



Setting Manufacturing Certificates and Installation Codes

For the EM250 and EM35x SoC Platforms, and EM260 Co-Processor

New in this Revision

Overriding the default EUI64 instruction for the EM2xx is updated.

File locations updated throughout, some minor corrections made.

Contents

Smart Energy Certificate Overview.....	4
Security Components	4
Certificate	4
Static private key	4
CA public key.....	5
Certificate Authorities	5
Test CA.....	5
Production CA.....	5
Certificate Validation	6
Matching the Components	6
Manufacturing test	6
Working with Certificates on the EM2xx.....	7
Overriding the default EUI64	7
Verifying the software version	7
Programming the Certificates into a Device	8
Checking the node information	8
Format of the certificate file.....	10



Writing the certificate into the manufacturing area	10
Verifying the stored certificate	11
Erasing the Certificate	12
Running em2xx_patch	12
Working with Certificates on the EM35x.....	13
Overriding the default EUI64	13
Verifying the software version	14
Programming the Certificates into a Device	14
Checking the node information	14
Format of the certificate file.....	15
Writing the certificate into the manufacturing area	16
Verifying the stored certificate	17
Erasing the Certificate	18
Smart Energy Installation Code Overview	18
Security Use	19
Installation Code Format.....	19
Installation Code CRC.....	20
Validation.....	20
Generation	20
Labels	20
Programming the Installation Code on an EM250	21
Checking the installation code on a device	21
Format of the installation code file	22
Writing the installation code into the manufacturing area	22
Verifying the stored installation code.....	23
Erasing the Installation Code.....	23
Running the em2xx_patch	23
Programming the Installation Code on an EM35x.....	24

Checking the installation code on a device	24
Format of the installation code file	25
Writing the installation code into the manufacturing area	26
Verifying the stored installation code.....	27
Erasing the Installation Code.....	28
Erase process.....	28

Smart Energy Certificate Overview

The ZigBee Smart Energy specification uses public/private key technology to authenticate a device joining a Smart Energy network and provides a means to securely establish encryption keys for future transactions. The Smart Energy specification uses Elliptical Curve Cryptography (ECC) for cryptographic authentication and key generation.

The Smart Energy profile uses ECC with the key establishment cluster to derive a link key. It also uses ECC for creating digital signatures for software upgrade images sent through the ZigBee Over-the-air bootloader cluster (OTA).

Currently Certicom (www.certicom.com) provides both the certificates and the ECC technology for use in ZigBee Smart Energy networks.

Security Components

When referring to Smart Energy Certificates, there are actually three components that make up the ECC security data contained within a node:

- Certificate
- Static Private Key
- CA Public Key

Certificate

The first component of the ECC security data is the certificate. This is the device's IEEE address and static public key, which are signed using the CA's private key. It is 48 bytes in length. The certificate is associated with a particular IEEE address. As a result, the IEEE address of the device must match the value contained in the certificate.

Although this is part of the security data, it is not secret, and does not need to be handled in the same fashion as the private key. It is transmitted over the air during key establishment. Table 1 summarizes the four different fields contained within the certificate body.

Table 1: Certificate body fields

Field name	Length (bytes)	Description
Public Key Reconstruction Data	22	Device's public key signed by the CA's private key.
Subject	8	Contains the IEEE address associated with the certificate, in big endian format.
Issuer	8	Identity of the CA that issued the certificate.
Attributes	10	An extra set of data associated with the device whose authenticity is guaranteed by the CA. It is not currently used by ZigBee Smart Energy.

Static private key

The second component of the ECC security data is the static private key. This is a secret piece of data that should be carefully protected. Ember recommends that during the manufacturing process only those computers that need to know this data to program the Ember chip have access to it.

Certificate Authorities

CA public key

This is the CA's public key that corresponds to the secret private key held by Certicom. It is used to authenticate certificates received by remote devices and validate the keys derived using the Smart Energy Key Establishment Cluster. While it is not transmitted over the air, it is also public information, and does not need to be kept secret.

Two Certificate Authorities (CA) are provided by Certicom for use in Smart Energy networks:

- A Test CA that is intended only for use in internal development environments.
- The Production CA that is intended for use in real Smart Energy deployments.

Note: Certificates from the Test CA and the Production CA are **not compatible**.

There are two CA's because developers and installers have different security needs. Developers need to be able to test and debug devices where they will have access to the private keys. Their interests lie in having access to internal data in order to accomplish their goal of writing software. On the other hand, installers want reasonable assurances that a device will not have compromised security data. Their goal is to have a secure installation. Certicom can meet both needs by providing different security for each CA.

Test CA

Certicom provides a Test CA for quickly and easily generating certificates for use in development units wishing to use ECC. The following link can be used to obtain certificates from the Test CA: <http://www.certicom.com/index.php/qencertregister>.

Ember provides two test certificates in their Smart Energy sample application, bound to IEEE addresses 0022080000000001 and 0022080000000002. These may be used in combination with the Test CA to perform key establishment. They can be found in the Ember ECC distribution ZIP file as *certicom-test-cert-1.txt* and *certicom-test-cert-2.txt*. The following is the CA Public Key for the Test CA:

```
0200FDE8A7F3D1084224962A4E7C54E69AC3F04DA6B8
```

Production CA

The Production CA is the official CA that is used by all deployed devices. All manufactured devices should contain the Production CA's public key, and a certificate and private key associated with the Production CA.

The following is the Production CA's Public Key, which should be installed on production devices:

```
0202264C5E4CBFA186A6B925B966B5B3A4D7A390344E
```

Production CAs are obtained through a different mechanism from Test CAs. Since manufacturing will involve hundreds or potentially thousands of devices, a different process is used to obtain bulk certificates for a block of IEEE addresses. In addition, security of the

transmission of the private keys is much more important and thus is handled differently from the Test CA.

It is recommended that a manufacturer obtain their own block of IEEE addresses for production units instead of using the internal Ember one programmed on the chip. Certificates identify a Smart Energy device and its vendor and Ember's chip may be used in many Smart Energy deployments. In addition, it is easier to issue certificates for blocks of IEEE addresses. Ember cannot guarantee a set of known IEEE addresses with a particular set of chips, so it is recommended that the manufacturer install their own to make it easier to manage them.

Contact Certicom for more information about obtaining production certificates.

Certificate Validation

It is important to note that ECC uses a technology called *implicit certificates*. Traditional SSL certificates used on the Internet contain the unencrypted public key of the device along with a separate signature field that ensures the authenticity of the data. However, with implicit certificates, the two pieces of data are combined together to shrink the size of the certificate for use in embedded devices. This is known as *Public Key Reconstruction data*.

As a result of the combination, it is not possible to verify an implicit certificate without using the certificate as part of the Key Establishment Cluster. Only by performing Key Establishment to derive a link key can a device determine if its certificate is valid.

Matching the Components

It is important to note that the four pieces of security data installed for Smart Energy must all match up correctly. The installed certificate must be the correct one for the private key that is also installed. The IEEE address of the device must match the certificate that is installed. Lastly, the certificate must have been issued by the same CA whose public key is installed along with it.

If any of the components is mismatched, the device will never be able to successfully perform key establishment to authenticate itself on a Smart Energy network.

Manufacturing test

Ember recommends that manufacturers validate their process by testing the installed certificate in their first batch of devices. A full test involves successfully joining a Smart Energy network and using the key establishment cluster to derive a link key. The devices in the test should be using Production Certificates issued by the Production CA.

The EM250/EM35x SoC platforms and EM260 co-processor can all store application-specific manufacturing data into an area of flash that can be used to store unique device information. This information is read-only at run-time and must be programmed via the InSight Adapter.

Note: Hard-coding a device's unique information in source code does not work when using the same application image on multiple devices. The preferred alternative is to have the software read in the device's unique information from the manufacturing area of flash.

Working with Certificates on the EM2xx

Overriding the default EU164

By default, an Ember EU164 address is pre-programmed into all EM2xx chips. Customers may override this by writing their own EU164 into the Custom EU164 area of the manufacturing flash.

When programming certificates and installation codes, the `em2xx_patch` tool allows you to override the Ember EU164 by supplying a custom EU164 value. When this is done, the Ember EU164 is still preserved, but runtime queries for the device's EU164 will now return the custom EU164 value. When a custom EU164 is already present, the certificate or installation code *must* use an EU164 that matches the custom one.

If you do not want to override the Ember EU164, then the certificate and installation codes must use Ember EU164s, and those EU164s *must* match what is already programmed on the chip. You cannot override the EU164 of the device with another Ember EU164 (one whose most significant bytes are 0x000D6F...).

The reason these requirements are in place is to prevent certificates or installation codes from being programmed on the wrong device. Both of these items are bound to the EU164 of the device and therefore a mismatch will cause problems when deploying an end product.

To program a device with a certificate or installation code, the device must be connected to an Ember InSight Adapter. For the 250 and 260 platforms, programming is done with the SIF tools (`em2xx_read.exe` and `em2xx_patch.exe`) provided with the xIDE 2.0 or later (for EM250 platform) or the EmberZNet PRO 3.3 or later stack installation (for EM260 platform). Refer to Ember Document 120-4020-000: *EM2xx Utilities Guide* for more information about how these command-line-based tools work.

Verifying the software version

To verify that you have the correct version of the tools, open a command prompt, and change to the directory where the tools are installed. The default for the xIDE 2.0 or xIDE 2.1 Installation is `C:\Program Files\Ember\xIDE_EM250\SIF\bin\`. For the EM260, the default location is `C:\Program Files\Ember\EmberZNet{version}\em260\tool\em2xx_load\`.

To verify the software version, run the tool without any arguments. You should see output similar to the following.

```
C:\Program Files\Ember\xIDE_EM250\SIF\bin> em2xx_patch.exe
em2xx_patch.exe Version 3.0 Build 1
Emsif Library Version 1.0.6
```

If the software version of the tool is at least 3.0 build 1, then it will support programming certificates and installation codes. If you have an older version, visit the Ember Customer Support portal at http://www.ember.com/support_index.html to download the latest copy of xIDE.

Programming the Certificates into a Device

Checking the node information

Prior to modifying the certificate it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. To do this, execute the following command:

```
em2xx_read.exe -id <ip-address> -Mfg -print -Run
```

```
em2xx_read.exe -id 192.168.221.51 -Mfg -print -Run
em2xx-read Version 3.2 Build 1
Emsif Library Version 1.0.6
Opening InSight Adapter with hostname 192.168.221.51...
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed

'General' token group
TOKEN_MFG_CUSTOM_VERSION [ 2 bytes] : FFFF
TOKEN_MFG_CUSTOM_EUI_64 [ 8 bytes] : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING [16 bytes] : " " (0 of 16 characters)
                                FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME [16 bytes] : " " (0 of 16 characters)
                                FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID [ 2 bytes] : FFFF
TOKEN_MFG_PHY_CONFIG [ 2 bytes] : FFFF
TOKEN_MFG_BOOTLOAD_AES_KEY [16 bytes] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE [ 8 bytes] : FFFFFFFFFFFFFFFF
TOKEN_MFG_ASH_CONFIG [ 2 bytes] : FFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [ 2 bytes] : FFFF

'Smart Energy CBKE (TOK_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 bytes] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CA Public Key [22 bytes] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                FFFFFFFFFFFFFFFF
```

```

Device Private Key  [21 bytes] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                               FFFFFFFFFFFFFFFF
CBKE Flags          [ 1 byte ] : FF

'Smart Energy Install Code (TOK_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 bytes] : FFFF
Install Code       [16 bytes] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC                [ 2 bytes] : FFFF

```

Note: The pre-programmed Ember EUI64 is not shown in this output as it is part of the Ember-programmed manufacturing data rather than the customer-programmed manufacturing data. To obtain the Ember EUI64 of an EM2xx device, use the following command (shown for a device with hostname "133bxi"):

```
em2xx_read -id 133bxi -Mfg @5014+8
```

This reads out the Ember EUI64 in least-significant byte (LSB) notation. For example, the command above produces the following output (indicating an Ember EUI64 value of 0x000D6F00000F1EB2):

```

em2xx_read.exe Version 3.2 Build 1
Emsif Library Version 1.0.6
Opening InSight Adapter with hostname 133bxi...
  XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= Reset performed
  XAP2= Issuing SW_RESET
  XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= Reset performed
Manuf:
###--> addrHi=d addrLo=a
#0008:      .      .B21E.0F00.006F.0D00.      .      .....0..
  XAP2= Stop performed
  XAP2= Reset performed
XAP is stopped, ready to debug ...

```

If the custom EUI64 (IEEE address) is set to all F's, then the Ember EUI64 address is used. If this value is not all F's, then it used instead of the Ember EUI64.

If the security data (certificate, private key, root CA public key, metadata) is set to all F's then the data is not valid and will not be used by the software.

Format of the certificate file

To program a certificate into the tokens, it is necessary to create a simple text file with the security information in it. The following is the format of that file expected by the `em2xx_patch.exe` tool.

```
CA Public Key: <hex-string>
Device Implicit Cert: <hex-string>
Device Private Key: <hex-string>
```

Note: Previously the tool required a field called "Device Public Key". This field was never used but its presence was required. 3.1 build 4 of the `em2xx_patch` tool no longer has this requirement.

The following is a sample certificate file issued by the Test CA, for a device with an IEEE of 0000000000000001 (big endian).

```
CA Public Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert:
03045fd8d85ffb8b3993cb72ddcaa55f00b3e87d6d0000000000000015445535453454
341010900060000000000000
Device Private Key: 00b8a900fcadebabbfa383b540fce9ed438395eaa7
```

Writing the certificate into the manufacturing area

To write the certificate into the manufacturing area, execute the following command:

```
em2xx_patch -id <host | ip-address> -Mfg -loadTokens<certificate-file> -Run
```

For example, you should see something similar to the following:

```
em2xx_patch.exe -id 192.168.221.51 -Mfg -loadTokens 0000000000000001.txt
-Run
em2xx-patch Version 3.2 Build 1
Emsif Library Version 1.0.6

Writing to address 0x00005128 for token 'Device Implicit Cert'
Writing to address 0x00005158 for token 'CA Public Key'
Writing to address 0x0000516E for token 'Device Private Key'
Writing to address 0x00005183 for token 'CBKE Flags'
Opening InSight Adapter with hostname 192.168.221.51...
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed

NOTE: Writing Custom EUI64 '0100000000000000' to match certificate.
Address 0x00005082.

Programming FLASH...
```

```

XAP2= Run performed
MFGcu 0041-0044 (0004)
MFGcu 0094-00c1 (002e)
Total Manuf Used=0032 words
XAP2= Stop performed
Verifying FLASH...

****Application load: SUCCESS****
****Verification:      SUCCESS****

XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET

```

In addition to writing the certificate, this will update the custom IEEE address so that it matches the data in the certificate. This ensures that there is parity between the data in the certificate and the IEEE address used by the local device for its identity on the network.

Verifying the stored certificate

After writing the certificate, it is best to verify the information by executing `em2xx_read` again. Note the **bold** values.

```

$ ./em2xx-read -id 192.168.221.51 -mfg -print -Run
em2xx-read Version 3.2 Build 1
Emsif Library Version 1.0.6
Opening InSight Adapter with hostname 192.168.221.51...
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed

'General' token group
TOKEN_MFG_CUSTOM_VERSION [ 2 bytes] : FFFF
TOKEN_MFG_CUSTOM_EUI_64  [ 8 bytes] : 0100000000000000
TOKEN_MFG_STRING         [16 bytes] : " " (0 of 16 characters)
                          FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME     [16 bytes] : " " (0 of 16 characters)
                          FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID       [ 2 bytes] : FFFF
TOKEN_MFG_PHY_CONFIG     [ 2 bytes] : FFFF
TOKEN_MFG_BOOTLOAD_AES_KEY [16 bytes] : FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE   [ 8 bytes] : FFFFFFFFFFFFFFFFFF
TOKEN_MFG_ASH_CONFIG     [ 2 bytes] : FFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [ 2 bytes] : FFFF

'Smart Energy CBKE (TOK_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 bytes] : 03045FDFC8D85FFB 8B3993CB72DDCAA5
                               5F00B3E87D6D0000 0000000000015445
                               5354534543410109 0006000000000000
CA Public Key [22 bytes] : 0200FDE8A7F3D108 4224962A4E7C54E6
                               9AC3F04DA6B8
Device Private Key [21 bytes] : 0B8A900FCADEBAB BFA383B540FCE9ED
                               438395EAA7

```

```

OCBKE Flags          [ 1 byte ] : 00

'Smart Energy Install Code (TOK_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 bytes] : FFFF
Install Code       [16 bytes] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC                [ 2 bytes] : FFFF

```

In this case, the device has been set with the data in test certificate #1 (associated with IEEE address (>)0000000000000001), which uses the Test CA.

Erasing the Certificate

If the wrong device is programmed with a certificate, or it is necessary to remove this security data from the device, complete the following procedures.

Warning: Incorrectly executing these steps can damage the device and prevent it from functioning.

Running em2xx_patch

In order to wipe the existing certificate data it is necessary to specify an additional option of `-Override` to `em2xx_patch`. The following is the command-line:

```
em2xx_patch -id <host | ip-address> -Mfg -loadTokens <erase-file> -Run -Override
```

Warning: The `-Override` option carries a slight risk of damaging your manufacturing data area if the loading/patching process is disrupted, which could render your device unusable. You should therefore only use this option sparingly and should take care not to disrupt the loading/patching process when this option is used.

This produces output similar to the following:

```

em2xx_patch.exe -id 192.168.221.51 -Mfg -loadTokens erase-cert.txt -Run -
Override
em2xx-patch Version 3.2 Build 1
Emsif Library Version 1.0.6

Writing to address 0x00005128 for token 'Device Implicit Cert'
Writing to address 0x00005158 for token 'CA Public Key'
Writing to address 0x0000516E for token 'Device Private Key'
Writing to address 0x00005183 for token 'CBKE Flags'
Opening InSight Adapter with hostname 192.168.221.51...
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= Reset performed

NOTE: Writing Custom EUI64 'FFFFFFFFFFFFFFFF' to match certificate.
Address 0x00005082.

*****
! WARNING ! WARNING ! WARNING ! WARNING ! WARNING ! WARNING ! WARNING

```

The '-Override' flag MUST be used with caution.

This flag enables re-programming of addresses in the manufacturing token area. This option should be used with caution and only in development scenarios. There is slight risk of a chip becoming corrupted and inoperable if a serious error (for example: power failure, crash, loss of connection, chip reset) occurs during the re-programming of the manufacturing tokens.

Addresses in the manufacturing token area can be written once without using the Override flag.

Are you sure you want to proceed? (yes/no): yes

! WARNING ! WARNING ! WARNING ! WARNING ! WARNING ! WARNING ! WARNING

```
Programming FLASH...
XAP2= Run performed
MFGcu 0041-0044 (0004)
MFGcu 0094-00c1 (002e)
Total Manuf Used=0032 words
XAP2= Stop performed
Verifying FLASH...
```

```
****Application load: SUCCESS****
****Verification:      SUCCESS****
```

```
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
```

Working with Certificates on the EM35x

Overriding the default EU164

By default, an Ember EU164 address is pre-programmed into all EM3xx chips. Customers may override this by writing their own EU164 into the Custom EU164 area of the manufacturing flash.

When programming certificates and installation codes the `em3xx_load.exe` tool allows you to override the Ember EU164 (see Ember document 120-4032-000: *EM3xx Utilities Guide* for more information). When a custom EU164 is already present, the certificate or installation code *must* use an EU164 that matches the custom one.

If you do not want to override the EU164, then the certificate and installation codes must use Ember EU164s, and those EU164s *must* match what is already programmed on the chip. You cannot override the EU164 of the device with another Ember EU164.

The reason these requirements are in place is to prevent certificates or installation codes from being programmed on the wrong device. Both of these items are bound to the EU164 of the device and therefore a mismatch will cause problems when deploying an end product.

To program a device with a certificate or installation code, the device must be connected via the debug cable to an Ember InSight Adapter. The tools provided with the ISA3 Utilities installer provide the basic elements needed to do this. The tool used is `em3xx_load.exe`.

Verifying the software version

To verify that you have the correct version of the tools, open a command prompt, and change to the directory where the tools are installed. The default for the Ember installation is C:\Program Files\Ember\ISA3 Utilities\bin\.

To verify the software version, run the tool without any arguments using this command:

```
$ em3xx_load.exe
```

You should see the following output:

```
$ em3xx_load.exe
em3xx_load (Version 1.0b13.1256666220)
```

Programming the Certificates into a Device

Checking the node information

Prior to modifying the certificate it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. To do this, execute the following command:

```
$ em3xx_load.exe --cibtokensprint
```

You should see the following output:

```
$ em3xx_load.exe --cibtokensprint
em3xx_load (Version 1.0b13.1256666220)

Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357

'General' token group
TOKEN_MFG_CIB_OBS          [16 byte array ] : A55AFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION  [16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64   [ 8 byte array ] : FFFFFFFFFFFFFFFFFF
TOKEN_MFG_STRING          [16 byte string] : " (0 of 16 chars)
                        FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME      [16 byte string] : " (0 of 16 chars)
                        FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID        [16-bit integer] : 0xFFFF
TOKEN_MFG_PHY_CONFIG      [16-bit integer] : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY [16 byte array ] : FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE    [ 8 byte array ] : FFFFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [16-bit integer] : 0xFFFF

'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 byte array] : FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
                        FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
```

```

CA Public Key      [22 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                   FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
Device Private Key [21 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                   FFFFFFFFFFFFFFFF
CBKE Flags        [ 1 byte array ] : FF

'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 byte array ] : FFFF
Install Code      [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC               [16-bit integer] : 0xFFFF

DONE

```

Note: The pre-programmed Ember EU164 is not shown in this output as it is part of the Ember-programmed manufacturing data rather than the customer-programmed manufacturing data. To obtain the Ember EU164 of an EM3xx device, use the following command (shown for a device whose InSight Adapter is attached to the PC via USB at Device ID 0):

```
em3xx_load --read @080407A2-080407A9
```

This reads out the Ember EU164 in least-significant byte (LSB) notation. For example, the command above produces the following output (indicating an Ember EU164 value of 0x000D6F000092E047):

```

em3xx_load (Version 1.0b13.125666220)

Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Getting memory from 0x080407A2 to 0x080407A9

{address: 0 1 2 3 4 5 6 7 8 9 A B C D E F}
080407A0: . . 47 E0 92 00 00 6F 0D 00 . . . . .

Run (by toggling nRESET)
DONE

```

If the custom EU164 (IEEE address) is set to all F's, then the Ember EU164 address is used. If this value is not all F's, then it used instead of the Ember EU164.

If the data within the Smart Energy token group (certificate, private key, root CA public key, metadata) is set to all F's, then the data is not valid and will not be used by the software.

Format of the certificate file

To program a certificate into the tokens, it is necessary to create a simple text file with the security information in it. The following is the format of that file expected by the em3xx_load.exe tool.

```
CA Public Key: <hex-string>
Device Implicit Cert: <hex-string>
Device Private Key: <hex-string>
```

The following is a sample certificate file issued by the Test CA, for a device with an IEEE of 0000000000000001 (big endian).

```
CA Public Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert:
03045fd8c8d85ffb8b3993cb72ddcaa55f00b3e87d6d0000000000000015445535453454
341010900060000000000000
Device Private Key: 00b8a900fcadebabbfa383b540fce9ed438395eaa7
```

Note: The code line starting with 'Device Implicit Cert:' and its value appear in the CA Public Key on a single line, not three lines as represented above.

Writing the certificate into the manufacturing area

To write the certificate into the manufacturing area, execute the following command:

```
$ em3xx_load.exe --cibtokenspatch sample_cert.txt
```

You should see the following output:

```
$ em3xx_load.exe --cibtokenspatch sample_cert.txt
em3xx_load (Version 1.0b13.125666220)

Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip

Writing to address 0x0804087E for token 'Device Implicit Cert'
Writing to address 0x080408AE for token 'CA Public Key'
Writing to address 0x080408C4 for token 'Device Private Key'
Writing to address 0x080408D9 for token 'CBKE Flags'

NOTE: Writing Custom EUI64 '0100000000000000' to match certificate.
Address 0x08040812.

Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

In addition to writing the certificate, this will update the custom IEEE address so that it matches the data in the certificate. This ensures that there is parity between the data in the certificate and the IEEE address used by the local device for its identity on the network.

Verifying the stored certificate

After writing the certificate, it is best to verify the information by executing `em3xx_load` again.

```
$ ./em3xx_load.exe --cibtokensprint
```

You should see the following output:

```
$ ./em3xx_load.exe --cibtokensprint
em3xx_load (Version 1.0b13.125666220)

Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357

'General' token group
TOKEN_MFG_CIB_OBS          [16 byte array ] : A55AFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION  [16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64   [ 8 byte array ] : 0100000000000000
TOKEN_MFG_STRING          [16 byte string] : " (0 of 16 chars)
                        FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME      [16 byte string] : " (0 of 16 chars)
                        FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID        [16-bit integer] : 0xFFFF
TOKEN_MFG_PHY_CONFIG      [16-bit integer] : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY [16 byte array ] : FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE    [ 8 byte array ] : FFFFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [16-bit integer] : 0xFFFF

'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 byte array ] : 03045FDFC8D85FFB 8B3993CB72DDCAA5
                        5F00B3E87D6D0000 0000000000015445
                        5354534543410109 0006000000000000
CA Public Key         [22 byte array ] : 0200FDE8A7F3D108 4224962A4E7C54E6
                        9AC3F04DA6B8
Device Private Key    [21 byte array ] : 00B8A900FCADEBAB BFA383B540FCE9ED
                        438395EAA7
CBKE Flags            [ 1 byte array ] : 00

'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 byte array ] : FFFF
Install Code       [16 byte array ] : FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
CRC                [16-bit integer] : 0xFFFF

DONE
```

Erasing the Certificate

If the wrong device is programmed with a certificate, or it is necessary to remove this security data from the device, complete the following procedures.

To erase token data create an erase file listing the token name using **!ERASE!** as the token data. For example:

```
CA Public Key: !ERASE!
Device Implicit Cert: !ERASE!
Device Private Key: !ERASE!
```

To erase a sample certificate, execute this command:

```
$ em3xx_load.exe --cibtokenspatch sample_erase_cert.txt
```

You should see the following output:

```
$ em3xx_load.exe --cibtokenspatch sample_erase_cert.txt
em3xx_load (Version 1.0b13.1256666220)

Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM3577
Reset Chip

Writing to address 0x0804087E for token 'Device Implicit Cert'
Writing to address 0x080408AE for token 'CA Public Key'
Writing to address 0x080408C4 for token 'Device Private Key'
Writing to address 0x080408D9 for token 'CBKE Flags'

NOTE: Writing Custom EUI64 'FFFFFFFFFFFFFFFF' to match certificate.
Address 0x08040812.

Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

Smart Energy Installation Code Overview

Smart Energy Installation codes are provided as a means for a device to join a ZigBee network in a reasonably secure fashion. The installation code itself is a random value printed on the joining device, which is used to encrypt the initial message exchange between it and the Energy Services Interface (ESI).

The installation code can be thought of as similar to the PIN code on Bluetooth devices when two devices are paired. The PIN code is provided as an authorization code for the parent device so that the joining device knows it is receiving information securely, such as when a handsfree headset is paired to a cellular phone.

The installation code for a Smart Energy device is printed on the outside of the device, and provided via an out-of-band mechanism to the utility along with the IEEE of the device. The utility then securely transports that information to the ESI.

The ESI and the joining device use the installation code as a shared key to establish an initial bond of trust allowing the new device to join the Smart Energy network.

Security Use

An installation code is used to create a preconfigured, link key. The installation code is transformed into a link key by use on an AES hash. For more information and sample code, consult the ZigBee Smart Energy Profile Specification (ZigBee Document Number 07-5356). Public access for ZigBee Technical Documents can be found at: <http://www.zigbee.org/Specifications/ZigBee/download.aspx>.

The derived ZigBee link will be known only by the ESI and the joining device. The ESI uses that key to securely transport the ZigBee network key to the device. Once the device has the network key, it can communicate at the network layer to the Smart Energy network. It has the ability to perform service discovery and begin the application's initialization process.

The installation code, while not exactly a secret, cannot be easily guessed by a malicious device that hears the initial exchange between the joining device and ESI. Without knowledge of the installation code and thus the key, it cannot decrypt the messages.

The initial link key derived from the installation code does not have full access privileges on a Smart Energy network. Attempts to use it for Smart Energy messaging are not allowed and will be ignored by other Smart Energy devices. Shortly after joining a network, a device must use the Key Establishment cluster to establish a new link key with the ESI. Only when key establishment completes successfully will a device have full privileges on the network and be able send and receive certain Smart Energy messages.

Installation Code Format

The installation code is made up of a 6-, 8-, 12-, or 16-byte random, hexadecimal number with a 2-byte CRC appended to the end. As far as the user is concerned, the CRC is part of the installation code and they do not need to know that it is there or why. Therefore, from the user's point of view, the length of the install code is 8, 10, 14, or 18 bytes.

The choice of 6-, 8-, 12-, or 16-byte length for the installation code is up to the device vendor. Manufacturing and managing the list of installation codes will play a part in choosing the size, security, and user experience in installing the device. A larger installation code size will mean less of a chance of an attacker "guessing" the installation code and eavesdropping on the initial join. However smaller installation code is much easier for a user to read off the device during installation.

Installation Code CRC

The installation code CRC is mechanism used to verify the integrity of an installation code when it is transmitted via an out-of-band mechanism to the utility. This transport mechanism involves human interaction in some way. As a result, the CRC was designed as a way to verify that an installation code is valid and was not mistakenly changed during transport.

The Smart Energy installation model enables users to install a device themselves. Users simply read the installation code on the back of the device and enter it into a webpage or provide it over the phone to a utility service. Because the number is a hexadecimal value, it is easy to transpose digits or read the wrong value.

Validation

The ZigBee Smart Energy Profile Specification expects that the utility will perform basic checking of the installation code for validity. The utility calculates the CRC over all bytes in the installation code except the final two. It then compares the final two bytes of the installation code with the calculated CRC to see if they match. If they do not match, the user entering the installation code can be informed immediately that it does not look valid. The user should then double check the value.

The ZigBee Smart Energy Profile Specification does not require the ESI to validate the installation code. The ESI expects to receive a pre-configured link key along with the IEEE address of the new joining device. It does not need to have any knowledge about how that key was derived. It is up to the particular utility how they wish to manage and transport the link key to the ESI.

For details on how the CRC is calculated, including sample code, consult the ZigBee Smart Energy Profile Specification (ZigBee Document Number 07-5356). Public document access can be found here: <http://www.zigbee.org/Specifications/ZigBee/download.aspx>.

Generation

Ember recommends that the installation code be a random number. This reduces the chances of an attacker guessing the installation code and compromising the initial join procedure. The installation code should **not** be based on the manufacturing process, such as tied to the IEEE address or sequential numbering based on the manufacturing lot.

If that were the case, an attacker with knowledge about the type of device being joined would have a known range of installation codes it could try to compromise the network and clone the device's identity. An installation code does not have to be unique across all Smart Energy devices for all manufacturers.

Labels

The device's installation code should be printed on a label on the outside of the device along with its IEEE address. Both elements should be identified with text indicating what they are. The installation code should **not** be printed on the outside of the box because that makes it easier for an attacker to gain knowledge of the installation code and potentially compromise the device. It is recommended that the installation code be printed in 2-byte blocks (for example, 83FE D340 7A93 9738 C552).

Note: The CRC should be appended to the installation code in **little endian** format on the label.

Format of the installation code file

To program the installation code, create a simple text file with the value of the installation code (without the CRC). This file is passed into the `em2xx_patch.exe` tool.

The format of the file is as follows:

```
Install Code: <ascii-hex>
```

Here is a sample installation code file. The CRC for that code is 0x52C5.

```
Install Code: 83FED3407A939738
```

Per the Smart Energy specification, the installation code may be 6, 8, 12, or 16 bytes in length (not including the two-byte CRC).

Note: Versions of `em2xx-patch` prior to 3.1 build 4 required the CRC value to be specified in the installation code file. This is no longer the case because the utility can derive the correct CRC for the installation code. It is now recommended that users *do not* specify the CRC in the file and let the utility do it for them.

Writing the installation code into the manufacturing area

To write the installation code into the manufacturing area, execute the following command:

```
em2xx_patch -Mfg -id <host|ip-address> -loadInst <install-code-file> -Run
```

For example, you should see something similar to the following:

```
C:\> em2xx_patch.exe -id 192.168.221.51 -Mfg -loadInst install.txt -Run
em2xx_patch.exe Version 3.1 Build 4
Emsif Library Version 1.0.6
Install code length: 8 bytes
Calculated CRC: 0x52c5
Non-ember EUI64 found in file
Install Code: 83FE D340 7A93 9738 FFFF FFFF FFFF FFFF
CRC: 52C5
EUI: 0000 0000 0000 0001
Writing Custom EUI64 Address: 0000000000000001
Opening InSight Adapter with hostname 192.168.221.51...
  XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= Reset performed
  XAP2= Issuing SW_RESET
  XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= Reset performed
EUI was changed, checking for the current value...
Current custom EUI on the chip: FFFFFFFFFFFFFFFF
New custom EUI to be written: 0000000000000001
Programming FLASH...
  XAP2= Run performed
  MFGcu 0041-0044 (0004)
  MFGcu 00c2-00cb (000a)
  Total Manuf Used=000e words
  XAP2= Stop performed
Verifying FLASH...
```


Warning: The “-Override” option carries a slight risk of damaging your manufacturing data area if the loading/patching process is disrupted, which could render your device unusable. You should therefore only use this option sparingly and should take care not to disrupt the loading/patching process when this option is used.

This produces output similar to the following:

```
C:\>em2xx_patch.exe -id 172.16.0.6 -Mfg -loadInst null -Run -Override
em2xx_patch.exe Version 3.0 Build 1
Emsif Library Version 1.0.6
Writing Custom EUI64 Address: FFFFFFFFFFFFFFFF
SIF Slave id change from 1 to 0 (pod 1) succeeded
  XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= DbgEna performed
  XAP2= Reset performed
Programming FLASH...
  XAP2= Run performed
  MFGcu 0041-0044 (0004)
  MFGcu 00c2-00cb (000a)
  Total Manuf Used=000e words
  XAP2= Stop performed
Verifying FLASH...

****Application load: SUCCESS****
****Verification:      SUCCESS****

  XAP2= Stop performed
  XAP2= Reset performed
  XAP2= Issuing SW_RESET
```

Programming the Installation Code on an EM35x

Checking the installation code on a device

Prior to modifying the certificate, it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. To do this, execute the following command:

```
$ ./em3xx_load.exe --cibtokensprint
```

You should see the following output:

```
$ ./em3xx_load.exe --cibtokensprint

em3xx_load version 2.0b14.1286916699
Connecting to ISA via IP address rob-350
DLL version 1.1.13, compiled Sep 24 2010 17:01:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357

'General' token group
```

```

TOKEN_MFG_CIB_OBS          [16 byte array ] : A55AFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION  [16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64   [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING          [16 byte string] : " " (0 of 16 chars)
                          FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME      [16 byte string] : " " (0 of 16 chars)
                          FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID        [16-bit integer] : 0xFFFF
TOKEN_MFG_PHY_CONFIG      [16-bit integer] : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY [16 byte array ] : FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE    [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [16-bit integer] : 0xFFFF

'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                          FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                          FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CA Public Key          [22 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                          FFFFFFFFFFFFFFFF
Device Private Key     [21 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                          FFFFFFFFFFFFFFFF
CBKE Flags             [ 1 byte array ] : FF

'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 byte array ] : FFFF
Install Code        [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC                 [16-bit integer] : 0xFFFF

DONE

```

The data stored at `TOKEN_MFG_CUSTOM_EUI_64` is the custom value for the IEEE address in little-endian format. If this value is set to all F's, then the value for the internal IEEE address is used. If this value is not all F's, then it used instead of the internal IEEE address.

The data in the last block is the installation code metadata, the installation code, and the CRC. The installation code metadata (Flags and CRC) is automatically programmed by the `em3xx_load` tool.

Format of the installation code file

To program the installation code, create a simple text file with the value of the installation code (without the CRC). This file is passed into the `em3xx_load.exe` tool.

The format of the file is as follows:

```
Install Code: <ascii-hex>
```

Here is a sample installation code file. The CRC for that code is 0x52C5.

```
Install Code: 83FED3407A939738
```

Per the Smart Energy specification, the installation code may be 6, 8, 12, or 16 bytes in length (not including the two-byte CRC).

Writing the installation code into the manufacturing area

To write the installation code into the manufacturing area, execute the following command:

```
$ em3xx_load.exe --cibtokenspatch install-code-file.txt
```

You should see the following output:

```
$ em3xx_load.exe --cibtokenspatch install-code-file.txt
em3xx_load version 2.0b14.1286916699
Connecting to ISA via IP address rob-350
DLL version 1.1.13, compiled Sep 24 2010 17:01:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip

NOTE: Calculated install code CRC is 0x52C5.
Writing to address 0x080408DA for token 'Install Code Flags'
Writing to address 0x080408DC for token 'Install Code'
Writing to address 0x080408EC for token 'CRC'

Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

Verifying the stored installation code

After writing the installation code, it is best to verify the information by executing `em3xx_load` again. Note the bold values.

```
$ ./em3xx_load.exe --cibtokensprint
```

You should see the following output:

```
$ ./em3xx_load.exe --cibtokensprint
em3xx_load version 2.0b14.1286916699
Connecting to ISA via IP address rob-350
DLL version 1.1.13, compiled Sep 24 2010 17:01:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357

'General' token group
TOKEN_MFG_CIB_OBS          [16 byte array ] : A55AFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION  [16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64   [ 8 byte array ] : 01000000006F0D00
TOKEN_MFG_STRING          [16 byte string] : " (0 of 16 chars)
                        FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME      [16 byte string] : " (0 of 16 chars)
                        FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID        [16-bit integer] : 0xFFFF
TOKEN_MFG_PHY_CONFIG      [16-bit integer] : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY [16 byte array ] : FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE    [ 8 byte array ] : FFFFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [16-bit integer] : 0xFFFF

'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 byte array ] : FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
                        FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
                        FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
CA Public Key         [22 byte array ] : FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
                        FFFFFFFFFF
Device Private Key    [21 byte array ] : FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF
                        FFFFFFFFFF
CBKE Flags            [ 1 byte array ] : FF

'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 byte array ] : 0200
Install Code       [16 byte array ] : 83FED3407A939738 FFFFFFFFFFFFFFFFFF
CRC                [16-bit integer] : 0x52C5

DONE
```

When the installation code is less than 16 bytes (not including the CRC), the rest of the words will be all F's. The final two bytes are the CRC *displayed* in **big endian** format. On the chip, it will be stored in the processor's native endianness (little-endian on the 35x).

Erasing the Installation Code

If the wrong device is programmed with a certificate, or you want to remove this security data from the device, complete the following steps.

Erase process

To erase token data create an erase file listing the token name using **!ERASE!** as the token data. For example:

```
Install Code: !ERASE!
TOKEN_MFG_CUSTOM_EUI_64: !ERASE!
```

This is the command-line:

```
$ em3xx_load.exe --cibtokenspatch install-code-file-erase.txt
```

You should see the following output:

```
$ em3xx_load.exe --cibtokenspatch install-code-file-erase.txt

em3xx_load version 2.0b14.1286916699
Connecting to ISA via IP address rob-350
DLL version 1.1.13, compiled Sep 24 2010 17:01:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip

Writing to address 0x080408DA for token 'Install Code Flags'
Writing to address 0x080408DC for token 'Install Code'
Writing to address 0x080408EC for token 'CRC'

NOTE: Writing Custom EUI64 'FFFFFFFFFFFFFFFF'. Address 0x08040812.

Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

After Reading this Document

If you have questions or require assistance with the procedures described in this document, contact Ember Customer Support. The Ember Customer Support portal provides a wide array of hardware and software documentation such as FAQ's, reference designs, user guides, application notes, and the latest software available to download. To obtain support on all Ember products and to gain access to the Ember Customer Support portal, visit http://www.ember.com/support_index.html.

Copyright © 2008-2012 Ember Corporation

All rights reserved. Neither this publication nor any part thereof can be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Ember Corporation. This documentation is furnished under license and can be used or copied only in accordance with the terms of such license.

The content of this documentation is furnished for informational use only, is subject to change without notice, and does not represent a commitment or guaranty by Ember Corporation. The statements, configurations, technical data, and recommendations in this document are believed to be accurate and reliable as of the time of publication, but Ember Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation. DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT, ARE DISCLAIMED. Users are responsible for their applications and for the use of any products specified in this document.

Title, ownership, and all rights in copyrights, patents, trademarks, and other intellectual property rights embodied in Ember Corporation's Products and any copy, portion, or modification thereof, shall not transfer to Purchaser or its customers and shall remain in Ember Corporation and its licensors.

No source code rights are granted to Purchaser or its customers with respect to any Ember software. Purchaser agrees not to copy, modify, alter, translate, decompile, disassemble, or reverse engineer Ember hardware (including without limitation any embedded software) or attempt to disable any security devices or codes incorporated in Ember hardware. Purchaser shall not alter, remove, or obscure any printed or displayed legal notices contained on or in Ember hardware.

Ember is a trademark of Ember Corporation. All other trademarks are the property of their respective holders.

