KNX Cookbook

Features of KNX and ETS

Load Controls

2

3

1

Summary

This document is a development help for KNX newcomers.

This part gives a closer description of some features of KNX and ETS. This Chapter describes the Load State Machines, Management Procedures and Load Controls.

This document is not part of the KNX Standard approval procedure.

This document is part of the KNX Specifications v2.1.

Document updates

Version	Date	Modifications
1.0.0	2011.05.13	Preparation of the final version.
01.00.01	2013.10.14	Editorial updates for the publication of KNX Specifications 2.1.
01.00.02	2013.12.10	Final editorial corrections.

References

[01] Chapter 3/5/2 “Management Procedures”
[02] Chapter 3/5/3 “Configuration Procedures”
[03] Volume 6 “Profiles”

Filename:	02_03_01 Load Controls v01.00.02.docx
Version:	01.00.02
State:	Final version
Savedate:	2013.12.10
Number of pages:	12

Contents

1 Load Controls **4**
1.1 Management Procedures 4
1.2 KNX Devices 4
1.3 Load State Machine 4
1.4 ADM1 5
2 Overview **8**
2.1 Load Controls for Management Procedures related to State Machines 8
2.2 Load Controls for common use 8
2.3 Load Controls for ETS internal use only 9
3 Example **10**

1 Load Controls

1.1 Management Procedures

Application Programs for KNX devices are downloaded by ETS. However, as an application developer, you do not have to tell ETS all details of how this shall be done. KNX implementations shall comply with any of the standard KNX Profiles ([03]) and for any KNX Profile, ETS knows a fixed, standard download Procedure. This is called a "Configuration Procedure". Chapter 3/5/3 "Configuration Procedures" ([02]) specifies all Configuration Procedures for all KNX Profiles.

The download of a KNX device, thus the Configuration Procedure, makes use of the Application Layer services for management that are required to be supported by the Profile.

EXAMPLE 1 A_Memory_Write, A_DeviceDescriptor_Read, A_PropertyValue_Write…

However, specifying Configuration Procedures in terms of Application Layer services, would lead to very long, hard to read sequences. Moreover, these Application Layer services for management need to be used in sometimes very specific ways.

EXAMPLE 2 Before reading a Device Descriptor (A_Memory_Read), it is for many Profiles required that a Transport Layer connection be established to the device.

EXAMPLE 3 Before writing any Property Value (A_PropertyValue_Write), it is needed that ETS knows the Object Index and sometimes the Property Description of the Property, to be sure to write the correctly formatted data at the correct Property.

Therefore, an intermediate definition level is specified in between the Application Layer services and the Configuration Procedures: the Management Procedures. These are typically short sequences of Application Layer services, that accomplish a well-defined task and take care of all the detailed service parameters and conditions that may occur.

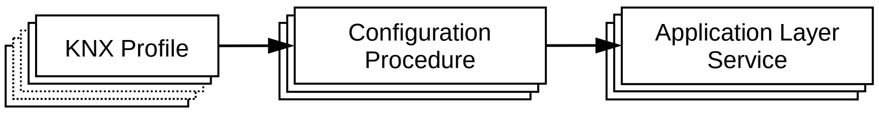


Figure 1 – Three levels of specifying how a KNX device will be downloaded

1.2 KNX Devices

These procedures all serve for managing and downloading data in the KNX device. This data can be various. The following types of data can be distinguished in a KNX device.

- Group Address Table
- Group Object Association Table
- AP
- …

1.3 Load State Machine

The access to the Resources should be controlled, to avoid that incomplete or faulty data is accessed and to make sure that the device knows which Resources resided where in memory. That is why many of the standard KNX Resources are controlled by a Load State Machine with several Load States.

EXAMPLE 4 Load States can be *Loaded*, *Loading*, *Unloaded*…

The Load State Machine goes from one Load State to another via events. Most of these events are requested by ETS, via Telegrams on the bus.

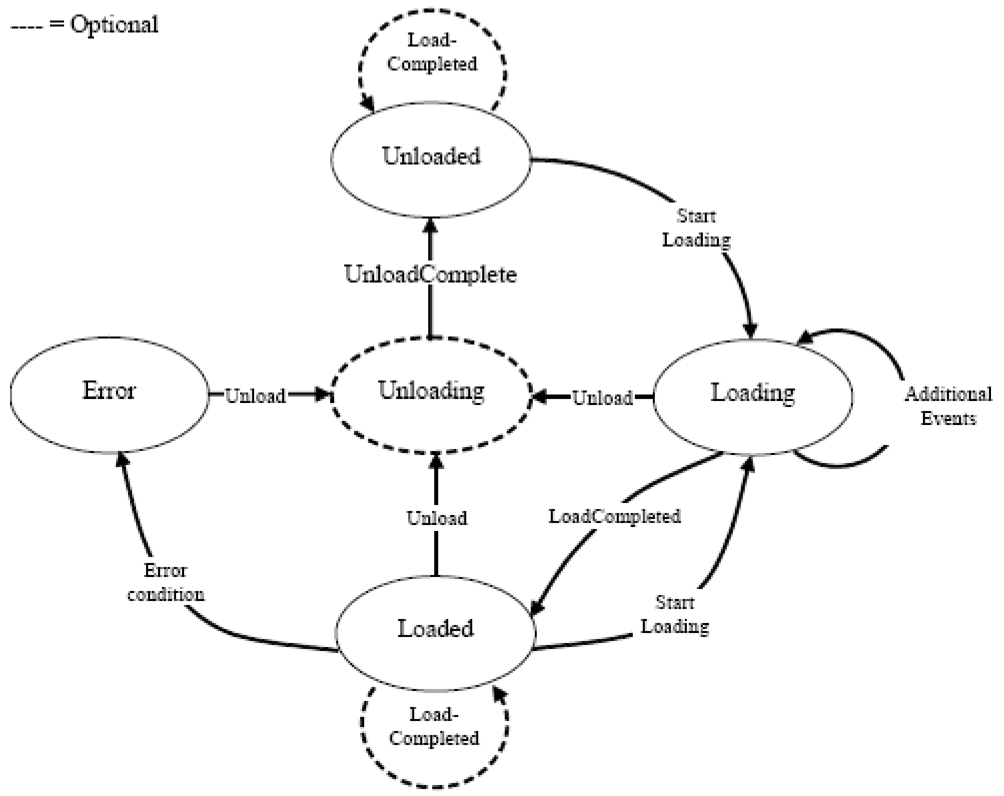


Figure 2 – Example of a Load State Machine

This is in summary what happens when a device Resource is programmed with ETS.

- Assume that a particular device Resource already has been configured. Its Load State will thus be "Loaded".
- In order to (re)program it, ETS will send a Telegram to request to change its Load State into "Unloading". This Telegram is an "Unload" event.
- The device will internally mark that the data is no longer valid. It will also make sure that the data of the Resource are no longer used by any internal functions it may have. When this is done, a device internal event "UnloadComplete" will set the Resource state to "Unloaded".
- ETS waits until the state has become "Unloaded" before starting the actual download.
- With the event "Start Loading", ETS brings the device Resource in "Loading" state.
- ETS now has the opportunity to download the device Resource.
- After completing the load process, ETS sets the Load State of the Resource state to "Loaded" by sending another Telegram to the device: this will be the "LoadCompleted" event.

1.4 ADM1

The KNX Specifications only define the Management Procedures on the bus, i.e. the entire communication flow between the Management Server - this is the target device - and the Management Client (ETS). This is of course only one side of the story: it is possible to instruct ETS when to use which Management Procedure, with what data. For each Management Procedure, the application developer writes a simple formatted instruction that can be understood by ETS. This is called a "Load Control" (LC) and it is expressed in a language called ADM1.

NOTE 1 ADM1 is an historical name and stands for 'Active Device Management, Version 1'. ADM1 is specified in the eteC SDK help file.

NOTE 2 Load Controls are elementary instructions for ETS how to download a device. Many of these instructions have to do with the manipulation of the Load State Machines, so many Load Controls relate one-to-one with Load State Machines. Yet, there are as many Load Controls that have to do with other Management Procedures than these manipulating the Load State Machines.

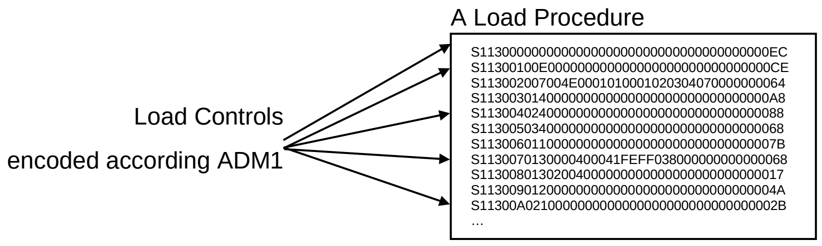
EXAMPLE 5 To Restart the device, to read- or write memory contents.

Load State Machines (something on the bus) and Load Controls (an instruction to ETS) are thus related, but should not be confused.

There are also Load Controls that are not related to Management Procedures, but that are purely handled internally by ETS, without effect on the bus.

EXAMPLE 6 To pause the execution of the Load Procedure for a while to give the device the time to handle the last events.

The entire download of a device can be directed through a small or large sequence of Load Controls. This is called a Load Procedure.



```
S1130000000000000000000000000000000000EC
S11300100E000000000000000000000000000000CE
S113002007004E00010100010203040700000000064
S113003014000000000000000000000000000000A8
S11300402400000000000000000000000000000088
S11300503400000000000000000000000000000068
S113006011000000000000000000000000000000007B
S1130070130000400041FEFF038000000000000068
S113008013020040000000000000000000000000017
S113009012000000000000000000000000000000004A
S11300A021000000000000000000000000000000002B
...
```

Figure 3 – Load Controls together build a Load procedure

Each Load Procedure starts with an ADM1_Header record. This header is stored at device memory address $0000 and contains the offset where all further LCs will be stored.

Each Load Procedure ends with an ADM1_End record.

The actual LCs of a Load Procedure are put between an ADM1_Header - and an ADM1_End record.

The ETS component responsible for deploying the Load Procedures is called ‘ADM1 Interpreter’. This is depicted in Figure 4.

In order to download a device, ETS also needs to know what data shall be written in the device’s memory. This collection of data is called a “Download Image”. The Load Controls can refer to the data in this download image. See the yellow note box in Figure 4.

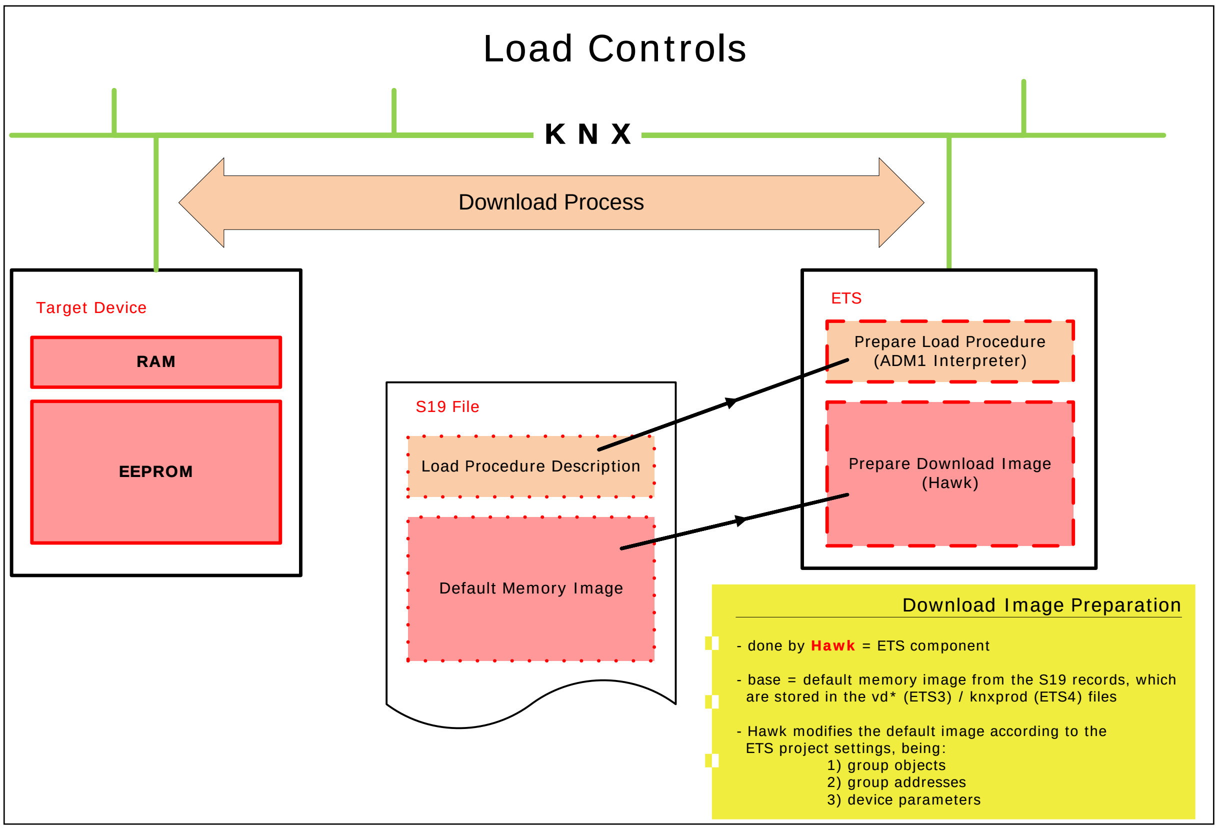
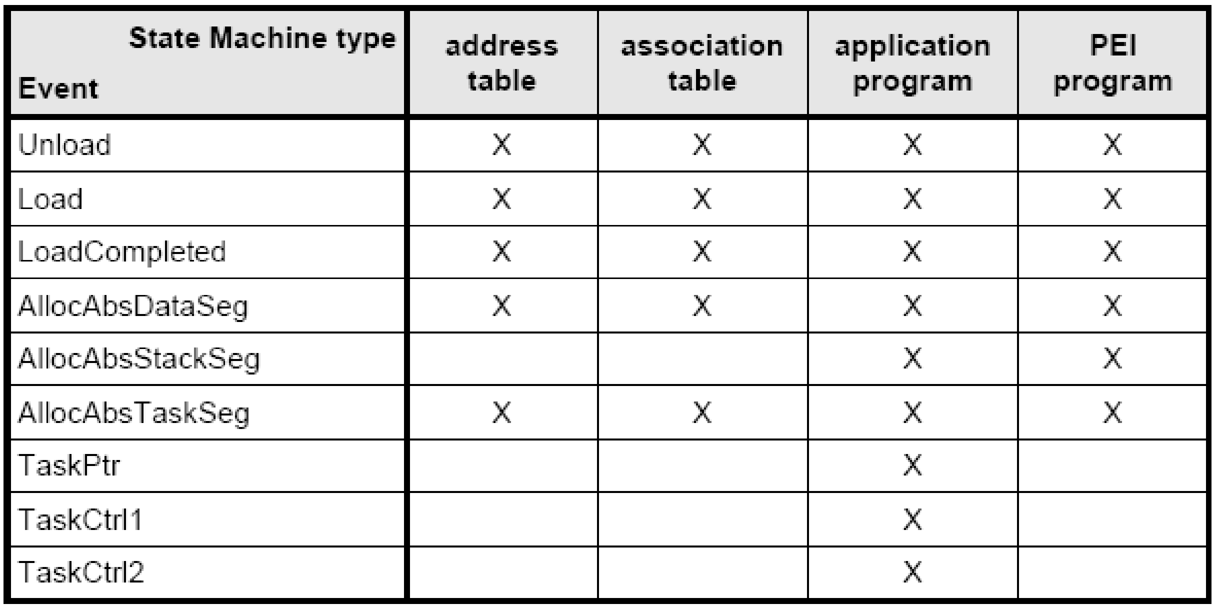

Load Controls
K N X
Download Process
Target Device
RAM
EEPROM
S19 File
Load Procedure Description
Default Memory Image
ETS
Prepare Load Procedure (ADM1 Interpreter)
Prepare Download Image (Hawk)
Download Image Preparation
- done by Hawk = ETS component
- base = default memory image from the S19 records, which are stored in the vd* (ETS3) / knxprod (ETS4) files
- Hawk modifies the default image according to the ETS project settings, being:
1) group objects
2) group addresses
3) device parameters

Figure 4 – Load Procedures interpreted by ETS

2 Overview

2.1 Load Controls for Management Procedures related to State Machines



State Machine type / Event	address table	association table	application program	PEI program
Unload	X	X	X	X
Load	X	X	X	X
LoadCompleted	X	X	X	X
AllocAbsDataSeg	X	X	X	X
AllocAbsStackSeg			X	X
AllocAbsTaskSeg	X	X	X	X
TaskPtr			X	
TaskCtrl1			X	
TaskCtrl2			X	

Figure 5 – Overview of the Load Controls for the Management Procedures for handling State Machines

- Unload: invokes the transition event ‘Unload’, LC = **ADM1_Unload**
- Load: invokes the transition event ‘Start Loading’, LC = **ADM1_Load**
- LoadCompleted: invokes the transition event ‘LoadCompleted’, LC = **AMD1_LoadCompleted**
- AllocAbsDataSeg (segment type = $00): allocates a data/code segment, LC = **AMD1_AbsSegment**
- AllocAbsStackSeg (segment type = $01): allocates a stack segment, LC = **AMD1_AbsSegment**
- AllocAbsTaskSeg(segment type = $02): sets the Resource start address or the AP’s main function entry address, PEI type, AP ID = Manufacturer ID, Device Type, AP Version, LC = **ADM1_TaskSegment**
- TaskPtr (segment type = $03): sets the initialization function entry address, the save function entry address and the Serial PEI handler function entry address, LC = **ADM1_TaskPtr**
- TaskCtrl1 (segment type = $04): sets the user KNX object table start address and the number of user KNX objects, LC = **ADM1_TaskControl1**
- TaskCtrl2 (segment type = $05): sets the start address of the AP’s callback procedure, CO table start address, address of CO segment #0, address of CO segment #2, LC = **ADM1_TaskControl2**

2.2 Load Controls for common use

- **ADM1_Connect**: makes ETS build up a transport layer connection with the target device
- **ADM1_Disconnect**: indicates ETS to close the transport layer connection with the target device
- **ADM1_Delay**: instructs ETS to wait before starting the next LC record, the delay time is specified in milliseconds

- **ADM1_Restart**: instructs ETS to restart the target device, this also closes the transport layer connection (as 'side effect')
- **ADM1_Write**: writes to memory from a specified data block, without verification
- **ADM1_WriteVerify**: writes to memory from a specified data block, with verification
- **ADM1_WriteDirect**: writes up to 11 bytes specified within the LC record to memory, without verification
- **ADM1_CompareDirect**: compares up to 11 bytes specified within the LC record, with the same number of bytes from a specified data block
- **ADM1_PropWrite**: writes up to 10 bytes specified within the LC record to a property, without verification
- **ADM1_PropWriteVerify**: writes up to 10 bytes specified within the LC record to a property, with verification
- **ADM1_PropCompare**: compares up to 10 bytes specified within the LC record, with the same number of bytes from a specified property

2.3 Load Controls for ETS internal use only

These Load Controls are as well interpreted by ETS, but do not lead to Telegrams on the bus.

AbsCObjSeg

- This LC is to announce the CO table to a software tool (typically MT). This record contains the start address of the CO table (2 bytes) and the CO table type ID consisting out of the manufacturer ID (2 bytes), the device ID (2 bytes) and the software version number (1 byte).
- Format: 53 02 00 ss ss 00 dd dd dd dd dd 00 00
- ss ss = start address of the CO table
- dd dd dd dd dd = CO table ID
- This record is for MT-information only. ETS ignores it and will hence not be transmitted on the bus.

3 Example

```
S11300000000000000000000000000000000000000EC
S11300100E000000000000000000000000000000CE
S113002007004E000101000102030407000000 0064
S113003014000000000000000000000000000000A8
S11300402400000000000000000000000000000088
S11300503400000000000000000000000000000068
S113006011000000000000000000000000000000 7B
S1130070130000400041FEFF038000000000000068
S1130080130200400000000000000000000000017
S11300901200000000000000000000000000000004A
S11300A021000000000000000000000000000000 2B
S11300B023000041FF43FBFF038000000000000019
S11300C023020041FF00000000000000000000C7
S11300D022000000000000000000000000000000FA
S11300E031000000000000000000000000000000DB
S11300F03300000700098F000200000000000000 28
S113010033010009900990000200000000000000 83
S113011033000043FC5E9BFF0380000000000000EE
S113012033020043FC0100720001010000000000E2
S113013053020043FC0000010001000000000000 25
S113014032000000000000000000000000000000 79
S11301500C000000000000000000000000000000 8F
S11301600F000000000000000000000000000000 7C
S1130170FF000000000000000000000000000000 7C
```

The above is an excerpt from an s19 file. Each line of such a file is called an s19 record.

First record:

```
S 1 13 0000 00 00 0000 000000000000000000000000 EC
```

- S: indicates this is a Motorola record
- 1: means this is a data record
- $13: is the number of bytes to follow => $13 = 19 (decimal)
- $0000: is the EEPROM address where this record needs to be stored
- $EC: is the checksum (last byte)

- Checksum calculation: add all successive bytes, starting from $13 (number of bytes), then truncate this result to one byte (if necessary), subtract this byte from $FF
- Example, for the last record : $13 + $01 + $70 + $FF = $183, $FF - $83 = $7C

➔ We know from ADM1 that a ADM1_Header of a Load Procedure is stored at address $0000
➔ So this means this record is an ADM1_Header
➔ According to the specifications:
  ➔ code = $00 (following the address) indeed confirms this being the ADM1_Header
  ➔ $00 -> reserved
  ➔ offset = $0000 mean that all next records (LCs) are stored from address $0010 onwards

Second record:

```
S113 0010 0E 000000000000000000000000000000 CE
```

➔ address = $0010 (as expected)
➔ code = $0E means this LC record = ADM1_Connect

Third record:

```
S113 0020 07 00 4E 0001 01 000102030407000000 00 64
```

➔ address = $0020
➔ code = $07 means this is LC record = ADM1_PropCompare

- ➔ Object Index = $00, Property ID = $4E, Start Element = $0001, Element Count = $01, Data = `000102030407000000`

Next 3 records:

```
S113 0030 14 000000000000000000000000000000 A8
S113 0040 24 000000000000000000000000000000 88
S113 0050 34 000000000000000000000000000000 68
```

- ➔ code = $14 means this LC = ADM1_Unload, Resource = Address Table
- ➔ code = $24 means this LC = ADM1_Unload, Resource = Association Table
- ➔ code = $34 means this LC = ADM1_Unload, Resource = AP

Next 4 records:

```
S113 0060 11 000000000000000000000000000000 7B
S113 0070 13 00 00 4000 41FE FF 03 80 000000000000 68
S113 0080 13 02 00 4000 00 0000 0000 00 0000000000 17
S113 0090 12 000000000000000000000000000000 4A
```

- ➔ code = $11: ADM1_Load, Resource = Address Table
- ➔ code = $13 + Segment Type = $00: ADM1_AbsSegment, Address Table
  - ➔ $00 -> reserved
  - ➔ Segment Start Address = $4000
  - ➔ Segment End Address = $41FE => Length = $01FF
  - ➔ Segment Access Attributes = $FF => read + write access
  - ➔ Segment Memory Type = $03 => EEPROM
  - ➔ Segment Flags = $80 => checksum control enabled
- ➔ code = $13 + Segment Type = $02: ADM1_TaskSegment, Address Table
  - ➔ $00 -> reserved
  - ➔ Address Table Start Address = $4000
  - ➔ PEI Type = $00 (not applicable)
  - ➔ Manufacturer ID = $00 (not applicable)
  - ➔ Device Type = $0000 (not applicable)
  - ➔ AP Version = $00 (not applicable)
- ➔ code = $12: ADM1_LoadCompleted, Address Table

Next 4 records:

```
S113 00A0 21 000000000000000000000000000000 2B
S113 00B0 23 00 00 41FF 43FB FF 03 80 000000000000 19
S113 00C0 23 02 00 41FF 00 0000 0000 00 0000000000 C7
S113 00D0 22 000000000000000000000000000000 FA
```

- ➔ code = $21: ADM1_Load, Resource = Association Table
- ➔ code = $23 + Segment Type = $00: ADM1_AbsSegment, Association Table
  - ➔ $00 -> reserved
  - ➔ Segment Start Address = $41FF
  - ➔ Segment End Address = $43FB => Length = $01FD
  - ➔ Segment Access Attributes = $FF => read + write access
  - ➔ Segment Memory Type = $03 => EEPROM
  - ➔ Segment Flags = $80 => checksum control enabled
- ➔ code = $23 + Segment Type = $02: ADM1_TaskSegment, Association Table
  - ➔ $00 -> reserved
  - ➔ Association Table Start Address = $41FF
  - ➔ PEI Type = $00 (not applicable)
  - ➔ Manufacturer ID = $00 (not applicable)
  - ➔ Device Type = $0000 (not applicable)
  - ➔ AP Version = $00 (not applicable)
- ➔ code = $22: ADM1_LoadCompleted, Association Table

Next 7 records:

```
S113 00E0 31 000000000000000000000000000000 DB
S113 00F0 33 00 00 0700 098F 00 02 00 000000000000 28
S113 0100 33 01 00 0990 0990 00 02 00 000000000000 83
S113 0110 33 00 00 43FC 5E9B FF 03 80 000000000000 EE
S113 0120 33 02 00 43FC 01 0072 0001 01 0000000000 E2
S113 0130 53 02 00 43FC 00 0001 00 0100 0000000000 25
S113 0140 32 000000000000000000000000000000 79
```

- ➔ code = $31: ADM1_Load, Resource = AP
- ➔ code = $33 + Segment Type = $00: ADM1_AbsSegment, AP
  - ➔ $00 -> reserved
  - ➔ Segment Start Address = $0700
  - ➔ Segment End Address = $098F => Length = $0290
  - ➔ Segment Access Attributes = $00 => no access
  - ➔ Segment Memory Type = $02 => RAM
  - ➔ Segment Flags = $00 => checksum control disabled
- ➔ code = $33 + Segment Type = $01: ADM1_AbsSegment, AP
  - ➔ $00 -> reserved
  - ➔ Segment Start Address = $0990
  - ➔ Segment End Address = $0990 => Length = $01
  - ➔ Segment Access Attributes = $00 => no access
  - ➔ Segment Memory Type = $02 => RAM
  - ➔ Segment Flags = $00 => checksum control disabled
- ➔ code = $33 + Segment Type = $00: ADM1_AbsSegment, AP
  - ➔ $00 -> reserved
  - ➔ Segment Start Address = $43FC
  - ➔ Segment End Address = $5E9B => Length = $1AA0
  - ➔ Segment Access Attributes = $00 => no access
  - ➔ Segment Memory Type = $03 => EEPROM
  - ➔ Segment Flags = $00 => checksum control disabled
- ➔ code = $33 + Segment Type = $02: ADM1_TaskSegment, AP
  - ➔ $00 -> reserved
  - ➔ AP Main Function Entry Address = $43FC
  - ➔ PEI Type = $01
  - ➔ Manufacturer ID = $72
  - ➔ Device Type = $0001
  - ➔ AP Version = $01
- ➔ code = $53 + Segment Type = $02: AbsCObjSeg
  - ➔ $00 -> reserved
  - ➔ CO Table Start Address = $43FC
  - ➔ CO Table ID = $0001 $00 $0100
- ➔ code = $32: ADM1_LoadCompleted, AP

Last 3 records**:**

```
S113 0150 0C 00000000000000000000000000000008F
S113 0160 0F 00000000000000000000000000000007C
S113 0170 FF 00000000000000000000000000000007C
```

- ➔ code = $0C: ADM1_Restart
- ➔ code = $0F: ADM1_Disconnect
- ➔ code= $FF: ADM_End record