Format

The first eight bytes in the EEPROM MUST be a NULL-terminated ASCII string TlvInfo. This identification string can be used as a simple, preliminary check that the EEPROM conforms to the TlvInfo format defined here. Additional validation checks should be performed to validate that the TlvInfo format is really being used, such as validating the CRC. But this string provides a good clue, when debugging or dumping memory, that what follows is in the TlvInfo EEPROM format.

The identification string MUST BE followed by a single-byte version value. This value is set to 0x01 for the TlvInfo EEPROM format described in this document. Since the format described herein is very flexible and extensible, this value is not expected to ever change, but is included just in case. Software SHOULD NOT assume anything about the format of the data that follows this byte if it has not been written to support the version it reads from this value. The values 0x00 and 0xFF are reserved and will never be used.

The version field MUST BE followed by two bytes, which give the total length of the data that follows. This field is in big endian order and includes the cumulative length of all of the TLV fields that follow. This field MAY be used to determine the amount of data to read, if the EEPROM data is read in bulk, following the first 11 bytes. This field can also be used to determine the location of the CRC, since the CRC-32 TLV is fixed length and always the last TLV in the EERPOM.

The total length field MUST BE followed by the EEPROM system data, known as *TLV fields*. Each TLV field is composed of three sub-fields: a type code field, a length field, and a value field, in that order.

- **Type** code: This is a single byte that defines the type and format of the value field. These types are defined in the table below. Since these type codes can be added over time, software that does not understand a particular type code MUST treat the value field as opaque data, not assigning any meaning to its type or format. Type codes 0x00 and 0xFF are reserved and will never be used. This allows for up to 254 type codes.

- **Length**: This is a single byte that contains the number of bytes in the value field. Valid values of this field range from 0 to 255. A length of zero means that there is no value field associated with this type code. In that case, the byte following the length field is the first byte of the next TLV, its type code field.

- **Value**: This field contains the value for the specified type code. It may range in size from 0 to 255 bytes. The format of this field is defined, below, for each of the individual type codes. Because each TLV contains a length field, ASCII strings are not NULL-terminated, unless otherwise specified as described below.

Only the CRC-32 TLV is required to be present by this specification, but some systems may not initialize properly without the presence of other fields. The CRC-32 TLV must occur last. This field can be quickly found at the following offset in the EEPROM by adding 11 (the length of the fixed header information) + the value of the total length field - 6 (the length of the CRC-32 TLV field).

The total length of the TlvInfo EEPROM data, from the first byte of the identification string to the last byte of the CRC TLV, must be less than or equal to 2048 bytes.

The layout of the entire EEPROM block looks like:

Field Name	**Size in Bytes**	**Value**
ID String	8	“TlvInfo”
Header Version	1	0x01
Total Length	2	Total number of bytes that follow
TLV 1	Varies	The data for TLV 1
TLV 2	Varies	The data for TLV 2
.....	.....	.....
TLV N	Varies	The data for TLV N
CRC-32 TLV	6	Type = 0xFE, Length = 4, Value = 4 byte CRC-32

Table 1- Layout of entire EEPROM block

Type Code Values

The following type codes are defined.

Type Code	Length	Description	Format
0x00	None	Reserved	This type code is illegal and will never be used, so that it will be easy to detect if a portion of the EEPROM is erased.
0x21	Variable	Product Name	An ASCII string containing the product name.
0x22	Variable	Part Number	An ASCII string containing the vendor's part number for the device.
0x23	Variable	Serial Number	An ASCII string containing the serial number of the device.
0x24	6 bytes	MAC #1 Base	Six bytes containing the base MAC address for this device. The first three bytes contain the OUI of the assigning authority.
0x25	19 bytes	Manufacture Date	An ASCII string that specifies when the device was manufactured. The format of this string is: MM/DD/YYYY HH:NN:SS where MM is the month (01-12), DD is the day of the month (01-31), YYYY is the year, HH is the hour (00-23), NN is the minute (00-59), and SS is the second (00-59).
0x26	1 byte	Device Version	A single byte indicating the version, or revision, of the device.
0x27	Variable	Label Revision	An ASCII string containing the label revision.
0x28	Variable	Platform Name	An ASCII string which identifies a CPU subsystem (CPU, architecture, DRAM, NOR flash). Very useful when the CPU resides on a daughter card. Typically this includes <arch>-<machine>-<machine_revision>.
0x29	Variable	ONIE Version	An ASCII string containing the version of the ONIE software installed by the manufacturer.
0x2A	2 bytes	Num MACs	A two-byte big-endian unsigned integer describing the number of sequential MAC addresses allocated to this device, starting with the value specified in the MAC #1 Base TLV (code 0x2A). Valid values for this field range from 1 to 65535.
0x2B	Variable	Manufacturer	An ASCII string containing the name of the entity that manufactured the device.
0x2C	2 bytes	Country Code	A two-byte ASCII string containing the ISO 3166-1 alpha-2 code of the country where the device was manufactured.
0x2D	Variable	Vendor	The name of the vendor who contracted with the manufacturer for the production of this device. This is typically the company name on the outside of the device.
0x2E	Variable	Diag Version	An ASCII string containing the version of the diagnostic software.

0x2F	Variable	Service Tag	An ASCII string containing a vendor defined service tag.
0xFD	Variable	Vendor Extension	This type code allows vendors to include extra information that is specific to the vendor and cannot be specified using the other type codes. The format of this value field is a four byte IANA enterprise number, followed by a vendor defined string of bytes. The format of the string of bytes is entirely up to the vendor, except that it can be, at most, 255 bytes long, including the IANA enterprise number. If more space is needed, then multiple TLVs with this type code can be used.
0xFE	4 bytes	CRC-32	A four-byte CRC which covers the EEPROM contents from the first byte of the EEPROM (the "T" in the "TlvInfo" identification string) to the length field of this TLV, inclusive. This CRC uses the crc32 algorithm (see Python's binascii.crc32() function).
0xFF	None	Reserved	This type code is illegal and will never be used, so that it will be easy to detect if a portion of the EEPROM is erased.

Table 2 - Type Code Value Definitions

Note about MAC Addresses

A very critical characteristic of a switching platform EEPROM is the number of MAC addresses allocated to the machine. The firmware requires allocating 1 MAC address for every *serdes* on the box.

For example, consider a machine that has one Ethernet management port and a switching ASIC with 48x10G ports plus 6x40G ports. Each 40G port could be broken out into 4x10G ports. Therefore, the total number of MAC addresses this machines requires is:

```
1  -- Ethernet management port

48 -- 1 for each 48x10G port

24 -- 4 for each 6x40G port

-----------------------------

73 Total MACs
```

To encode that in the EEPROM, set TLV code 0x2A (Num MACs) to 73.

Hardware Face Plate and FRU Numbering

The following conventions for face plate and FRU numbering shall be used on the sheet metal silkscreen for the switch:

1. All enumerations start at 1, not 0. This is due to the historical nature of ONIE, where the first supported platforms where ODM based networking hardware, which chose this scheme.
2. Switch ports are labeled starting with the number "1". This scheme assumes facing at the front of the switch, or whichever side has the most predominate number of ports.
3. Switch ports are labeled top to bottom, left to right. For example, consider a 48 port switch with two rows of switch ports (two rows of 24 ports). The ports along the top row are labeled

"1, 3, 5…47" and the bottom row is labeled "2, 4, 8..48". Port should be labeled only with a number designation, no "eth#" or "swp#" pre or appended name – as each operating system my name this differently.

4. Field Replaceable Units (FRUs) are labeled starting with the number "1". This typically applies to field pluggable power supplies, fan modules, or other expansion slots. For example, a system with 2 PSUs and 3 fan modules would label the PSUs "1, 2" and the fans "1, 2, 3".

Hardware Documentation

As ONIE becomes ubiquitous in switching hardware, the hardware vendor shall include a one page (two sides printed) **Quick Start Guide** flyer.

This flyer MUST include at least the following:

- Installed ONIE version number and certification number/date
- Stencil of front and rear switch view that calls out or arrows for the follow locations:
  - First usable management Ethernet port
  - Out of band serial or console port
  - FRU's (such as fans and power supplies, enumerated as specified above)
- If the baud rate or settings information is changed from the default (115200 baud, 8/n/1 – no flow control), this data should be noted.
- Console port pin out or wiring scheme to a standard DB9 cable.
- Generic ONIE installation instructions (visual of network waterfall, file name discovery) – pointers to the ONIE website or similar tiny URL with more verbose instructions. A template of this graphical element will be available in a future specification standard.
- Generic ONIE debugging (ssh/telnet access, syslog) – pointers to ONIE website.

Access to ONIE images

Over the course of the lifetime of a given switch product, new features may be released in upstream ONIE that a vendor may want to make available to all of their relevant products.

- All ONIE released and upgrade images much be made available from a public accessible HTTP and/or FTP server. Such access MUST be freely accessible without any credentials or special login information.
- All shipping, archival, and upgrades versions SHOULD be available to download.
- Such files should be offered in a directory tree starting with the SKU name, followed by ONIE version number.

Asset Tracking and Labeling

Asset tracking is a vital part of an operator's work flow and as such, the following asset tracking labels are required:

- Human and machine readable barcodes that represent the first usable Ethernet management port (usually eth0) MAC address, system serial number, product family or SKU identification and CPU ID. Acceptable barcode encoding MUST be: Code 38, Code 128, or QR Code as ASCII text. Such encoding between human and machine-readable should be consistent (for example, a serial number printed as ASCII but encoded as HEX is not acceptable).

- For products that will be offered in a 2 or 4 post rack environment, these labels SHOULD be available on the sides of the switches, **not** on the top or bottom of the switch – as to make them readable once racked. A perfect example would be the "luggage tag" that many server vendors currently use.
- Wherever the asset locators are placed on the device there needs to be enough space for an end user to place their own asset locators. An ideal area is a "luggage tag" with enough space for an end user's asset locator.
- On platforms that support the SMBIOS/DMI standard, the serial number and product family / SKU identifier information must be encoded to match human-readable labels. For example, the encoding of the serial number (SMBIOS type 1, offset 07h) field MUST NOT be null or fake (123456789).

NOS image discovery and Installation

The firmware is required to discover a NOS image through the following methods (in order):

1. Statically configured (passed from boot loader)
2. Local file systems (USB for example)
3. Exact URLs from DHCPv4
4. Inexact URLs based on DHCP responses
5. IPv6 neighbors
6. TFTP waterfall

Once the image has been downloaded, the firmware is required to perform the installation of the NOS. At the end of installation, the firmware shall not attempt to auto discover another image, unless instructed to by the NOS or user but rather should boot the NOS installed.

In the event of an installation failure, the firmware shall repeat the NOS image discovery and installation process.

Default File Name Search Order

In a number of the following methods, ONIE searches for default file names in a specific order. All the methods use the same default file names and search order, which are described in this section.

The default installer file names are searched for in the following order:

1. `onie-installer-<arch>-<vendor>_<machine>-r<machine_revision>`
2. `onie-installer-<arch>-<vendor>_<machine>`
3. `onie-installer-<vendor>_<machine>`
4. `onie-installer-<arch>`
5. `onie-installer`

Statically configured (passed from boot loader)

The firmware shall provide this static method as an engineering function to be used during the porting of ONIE to a new platform. To use this method, a statically configured installer URL (`install_url`) shall be set by the user on the kernel command line argument prior to booting ONIE. Additional kernel command line arguments can be added by setting the `onie_debugargs` environment variable.

Local file systems (USB for example)

The firmware shall provide a method to identify a locally attached storage device to obtain a NOS installer. This method is intended for the case where the NOS installer is available on a USB memory stick (if supported by the device) plugged into the front panel. The supported file systems the firmware will support shall be vfat (common on commercially available USB sticks) and ext2.

The general algorithm for locating the installer on local storage proceeds as follows:

```
foreach $partition in /proc/partitions {
  if able to mount $partition then {
    if default file name exists {
      Add partition to found_list
    }
  }
}
foreach $partition in found_list {
  Run installer from $partition
}
```

Exact URLs from DHCPv4

The DHCP options discussed *below*, provide a number of ways to express the exact URL of the NOS installer. When interpreting URLs, the firmware accepts the following URI schemes:

- http://server/path/...
- https://server/path/...
- ftp://server/path/...
- tftp://server/path/…

The following options can be used to form the exact URL.

Option	Name	Comments
125	VIVSO	The installer URL option (code = 1). Options yields an exact URL.
114	Default URL	Intended for HTTP, but other URLs are accepted.
150 + 67	TFTP server IP and TFTP bootfile	Both options are required for an exact URL.
66 + 67	TFTP server name and TFTP bootfile	Both options are required for an exact URL. Requires DNS.

Table 3 - DHCP Options for Exact URL

Inexact URLs based on DHCP responses

The firmware can find an installer using partial DHCP information by using a default sequence of URL paths and default file names in conjunction with partial DHCP information available to find an installer.

The following DHCP option responses are used to locate an installer in conjunction with the default file names:

DHCP Options	Name	URL
67	TFTP Bootfile	Contents of bootfile
72	HTTP Server IP	http://$http_server_ip/${onie_default_installer_names}
66	TFTP Server IP	http://$tftp_server_ip/${onie_default_installer_names}
54	DHCP Server IP	http://$dhcp_server_ip/${onie_default_installer_names}

Table 4 - DHCP Options used for Inexact URLs

DHCP Reference Information

DHCP provides a powerful and flexible mechanism for specifying the installer URL. During the DHCP request, the firmware sets a number of options to help the DHCP server determine an appropriate response. The following table illustrates what options are set during the request phase.

Option	Name	ISC option-name	RFC
60	Vendor Class Identifier	vendor-class-identifier	RFC 2132
77	User Class	user-class	RFC 2132
125	Vendor-Identifying Vendor-Specific Information	vivso	RFC 3925
55	Parameter Request List	dhcp-parameter-request-list	RFC 2132

Table 5 - DHCP Request Options

Vendor Class Identifier – Option 60

The vendor class identifier option is the concatenation of two strings, separated by the colon ':' character.

1. The static string onie_vendor
2. <arch>-<vendor>_<machine>-r<machine_revision>

For example, using the example PowerPC machine, the string would be:

```
onie_vendor:powerpc-VENDOR_MACHINE-r0
```

At this time, the only valid values for CPU architecture are:

- powerpc
- x86_64

User Class – Option 77

The user class option is set to the static string

```
onie_dhpc_user_class
```

Vendor-Identifying Vendor-Specific Information – Option 125

The VIVSO option allows for custom namespaces, where the namespace is identified by the 32-bit IANA Private Enterprise Number. The firmware currently uses the enterprise number `42623` to identify its custom namespace.

The option codes within the firmware namespace have a size of 1 byte. The option payload length is also 1 byte.

Within this namespace, the following option codes are defined:

Option Code	Name	Type	Example
1	Installer URL	string	http://10.0.0.1/nos_installer.bin
2	Updater URL	string	http://10.0.0.1/onie_update.bin
3	Platform Name	string	VENDOR_MACHINE
4	CPU Architecture	string	powerpc
5	Machine Revision	string	0

Table 6 - VIVSO Namespace Option Codes

Parameter Request List – Option 55

The parameter request list option encodes a list of requested options. The firmware requests the following options:

Option	Name	ISC option-name	Option Type	RFC	Example
1	Subnet Mask	subnet-mask	dotted quad	2132	255.255.255.0
3	Default Gateway	routers	dotted quad	2132	10.0.0.1
6	Domain Server	Domain-name-servers	dotted quad	2132	10.0.0.1
7	Log Server	log-servers	dotted quad	2132	10.0.0.1
12	Hostname	host-name	string	2132	switch-01
15	Domain Name	domain-name	string	2132	example.com
42	NTP Servers	ntp-servers	dotted quad	2132	10.0.0.1
54	DHCP Server Identifier	dhcp-server-identifier	dotted quad	2132	10.0.0.1
66	TFTP Server Name	tftp-server-name	string	2132	bootserver
67	TFTP Bootfile Name	bootfile-name or filename	string	2132	tftp/installer.sh
72	HTTP Server IP	www-server	dotted quad	2132	10.0.0.1
114	Default URL	default-url	string		http://server/install
150	TFTP Server IP Address	next-server	dotted quad		10.0.0.2

Table 7 - DHCP Parameter Request List Options

HTTP Requests and Headers

All HTTP requests made by the firmware include a set of standard HTTP headers, which an HTTP automation system could utilize. The headers sent on each HTTP request are:

Header	Value	Example
ONIE-SERIAL-NUMBER	Serial number	XYZ123004
ONIE-ETH-ADDR	Management MAC address	08:9e:01:62:d1:93
ONIE-VENDOR-ID	32-bit IANA Private Enterprise Number	12345
ONIE-MACHINE	<vendor>_<machine>	VENDOR_MACHINE
ONIE-MACHINE-REV	<machine_revision>	0
ONIE-ARCH	CPU architecture	powerpc
ONIE-OPERATION	ONIE mode of operation	`os-install` or `onie-update`

ONIE-VERSION	Version of ONIE	`onie/1.0 (2.6.13)`

Table 8 - HTTP Headers

IPv6 neighbors

The firmware shall also query its IPv6 link-local neighbors via HTTP for an installer. The general algorithm follows:

`Ping6` the "all nodes" link local IPv6 multicast address, `ff02::1`

For each responding neighbor, try to download the default file names from the root of the web server

The following is an example of URLs used by this method:

```
http://fe80::4638:39ff:fe00:139e%eth0/onie-installer-powerpc-VENDOR_MACHINE-r0

http://fe80::4638:39ff:fe00:139e%eth0/onie-installer-powerpc-VENDOR_MACHINE

http://fe80::4638:39ff:fe00:139e%eth0/onie-installer-VENDOR_MACHINE

http://fe80::4638:39ff:fe00:139e%eth0/onie-installer-powerpc

http://fe80::4638:39ff:fe00:139e%eth0/onie-installer

http://fe80::4638:39ff:fe00:2659%eth0/onie-installer-powerpc-VENDOR_MACHINE-r0

http://fe80::4638:39ff:fe00:2659%eth0/onie-installer-powerpc-VENDOR_MACHINE

http://fe80::4638:39ff:fe00:2659%eth0/onie-installer-VENDOR_MACHINE

http://fe80::4638:39ff:fe00:2659%eth0/onie-installer-powerpc

http://fe80::4638:39ff:fe00:2659%eth0/onie-installer

http://fe80::230:48ff:fe9f:1547%eth0/onie-installer-powerpc-VENDOR_MACHINE-r0

http://fe80::230:48ff:fe9f:1547%eth0/onie-installer-powerpc-VENDOR_MACHINE

http://fe80::230:48ff:fe9f:1547%eth0/onie-installer-VENDOR_MACHINE

http://fe80::230:48ff:fe9f:1547%eth0/onie-installer-powerpc

http://fe80::230:48ff:fe9f:1547%eth0/onie-installer
```

TFTP waterfall

The firmware shall include a classic PXE-like TFTP waterfall method for locating a NOS installer image. Given a TFTP server address, the firmware will attempt to download the installer using a sequence of TFTP paths with decreasing levels of specificity.

An example of this method is as follows:

```
55-66-aa-bb-cc-dd/onie-installer-<arch>-<vendor>_<machine>

C0A801B2/onie-installer-<arch>-<vendor>_<machine>

C0A801B/onie-installer-<arch>-<vendor>_<machine>

C0A801/onie-installer-<arch>-<vendor>_<machine>
```

```
C0A80/onie-installer-<arch>-<vendor>_<machine>

C0A8/onie-installer-<arch>-<vendor>_<machine>

C0A/onie-installer-<arch>-<vendor>_<machine>

C0/onie-installer-<arch>-<vendor>_<machine>

C/onie-installer-<arch>-<vendor>_<machine>

onie-installer-<arch>-<vendor>_<machine>-<machine_revision>

onie-installer-<arch>-<vendor>_<machine>

onie-installer-<vendor>_<machine>

onie-installer-<arch>

onie-installer
```

NOS Uninstallation

The firmware is required to allow a user to completely remove the current NOS installed leaving only the firmware intact. The firmware will wipe out all unused portions of NOR flash and the attached mass storage device (like an SD card or USB NAND flash). The only thing untouched is the firmware itself. This is akin to 'reset to factory defaults'.

Update ONIE

The firmware is required to provide a facility to self-update via the same method used in NOS image discovery and installation except for a firmware image rather than a NOS image.

The firmware is required to discover an ONIE image through the following methods (in order):

1. Statically configured (passed from boot loader)
2. Local file systems (USB for example)
3. Exact URLs from DHCPv4
4. Inexact URLs based on DHCP responses
5. IPv6 neighbors
6. TFTP waterfall

Default File Name Search Order

ONIE searches for default file name for the updated image in a specific order. All update methods use the same default file names and search order, which are described in this section.

The default ONIE image file names are searched for in the following order:

1. `onie-updater-<arch>-<vendor>_<machine>-r<machine_revision>`
2. `onie-updater-<arch>-<vendor>_<machine>`
3. `onie-updater-<vendor>_<machine>`
4. `onie-updater-<arch>`
5. `onie-updater`

Statically configured (passed from boot loader)

The firmware shall provide this static method as an engineering function to be used during the porting of ONIE to a new platform. To use this method, a statically configured updater URL (`install_url`) shall be set by the user on the kernel command line argument prior to booting ONIE. Additional kernel command line arguments can be added by setting the `onie_debugargs` environment variable.

Local file systems (USB for example)

The firmware shall provide a method to identify a locally attached storage device to obtain a firmware updater. This method is intended for the case where the new firmware image is available on a USB memory stick (if supported by the device) plugged into the front panel. The supported file systems the firmware will support shall be vfat (common on commercially available USB sticks) and ext2.

The general algorithm for locating the installer on local storage proceeds as follows:

```
foreach $partition in /proc/partitions {
  if able to mount $partition then {
    if default file name exists {
      Add partition to found_list
    }
  }
}
foreach $partition in found_list {
  Run updater from $partition
}
```

Exact URLs from DHCPv4

The firmware shall provide a method to use a URL from a DHCPv4 response. Please refer to the section *above* for details.

Inexact URLs based on DHCP responses

The firmware can find an updater using partial DHCP information by using a default sequence of URL paths and default file names in conjunction with partial DHCP information available to find an updater image. Please refer to the section *above* for details.

IPv6 neighbors

The firmware shall also query its IPv6 link-local neighbors via HTTP for an updater image. Please refer to the section *above* for details.

TFTP waterfall

The firmware shall include a classic PXE-like TFTP waterfall method for locating an updated firmware image. Given a TFTP server address, the firmware will attempt to download the updater using a sequence of TFTP paths with decreasing levels of specificity.

An example of this method is as follows:

```
55-66-aa-bb-cc-dd/onie-updater-<arch>-<vendor>_<machine>

C0A801B2/onie-updater-<arch>-<vendor>_<machine>

C0A801B/onie-updater-<arch>-<vendor>_<machine>

C0A801/onie-updater-<arch>-<vendor>_<machine>

C0A80/onie-updater-<arch>-<vendor>_<machine>

C0A8/onie-updater-<arch>-<vendor>_<machine>

C0A/onie-updater-<arch>-<vendor>_<machine>

C0/onie-updater-<arch>-<vendor>_<machine>

C/onie-updater-<arch>-<vendor>_<machine>

onie-updater-<arch>-<vendor>_<machine>-<machine_revision>

onie-updater-<arch>-<vendor>_<machine>

onie-updater-<vendor>_<machine>

onie-updater-<arch>

onie-updater
```

Rescue Mode

The firmware is required to provide a rescue mode as a failsafe, to perform diagnostics and to reload a new NOS. The rescue mode is the same as the discovery and installation phase, but the discovery mechanism is disabled. The firmware shall not try to locate and install a NOS image, but rather allow troubleshooting of the current system.

In this mode of operation, the firmware is accessible via the serial console or via *telnet*. A user can use the available BusyBox toolset to attempt to fix the problem or use *wget* to retrieve additional resources.

Execution Environment

After the firmware locates and downloads an installer, the next step is to execute the installer.

Prior to execution, the firmware prepares an execution environment:

1. `chmod +x` on the downloaded installer
2. Export a number of environment variables (defined *below*) which are usable by the installer
3. Executes the installer

Variable Name	Meaning
onie_exec_url	Currently executing URL
onie_platform	CPU architecture, vendor and machine name
onie_vendor_id	32-bit IANA Private Enterprise Number
onie_serial_num	Device serial number
onie_eth_addr	MAC address for Ethernet management port
onie_version	ONIE build version number

Table 9 - Installer Core Environment Variables

In addition to the environment variables, any and all DHCP response options are exported, in the style of BusyBox's `udhcpc`. Those variables are as follows:

Variable Name	**Meaning**
onie_disco_dns	DNS Server
onie_disco_domain	Domain name from DNS
onie_disco_hostname	Switch hostname
onie_disco_interface	Ethernet management interface (i.e. eth0)
onie_disco_ip	Ethernet management IP address
onie_disco_router	Gateway
onie_disco_serverid	DHCP server IP
onie_disco_siaddr	TFTP server IP
onie_disco_subnet	IP netmask
onie_disco_vivso	VIVSO option data

Table 10 - Installer DHCP Environment Variables

ONIE Testing

Test Environment

In order to test an ONIE device, the following is required:

- ONIE device
- Vendor provided serial console cable (for device interaction and recording of session)
- CAT5/CAT6 RJ45 cable (for image discovery and delivery)
- PC with serial terminal and RJ45 NIC

Recommended environment - Linux

- Latest Linux distribution (e.g. Debian) with IPv6 enabled
- screen(1) or minicom(1)
  - 115200 baud 8N1, no flow control
  - Logging enabled
- ISC DHCP Server
  - See *Appendix A – dhcpd.conf example file* or https://github.com/onie/onie/blob/master/contrib/isc-dhcpd/dhcpd.conf
- Web Server
  - Apache httpd
  - nginx
  - lighttpd
- TFTP server
  - atftpd
- USB memory stick (if supported by the device)
  - 2GB is sufficient

Recommended environment - Windows

- Windows 8.1 Update
- PuTTY or Tera Term

  - 115200 baud 8N1, no flow control
  - Logging enabled
- dhcpsrv (http://www.dhcpserver.de/dhcpsrv.htm)
  - Also includes web server and tftp server
- Web Server
  - IIS
  - Apache httpd
- TFTP server
  - WinAgents TFTP server
- USB memory stick (if supported by the device)
  - 2GB is sufficient

Dead on Arrival Testing

Prior to the start of the testing the ONIE device, the ONIE Certification lab will be performing the following functionality testing

1. Power on the switch, perform any vendor diagnostic test, if applicable.
2. Verify the hardware configuration such as CPU, memory, flash storage, USB, Network ports, labeling and asset tracking
3. Perform warm reboot 20 x AC power cycles
4. Perform cold boot 20 x AC power cycles

PASS Criteria: Switch boots from warm and cold boot. Tests 0 and 1.

Manual Testing

Preliminary

Before putting a device under test, it is best to know the available environment variables and options as they will be used in a variety of ways (naming schemes for install and update images, host options, etc…). Below is a print out of using the `printenv` command from UBoot.

```
LOADER=> printenv

autoload=no

baudrate=115200

bootargs=root=/dev/ram rw console=ttyS0,115200 quiet

bootcmd=run check_boot_reason; run nos_bootcmd; run onie_bootcmd

bootdelay=10

check_boot_reason=if test -n $onie_boot_reason; then setenv onie_bootargs
boot_reason=$onie_boot_reason; run onie_bootcmd; fi;

consoledev=ttyS0

dhcp_user-class=powerpc-as4600_54t_uboot

dhcp_vendor-class-identifier=powerpc-as4600_54t
```

```
ethact=eth0

ethaddr=70:72:CF:AA:34:FA

ethprime=eth0

gatewayip=192.168.1.10

hostname=es4654bf_zz-unknown

ipaddr=192.168.1.10

loadaddr=0x2000000

loads_echo=1

netmask=255.255.255.0

nos_bootcmd=echo

onie_args=run onie_initargs onie_platformargs

onie_bootcmd=echo Loading Open Network Install Environment ...; echo
Platform: $onie_platform ; echo Version : $onie_version ; cp.b $onie_start
$loadaddr ${onie_sz.b} && run onie_args && bootm ${loadaddr}#$platform

onie_initargs=setenv bootargs quiet console=$consoledev,$baudrate

onie_machine=as4600_54t

onie_machine_rev=0

onie_platform=powerpc-as4600_54t

onie_platformargs=setenv bootargs $bootargs serial_num=${serial#}
eth_addr=$ethaddr $onie_bootargs $onie_debugargs

onie_rescue=setenv onie_boot_reason rescue && boot

onie_start=0xefB60000

onie_sz.b=0x00400000

onie_uninstall=setenv onie_boot_reason uninstall && boot

onie_update=setenv onie_boot_reason update && boot

onie_vendor_id=259

platform=as4600_54t

serial#=460054T1406013

serverip=192.168.1.99

stderr=serial

stdin=serial

stdout=serial

ver=U-Boot 2013.01.01-g73423af-dirty (Jan 10 2014 - 21:00:23) - 3.0.1.6
```

```
Environment size: 1584/65532 bytes

LOADER=>
```

Of interest for testing,

Variable	Value
MAC Address	70:72:CF:AA:34:FA
arch	powerpc
vendor	259
machine	as4600_54t
machine_revision	0

Table 11- ONIE Variables from Device

NOS image discovery and Installation

Statically configured (passed from boot loader)

Prior to booting into ONIE, the environment variable `install_url` needs to be set. To ensure ONIE will perform the installation regardless if there is a NOS installed, the variable `onie_boot_reason` needs to be set to `install`.

```
LOADER=> setenv onie_boot_reason install

LOADER=> setenv onie_debugargs install_url=<URL>

LOADER=> boot
```

PASS Criteria: ONIE installs the specified image. Test 2.

Local file systems (USB for example)

Prior to booting ONIE, a USB memory stick with an ONIE image conforming to the naming scheme *above*. Boot device. This test is only valid for those devices that contain a USB port.

PASS Criteria: ONIE installs image from USB device using all options of the naming scheme. Tests 3 – 7.

Exact URLs from DHCPv4

Prior to booting ONIE, ensure the ONIE image server has the DHCP server configured to parse out VIVSO (defined *above*) and other DHCP options. When using `default-url`, please ensure the appropriate service (ftp, http, or tftp) is enabled.

PASS Criteria: ONIE installs image using Exact URLs from DHCPv4 (all 4 targets). Tests 8 – 11.

Inexact URLs based on DHCP responses

Prior to booting ONIE, ensure the ONIE image server has the DHCP server configured with four options (configured one at a time). Please refer to *Inexact URLs based on DHCP responses* for the four options. All options except for the TFTP bootfile, will locate the image by conforming to the naming scheme defined *above*.

PASS Criteria: ONIE installs image using Inexact URLs from 4 DHCPv4 options (all 16 tests). Tests 12 – 27.

IPv6 neighbors

Prior to booting ONIE, ensure the ONIE image server has IPv6 configured and running a web server with the ONIE images conforming to the naming scheme *above*.

PASS Criteria: ONIE installs image from IPv6 neighbor device using all options of the naming scheme. Tests 28 – 32.

TFTP waterfall

Prior to booting ONIE, ensure the ONIE image server has the TFTP service enabled and configured.

PASS Criteria: ONIE installs image from TFTP waterfall using all options of the naming scheme. Tests 33 – 37.

NOS Uninstallation

To perform the NOS uninstallation will depend on where in the boot process the device is in.

If the device is powered off:

- Boot device
- Break out to UBoot prompt

```
LOADER=> run onie_uninstall
```

Otherwise

```
# fw_setenv onie_boot_reason uninstall

# reboot
```

PASS Criteria: ONIE boots up and performs the uninstallation phase erasing all blocks of the previous image. Tests 74 – 76.

Update ONIE

Statically configured (passed from boot loader)

Prior to booting into ONIE, the environment variable `install_url` needs to be set. To ensure ONIE will perform the upgrade regardless if there is a NOS installed, the variable `onie_boot_reason` needs to be set to `update` .

```
LOADER=> setenv onie_boot_reason update

LOADER=> setenv onie_debugargs install_url=<URL>

LOADER=> boot
```

PASS Criteria: ONIE installs the specified image. Test 38.

Local file systems (USB for example)

Prior to booting ONIE, a USB memory stick with an ONIE image conforming to the naming scheme *above*. Boot device. This test is only valid for those devices that contain a USB port.

PASS Criteria: ONIE updates image from USB device using all options of the naming scheme. Tests 39 – 43.

Exact URLs from DHCPv4

Prior to booting ONIE, ensure the ONIE image server has the DHCP server configured to parse out VIVSO (defined *above*) and other DHCP options. When using `default-url`, please ensure the appropriate service (ftp, http, or tftp) is enabled.

PASS Criteria: ONIE updates image using Exact URLs from DHCPv4 (all 4 targets). Tests 44 – 47.

Inexact URLs based on DHCP responses

Prior to booting ONIE, ensure the ONIE image server has the DHCP server configured with four options (configured one at a time). Please refer to *Inexact URLs based on DHCP responses* for the four options. All options except for the TFTP bootfile, will locate the image by conforming to the naming scheme defined *above*.

PASS Criteria: ONIE updates image using Inexact URLs from 4 DHCPv4 options (all 16 tests). Tests 48 – 63.

IPv6 neighbors

Prior to booting ONIE, ensure the ONIE image server has IPv6 configured and running a web server with the ONIE images conforming to the naming scheme *above*.

PASS Criteria: ONIE updates image from IPv6 neighbor device using all options of the naming scheme. Tests 64 – 68.

TFTP waterfall

Prior to booting ONIE, ensure the ONIE image server has the TFTP service enabled and configured.

PASS Criteria: ONIE updates image from TFTP waterfall using all options of the naming scheme. Tests 69 – 73.

Rescue Mode

To enter rescue mode will depend on where in the boot process the device is in.

If the device is powered off:

- Boot device
- Break out to UBoot prompt

```
LOADER=> run onie_rescue
```

Otherwise:

```
# fw_setenv onie_boot_reason rescue
```

```
# reboot
```

PASS Criteria: ONIE boots up without the discover mechanism running. Tests 77 – 79. Verify with boot screen saying:

```
discover: Rescue mode detected.  Installer disabled.
```

Or via `ps w`**.**

Appendix A – dhcpd.conf example file

```
# Sample configuration demonstrating many ONIE install options

ddns-update-style none;
option domain-name "ocp-labs.local";
option domain-name-servers 192.168.1.1;

default-lease-time 600;
max-lease-time 7200;

# Create an option namespace called ONIE for VIVSO (option 125)
option space onie code width 1 length width 1;

# Define the code names and data types within the ONIE namespace
option onie.installer_url code 1 = text;
option onie.updater_url   code 2 = text;
option onie.machine       code 3 = text;
option onie.arch          code 4 = text;
option onie.machine_rev   code 5 = text;

# Package the ONIE namespace into option 125
option space vivso code width 4 length width 1;
option vivso.onie code 42623 = encapsulate onie;
option vivso.iana code 0 = string;
option op125 code 125 = encapsulate vivso;

# Optionally add syslog server for logging
# option log-servers 192.168.1.3;

log-facility local7;

# Logging constructs to assist with debugging
```

```
log(error, concat("vendor-class: ", substring(option vendor-class-identifier,
0, 11)));
log(error, concat("platform    : ", substring(option vendor-class-identifier,
12, 999)));

# Parses vendor-class-identifier and adjusts the default-url
class "onie-vendor-X-class" {
  match if substring(option vendor-class-identifier, 0, 27) =
"onie_vendor:powerpc-VendorX";
  option default-url = "http://onie-server/VendorX-onie-installer";
}

# VIVSO example
class "onie-vendor-classes" {
  # Limit the matching to a request we know originated from ONIE
  match if substring(option vendor-class-identifier, 0, 11) = "onie_vendor";

  # Required to use VIVSO
  option vivso.iana 01:01:01;

  # generic CPU architecture matching
  if option onie.arch = "powerpc" {
    option onie.installer_url = "http://onie-server/generic-powerpc-onie-
installer";
  }

  # matching on CPU architecture and machine type
  if option onie.arch = "powerpc" and option onie.machine = "XYZ1234" {
    option onie.installer_url = "http://onie-server/powerpc-xyz1234-onie-
installer";
  }

  # The contents of an option can also be used to create the response text
  if exists onie.arch and exists onie.machine and exists onie.machine_rev {
    option onie.installer_url = concat("http://onie-server/image-installer-",
                                     option onie.arch, "-", option
onie.machine,
                                     "-r", option onie.machine_rev);
  }

  # When operating in ONIE 'update' mode ONIE will check the
  # onie.updater_url response option
  if option onie.arch = "powerpc" and option onie.machine = "XYZ1234" {
    option onie.updater_url = "http://onie-server/onie-updater-
VENDOR_XYZ1234-powerpc.bin";
  }

}

# Uses the default-url option for ONIE
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.150 192.168.1.240;
    authoritative;
    option routers 192.168.1.1;
    option broadcast-address 192.168.1.255;
    option default-url = "http://192.168.1.2/custom-onie-installer";
    # Below is the same but uses DNS resolution
    # option default-url = "http://onie-server/custom-onie-installer";
}

# Typical tftp waterfall example
# ONIE will also try to use HTTP on the next-server and dhcpd server
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.150 192.168.1.240;
    authoritative;
    option routers 192.168.1.1;
    option broadcast-address 192.168.1.255;
    next-server 192.168.1.1;
}
```