|  |
| --- |
| 3D Printer Module Message Document |
|  |
| 3D Printer  Module Message Specification |

|  |  |
| --- | --- |
| Document Number: |  |
| Version: | 1.0 |
| Publication Date: | 06/28/2025 |

**Table Of Contents**

[1 Introduction 5](#_Toc202436038)

[**1.1** Revision History 5](#_Toc202436039)

[**1.2** References 5](#_Toc202436040)

[**1.3** Glossary/Acronyms 5](#_Toc202436041)

[**1.4** Document Conventions 5](#_Toc202436042)

[2 Features Overview and Host Usage 6](#_Toc202436043)

[**2.1** 6](#_Toc202436044)

[**2.2** 6](#_Toc202436045)

[**2.3** 6](#_Toc202436046)

[3 Message Processing 7](#_Toc202436047)

[**3.1** Error Responses 7](#_Toc202436048)

[**3.2** Message Extensions 7](#_Toc202436049)

[4 Common Message Fields 8](#_Toc202436050)

[**4.1** Sync 8](#_Toc202436051)

[**4.2** NumBytes 8](#_Toc202436052)

[**4.3** TransactionID 8](#_Toc202436053)

[**4.4** CmdType 8](#_Toc202436054)

[**4.5** CmdID 9](#_Toc202436055)

[**4.6** Checksum 9](#_Toc202436056)

[**4.7** Parameter field 9](#_Toc202436057)

[**4.8** Message Format 9](#_Toc202436058)

[5 Module Messages 10](#_Toc202436059)

[**5.1** Module Identify Command 10](#_Toc202436060)

[**5.2** Module Identify Response 10](#_Toc202436061)

[**5.3** Generic Response 11](#_Toc202436062)

[**5.4** Power Down Dispatch 12](#_Toc202436063)

[**5.5** Power Down Response 12](#_Toc202436064)

[**5.6** Package Dispatch 13](#_Toc202436065)

[**5.7** Package Response 14](#_Toc202436066)

[**5.8** Package Error Response 16](#_Toc202436067)

[**5.9** IP Device Authentication Dispatch 17](#_Toc202436068)

[**5.10** IP Device Authentication Response 18](#_Toc202436069)

[**5.11** IP Device Authentication Error Response 18](#_Toc202436070)

[**5.12** Authorization Pellet Dispatch 19](#_Toc202436071)

[**5.13** Authorization Pellet Response 20](#_Toc202436072)

[**5.14** Authorization Pellet Error Response 20](#_Toc202436073)

[**5.15** CA Container Dispatch 21](#_Toc202436074)

[**5.16** CA Container Error Response 22](#_Toc202436075)

[**5.17** Tunnel Dispatch 23](#_Toc202436076)

[**5.18** Tunnel Response 23](#_Toc202436077)

[6 Status Messages 25](#_Toc202436078)

[**6.1** Time Dispatch 25](#_Toc202436079)

[**6.2** System Status Dispatch 26](#_Toc202436080)

[**6.3** Subscription Status Dispatch 27](#_Toc202436081)

[**6.4** Authorization Status Dispatch 29](#_Toc202436082)

[**6.5** Channel Status Dispatch 30](#_Toc202436083)

[**6.6** Event Report Dispatch 31](#_Toc202436084)

[**6.7** Status Monitor Dispatch 32](#_Toc202436085)

[**6.8** Status Monitor Error Response 33](#_Toc202436086)

[**6.9** Antenna Aiming Dispatch 34](#_Toc202436087)

[**6.10** Signal Status Summary Dispatch 35](#_Toc202436088)

[7 Audio Services Management Messages 41](#_Toc202436089)

[**7.1** Audio Select Dispatch 41](#_Toc202436090)

[**7.2** Audio Select Error Response 43](#_Toc202436091)

[**7.3** Now Playing Dispatch 44](#_Toc202436092)

[**7.4** Playback Control Dispatch 47](#_Toc202436093)

[**7.5** Playback Control Error Response 48](#_Toc202436094)

[**7.6** Smart Favorites Configure Dispatch 49](#_Toc202436095)

[**7.7** Smart Favorites Configure Error Response 50](#_Toc202436096)

[**7.8** Sequential Content Dispatch 51](#_Toc202436097)

[**7.9** SportsFlash Configure Dispatch 53](#_Toc202436098)

[**7.10** SportsFlash Configure Error Response 54](#_Toc202436099)

[**7.11** SportsFlash Event Dispatch 54](#_Toc202436100)

[**7.12** TuneMix Content Dispatch 55](#_Toc202436101)

[**7.13** Alert Packet Dispatch 56](#_Toc202436102)

[**7.14** Alert Packet Error Response 58](#_Toc202436103)

[**7.15** Ad Configure Dispatch 59](#_Toc202436104)

[**7.16** Ad Configure Error Response 60](#_Toc202436105)

[**7.17** Ad Control Dispatch 61](#_Toc202436106)

[**7.18** Ad Control Error Response 62](#_Toc202436107)

[**7.19** Metadata Reference Dispatch 62](#_Toc202436108)

[**7.20** Lookahead Metadata Reference Dispatch 64](#_Toc202436109)

[8 Firmware Update Messages 65](#_Toc202436110)

[**8.1** FWU Start Dispatch 65](#_Toc202436111)

[**8.2** FWU Start Response 66](#_Toc202436112)

[**8.3** FWU Start Error Response 66](#_Toc202436113)

[**8.4** FWU Packet Dispatch 67](#_Toc202436114)

[**8.5** FWU Packet Response 68](#_Toc202436115)

[**8.6** FWU Packet Error Response 68](#_Toc202436116)

[**8.7** FWU Status Dispatch 69](#_Toc202436117)

[9 Production Support Messages 71](#_Toc202436118)

[**9.1** Set Production Data Dispatch 71](#_Toc202436119)

[**9.2** Set Production Data Error Response 71](#_Toc202436120)

[**9.3** Set Service Data Dispatch 72](#_Toc202436121)

[**9.4** Set Service Data Error Response 73](#_Toc202436122)

[**9.5** Get Service Data Dispatch 74](#_Toc202436123)

[**9.6** Get Service Data Response 74](#_Toc202436124)

[**9.7** Tuner Calibration Dispatch 75](#_Toc202436125)

[**9.8** ADI Tuner Calibration Response 75](#_Toc202436126)

[**9.9** Broadcom Tuner Calibration Response 76](#_Toc202436127)

[**9.10** NXP Tuner Calibration Response 77](#_Toc202436128)

[**9.11** GPIO Control Dispatch 78](#_Toc202436129)

[**9.12** GPIO Control Response 79](#_Toc202436130)

[**9.13** Set Operating Mode Dispatch 79](#_Toc202436131)

[**9.14** Set Operating Mode Response 80](#_Toc202436132)

[**9.15** UART1 Control Dispatch 81](#_Toc202436133)

[**9.16** UART1 Control Response 82](#_Toc202436134)

[**9.17** DDR3 Test Control Dispatch 82](#_Toc202436135)

[**9.18** DDR3 Test Control Response 83](#_Toc202436136)

[**9.19** Maintenance Status Dispatch 84](#_Toc202436137)

[**9.20** Maintenance Status Response 84](#_Toc202436138)

[**9.21** C/N Status Dispatch 86](#_Toc202436139)

[**9.22** Messaging Status Dispatch 87](#_Toc202436140)

[**9.23** NVM Test Control Dispatch 88](#_Toc202436141)

[**9.24** NVM Test Control Response 89](#_Toc202436142)

[13 Engineering Messages – Memory 91](#_Toc202436143)

[13.1 Write Memory Dispatch 91](#_Toc202436144)

[13.2 Write Memory Error Response 92](#_Toc202436145)

[13.3 Read Memory Dispatch 92](#_Toc202436146)

[13.4 Read Memory Response 93](#_Toc202436147)

[13.5 Write Serial Flash Dispatch 94](#_Toc202436148)

[13.6 Write Serial Flash Error Response 95](#_Toc202436149)

[13.7 Read Serial Flash Dispatch 95](#_Toc202436150)

[13.8 Read Serial Flash Response 96](#_Toc202436151)

# Introduction

This document describes the first edition of 3D\_Printer\_GCODE\_Commander and the 3D\_Printer\_Module, which will both be used to operate a 3D printer with X, Y, Z axis stepper motors given a GCODE file with valid GCODE commands.

## Revision History

## References

## Glossary/Acronyms

|  |  |
| --- | --- |
| – (hyphen character) | Symbol to represent an inclusive range of values. |
| : (colon character) | Symbol to represent a range of ordered values with first value representing the first value in the group. For example, 0:3 represents values 0, 1, 2, and 3 in that order; 7:4 represents values 7, 6, 5, and 4 in that order. |
| 0x | Prefix indicating hexadecimal notation. |
| LSB | Least Significant Bit |
| Module | The *Module* refers to the 3D\_Printer\_module which runs on a ATSAMD21G18A processor on a Arduino M0 Pro board. The Module is controlled by the 3D\_Printer\_GCODE\_Commander, which supplies it with instructions derived from GCODE commands. |
| Commander | The 3D\_Printer\_GCODE\_Commander desktop application. This application translates GCODE commands into instructions that the Module will be able to understand. |
| UART | Universal Asynchronous Receiver/Transmitter |
| User | The 3D printer operator that interacts with the Commander User Interface. |

## Document Conventions

All numerical values are represented in decimal radix unless stated otherwise or prefixed with “0x” notation representing hexadecimal notation. Numeric values are 8-bit, 16-bit, or 32-bit unsigned or signed (2’s complement) integers. Bit-0 is always the least-significant bit of the least-significant byte of the numeric.

|  |  |  |
| --- | --- | --- |
| **Type (C/C#)** | **# Bytes** | **Definition** |
| uint8 | 1 | integer, 8-bit, unsigned |
| int8, byte | 1 | integer, 8-bit, signed (two’s complement) |
| uint16, ushort | 2 | integer, 16-bit, unsigned |
| int16, short | 2 | integer, 16-bit, signed (two’s complement) |
| uint32, uint(C#) | 4 | integer, 32-bit, unsigned |
| int32, int(C#) | 4 | integer, 32-bit, signed (two’s complement) |

Bitfields are a specified number of bits that are not even multiples of 8 bits. Bit 0 of a bitfield refers to the least-significant bit of the last (rightmost) byte in the representation. All bitfields are designed to fit within an integral number of 8-bit bytes.

# Features Overview and Host Usage

This section provides a brief introduction to the Module, and a brief description of how the Host is expected to use the Module’s capabilities with the Commander main application.

## 

## 

## 

# Message Processing

There are two types of messages in this two device system:

1. Command: Sent from the Commander to the module, contains instructions for the module.
2. Acknowledgment: Sent from module to the Commander as a reply to a command.

Most Commands sent by the Commander Main Application generate a corresponding Acknowledgement from the receiving device(module).

## Error Responses

## Message Extensions

# Common Message Fields

This section provides details of parameters common to more than one message.

## Sync

The Sync provides a fixed bit pattern to identify the start of a message within the byte stream. The Sync is 0xB7 and is the same for all messages. There is nothing to prevent the Sync from otherwise being present in the byte stream, so other fields (e.g., the checksum) within the message must also be checked to confirm the message is valid.

## NumBytes

NumBytes is the number of bytes in the indicated message. It includes the Sync, NumBytes, Checksum, TransactionID, CmdType, CmdID, and all message-specific fields. In all cases, the minimum value of NumBytes is 9, also called the baseMessage size.

The number of Message parameters may range from 0 to 5+, so a message’s size can vary depending on the type of command it contains.

## TransactionID

A TransactionID is an 8-bit value included in each message. The TransactionID in a Dispatch will be echoed by the receiver of the message in the corresponding Response(Acknowledgment), allowing the Sender(Commander) to match up messages with their replies:

The Commander shall set the TransactionID to 0 in all Module Configure Dispatches sent to the Module. For subsequent Dispatches, the Host shall increment the TransactionID by 1 and roll it over back to 0 when it reaches 256 (i.e., it takes on values of 0-255).

The Module shall create an acknowledgement response for the command and assign to it the same TransactionID as the command it received. Then it will be sent to the Commander.

The TransactionID can also be used to detect when messages have been “lost”.

## CmdType

A unique 8-bit CmdType is assigned to each message to identify it as either a G, M, I, R, or E etc. , allowing interpretation of the associated message fields.

G (0) – GCODE G- type, there are several commands that a 3D printer takes that may start with G or M.

M (1) – GCODE M-type

I (2) – Identify Command, sent by Commander to module make a connection between it and the module.

R (3) – Generic Response, sent by Module, received by Commander. Informs Commander of any error codes.

D (4) – Diagnostic report Command, used to diagnose the module and the 3D printer’s overall status.

E (0xF) – Error, can be sent by the Commander or the Module if an error has occurred in either.

## CmdID

A 2 byte value to determine a specific GCODE command. Used in conjunction with the CmdType.

## Checksum

A Checksum is included in each message to allow verification that the message has not been corrupted. An algorithm will generates a 16-bit checksum, computed over all bytes in the message with the checksum bytes set to 0.

## Parameter field

An optional byte field appended to the end of the message carrying parameters that a GCODE may need. These are stored in 4 byte float values as the parameter values can have decimals. Each 4 byte float value will be prefaced by a byte value that contains an ASCII code for a parameter type, e.g. X, Y, Z, E … etc. A message can have more than one parameter.

To determine how many parameter values are in a message, do : (numBytes – baseMessageSize) / (sizeof(float) +1).

## Message Format

Below is the format of what a message to the module will look like, in a byte stream format.

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumBytes | uint16 | 9 |
| 3 | TransactionID | uint8 | 0 |
| 4-5 | Checksum | uint16 | See Section 4.6 |
| 6 | CmdType | uint8 | 2 |
| 7-8 | CmdID | uint16 | 0 |
| 8+ | Optional Command Parameters: | uint8  +  uint32 | See Section 4.7 |

# Module Messages

## Module Identify Command

The Commander sends the ‘Module Identify Command’ to the Module to establish communication with the Module and to configure its operational parameters.

The Module ignores any data received on the UART interface during its startup process. Once it is fully operational, it will begin interpreting data on the UART interface. The Module Identify Command is the first message sent from the Host that the Module will accept after the Module is reset.

If the Module is not ready to process the Module Identify Command, then the Module shall not send any response to that Dispatch. The sender should continuously transmit the Module Configure Dispatch to the Module until it receives a Module Identify Response. If no Module Identify Response is received by the Host after retrying for 10 seconds after powering and/or resetting the Module, the Host shall reset the Module and restart the process.

Message Usage: Establish Module Connection

Message Direction: Sent to Module from Commander

Response: Module Identify Response

Response Timeout: 2 seconds

Minimum **NumBytes**: 9

Maximum **NumBytes**: 9

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumBytes | uint16 | 9 |
| 3 | TransactionID | uint8 | 0 |
| 4-5 | Checksum | uint16 | See Section 4.6 |
| 6 | CmdType | uint8 | 2 |
| 7-8 | CmdID | uint16 | 0 |
| 8+ | Reserved, Not in Use | uint8  +  uint32 | See Section 4.7 |

## Module Identify Response

When the Module receives a Module Identify Dispatch, it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Module Configure Response to the Host.

Once this message is received by the sender of the Module Configure Dispatch, it is free to send any other valid SXi8 Dispatches to the Module. Also, the Module may start sending Dispatches to the sender to report status, etc.

Message Usage: Establish Module Connection

Message Direction: Sent from Module

Minimum **NumBytes**: 9

Maximum **NumBytes**: 9

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumBytes | uint16 | 9 |
| 3 | TransactionID | uint8 | 0 |
| 4-5 | Checksum | uint16 | See Section 4.6 |
| 6 | CmdType | uint8 | 2 |
| 7-8 | ModuleVersion | uint16 | See below |
| 8+ | Reserved, Not in Use | uint8  +  uint32 | See Section 4.7 |

**Module Version**:

This entry will be used to store the major and minor numbers of the module version. This version number is returned in the Module Identify Response by the module when the Identify Command is sent and processed. The major and minor numbers at uint8 byte values, the major is stored in the upper 8 bits, whereas the minor number is stored in the lower 8 bits.

## Generic Response

For G, or M Commands sent to the module, only an acknowledgement that the Dispatch was received is necessary to send back to the sender. CmdType and CmdID values in the response are mirrored from the Generic Command sent to the Module. If there are any error codes corresponding to processing problems on the module side, this is where it would be placed to alert the Commander. Should the Host (Commander) notice warnings, a Diagnostic command should be sent to inquire about the specific errors.

Message Usage: Respond to Generic Command

Message Direction: Sent from Module, Sent to Module

Test: Full validation by Test Team

Minimum **NumBytes**: 10

Maximum **NumBytes**: 10

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumBytes | uint16 | 63 |
| 3 | TransactionID | uint8 | 0 |
| 4-5 | Checksum | uint16 | See Section 4.6 |
| 6 | CmdType | uint8 | 3 |
| 7-8 | CmdID | uint16 | See below |
| 9 | Status | uint8 | See below |

**Status**:

Indicates status of the Dispatch/Response. The valid values are:

0 = Dispatch received and processed

1 = Dispatch received but not processed

TO DO:

## Power Down Dispatch

The Host sends the Power Down Dispatch to the Module to have it perform an orderly shutdown of the Module. Should the Module not be shutdown in an orderly manner (e.g., the reset line is toggled or power is removed prior to the Host receiving the Power Down Response), then the Module will still operate properly on the next startup.

If the Module receives an additional Power Down Dispatch, it will send the Power Down Response to the sender but will otherwise ignore the message. The Module shall ignore any other Dispatch it receives once it has processed the Power Down Dispatch.

When a Factory Default power down is commanded, the Module will cause the appropriate NVM data to be reset to its default values prior to the completion of the Module’s next initialization. The **ProductionData,** **ServiceData,** Tuner Calibration Data, and Maintenance Status are not reset by a Factory Default.

As part of the shutdown process the Module shall mute the audio output if it is not already muted. A fading profile will be applied by the Module when necessary to effect the mute. That is, the Module will apply intermediate gains over a brief period of time when adjusting the gain from one value to another.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Power Down Response

Response Timeout: 1100 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 4 |
| 7 | Command | uint8 | See below |

**Command**: Type of power down to perform. The valid values are:

0 = Normal Shutdown

1 = Factory Default: mark applicable NVM parameters as needing to be reset to their default values and shutdown

## Power Down Response

When the Module receives a Power Down Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, performs any processing associated with that command, and sends the Power Down Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 8

The CAPPowerDownStatus field was a late addition to this message. Module FW using v1.29 and earlier will transmit this message without this field and the message will have a length of 8 bytes.

Maximum **NumMsgBytes**: 32

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 9 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 5 |
| 7 | Status | uint8 | See below |
| 8 | CAPPowerDownStatus | uint8 | See below |

**Status**: Provides status on the Module power down. The valid values are:

0 = Power Down Dispatch processed

1 = Power Down Dispatch was previously received; Dispatch ignored

2 = **Command** was invalid; Power Down Dispatch processed as if **Command**=0

**CAPPowerDownStatus**: Indicates the CAP’s power down status. The valid values are:

0 = CAP successfully powered down

1 = CAP successfully powered down, but required more than one attempt

2 = CAP did not successfully power down

## Package Dispatch

The Host sends the Package Dispatch to query or select the subscription package in a Module that has multiple Factory Activation packages.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Package Response or Package Error Response

Response Timeout: 2000 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 9

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 9 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | 0 |
| 6 | MessageID | uint8 | 7 |
| 7 | Command | uint8 | 0 |
| 8 | PackageIndex | uint8 | See below |

**Command**: Action to perform. The valid values are:

0 = Pellet: request the MPFA pellet which identifies the current state of the Factory Activation Multi-Package process. See the Package Response for details on the pellet content and use cases.

1 = Select: selection of a package from the Factory Activation Multi-Package array

**PackageIndex**: Index of the selected package. This parameter is only valid for **Command**=1, and is Reserved for all other **Command** values. A value of 0 forces the Module to exit the package selection state (pre-selection) and clear the subscription package. The range of values is 0 to 252.

## Package Response

When the Module receives a Package Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Package Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 36

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 36 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 8 |
| 7-31 | SelectionPellet[25] | uint8 | See below |
| 32-33 | ASID1 | uint16 | See below |
| 34-35 | ASID2 | uint16 | See below |

**SelectionPellet**: An identification block reported to Sirius XM and used for supply chain management and tracking of authorized Modules. It contains Module identification, MASA identification, selected packet identification, and a signature to verify stat and reported information.

**ASID1**: Authorization State ID for the first Authorization State Unit of the selected packets. Prior to selection, value set to 0 when responding to **Command**=0 (i.e., Pellet) in Package Dispatch.

**ASID2**: Authorization State ID for the second Authorization State Unit of the selected packets. Prior to selection, value set to 0 when responding to **Command**=0 (i.e., Pellet) in Package Dispatch.

<<TODO: Need to move the following material to somewhere more appropriate (Jama?).>>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MPFA State | Action | Return Code/Results | Comment | Example Selection byte & Hash (or MAC)[[1]](#footnote-1) |
| No MASA present | MpsPellet | 153:CyMpsMissing | Single Package Factory Activated state (SPFA) | N/A  (returned Pellet is all zeros) |
| MpsSelect | 153:CyMpsMissing |
| MASA Loaded and Unselected | MpsPellet | * 0:Success * MPFA Selection Pellet * ASIDs empty | Normal NPFA response before making package selection at OEM | Selection byte: ff  Hash:99dd85cb3540a6c7e672  ASIDs { 0, 0 } |
| MpsSelect | * 0:Success * MPFA Selection Pellet * Selected ASIDs | Normal response to making package selection at OEM | Selection byte: 01  MAC: 5de6d50ec8204f05fb27 ASIDs: { 257, 514 } |
| MASA Loaded and Selected | MpsPellet | * 0:Success * MPFA Selection Pellet * Previously selected ASIDs | Possible normal post-selection pellet extraction at OEM | Selection byte: 01  MAC: 5de6d50ec8204f05fb27 ASIDs: { 257, 514 } |
| MpsSelect | 149:CyMpsAlreadySelected | Erroneous reselection attempt | N/A  (returned Pellet is all zeros) |
| MASA Loaded but revoked | MpsPellet | * 0:Success * MPFA Selection Pellet | Radio was authorized over SCADS or IP before selection was made | Selection byte: fe  MAC: 6452c44aff49d442c8b4  ASIDs { 0, 0 } |
| MpsSelect | 148:CyMpsRevoked | Erroneous reselection attempt | N/A  (returned Pellet is all zeros) |
| MASA Loaded and Selected, & Replaced | MpsPellet | After OTA or IP assignment, selection pellet and selected ASIDs remain available even though the reported values no longer reflect actual radio state. (See MASA Loaded and Selected row above) | | |
| MpsSelect |

State Action Return code/Results

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

No MPFA MpsQuery 153:CyMpsMissing (MASA Invalid)

MpsVerify 153:CyMpsMissing

MpsSelect 153:CyMpsMissing

MPFA Loaded, MpsQuery 0:Success

unselected Pellet (eg a0012345678900001234b032f305ffb7e4ddc16a88edaa45be)

ASIDs empty ({ 0, 0 })

MpsVerify 151:CyMpsUnselected (Selection Required)

MpsSelect 0:Success

Pellet (eg a0012345678900001234b032f3050155aabcf9d3702467c63b)

ASIDs not empty (eg { 9029, 13398 })

MPFA Loaded, MpsQuery 0:Success

selected Pellet (eg a0012345678900001234b032f30501b7e4ddc16a88edaaxxxx)

ASIDs not empty (eg { 9029, 13398 })

MpsVerify 0:Success

Pellet (eg a0012345678900001234b032f3050155aabcf9d3702467c63b)

ASIDs not empty (eg { 9029, 13398 })

MpsSelect 149:CyMpsAlreadySelected (Already Selected)

MPFA Loaded, MpsQuery 0:Success

revoked Pellet (eg a0012345678900001234b032f305fe1dfbab957465f8c60527)

ASIDs empty ({ 0, 0 })

MpsVerify 0:Success

Pellet (eg a0012345678900001234b032f305feba96e3d677ffa15f9573)

ASIDs empty ({ 0, 0 })

MpsSelect 148:CyMpsRevoked

MPFA Loaded, MpsQuery 0:Success

selected, Pellet (eg a0012345678900001234b032f30501051956d572bac81ed395)

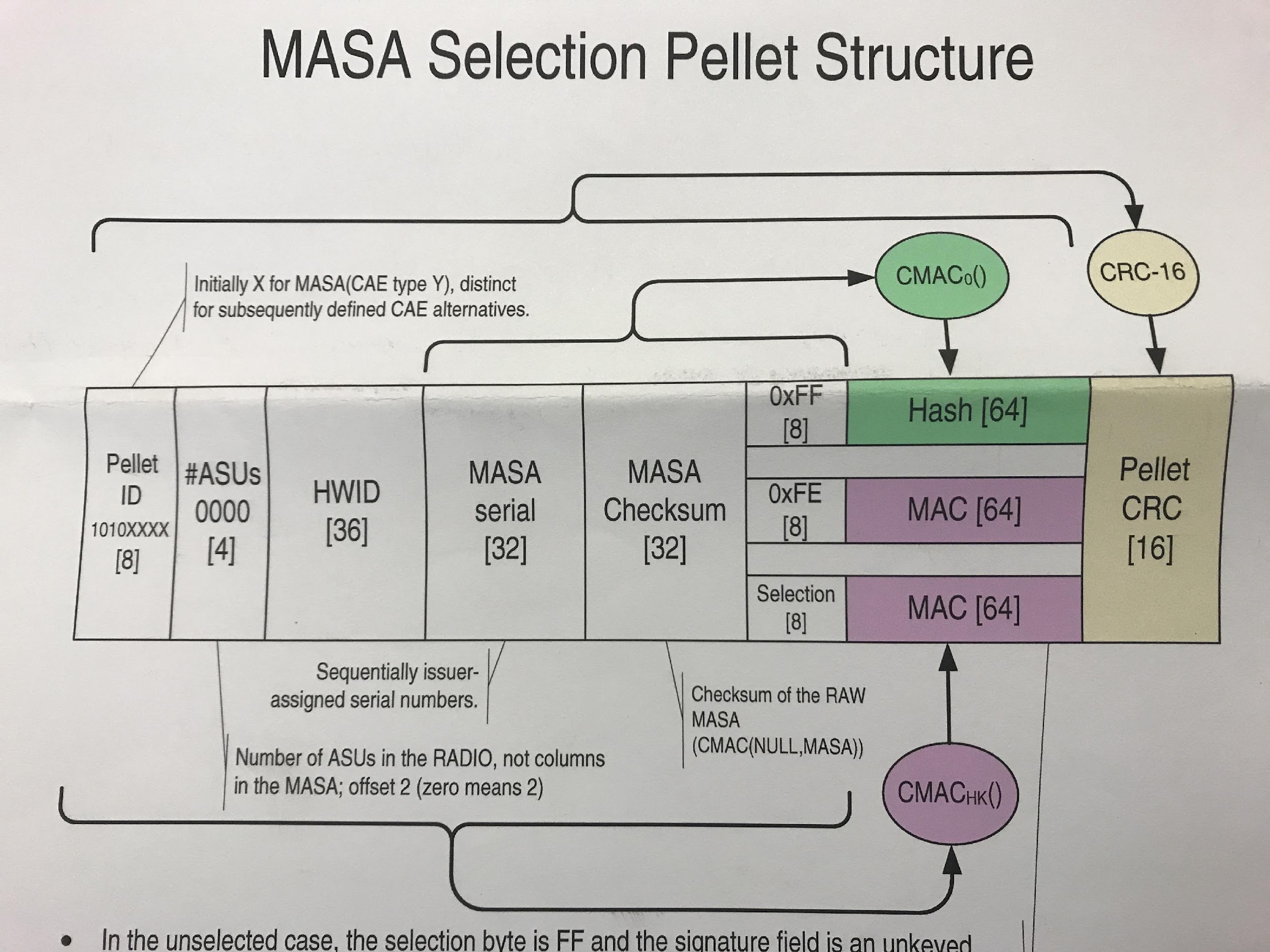
CADS changed ASIDs empty ({ 0, 0 })

MpsVerify 0:Success

Pellet (eg a0012345678900001234b032f3050155aabcf9d3702467c63b)

ASIDs stale (eg { 9029, 13398 })

MpsSelect 149:CyMpsAlreadySelected



## Package Error Response

When the Module receives a Package Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Package Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

The **CAPErrorCode** field was a late addition to this message. Module FW using SXi8 v1.20 and before will transmit this message without this field and the message will have a length of 56 bytes.

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 60 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 9 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |
| 56-59 | CAPErrorCode | uint32 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = Invalid **Command**

1 = Invalid **PackageIndex**

2 = CAP returned an error code

3 = CAP did not respond to command

4 = CAP fatally unresponsive

5 = CAP failed to boot or initialize

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

**CAPErrorCode**: Error code returned by the CAP. This field is only valid when **ErrorCode** = 2, otherwise the value should be ignored.

## IP Device Authentication Dispatch

The Host sends the IP Device Authentication Dispatchto request a cryptographic signature of a Challenge Request parameter (provided to the Host via an IP message) in order to authenticate the Host for SiriusXM applications involving IP connectivity. The IP Device Authentication Dispatchdelivers the **ChallengeRequest** to the Module, and the Module returns the signed **ChallengeResponse** via the IP Device Authentication Response*.*

When the Module receives an IP Device Authentication Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the IP Device Authentication Response to the Host. If these operations are not successful then the Module sends the IP Device Authentication Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: IP Device Authentication Response or IP Device Authentication Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 15

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 15 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 74 |
| 7-14 | ChallengeRequest[8] | uint8 | See below |

**ChallengeRequest**: Challenge parameter.

## IP Device Authentication Response

When the Module receives an IP Device Authentication Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command (i.e., the Module authenticates the **ChallengeRequest**, adds a cryptographic signature, and sets the result in the **ChallengeResponse**). If these operations are successful then the Module sends the IP Device Authentication Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 37

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 37 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 75 |
| 7-36 | ChallengeResponse[30] | uint8 | See below |

**ChallengeResponse**: Received **ChallengeRequest** with signature. The valid values for each byte are 0-255.

## IP Device Authentication Error Response

When the Module receives an IP Device Authentication Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the IP Device Authentication Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

The **CAPErrorCode** field was a late addition to this message. Module FW using SXi8 v1.20 and before will transmit this message without this field and the message will have a length of 56 bytes.

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 60 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 76 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |
| 56-59 | CAPErrorCode | uint32 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = Invalid ChallengeRequest size

2 = CAP returned an error code

3 = CAP did not respond to command

4 = CAP fatally unresponsive

5 = CAP failed to boot or initialize

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

**CAPErrorCode**: Error code returned by the CAP. This field is only valid when **ErrorCode** = 2, otherwise the value should be ignored.

## Authorization Pellet Dispatch

The Host sends the Authorization Pellet Dispatchto request a signed Authorization State for Sirius XM applications involving IP connectivity. The Module returns the signed Authorization State (**AuthStatePellet**) via the Authorization Pellet Response*.*

When the Module receives an Authorization Pellet Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Authorization Pellet Response to the Host. If these operations are not successful then the Module sends the Authorization Pellet Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Authorization Pellet Response or Authorization Pellet Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 7

Maximum **NumMsgBytes**: 32

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 7 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 77 |

## Authorization Pellet Response

When the Module receives an Authorization Pellet Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Authorization Pellet Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 79

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1:2 | NumMsgBytes | uint16 | 79 |
| 3:4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 78 |
| 7-78 | AuthStatePellet[72] | uint8 | See below |

**AuthStatePellet:** Signed authorization state.

## Authorization Pellet Error Response

When the Module receives an Authorization Pellet Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Authorization Pellet Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

The **CAPErrorCode** field was a late addition to this message. Module FW using SXi8 v1.20 and before will transmit this message without this field and the message will have a length of 56 bytes.

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 60 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 79 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |
| 56-59 | CAPErrorCode | uint32 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

2 = CAP returned an error code

3 = CAP did not respond to command

4 = CAP fatally unresponsive

5 = CAP failed to boot or initialize

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

**CAPErrorCode**: Error code returned by the CAP. This field is only valid when **ErrorCode** = 2, otherwise the value should be ignored.

## CA Container Dispatch

The Host sends the CA Container Dispatchto deliver a Conditional Access Container to the Module. Any CA Container or container fragment can be delivered to the Module with this message, but the primary use cases are:

1. Deliver a CA Container recovered via an IP connection
2. Deliver a Factory Activation Container for either a single image or MPFA image array
3. Deliver CA Containers collected by the Host via an RFD data service.

When the Module receives a CA Container Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the CA Container Response to the Host. If these operations are not successful then the Module sends the CA Container Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or CA Container Error Response

Response Timeout: 1500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 10

Maximum **NumMsgBytes**: 8192

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | Varies |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 83 |
| 7-8 | NumBytes | uint16 | See below |
| 9-8+NumBytes | Data[NumBytes] | uint8 | See below |

**NumBytes**: Number of bytes in **Data**. The valid values are 1 thru 8183.

**Data**: CA Container data.

## CA Container Error Response

When the Module receives a CA Container Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the CA Container Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

The **CAPErrorCode** field was a late addition to this message. Module FW using SXi8 v1.20 and before will transmit this message without this field and the message will have a length of 56 bytes.

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 60 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 84 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |
| 56-59 | CAPErrorCode | uint32 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = No fragments or NULL pointer

2 = CAP returned an error code

3 = CAP did not respond to command

4 = CAP fatally unresponsive

5 = CAP failed to boot or initialize

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

**CAPErrorCode**: Error code returned by the CAP. This field is only valid when **ErrorCode** = 2, otherwise the value should be ignored.

## Tunnel Dispatch

This message is reserved for internal use by the Module and Host to allow each of these devices to augment the available SXi8 messages with their own messages. Each device can develop a set of proprietary, device-specific messages under the umbrella of this message. As an example, the Module FW uses the Tunnel Dispatch as an envelope for its internal inter-task messages, however, if the Module receives this message on the SXi8 interface it will ignore it. There is no requirement that any device use this message for its internal processing, and each device can choose how to reply to this message.

Message Usage: Internal Host or Internal Module Use

Message Direction: Not applicable

Response: Device specific

Response Timeout: Device specific

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: Device specific, but subject to overall maximum message size

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | See below |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | 0 |
| 6 | MessageID | uint8 | 10 |
| 7-NumMsgBytes-1 | Payload[NumMsgBytes-7] | uint8 | See below |

**NumMsgBytes**: The number of bytes in the message is variable based on the device’s use of the message. LSB = 1 byte. The range of values is defined internally by each device.

**Payload**: The message payload. It’s format, number of bytes, etc., are internally defined by each device.

## Tunnel Response

This message is reserved for internal use by the Module and Host to allow each of these devices to augment the available SXi8 messages with their own messages. Each device can develop a set of proprietary, device-specific messages under the umbrella of this message. If the Module receives this message on the SXi8 interface it will ignore it. There is no requirement that any device use this message for its internal processing, and each device can choose how to reply to this message.

Message Usage: Internal Host or Internal Module Use

Message Direction: Not applicable

Response: Device specific

Response Timeout: Device specific

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: Device specific, but subject to overall maximum message size

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | See below |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | 0 |
| 6 | MessageID | uint8 | 126 |
| 7 | ResponseStatus | uint8 | See below |

**ResponseStatus**: Processing status of the Tunnel Dispatch to which this is a response. The range of values is defined internally by each device.

# Status Messages

The Module sends status messages to the Host to inform it of various Module and System statuses:

* Some statuses are essential for implementation of all Production products, so the corresponding status dispatches are automatically enabled at startup and cannot be disabled.
* Some statuses are needed for Production products, but only under certain conditions, so the Host is allowed to enable and disable the corresponding status dispatches as needed.
* Some statuses are only needed in an Engineering & Test environment, so the Test Host is allowed to enable and disable the corresponding status dispatches as needed.
* Some Engineering statuses change at a high rate. It is assumed that the Test Host is capable of receiving all messages without loss and that acknowledgement of certain high-frequency Engineering messages, and subsequent resends of unacknowledged messages by the Module, is unnecessary.

## Time Dispatch

The Module sends the Time Dispatch to report the current SiriusXM High Band network date and time to the Host. The Module shall send this message once at startup, that is, after the Module Configure Response is sent to the Host and after the Module synchronizes with SiriusXM High Band network time. If the SiriusXM signal is never acquired then the message will never be sent.

Thereafter, the message is sent automatically whenever any of the fields contained in the message changes. Typically this will be on minute boundaries (i.e., when the SiriusXM network time transitions from 59 seconds to 00 seconds). The initial Time Dispatch might not be aligned to “0 seconds” but subsequent Time Dispatches will be aligned to “0 seconds”. This may result in two Time Dispatches within the first minute after the Sirius XM signal is acquired.

If the Module becomes synchronized with SiriusXM network time and then subsequently loses reception of the SiriusXM network signal, the Module will continue to send Time Dispatches in accordance with its internally generated clock. Any time drift will be corrected when SiriusXM signal is again available.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Time Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 12

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 11 |
| 7 | Minute | uint8 | See below |
| 8 | Hour | uint8 | See below |
| 9 | Day | uint8 | See below |
| 10 | Month | uint8 | See below |
| 11 | Year | uint8 | See below |

**Minute**: Current minute of Sirius XM network (UTC) time. LSB=1 minute. The range of values is 0 to 59.

**Hour**: Current hour of Sirius XM network (UTC) time. LSB=1 hour. The range of values is 0 to 23.

**Day**: Current day of Sirius XM network (UTC) time. LSB=1 day. The range of values is 1 to 31.

**Month**: Current month of Sirius XM network (UTC) time. LSB=1 month. The range of values is 1 to 12.

**Year**: Current year of Sirius XM network (UTC) time, offset from 2000. E.g., the year 2016 would be reported as 16. LSB=1 year. The range of values is 16 to 99.

## System Status Dispatch

The Module sends the System Status Dispatch to report the various statuses always needed for User display or other routine operational needs. The Module shall send this message once at startup, that is, after the Module Configure Response is sent to the Host. Thereafter, the message is sent automatically whenever any of the fields contained in the message changes.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the System Status Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 12

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 12 |
| 7 | SummaryStatus | uint8 | See below |
| 8-9 | ScannableUnplayedTracks | uint16 | See below |
| 10-11 | TuneMixStatus | uint16 | See below |

**SummaryStatus**: Reports the Module’s summary status, with bit fields defined as:

| **Bit7** | **Bit 6** | **Bits 5** | **Bit 4** | **Bits 3-2** | **Bits 1-0** |
| --- | --- | --- | --- | --- | --- |
| AudioPresent | AdPending | AdImminent | HiTemp | AntennaStatus | NumBars |

AudioPresent: Audio present status, with values:

0 = Non-zero audio is not being output on the Module’s I2S interface

1 = Non-zero audio is being output on the Module’s I2S interface

AdPending: Indicates if there is an Ad Insertion or Ad Replacement currently scheduled, with values:

0 = no Ad Insertion or Ad Replacement is scheduled

1 = Ad Insertion or Ad Replacement is scheduled

AdImminent: Indicates that an Ad Insertion or Ad Replacement is expected to happen within the next second, with values:

0 = Ad Insertion or Ad Replacement is not imminent

1 = Ad Insertion or Ad Replacement is imminent

When the Ad Replacement feature is used, AdImminent is only set when an new ad replacement session is imminent. AdImminent will not be set if an existing ad replacement session is re-played.

HiTemp: Reports the Module’s temperature warning status, with values:

0 = the Module’s temperature is below the warning temperature threshold

1 = the Module’s temperature is at or above the warning temperature threshold

NumBars: Reports the Module’s summary reception status, with values:

0 = 0 bars – very weak or no signal

1 = 1 bar – weak signal

2 = 2 bars – marginal signal

3 = 3 bars – good signal

AntennaStatus: Reports the antenna status, with values:

0 = Antenna okay

1 = Antenna disconnected

2 = Antenna shorted

**ScannableUnplayedTracks**: Indicates if there are enough scannable tracks available (i.e., two or more) to initiate TuneScan. The valid values are:

0 = Insufficient scannable tracks are available to initiate TuneScan

1 = Sufficient scannable tracks are available to initiate TuneScan

**TuneMixStatus**: Summary status for TuneMix, with bit fields defined as:

| **Bit 15** | **Bit 14** | **Bits 13-6** | **Bits 5-0** |
| --- | --- | --- | --- |
| Playable | HasStart | Reserved | NumComponents |

NumComponents: The number of valid, available component channels assigned to TuneMix. The valid values are 0-24.

HasStart: Indicates if the Module has the start of at least one track from one of TuneMix’s component channels:

0 = does not have start of any tracks

1 = has start for at least one track

Playable: Indicates if the Host can select TuneMix for playback:

0 = TuneMix is not playable (i.e., does not have sufficient component music channels)

1 = TuneMix is playable

## Subscription Status Dispatch

The Module sends the Subscription Status Dispatch to report various statuses related to the Module’s overall subscription state. This message does not report individual channel subscription statuses. The Module shall send this message once at startup, that is, after the Module Configure Response is sent to the Host. Thereafter, the message is automatically sent whenever any of the fields contained in the message changes.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Subscription Status Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 13

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 13 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 13 |
| 7 | Status | uint8 | See below |
| 8 | SubscriptionStatus | uint8 | See below |
| 9-12 | FCADSVersion | uint32 | See below |

**Status**: Reports the validity of the subscription status. A value of 0 indicates that **SubscriptionStatus** is valid. Any other value indicates that **SubscriptionStatus** is not valid and the subscriptions status of the Module is uncertain. See the CAP FW Development Team for the interpretation of non-zero values of this field.

**SubscriptionStatus**: Reports the Module’s subscription status. If **Status** indicates that the subscription status is invalid, then this field is set to 0. **SubscriptionStatus** has bit fields defined as:

| **Bit7** | **Bits 6-5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bits 1-0** |
| --- | --- | --- | --- | --- | --- |
| AudioSub | Reserved | L2Reconfig | CAPFWOp | SubUpdate | SubStatus |

AudioSub: Audio subscription summary, with values:

0 = Not subscribed to any audio services

1 = Subscribed to at least one audio service that is not FTA

L2Reconfig: Reports that the Module’s Layer 2 (Low Band Base Layer) channel lineup is changing and is affecting audio playback. During Layer 2 channel lineup changes there is a brief period where service decryption may be blocked until the system is again synchronized. L2Reconfig is valid only when OpMode=0. Valid values:

0 = Audio playback is not being affected by a Layer 2 reconfiguration.

1 = Audio playback is being affected by a Layer 2 reconfiguration (only happens when playing from the Live point of a Layer 2 channel during a Layer 2 reconfiguration).

CAPFWOp: Reports if the CAP FW is operational. The CAP FW could be non-operational either because a valid version is not loaded, or the loaded version is blocked from running for security reasons. Valid values:

0 = CAP FW is operational

1 = CAP FW is not operational

SubUpdate: Reports that the Module’s subscription package has changed since it was last reported, with values:

0 = Subscription package has not changed since last report

1 = Subscription package has changed since last report

SubStatus: Reports the overall Module subscription status, with values:

0 = Not subscribed

1 = Subscribed

2 = MPFA Pre-Select

**FCADSVersion**: Identifies the version of the last FCADS file processed.

## Authorization Status Dispatch

The Module sends the Authorization Status Dispatch to report the current Authorization State Unit (ASU) values in the Module and corresponding details. The Module shall send this message once at startup, that is, after the Module Configure Response is sent to the Host. Thereafter, the message is automatically sent whenever any of the fields contained in the message changes.

This message is automatically enabled and cannot be disabled by the Host.

Gen8 supports two user ASUs with values assigned via subscription processes. The Module is authorized for the combination of all services authorized (logical OR of all SID or DMI vectors) by the two user ASUs and a default ASU. A user ASU might not be present for a given package, and this is indicated with an ASID value of 0. When ASID=0, all other parameters related to that ASU will be set to 0.

The sending of this message is controlled by the Status Monitor Dispatch.

The receiver sends the Generic Response to the Module to confirm receipt of the Authorization Status Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 13

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 13 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 27 |
| 7-8 | ASID1 | uint16 | See below |
| 9 | ASID1SlotStatus | uint8 | See below |
| 10-11 | ASID2 | uint16 | See below |
| 12 | ASID2SlotStatus | uint8 | See below |

**ASID1**: Authorization State Identifier for ASU1.

**ASID1SlotStatus**: ASID1 Status.

| **Bit 7-3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- |
| Reserved | MPFA\_Pending | Pending | Expired |

Expired: Authorization State expiry status:

0 = Not expired

1 = Expired

Pending: Authorization State Definition (ASD) pending status:

0 = Not Pending

1 = Pending

MPFA\_Pending: After a successful MPFA selection; Authorization State Definition pending status:

0 = Not Pending

1 = Pending

**ASID2**; **ASID2SlotStatus**: Same as above but for ASU2/ASID2.

## Channel Status Dispatch

The Module sends the Channel Status Dispatch to the Host to report the availability status of each of the Sirius XM channels. The Module shall send this message once at startup, that is, after the Module Configure Response is sent to the Host. In general, this will contain the last channel status the Module has stored in its NVM. Thereafter, the Module will send the message whenever any of the fields contained in the message changes (e.g., as driven by received OTA channel lineup updates).

If the Host attempts to tune to an unavailable channel the Module will reject the request. The Host is responsible for examining a newly received Channel Status Dispatch and taking any appropriate actions with regard to channel management.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Channel Status Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 135

The **ChNotSubscribedMap** field was a late addition to this message. Module FW using SXi8 v1.16 and before will transmit this message without this field and the message will have a length of 135 bytes.

The **ChFTAMap** field was a late addition to this message. Module FW using SXi8 v2.2 and before will transmit this message without this field and the message will have a length of 263 bytes.

Maximum **NumMsgBytes**: 512

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 391 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 88 |
| 7-134 | ChAvailMap[128] | uint8 | See below |
| 135-262 | ChNotSubscribedMap[128] | uint8 | See below |
| 263-390 | ChFTAMap[128] | uint8 | See below |

**ChAvailMap**: A SID vector, with each bit indicating:

1 = the SID is an audio channel and the user is capable of tuning to it

0 = otherwise

See 4.10 for the definition of the SID vector.

**ChNotSubscribedMap**: A SID vector, with each bit indicating:

1 = the SID is an audio channel, the user is capable of tuning to it, and the Module is not subscribed to it

0 = otherwise, and for SID=1

See 4.10 for the definition of the SID vector.

**ChFTAMap**: A SID vector, with each bit indicating:

1 = the Module has identified this SID as a free-to-air (FTA) audio channel

0 = otherwise

See 4.10 for the definition of the SID vector.

## Event Report Dispatch

The Module sends the Event Report Dispatch to report an unexpected event has occurred. The Host should take the appropriate action based on the EventType. Typically this Dispatch will only be issued by production Module code under rare circumstances.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Event Report Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 60

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 60 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 14 |
| 7 | EventType | uint8 | See below |
| 8-11 | EventTime | uint32 | See below |
| 12-59 | EventText[48] | uint8 | See below |

**EventType**: Indicates what type of event is being reported. The valid values are:

0 = Fatal Event: the Module has detected an unrecoverable system error. The Host shall reset the Module to recover proper operation.

1 = Overtemp Event: the Module has detected it is beyond its safe operating temperature. The Host shall take the proper action to protect the Module from permanent damage and to safely recover proper operation.

2 = Reconfiguration Event: The Low Band Operational Mode has changed. The Module needs to be reset to properly receive Low Band channels. The Host does not have to immediately reset the Module (e.g., if the User is not using Low Band services), and can wait until the next ignition cycle if desired.

All others = Non-Fatal Event: The Module should continue to operate normally, but information is being provided to report that unexpected Module behavior was detected. This is intended for SiriusXM Engineering Use Only. All Events of these types should be ignored by the Host.

**EventTime**: Time elapsed from Module startup that the event was detected. LSB=1 second.

**EventText**: ASCII characters describing the event in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Status Monitor Dispatch

The Host sends the Status Monitor Dispatch to the Module to control the reporting of specified Module statuses. On startup, all status monitors that can be enabled/disabled are disabled.

When a status message becomes enabled, the Module will respond with the current state of the status item in an appropriate Dispatch. Thereafter, the Module will check the status item periodically and if the value has changed, in general the Module will send an appropriate Dispatch with the new value of the status item(s).

When the Module receives a Status Monitor Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Status Monitor Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Status Monitor Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 9

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 9 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 15 |
| 7 | Command | uint8 | See below |
| 8 | StatusMessage | uint8 | See below |

**Command**: Action to perform. The valid values are:

0 = Disable the sending of the specified status message

1 = Enable the sending of the specified status message

2 = Send the specified status message once (i.e., one-shot), ignore any response, and do not send retries if no response is received

3 = Reset any latched status associated with the specified status message

**StatusMessage**: Status message to perform the requested action on. The valid values are:

Production Host dispatches, valid for **Command**=one-shot:

0 = Time Dispatch

1 = System Status Dispatch

2 = Subscription Status Dispatch

3 = Authorization Status Dispatch

4 = Channel Status Dispatch

5 = Event Report Dispatches (one-shot sends last 10 events)

Production Host dispatches, valid for **Command**=disable, enable, one-shot:

40 = Antenna Aiming Dispatch

41 = Signal Status Summary Dispatch

Production Host dispatches, valid for **Command**=disable, enable, one-shot, reset:

100 = Messaging Status Dispatch

Engineering Only dispatches, for **Command**=disable, enable, one-shot, reset:

101 = Queue Status Dispatch

102 = Audio Decoder Status Dispatch

103 = Audio Decoder Detailed Status Dispatch

104 = SPI Status Dispatch

105 = Authorization Detailed Status Dispatch

106 = Application Payload Element Dispatch

107 = Channel Info Dispatch

108 = BIC Status Dispatch

109 = CAP Status Dispatch

110 = DMA Status Dispatch

111 = CA debug dispatches (CA Error Indication Dispatch, CA Debug Indication Dispatch, and CA Diagnostic Indication Dispatch)

120 = AVS Status Dispatch

140 = ADI, NXP, or Broadcom Tuner Status Dispatch

150 = C/N Status Dispatch

160 = High Band MFC Dispatch

161 = High Band Signal Quality Dispatch

162 = High Band ATE Dispatch

163 = Service Layer Status Dispatch

164 = High Band FFT Dispatch

165 = High Band Overlay Pipe Status Dispatch

180 = Low Band MFC Dispatch

181 = Low Band Signal Quality Dispatch

182 = Low Band ATE Dispatch

184 = Low Band Overlay Pipe Status Dispatch

185 = Low Band Pipe Header Dispatch

186 = High Band Fast C/N Dispatch

Engineering Only dispatches, for **Command**=disable, enable:200 = TuneMix Track Evaluation Dispatch

201 = Ad Status Dispatch

## Status Monitor Error Response

When the Module receives a Status Monitor Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Status Monitor Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 16 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = requested **Command** is unknown; message ignored

2 = requested **StatusMessage** is unknown; message ignored

3 = requested **StatusMessage** is not valid for the specified **Command**; message ignored

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Antenna Aiming Dispatch

The Module sends the Antenna Aiming Dispatch to report High Band and Low Band signal strengths to aid the User in locating their antenna for a fixed (i.e., non-automotive) installation. The Module shall send this message once when the Antenna Aiming Dispatch becomes enabled, and thereafter, the message is sent once per second while it remains enabled, regardless of whether or not any data in the message changes.

The sending of this message is controlled by the Status Monitor Dispatch.

The Module ignores any Host response to this message, and the Module will not attempt resends of this message.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: None

Response Timeout: NA

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 9

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 11 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 18 |
| 7 | HBSignalStrength | uint8 | See below |
| 8 | Reserved0 | uint8 | See below |
| 9 | LBSignalStrength | uint8 | See below |
| 10 | Reserved1 | uint8 | See below |

**HBSignalStrength**: The High Band signal strength, represented as a percentage of the maximum signal strength. LSB = 1 percent. The range of values is 0 (no signal) to 100 (maximum signal).

**LBSignalStrength**: The Low Band signal strength, represented as a percentage of the maximum signal strength. LSB = 1 percent. The range of values is 0 (no signal) to 100 (maximum signal).

## Signal Status Summary Dispatch

The Module sends the Signal Status Summary Dispatch to report all of the information needed for display on the TA1 Diagnostic Signal Quality display as specified by the MFFR. The Module shall send this message once when the Signal Status Summary Dispatch becomes enabled, and thereafter, the message is sent once per second while it remains enabled, regardless of whether or not any data in the message changes.

The sending of this message is controlled by the Status Monitor Dispatch.

The Module ignores any Host response to this message, and the Module will not attempt resends of this message.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: None

Response Timeout: NA

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 116

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 116 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 19 |
| 7 | TunerStatus | uint8 | See below |
| 8-9 | ReferenceFreqOffset | int16 | See below |
| 10-11 | RSSI | int16 | See below |
| 12 | HBSignalStrength | uint8 | See below |
| 13 | HBLockStatusA | uint8 | See below |
| 14 | HBLockStatusB | uint8 | See below |
| 15 | HBOLockStatus | uint8 | See below |
| 16-17 | HBBERSat1A | uint16 | See below |
| 18-19 | HBBERSat2A | uint16 | See below |
| 20-21 | HBBERTerrA | uint16 | See below |
| 22-23 | HBBERSat1B | uint16 | See below |
| 24-25 | HBBERSat2B | uint16 | See below |
| 26-27 | HBBERTerrB | uint16 | See below |
| 28-29 | HBOBERSat1A | uint16 | See below |
| 30-31 | HBOBERSat2A | uint16 | See below |
| 32-33 | HBOBERTerrA | uint16 | See below |
| 34-35 | HBOBERSat1B | uint16 | See below |
| 36-37 | HBOBERSat2B | uint16 | See below |
| 38-39 | HBOBERTerrB | uint16 | See below |
| 40 | HBCNSat1A | uint8 | See below |
| 41 | HBCNSat2A | uint8 | See below |
| 42 | HBCNTerrA | uint8 | See below |
| 43 | HBCNSat1B | uint8 | See below |
| 44 | HBCNSat2B | uint8 | See below |
| 45 | HBCNTerrB | uint8 | See below |
| 46-47 | HBRSWordErrorRate | uint16 | See below |
| 48-49 | HBRSWordErrorRateA | uint16 | See below |
| 50-51 | HBRSWordErrorRateSatA | uint16 | See below |
| 52-53 | HBRSWordErrorRateTerrA | uint16 | See below |
| 54-55 | HBRSWordErrorRateB | uint16 | See below |
| 56-57 | HBRSWordErrorRateSatB | uint16 | See below |
| 58-59 | HBRSWordErrorRateTerrB | uint16 | See below |
| 60-61 | HBTurboWordErrorRatePipe0A | uint16 | See below |
| 62-63 | HBTurboWordErrorRatePipe1A | uint16 | See below |
| 64-65 | HBTurboWordErrorRatePipe2A | uint16 | See below |
| 66-67 | HBTurboWordErrorRatePipe3A | uint16 | See below |
| 68-69 | HBTurboWordErrorRatePipe0B | uint16 | See below |
| 70-71 | HBTurboWordErrorRatePipe1B | uint16 | See below |
| 72-73 | HBTurboWordErrorRatePipe2B | uint16 | See below |
| 74-75 | HBTurboWordErrorRatePipe3B | uint16 | See below |
| 76 | LBSignalStrength | uint8 | See below |
| 77 | LBLockStatus | uint8 | See below |
| 78 | LBOLockStatus | uint8 | See below |
| 79-80 | LBBERSat1 | uint16 | See below |
| 81-82 | LBBERSat2 | uint16 | See below |
| 83-84 | LBBERTerr | uint16 | See below |
| 85-86 | LBOBERPipe | uint16 | See below |
| 87 | LBCNSat1 | uint8 | See below |
| 88 | LBCNSat2 | uint8 | See below |
| 89 | LBCNTerr | uint8 | See below |
| 90-91 | LBRSWordErrorRate | uint16 | See below |
| 92-93 | LBTurboWordErrorRatePipe0 | uint16 | See below |
| 94-95 | LBTurboWordErrorRatePipe1 | uint16 | See below |
| 96-97 | LBTurboWordErrorRatePipe2 | uint16 | See below |
| 98-99 | LBTurboWordErrorRatePipe3 | uint16 | See below |
| 100-101 | HBTurboWordErrorRatePipe4A | uint16 | See below |
| 102-103 | HBTurboWordErrorRatePipe5A | uint16 | See below |
| 104-105 | HBTurboWordErrorRatePipe6A | uint16 | See below |
| 106-107 | HBTurboWordErrorRatePipe7A | uint16 | See below |
| 108-109 | HBTurboWordErrorRatePipe4B | uint16 | See below |
| 110-111 | HBTurboWordErrorRatePipe5B | uint16 | See below |
| 112-113 | HBTurboWordErrorRatePipe6B | uint16 | See below |
| 114-115 | HBTurboWordErrorRatePipe7B | uint16 | See below |

**TunerStatus**: Reports the Tuner’s status, with bit fields defined as:

| **Bit 7** | **Bits 6** | **Bit 5** | **Bit 4** | **Bits 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Reserved | PulseBlocker | IFAGC | RFAGC | AntOver | AntUnder | AntDetect | PLLLock |

PulseBlocker: Pulse blocker detection status, with values:

0 = Not detected

1 = Detected

IFAGC: IF AGC engaged status, with values:

0 = Not engaged

1 = Engaged

RFAGC: RF AGC engaged status, with values:

0 = Not engaged

1 = Engaged

AntOver: Antenna overcurrent status, with values:

0 = Not overcurrent

1 = Overcurrent

AntUnder: Antenna undercurrent status, with values:

0 = Not undercurrent

1 = Undercurrent

AntDetect: Antenna detected status, with values:

0 = Not detected

1 = Detected

PLLLock: PLL Lock status, with values:

0 = Not locked

1 = Locked

**ReferenceFreqOffset**: Tuner carrier frequency offset. LSB=100 Hz. The valid values are -1400-1400.

**RSSI**: Received signal strength indication. LSB=1 dBm. The valid values are -128-0.

**HBSignalStrength**: Summary of High Band signal strength. The valid values are:

0 = No signal

1 = Very weak signal

2 = Weak signal

3 = Marginal signal

4 = Good signal

**HBLockStatusA**: Reports the High Band Ensemble A lock status, with bit fields defined as:

| **Bits 7-6** | **Bit 5** | **Bit 4** | **Bits 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- |
| Reserved | S1TDM | S1QPSK | S2TDM | S2QPSK | TerrTDM | TerrMCM |

S1TDM: Satellite 1 TDM lock status, with values:

0 = Not locked

1 = Locked

S1QPSK: Satellite 1 QPSK lock status, with values:

0 = Not locked

1 = Locked

S2TDM: Satellite 2 TDM lock status, with values:

0 = Not locked

1 = Locked

S2QPSK: Satellite 2 QPSK lock status, with values:

0 = Not locked

1 = Locked

TerrTDM: Terrestrial TDM lock status, with values:

0 = Not locked

1 = Locked

TerrMCM: Terrestrial MCM lock status, with values:

0 = Not locked

1 = Locked

**HBLockStatusB**: Reports the High Band Ensemble B lock status, with bit fields defined as:

| **Bits 7-6** | **Bit 5** | **Bit 4** | **Bits 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- |
| Reserved | S1TDM | S1QPSK | S2TDM | S2QPSK | TerrTDM | TerrMCM |

S1TDM: Satellite 1 TDM lock status, with values:

0 = Not locked

1 = Locked

S1QPSK: Satellite 1 QPSK lock status, with values:

0 = Not locked

1 = Locked

S2TDM: Satellite 2 TDM lock status, with values:

0 = Not locked

1 = Locked

S2QPSK: Satellite 2 QPSK lock status, with values:

0 = Not locked

1 = Locked

TerrTDM: Terrestrial TDM lock status, with values:

0 = Not locked

1 = Locked

TerrMCM: Terrestrial MCM lock status, with values:

0 = Not locked

1 = Locked

**HBOLockStatus**: Reports the High Band Overlay lock status, with bit fields defined as:

| **Bits 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bits 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PipeA | S1ATDM | S2ATDM | TATDM | PipeB | S1BTDM | S2BTDM | TBTDM |

PipeA: Pipe A lock status, with values:

0 = Not locked

1 = Locked

S1ATDM: Satellite 1 pipe A TDM lock status, with values:

0 = Not locked

1 = Locked

S2ATDM: Satellite 2 pipe A TDM lock status, with values:

0 = Not locked

1 = Locked

TATDM: Terrestrial pipe A TDM lock status, with values:

0 = Not locked

1 = Locked

PipeB: Pipe B lock status, with values:

0 = Not locked

1 = Locked

S1BTDM: Satellite 1 pipe B TDM lock status, with values:

0 = Not locked

1 = Locked

S2BTDM: Satellite 2 pipe B TDM lock status, with values:

0 = Not locked

1 = Locked

TBTDM: Terrestrial pipe B TDM lock status, with values:

0 = Not locked

1 = Locked

**HBBERSat1A, HBBERSat2A, HBBERTerrA, HBBERSat1B, HBBERSat2B, HBBERTerrB**: Satellite 1, satellite 2, and terrestrial bit-error rate (BER) measurement for High Band Base Layer ensembles A and B. LSB=0.1%. The valid values are 0-1000.

**HBOBERSat1A, HBOBERSat2A, HBOBERTerrA, HBOBERSat1B, HBOBERSat2B, HBOBERTerrB**: Satellite 1, satellite 2, and terrestrial bit-error rate (BER) measurement for High Band Overlay ensembles A and B. LSB=0.1%. The valid values are 0-1000.

**HBCNSat1A, HBCNSat2A, HBCNTerrA, HBCNSat1B, HBCNSat2B, HBCNTerrB**: Satellite 1, satellite 2, and terrestrial carrier/noise ratio for High Band ensembles A and B. LSB=0.25dB. The valid values are 0-255.

**HBRSWordErrorRate**: The per frame Reed-Solomon (RS) code word error rate averaged over both High Band ensembles and branches. LSB=0.1%. The valid values are 0-1000.

**HBRSWordErrRateA, HBRSWordErrRateB**: The per frame Reed-Solomon (RS) code word error rate on High Band ensembles A and B. LSB=0.1%. The valid values are 0-1000.

**HBRSWordErrRateSatA, HBRSWordErrRateTerrA, HBRSWordErrRateSatB, HBRSWordErrRateTerrB**: The per frame and per branch Reed-Solomon (RS) code word error rate for satellite and terrestrial branches on High Band ensembles A and B. LSB=0.1%. The valid values are 0-1000.

**HBTurboWordErrorRatePipe0A, HBTurboWordErrorRatePipe1A, HBTurboWordErrorRatePipe2A, HBTurboWordErrorRatePipe3A, HBTurboWordErrorRatePipe0B, HBTurboWordErrorRatePipe1B, HBTurboWordErrorRatePipe2B, HBTurboWordErrorRatePipe3B:** The per frame rate of uncorrectable Turbo words for High Band Overlay pipes 0, 1, 2, 3, and 4 on ensembles A and B. LSB=0.1%. The valid values are 0-1000, with all other values meaning the pipe is inactive.

**LBSignalStrength**: Summary of Low Band signal strength. The valid values are:

0 = No signal

1 = Very weak signal

2 = Weak signal

3 = Marginal signal

4 = Good signal

**LBLockStatus**: Reports the Low Band lock status, with bit fields defined as:

| **Bits 7-6** | **Bit 5** | **Bit 4** | **Bits 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- |
| Reserved | S1TDM | S1QPSK | S2TDM | S2QPSK | TerrTDM | TerrMCM |

S1TDM: Satellite 1 TDM lock status, with values:

0 = Not locked

1 = Locked

S1QPSK: Satellite 1 QPSK lock status, with values:

0 = Not locked

1 = Locked

S2TDM: Satellite 2 TDM lock status, with values:

0 = Not locked

1 = Locked

S2QPSK: Satellite 2 QPSK lock status, with values:

0 = Not locked

1 = Locked

TerrTDM: Terrestrial TDM lock status, with values:

0 = Not locked

1 = Locked

TerrMCM: Terrestrial MCM lock status, with values:

0 = Not locked

1 = Locked

**OLockStatus**: Reports the Low Band Overlay lock status, with bit fields defined as:

| **Bits 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bits 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Reserved | Reserved | Reserved | Reserved | Pipe | S1TDM | S2TDM | TTDM |

Pipe: Pipe lock status, with values:

0 = Not locked

1 = Locked

S1TDM: Satellite 1 pipe TDM lock status, with values:

0 = Not locked

1 = Locked

S2TDM: Satellite 2 pipe TDM lock status, with values:

0 = Not locked

1 = Locked

TTDM: Terrestrial pipe TDM lock status, with values:

0 = Not locked

1 = Locked

**LBBERSat1, LBBERSat2, LBBERTerr**: Bit-error rate (BER) measurement for satellite 1, satellite 2, and terrestrial for Low Band Base Layer when OpMode=0, or Full Band Replacment when OpMode=2. LSB=0.1%. The valid values are 0-1000.

**LBOBERPipe**: Bit-error rate (BER) measurement for the signaling pipe (i.e., pipe 4) for Low Band Overlay when OpMode=0 or for Full Band Replacement when OpMode=2. LSB=0.1%. The valid values are 0-1000.

**LBCNSat1, LBCNSat2, LBCNTerr**: Carrier/noise ratio for Low Band satellite 1, satellite 2, and terrestrial. LSB=0.25dB. The valid values are 0-255.

**LBRSWordErrorRate**: The averaged Reed-Solomon (RS) word error rate for all Low Band clusters. LSB=0.1%. The valid values are 0-1000. This value is valid only when OpMode=0.

**LBTurboWordErrorRatePipe0, LBTurboWordErrorRatePipe1, LBTurboWordErrorRatePipe2, LBTurboWordErrorRatePipe3**: The averaged pipes 0, 1, 2 and 3 Turbo word error rate for Low Band Overlay when OpMode=0 or for Full Band Replacement when OpMode=2. LSB=0.1%. The valid values are 0-1000, with all other values meaning the pipe is inactive.

**HBTurboWordErrorRatePipe4A, HBTurboWordErrorRatePipe5A, HBTurboWordErrorRatePipe6A, HBTurboWordErrorRatePipe7A, HBTurboWordErrorRatePipe4B, HBTurboWordErrorRatePipe5B, HBTurboWordErrorRatePipe6B, HBTurboWordErrorRatePipe7B**: The per frame rate of uncorrectable Turbo words for High Band Overlay pipes 4, 5, 6 and 7 on ensembles A and B. LSB=0.1%. The valid values are 0-1000, with all other values meaning the pipe is inactive.

# Audio Services Management Messages

## Audio Select Dispatch

The Host sends the Audio Select Dispatch to select the audio service whose audio the Module will output on the I2S interface. Any other audio services that were in progress will be layered or terminated, as appropriate. See section 2 for an overview of the various audio services and their layering.

If audio playback is currently paused as a result of the Host sending a Playback Control Dispatch to the Module with a pause command or a jump to time offset command, the Module will automatically unpause and start playing in the newly selected mode.

If the audio is currently muted as a result of the Host sending an Audio Mute Dispatch to the Module, the Module will perform the requested audio service change but will not unmute the audio. The Host must explicitly unmute the audio.

When the Module receives an Audio Select Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Audio Select Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Audio Select Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 19

The **AlertPB** and **AlertPBVolAdj** fields were late additions to this message. Hosts using SXi8 versions earlier than v2.0 will transmit this message without these fields and the message will have a length of 19 bytes.

The **AlertPB** and **AlertPBVolAdj** fields were removed from this message. Modules using SXi8 versions later than v2.2 will ignore these fields if they are present.

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 19 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 32 |
| 7 | Command | uint8 | See below |
| 8-9 | SID | uint16 | See below |
| 10 | ExtCommand | uint8 | See below |
| 11 | AlertID | uint8 | See below |
| 12-13 | SportsFlashEventID | uint16 | See below |
| 14 | ToneFrequency | uint8 | See below |
| 15 | ToneOptions | uint8 | See below |
| 16-17 | MRef | uint16 | See below |
| 18 | VolumeAdj | int8 | See below |

**Command**: Selects the audio service that will be the source of the Module’s audio output. The valid values are:

0 = Discontinue the current audio service (Alert, SportsFlash, Tune Scanning, or Tone) and resume the underlying audio service

1 = Begin Sequential playback using the Module’s Tune Start rules, with the channel being selected by **SID.** Related action to be taken on any pending Ad Insertion or Ad Replacement is specified by **ExtCommand**.

2 = Begin TuneMix playback

3 = Begin a single Alert playback, with the Alert to be played being selected by **AlertID**

5 = Begin SportsFlash playback, with the SportsFlash event being selected by **SportsFlashEventID**

6 = Discontinue SportsFlash playback, but continue on the current channel at the current audio point as Sequential playback

7 = Begin Tune Scanning of Music Only Smart Favorite audio tracks

10 = Discontinue Tune Scanning, but continue on the current channel at the current audio point as Sequential playback

11 = Begin or continue Tone playback, with the tone’s frequency specified by **ToneFrequency** and the tone’s options specified by **ToneOptions**.

12 = Begin Sequential playback at the specified track, with the channel being selected by **SID** and the track selected by **SID** and **MRef**. Related action to be taken on any pending Ad Insertion or Ad Replacement is specified by **ExtCommand**.

13 = Begin playback of the exposed TuneMix track when TuneMix is currently playing or overlayed, specified by **SID** and **MRef**.

14 = Begin Host Audio Playback

**SID**: Specifies the SID to begin playing. This field is only valid when **Command** = 1, 12, or 13, otherwise the Module ignores it. See Section 4.8 for more information.

**ExtCommand**: Bitfield of options applied to the selected **Command**.

**Retune** field applies to **Command** = 1 or 12, otherwise the Module ignores it.

**RemainInPause**  field applies to **Command** = 3, otherwise the Module ignores it.

|  |  |  |
| --- | --- | --- |
| **Bits 7-6** | **Bit 5** | **Bits 4-0** |
| Retune | RemainInPause | Reserved |

Retune: Controls what action the Module should take as part of the Sequential audio selection (i.e., Command=1 or 12 and a new SID is selected) if an Ad Insertion or Ad Replacement is pending. If the channel tune is to a Layer 2 SID, then any Ad Insertion or Ad Replacement is automatically cancelled and Retune is ignored. The valid values are:

0 = cancel the scheduled Ad Insertion or Ad Replacement

1 = insert the Ad Alerts’ audio at the start of the audio to be played for the new channel (only valid for a pending Ad Insertion – i.e., not valid for a pending Ad Replacement)

2 = begin playing the new channel and insert/replace the audio at the next insertion/replacement   
point for that channel, following established insertion/replacement rules

RemainInPause: Controls if the underlying stream, overlayed by alert playback, will remain paused after alert playback is completed. Any audio service selection initiated before the conclusion of alert playback, including Discontinue, will reset ‘remain in pause’ and module will resume playing the underlying audio service.  
 0 = Resume playback of the underlying stream which was active before alert playback   
 started.  
 1 = Do not resume playback of underlying stream if it was paused before alert playback was   
 started.

**AlertID**: Specifies the Alert audio segment to begin playing. This field is only valid when **Command** = 3, otherwise the Module ignores it. The valid values are 1-2 for Module-defined Alerts 1 and 2, and 3-16 for Host-defined Alerts 1-14.

**SportsFlashEventID**: A unique handle assigned to a SportsFlash event for communicating that event’s status to the Host and allowing the Host to select the desired event for playback. This field is only valid when **Command** = 5, otherwise the Module ignores it.

**ToneFrequency**: Sets the Tone frequency in 100 Hz steps. This field is only valid when **Command** = 11, otherwise the Module ignores it. The valid values are 1 to 200, corresponding to a frequency range of 100 Hz to 20 kHz.

**ToneOptions**: Bitfield of options applied to the Module tone generator. This field is only valid when **Command** = 11, otherwise the Module ignores it.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| Reserved | | | | | | Left | Right |

Left: Controls whether the Tone is generated for the left audio channel. The valid values are 0=disable audio or 1=enable audio.

Right: Controls whether the Tone is generated for the right audio channel. The valid values are 0=disable audio or 1=enable audio.

**MRef**: Metadata Reference. Contains the following parameters defined in the Metadata Reference Block Structure of the BIC Metadata Reference Message in [2]**Error! Reference source not found.**: Transition ID, Hour, RFU, and Type. The parameters are ordered as shown below. This field is only valid when **Command** = 12 or 13, otherwise the Module ignores it.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bits 15-14** | **Bits 13-12** | **Bit 11** | **Bits 10-9** | **Bits 8-0** |
| Type | Reserved | RFU | Hour | TransitionID |

**VolumeAdj**: Provides an adjustment to the base audio volume set by the Host. This volume adjustment will apply as long as the current audio service is playing. This field is only valid when **Command** = 3 or 11 or 14, otherwise the Module ignores it. Positive values increase the audio volume up to the maximum possible value specified in the Audio Volume Dispatch (i.e., 24 dB), and negative values decrease the audio volume to the minimum possible value specified in the Audio Volume Dispatch (i.e., -95 dB). LSB = 1dB. The range of values is -20 thru 20.

## Audio Select Error Response

This message can be sent in response to an Audio Select Dispatch. See the description of the Audio Select Dispatch for processing details.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 33 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = **Command** is outside its allowed range

1 = **SID** is outside its allowed range

2 = requested **SID** is unavailable for selection

4 = TuneMix does not have sufficient available component channels

5 = **AlertID** is outside its allowed range

6 = audio for the requested Alert(s) has not been fully downloaded to the Module

8 = requested **SportsFlashEventID** is unknown

9 = **ToneFrequency** is outside its allowed range

10 = **VolumeAdj** is outside its allowed range

11 = insufficient audio content available for scanning

12 = cannot discontinue the currently playing audio service

13 = cannot select track playback of audio that is not continuous

16 = Sportsflash is not the active audio service, so it cannot be discontinued

17 = TuneScan is not the active audio service, so it cannot be discontinued

18 = TuneMix is currently playing, so it cannot be reselected

19 = TuneScan is already active, so it cannot be reselected

20 = TuneMix is not active or overlayed, so cannot jump to the requested **SID**/**MRef**

21 = requested **SID**/**MRef** is not an exposed TuneMix track

24 = Retune field within **ExtCommand** is outside its allowed range

25 = Invalid selection while in FAST mode (see Smart Favorites Configure Dispatch)

26 = Host Audio service is already active

27 = Host Audio data is invalid

28 = SportsFlash feature is not supported

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Now Playing Dispatch

The Module sends the Now Playing Dispatch to report the status of the currently playing audio to the Host, providing the Host the information it needs for displaying the corresponding metadata and audio track and buffer state. The Module shall send this message once at startup, that is, after the Module Configure Response is sent to the Host, and thereafter it will send it:

* Immediately if the **ServiceState**, **SID**, **CutMRef**, **SegmentMRef**, or **ProgramMRef** changes.
* A maximum of once per second if any of the other statuses in the message changes.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Now Playing Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 24

The **AdID** field was a late additions to this message. Module FW using SXi8 versions earlier than v2.0 will transmit this message without this field and the message will have a length of 27 bytes.

The **HostPlayTime** and **HostFreeBytes** fields were late additions to this message. Module FW using SXi8 v2.0 or later, but earlier than v2.6, will transmit this message without these fields and the message will have a length of 31 bytes.

Maximum **NumMsgBytes**: 95

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 37 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 55 |
| 7 | ServiceState | uint8 | See below |
| 8-9 | SID | uint16 | See below |
| 10-11 | CutMRef | uint16 | See below |
| 12-13 | SegmentMRef | uint16 | See below |
| 14-15 | ProgramMRef | uint16 | See below |
| 16-17 | DurationOfTrack | uint16 | See below |
| 18-19 | TimeFromStartOfTrack | uint16 | See below |
| 20 | TracksRemaining | uint8 | See below |
| 21-22 | TimeRemaining | uint16 | See below |
| 23-24 | TimeBefore | uint16 | See below |
| 25-26 | ActionMRef | uint16 | See below |
| 27-30 | AdID | uint32 | See below |
| 31-34 | HostPlayingByte | uint32 | See below |
| 35-36 | HostFreeBytes | uint16 | See below |

**ServiceState**: Bitfield of status regarding the current audio service and its state.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| State | | | Service | | | | |

Service: Indicates which audio service is currently playing:

0 = no audio service has been selected

1 = Sequential

3 = TuneMix

4 = SportsFlash

5 = Tune Scan

6 = Alert

7 = Tone

8 = Sequential Ad Alert Sequence

9 = Host Playback

State: Indicates the playback state of the audio:

0 = no audio service has been selected

1 = playing from the Live point

2 = playing from stored audio

3 = paused

4 = awaiting sufficient Host Audio to begin Host Audio Playback

5 = Host Audio underflow after Host Audio Playback had begun

6 = erred Host Audio encountered, Host should restart Host Audio stream

**SID**: For **Service** = 1, 3, 4, 5, and 8 this is the SID from which the audio content was received. See Section 4.8. For **Service** = 6, this is the **AlertID** of the Alert being played, with valid values as defined in the Audio Select Dispatch. For **Service** = 0, 7, and 9 this field is set to 0.

**CutMRef**: Bit field delivering the Cut MRef. See bitfield format below. For **Service** = 0, 6, 7, 8, and 9 this field is set to 0.

**SegmentMRef**: Bit field delivering the Segment MRef. See bitfield format below. For **Service** = 0, 6, 7, 8, and 9 this field is set to 0.

**ProgramMRef**: Bit field delivering the Program MRef. See bitfield format below. For **Service** = 0, 6, 7, 8, and 9 this field is set to 0.

**ActionMRef**: Bit field delivering the Action MRef. See bitfield format below. For **Service** = 0, 6, 7, 8, and 9 this field is set to 0.

Bitfield format for **CutMRef**, **SegmentMRef**, **ProgramMRef** and **ActionMRef**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit 15** | **Bits 14-12** | **Bit 11** | **Bits 10-9** | **Bits 8-0** |
| MRefValid | Reserved | RFU | Hour | TransitionID |

TransitionID, Hour, RFU: These parameters are defined in the Metadata Reference Block Structure of the BIC Metadata Reference Message in [2].

MRefValid: Indicates if the MRef is valid (i.e., whether or not the Module has received the corresponding MRef via the SiriusXM broadcast). The valid values are:

0 = invalid

1 = valid

**DurationOfTrack**: Duration of the currently playing track. LSB=1 second. Valid range is 0 through 65535. For **Service** = 0, 7, and 9 this field is set to 0.

**TimeFromStartOfTrack**: Time that has elapsed from the start of the currently playing track. LSB= 1 second. Valid range is 0 through 65535. For **Service** = 0, 7, and 9 this field is set to 0.

**TracksRemaining**: Indicates the number of tracks remaining in the audio buffer being played, not including the current track. E.g., a value of 0 would indicate the most recent track is being played. Valid range is 0 thru 50, with 50 indicating there are 50 or more tracks remaining. For **Service** = 0, 5, 6, 7, 8, and 9 this field is set to 0.

**TimeRemaining**: Indicates the total number of seconds of stored content remaining in the audio buffer being played. This includes time remaining in the currently playing track as well as all tracks recorded after the current track. LSB=1 second. Valid range is 0 through 3600. For **Service** = 0, 5, 7, and 9 this field is set to 0.

**TimeBefore**: Indicates the total number of seconds of stored content in the audio buffer prior to the current play point. This includes time from the start of the currently playing track to the current play point as well as all tracks recorded before the current track. LSB=1 second. Valid range is 0 through 3600. For **Service** = 0, 5, 7, and 9 this field is set to 0.

**AdID**: A global ID, provided by the Host in the Alert Packet Dispatch for Host-defined Alerts and set to 1 or 2 for the two Module-defined Alerts, to identify the Ad corresponding to the Alert being played back. For **Service** not equal to 8, this field is set to 0.

**HostPlayingByte**: The byte number corresponding to the currently playing Host audio (i.e., the last successfully decoded Host audio), as measured from the first byte of audio sent from the Host in the most recent Host Audio Control Dispatch with Control=0. Valid range is 0 through 0xFFFFFFFF. For **Service** not equal to 9, this field is set to 0.

**HostFreeBytes**: The number of free bytes in the Module’s Host audio buffer. This is how many additional bytes of audio the Host can send to the Module without overflowing its audio buffer. LSB=1024 bytes. Valid range is 0 through 512. For **Service** not equal to 9, this field is set to 0.

## Playback Control Dispatch

The Host sends the Playback Control Dispatch to the Module to navigate playback of audio within the context of the current audio service. The possible audio services are:

* Sequential (includes Sequential Ad Alert Sequence)
* TuneMix
* SportsFlash
* Tune Scanning
* Alert
* Tone
* Host Playback

The validity and behavior of each audio playback control can vary for each audio service. The differences are described below.

When the Module receives a Playback Control Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Playback Control Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Playback Control Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 10

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 10 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 72 |
| 7 | Control | uint8 | See below |
| 8-9 | TimeOffset | int16 | See below |

**Control** : Selects the audio navigation action to take for the currently playing audio service. Each value of **Control** is only valid for the audio services listed below. The Module will report an error for **Control** values that are invalid for the current audio service. The valid values for each indicated audio service are:

0 = **Pause**:

*Sequential, TuneMix, SportsFlash, Alert, Host Playback*: Pause audio playback at the current play point.

1 = **Play**:

*Sequential, TuneMix, SportsFlash, Alert, Host Playback*: Resume audio playback from the paused play point. This command is sent after a navigation command that has caused audio playback to be paused (i.e., Pause or Jump To TimeOffset).

2 = **Jump To Time Offset**:

*Sequential, TuneMix, SportsFlash, Alert*: Jump **TimeOffset** seconds from the current position in the stored playback content and pause there. If the new play point would be before the start of the stored playback content then jump to the start of the stored content. If the new play point would be after the end of the stored content then treat this as a **Live** command and perform the appropriate processing for the current audio service. The Host implements a **Live** command by using the Jump To Time Offset command with a large, positive time offset (e.g., 32767).

3 = **Jump To Next Track**:

*Sequential, SportsFlash, TuneMix*: Jump to the next track boundary in the stored playback content and pause there. If there is no next track boundary then treat this as a **Live** command and perform the appropriate processing for the current audio service.

*Tune Scanning*: Discontinue playing the currently playing scan track, begin playing the next scan track, and continue scanning.

4 = **Jump To Previous Track**:

*Sequential, SportsFlash, TuneMix*: Jump to the previous track boundary in the stored playback content and pause there. If this command is received less than 3 seconds into playing the current track the Module should jump to the track boundary prior to the current track. If there is no previous track boundary then jump to the start of the stored content. If the previous track duration is less than 3 seconds the Module should again jump to the previous track and iterate this check (i.e., in case there are consecutive short tracks) until it gets to a track with duration greater than 3 seconds or the start of the stored content.

**TimeOffset**: When the command is Jump By Time Offset (i.e., **Control** = 2) this field provides the time offset in seconds to be applied from the current audio play point. Negative values cause a jump backwards in time and positive values cause a jump forward. This field is only valid when **Control** = 2, otherwise the Module ignores it. The valid values are -32768 thru 32767.

## Playback Control Error Response

This message can be sent in response to a Playback Control Dispatch. See the description of the Playback Control Dispatch for processing details.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 73 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = **Control** is outside its allowed range

1 = **Control** is invalid for the current audio service

2 = audio playback is already paused; pause command ignored

3 = audio playback is already in progress; play command ignored

4 = audio already being played from start of content; jump backward command ignored

5 = audio already being played from Live; jump forward command ignored

6 = Jump To Time Offset of 0 ignored

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Smart Favorites Configure Dispatch

The Host sends the Smart Favorites Configure Dispatch to the Module to setup the Smart Favorites and TuneMix features of the Module. The setup includes selecting which channels are requested to be treated as Smart Favorites and as TuneMix component channels. The Module buffers a certain amount of audio for each Smart Favorites channel, providing the following capabilities that normal channels do not have:

* In general, upon selecting a Smart Favorite channel the Module begins playing audio from the start of the most recent track on that channel (i.e., the TuneStart feature).
* Upon selecting a Smart Favorite channel the user can rewind into the channel’s buffered audio to play previously received audio.
* The stored tracks for Smart Favorites can be scanned using the Tune Scan feature, allowing the user to select for playback any stored Smart Favorite track.
* The stored tracks for Smart Favorites can be used for TuneMix playback.

In the Smart Favorites Configure Dispatch the Host provides **SIDList**, a list of requested Smart Favorites channels, to the Module. The Module will make Smart Favorites of all valid channels in **SIDList**. Entries in **SIDList** that are not tunable or otherwise invalid (e.g., channel 0, channel 1, data channels, unauthorized channels, unavailable channels, etc.) will be skipped.

If a current Smart Favorite channel becomes invalid (e.g., its free-to-air trial ends), the Module remove that Smart Favorite. If a channel in **SIDList** that was invalid and had been skipped becomes valid (e.g., it becomes subscribed), the Module make that channel a Smart Favorite.

The Host can change its list of Smart Favorites by sending the Smart Favorites Configure Dispatch to the Module with a new **SIDList**. The new **SIDList** replaces the old **SIDList**. Any buffered content is retained for a channel that was in the old **SIDList** and is in the new **SIDList**. Any buffered content is deleted for a channel that was in the old **SIDList** but is not in the new **SIDList**. The Host cannot update the list of Smart Favorites while Tune Scan is in progress.

The Module does not store the list of Smart Favorites in its NVM. The Host is required to store the list of Smart Favorites in its NVM and to resend the Smart Favorites Configure Dispatch upon each Module initialization.

When the Module receives a Smart Favorites Configure Dispatch it performs any processing associated with that command and sends the Generic Response or the Smart Favorites Configure Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Smart Favorites Configure Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 55

The **Options** field was a late additions to this message. Hosts using SXi8 versions earlier than v2.1 will transmit this message without this field and the message will have a length of 55 bytes.

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 67 |
| 7-54 | SIDList[24] | uint16 | See below |
| 55 | Options | uint8 | See below |

**SIDList**: A list of candidate Smart Favorite and TuneMix SID’s. The Host should set all unused entries to 0, and can effectively disable Smart Favorites and TuneMix by setting all entries in **SIDList** to 0. The bit fields are defined as:

| **Bit 15** | **Bit 14** | **Bits 13-0** |
| --- | --- | --- |
| TuneMixIt | Reserved | SID |

SID: Component channel’s SID. The valid values are 0-1023.

TuneMixIt: Selects if this channel should be included when TuneMix is played, with values:

0 = do not include this channel in TuneMix

1 = include this channel in TuneMix (if is a valid TuneMixable component channel)

**Options**: Smart Favorite configuration options. The bit fields are defined as:

| **Bits 7-2** | **Bit 1** | **Bit 0** |
| --- | --- | --- |
| Reserved | PlayPoint | EnableFAST |

EnableFAST: Selects if the Smart Favorites should be treated as Free Ad-Supported Tier (FAST) channels, with values:

0 = do not enable FAST behaviors (default)

1 = enable FAST behaviors

PlayPoint: Disables the TuneStart feature:

0 = do not disable TuneStart (default)

1 = disable TuneStart

## Smart Favorites Configure Error Response

When the Module receives a Smart Favorites Configure Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Smart Favorites Configure Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 93 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = one or more entries in **SIDList** is outside its valid range.

1 = one or more entries in **SIDList** is not a valid Smart Favorites SID.

2 = cannot update Smart Favorites while Tune Scan is in progress

3 = one or more channels selected for TuneMix is not a valid TuneMix SID

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Sequential Content Dispatch

The Module sends Sequential Content Dispatches to the Host to provide it with a list of all audio cuts the Module has stored for the Live channel, Smart Favorite channels, and SportsFlash channels. From this list the Host can:

* Determine which metadata, album art, etc., it needs to retain
* Present a list of the navigable audio content to the User
* Present a list of advertisements that have been inserted into the audio buffer through Ad Insertion or Ad Replacement

On startup the Module does not have any audio stored. The Module will send a Sequential Content Dispatch for a relevant SID each time a cut with a valid MRef from that SID is added to or deleted from the Module’s audio storage.

The cuts in the Sequential Content Dispatch are listed in sequential order, from oldest to newest. Only the MRef for the start of each cut is provided. Specifically, if there are multiple MRefs for a cut then only the first one is provided. The Host should retain subsequent MRef database entries up to the next sequential navigable MRef.

A CutInfo entry will refer to an MRef or an advertisement.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Sequential Content Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 10

Maximum **NumMsgBytes**: 1200

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | Varies |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 89 |
| 7-8 | SID | uint16 | See Section 4.8 |
| 9 | NumCuts | uint8 | See below |
| 10-9+  2\*NumCuts | CutInfo[NumCuts] | uint16 | See below |
| 10+  2\*NumCuts  –  9+2\*NumCuts + 4\*NumAds | AlertID[NumAdPresent] | uint32 | See below |

**NumCuts**: Number of cuts listed in **CutInfo**. A value of 0 indicates that no cuts are currently stored for the specified **SID**. The valid values are 0 thru 120. The Module stores a maximum of 120 cuts for each Sequential channel.

**CutInfo**: Each entry is a bitfield describing a cut stored for this SID. If **NumCuts** then this field is absent from the message.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bit 15** | **Bit 14** | **Bit 13** | **Bit 12** | **Bit 11** | **Bits 10-9** | **Bits 8-0** |
| MRefValid | HaveStart | Reserved | AdPresent | RFU | Hour | TransitionID |

TransitionID, Hour, RFU: These parameters are defined in the Metadata Reference Block Structure of the BIC Metadata Reference Message in [2]**Error! Reference source not found.**.

AdPresent: Indicates that the audio cut is an inserted advertisement as a result of Ad Insertion or Ad Replacement. The host may examine the corresponding **AlertID** entry to retrieve the AdID of the inserted advertisement.

0 = ad not present

1 = ad present

MRefValid: Indicates if the CutMRef is valid (i.e., whether or not the Module has received the corresponding CutMRef via the SiriusXM broadcast). The valid values are:

0 = invalid

1 = valid

MRefValid and AdPresent are mutually exclusive.

HaveStart: Indicates if the start of the cut is available for playback. The valid values are:

0 = start of cut is not available

1 = start of cut is available

**AlertID**: Each entry is the AdID of the inserted advertisement. If **NumCuts** is 0 then this field is absent from the message. If **NumCuts** is not 0 and AdPresent is 0 in the **CutInfo** field for all cuts then this field is absent from the message. NumAdPresent is the count of non-zero AdPresent values across each cut, where the total number of cuts is given by **NumCuts**.

## SportsFlash Configure Dispatch

The Host sends the SportsFlash Configure Dispatch to the Module to setup the SportsFlash feature of the Module. The setup includes selecting which channels are requested to be treated as SportsFlash channels.

The Module buffers a certain amount of audio for SportsFlash channels and monitors those channels for SportsFlash events. When the Module detects a SportsFlash event on a SportsFlash channel, it internally marks the location of the event in the audio buffer and sends a SportsFlash Event Dispatch to inform the Host of the event. The SportsFlash Event Dispatch includes an event identifier that the Host can use in the Audio Select Dispatch to begin playback of the SportsFlash channel at a point prior to the SportsFlash event, allowing the user to hear the audio leading up to the key event.

In the SportsFlash Configure Dispatch the Host provides **SIDList**, a list of requested SportsFlash channels, to the Module. Entries in **SIDList** that are not tunable or otherwise invalid (e.g., channel 0, channel 1, data channels, unauthorized channels, unavailable channels, etc.) will be skipped.

The Host can change its list of SportsFlash channels at any time by sending the SportsFlash Configure Dispatch to the Module with a new **SIDList**. The new **SIDList** replaces the old **SIDList**. Any buffered content is retained for a channel that was in the old **SIDList** and is in the new **SIDList**. Any buffered content is deleted for a channel that was in the old **SIDList** but is not in the new **SIDList**.

The Module does not store the list of SportsFlash channels in its NVM. The Host is required to store the list of SportsFlash channels in its NVM and to resend the SportsFlash Configure Dispatch upon each Module initialization.

When the Module receives a SportsFlash Configure Dispatch it performs any processing associated with that command and sends the Generic Response or the SportsFlash Configure Error Response to the Host.

The SportsFlash feature is deprecated in the X28.1F and X28.1M module types. If an X28.1F or X28.1M module receives a SportsFlash Configure Dispatch the module will return a SportsFlash Configure Error Response.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or SportsFlash Configure Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 17

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 17 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 68 |
| 7-16 | SIDList[5] | uint16 | See below |

**SIDList**: The list of candidate SportsFlash SID’s. The valid values for each entry are 0-1023. A value of 0 indicates the entry is unused, otherwise the entry is a SID to be monitored for SportsFlash events. The Host should set all unused entries to 0, and can effectively disable SportsFlash by setting all entries in **SIDList** to 0.

## SportsFlash Configure Error Response

When the Module receives a SportsFlash Configure Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the SportsFlash Configure Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 92 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = one or more enties in **SIDList** is outside its valid range. Entry ignored.

1 = one or more entries in **SIDList** is not a valid SportsFlash SID.

2 = SportsFlash is currently playing and the playing SID is no longer in **SIDList**. SportsFlash playback aborted.

3 = SportsFlash Feature is not supported

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## SportsFlash Event Dispatch

The Module sends the SportsFlash Event Dispatch to the Host report information about a Module-detected SportsFlash event for the indicated SID as a results of a SportsFlash Monitor previously established thru the SportsFlash Monitor Dispatch. The Module will send the SportsFlash Event Dispatch to the Host initially when the event is first detected, and thereafter whenever any of the fields contained in the message changes.

The SportsFlash feature is deprecated in the X28.1F and X28.1M module types.

The Host sends the Generic Response to the Module to confirm receipt of the SportsFlash Event Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 18

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 18 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 69 |
| 7-8 | SID | uint16 | See Section 4.8 |
| 9-10 | CutMRef | uint16 | See below |
| 11 | Status | uint8 | See below |
| 12-13 | SportsFlashEventID | uint16 | See below |
| 14-17 | EventData | uint32 | See below |

**CutMRef**: Bit field delivering the Cut MRef corresponding to the SportsFlash event. If the **Status** is “event expired”, then CutMRef is set to 0. The field is defined as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit 15** | **Bits 14-12** | **Bit 11** | **Bits 10-9** | **Bits 8-0** |
| MRefValid | Reserved | RFU | Hour | TransitionID |

TransitionID, Hour, RFU: These parameters are defined in the Metadata Reference Block Structure of the BIC Metadata Reference Message in [2].

MRefValid: Indicates if the MRef is valid (i.e., whether or not the Module has received the corresponding MRef via the SiriusXM broadcast). The valid values are:

0 = invalid

1 = valid

**Status**: SportsFlash status for the event being reported. The valid values are:

0 = New SportsFlash event

1 = SportsFlash event has expired

2 = SportsFlash event has finished playing

**SportsFlashEventID**: A unique handle assigned to a SportsFlash event for communicating status to the Host and allowing the Host to select the desired event.

**EventData**: Reports data for the SportsFlash event. Interpretation of this field depends on the type of SportsFlash event reported, and is detailed in document [3]. If the **Status** is “event expired”, then **EventData** is set to 0.

## TuneMix Content Dispatch

The Module sends TuneMix Content Dispatches to the Host to provide it with a list of all the non-Sequential audio cuts the Module has stored for TuneMix. Since all TuneMix future tracks are culled from the Smart Favorites and are identified in the Sequential Content Dispatch, only the exposed TuneMix tracks need to be identified in the TuneMix Content Dispatch. From this list the Host can:

* Determine which metadata, album art, etc., it needs to retain
* Present a list of the navigable TuneMix audio content to the User

The Module will send a TuneMix Content Dispatch to the Host whenever the audio content or the associated status for the non-Sequential TuneMix content changes. On startup the Module does not have audio stored for TuneMix.

Except during a startup condition where TuneMix has nothing else to play, TuneMix will only store cuts for which it has the beginning of the cut.

The cuts in the TuneMix Content Dispatch are listed in sequential play order, from oldest to newest. Only the MRef for the start of each cut is provided. Specifically, if there are multiple MRefs for a cut then only the first one is provided. The Host should retain subsequent MRef database entries up to the next sequential navigable MRef.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the TuneMix Content Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 9

Maximum **NumMsgBytes**: 1200

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | Varies |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 91 |
| 7 | Reserved | uint8 | See Section 4.5 |
| 8 | NumCuts | uint8 | See below |
| 9-(8+  4\*NumCuts) | CutInfo[NumCuts], with each entry defined as:  uint16 SID  uint16 CutData | Structure  (4 bytes) | See below |

**NumCuts**: Number of cuts listed in **CutInfo**. A value of 0 indicates that no exposed cuts are currently stored for TuneMix. The valid values are 0 thru 80.

**CutInfo**: Each entry is a structure with fields describing a non-Sequential cut stored for TuneMix**.** If **NumCuts** is 0 then this field is absent from the message. The constituent fields are:

**SID**: Service ID the audio was captured from. See Section 4.8.

**CutData**: A bitfield describing the cut:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit 15** | **Bits 14-12** | **Bit 11** | **Bits 10-9** | **Bits 8-0** |
| MRefValid | Reserved | RFU | Hour | TransitionID |

TransitionID, Hour, RFU: These parameters are defined in the Metadata Reference Block Structure of the BIC Metadata Reference Message in [2].

MRefValid: Indicates if the CutMRef is valid (i.e., whether or not the Module has received the corresponding CutMRef via the SiriusXM broadcast). The valid values are:

0 = invalid

1 = valid

## Alert Packet Dispatch

Prior to playing back a Host-defined Alert, the Host must download the entire audio clip to the Module. The Host uses the Alert Packet Dispatch to download the Alert audio to the Module. The entire clip must be encoded using the same encoder and bitrate. The encoded audio being downloaded is broken into 1024-byte snippets by the Host. The Host sends the Alert Packet Dispatch containing the first of those snippets to the Module. The Module responds with a Generic Response confirming that snippet has been received. The Host then sends the next snippet, and so on, until the entire clip has been transferred. The Host must send the snippets in sequential order.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Alert Packet Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 1037

The **AdID** field was a late additions to this message. Hosts using SXi8 versions earlier than v2.0 will transmit this message without this field and the message will have a length of 1037 bytes.

Maximum **NumMsgBytes**: 1400

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 1041 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 57 |
| 7 | AlertID | uint8 | See below |
| 8 | Status | uint8 | See below |
| 9 | PacketID | uint8 | See below |
| 10 | Reserved | uint8 | See below |
| 11-12 | NumAudioBytes | uint16 | See below |
| 13-1036 | Audio[1024] | uint8 | See below |
| 1037-1040 | AdID | uint32 | See below |

**AlertID**: Specifies the Alert the audio is being download for. Alerts 1 and 2 are hardcoded in the Module and cannot be overwritten by the Host. Alerts 3 thru 16 are Host-defined and must be downloaded prior to being played back. The valid values are 3-16.

**Status**: Alert status, with bit fields defined as:

| **Bits 7-5** | **Bits 4-2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- |
| Encoder | Reserved | Last | First |

First: Identifies the first packet of the audio clip:

0 = Not first audio packet. **Audio** will be appended to the audio previously loaded for this Alert.

1 = First audio packet. Any previously loaded audio for this Alert will be deleted.

Last: Identifies the last packet of the audio clip:

0 = Not the last audio packet

1 = Last audio packet

Encoder: Identifies the encoder used to encode the audio clip:

0 = Hawaii

1 = AAC+

3 = PAC

4 = MPEG-4 (i.e., AAC encoded with either ADTS or MP4FF transport)

**PacketID**: Number of the audio packet. The first packet is 0 and each subsequent packet is incremented by 1, wrapping back to 0 as necessary. The valid values are 0-255.

**NumAudioBytes**: Number of bytes of valid audio contained in the **Audio** field of this message. For packets other than the last packet this value should be 1024. For the last packet valid values are 1 thru 1024.

**Audio**: Encoded audio. For the last audio packet, the Module will ignore the last (1024 - **NumAudioBytes**) bytes. The overall clip must contain at least 128 bytes of audio.

**AdID**: A global ID provided by the Host to identify the Ad corresponding to this Alert. This value is not used by the Module, but is echoed back to the Host in the Now Playing Dispatch when this Alert is being played. All values except 0 (invalid), 1 (Module-defined alert), and 2 (Module-defined Alert) are valid.

## Alert Packet Error Response

When the Module receives an Alert Packet Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Alert Packet Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 57

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 57 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 58 |
| 7 | ErrorCode | uint8 | See below |
| 8 | PacketID | uint8 | See below |
| 9-56 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = **AlertID** is outside its valid range

1 = the Alert is currently being played, so its audio cannot be updated

2 = the first packet received for this Alert was not marked as being the first packet

4 = Encoder (a component of **Status**) is outside its valid range

5 = the specified encoder does not match the previously specified encoder for this Alert

8 = **NumAudioBytes** is outside its valid range

9 = too few audio bytes in the clip

10 = **NumAudioBytes** for a packet other than the last packet is not 1024

11 = too many audio bytes in the clip; the clip has been limited to its maximum size and all subsequent bytes have been discarded

12 = Packet ignored. The first packet did not have a **PacketID** of 0.

13 = Packet ignored. The **PacketID** was out of sequence. **PacketID’s** must be monotonically increasing.

14 = the specified AdID does not match the previously specified AdID for this Alert

15 = the Alert is currently scheduled for insertion or replacement, so its audio cannot be updated

16 = The AdID is outside its valid range.

17 = MP4FF Alert does not conform to supported MP4FF format

**PacketID**: **PacketID** of the last packet that was successfully processed. If no packet has been successfully processed then **PacketID** is set to 0xFF.

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Ad Configure Dispatch

The Host sends the Ad Configure Dispatch to the Module to configure the insertion and replacement of Ads into the audio stream.

If the Host changes the Ad configuration, then the new configuration will apply to any existing Ads and any future Ads.

The Module does not store the Ad configuration data in its NVM, but does have a default set of configuration data. The Module will use its default Ad configuration data after each Module reset, so the Host should resend the Ad Configure Dispatch to the Module upon each Module initialization.

When the Module receives an Ad Configure Dispatch it performs any processing associated with that command and sends the Generic Response or the Ad Configure Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Ad Configure Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 39

The **AdRepFadeInTime** and **AdRepFadeOutTime** fields were late additions to this message. To retain backward compatibility, the Module must allow for Ad Configure Dispatch to contain only 39 bytes and use default values for the Ad Replacement fade profiles.

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 43 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 157 |
| 7-8 | T0FadeOutTime0 | uint16 | See below |
| 9-10 | T0FadeOutTime1 | uint16 | See below |
| 11-12 | T0FadeOutTime2 | uint16 | See below |
| 13-14 | T0FadeOutTime3 | uint16 | See below |
| 15-16 | T0FadeInTime0 | uint16 | See below |
| 17-18 | T0FadeInTime1 | uint16 | See below |
| 19-20 | T0FadeInTime2 | uint16 | See below |
| 21-22 | T0FadeInTime3 | uint16 | See below |
| 23-24 | T1FadeOutTime0 | uint16 | See below |
| 25-26 | T1FadeOutTime1 | uint16 | See below |
| 27-28 | T1FadeOutTime2 | uint16 | See below |
| 29-30 | T1FadeOutTime3 | uint16 | See below |
| 31-32 | T1FadeInTime0 | uint16 | See below |
| 33-34 | T1FadeInTime1 | uint16 | See below |
| 35-36 | T1FadeInTime2 | uint16 | See below |
| 37-38 | T1FadeInTime3 | uint16 | See below |
| 39-40 | AdRepFadeOutTime | uint16 | See below |
| 41-42 | AdRepFadeInTime | uint16 | See below |

**T0FadeOutTimei** and **T0FadeInTimei**: The fade-out and fade-in times corresponding to Fade Profiles 0-3 in the BIC Ad Splice Message when Type=0 (i.e., Insert). LSB=1 msec. Valid values are 60-360 (i.e., 60 msec to 360 msec).

**T1FadeOutTimei** and **T1FadeInTimei:** The fade-out and fade-in times corresponding to Fade Profiles 0-3 in the BIC Ad Splice Message when Type=1 (i.e., Replace). LSB=1 msec. Valid values are 60-360 (i.e., 60 msec to 360 msec).

**AdRepFadeOutTime:** The fade-out times corresponding to Ad Replacement. LSB=1 msec. Valid values are 60-360 (i.e., 60 msec to 360 msec).

**AdRepFadeInTime:** The fade-in times corresponding to Ad Replacement. LSB=1 msec. Valid values are 60-360 (i.e., 60 msec to 360 msec).

## Ad Configure Error Response

When the Module receives an Ad Configure Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Ad Configure Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 158 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = one or more fade times were outside allowed range; erred fade time(s) were set to min or max limit as appropriate

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Ad Control Dispatch

The Host sends the Ad Control Dispatch to control the scheduling of the insertion or replacement of Ads into the audio stream. Actual Ad insertion timing and its low-level control are dictated by the BIC Ad Splice Messages received for the currently playing channel. Actual Ad replacement timing and its low-level control are dictated by the BIC PAM Messages received for the currently playing channel. An Ad can only be scheduled for insertion or replacement if the Module is currently in Sequential playback and is not playing a Layer 2 SID.

When the Module receives an Ad Control Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Ad Control Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: Generic Response or Ad Control Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 32

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 32 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 149 |
| 7 | AdControl | uint8 | See below |
| 8-19 | AlertList[12] | uint8 | See below |
| 20-31 | VolAdj[12] | int8 | See below |

**AdControl**: Bitfield of options for controlling Ad Insertion and Ad Replacement:

|  |  |
| --- | --- |
| **Bits 7-4** | **Bits 3-0** |
| Reserved | Command |

Command: Ad control command to perform. The valid values are:

0 = Schedule insertion of the list of Ad Alerts, as specified by the **AlertID** and **VolAdj** fields, at the next available insertion point for the currently playing channel. This selection is only valid if the Module is currently in Sequential playback and not playing a Layer 2 SID.

1 = cancel any pending Ad Insertion or Ad Replacement.

2 = Schedule replacement of the list of Ad Alerts, as specified by the **AlertID** and **VolAdj** fields, at the next available replacement point for the currently playing channel. This selection is only valid if the Module is currently in Sequential playback and not playing a Layer 2 SID.

3 = Inject a simulated BIC PAM Type=1 (i.e., End) Message for the currently playing channel that is applicable to its current audio play point. This selection is only intended for testing, and is ignored if an Ad Replacement is not currently in progress.

**AlertList**: The ordered list of Alerts to be used for Ad Insertion or Ad Replacement. The Alerts will be used in the listed order until all Alerts have been used or an AlertID of 0 is found in the list. The bit fields are defined as:

| **Bits 7-5** | **Bits 4-0** |
| --- | --- |
| Reserved | AlertID |

AlertID: Specifies the ID of the Alert. The valid values are 1-2 for Module-defined Alerts 1 and 2, and 3-16 for Host-defined Alerts 1-14. A value of 0 terminates the list. To schedule an Ad Insertion or Ad Replacement, at a minimum the first entry in the list must be non-zero.

**VolAdj**: Provides an adjustment to the base audio volume set by the Host for the i-th scheduled Alert. This volume adjustment will apply when the corresponding Alert is played back. This field is only valid when the corresponding **AlertList** field is non-zero, otherwise the Module ignores it. Positive values increase the audio volume up to the maximum possible value specified in the Audio Volume Dispatch (i.e., 24 dB), and negative values decrease the audio volume to the minimum possible value specified in the Audio Volume Dispatch (i.e., -95 dB). LSB = 1dB. The range of values is -20 thru 20.

## Ad Control Error Response

This message can be sent in response to an Ad Control Dispatch. See the description of the Ad Control Dispatch for processing details.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 150 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = Command field within **AdControl** is outside its allowed range

1 = one or more AlertID fields within **AlertList** is outside its allowed range

2 = one or more **VolAdj** values are outside their allowed range

3 = cannot schedule Ad Insertion or Ad Replacement unless in Sequential playback

4 = cannot schedule Ad Insertion or Ad Replacement on a Layer 2 SID

5 = audio for the requested Alert(s) has not been fully downloaded to the Module

6 = there is no pending scheduled Ad Insertion or Ad Replacement to cancel

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Metadata Reference Dispatch

The Module sends the Metadata Reference Dispatch to the Host to report MRef’s associated with the current Live content of the indicated channel.

The Module will send the Metadata Reference Dispatch for any channel when new MRef’s are detected for the Live content.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Metadata Reference Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 15

The **ActionMRef** field was a late addition to this message. Module FW using SXi8 v1.20 and before will transmit this message without this field and the message will have a length of 15 bytes.

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 17 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 59 |
| 7-8 | SID | uint16 | See below |
| 9-10 | CutMRef | uint16 | See below |
| 11-12 | SegmentMRef | uint16 | See below |
| 13-14 | ProgramMRef | uint16 | See below |
| 15-16 | ActionMRef | uint16 | See below |

**SID**: SID associated with these MRef’s. See Section 4.8.

**CutMRef**: Bit field delivering the Cut MRef. See bitfield format below.

**SegmentMRef**: Bit field delivering the Segment MRef. See bitfield format below.

**ProgramMRef**: Bit field delivering the Program MRef. See bitfield format below.

**ActionMRef**: Bit field delivering the Program MRef. See bitfield format below.

Bitfield format for **CutMRef**, **SegmentMRef**, **ProgramMRef** and **ActionMRef**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bit 15** | **Bits 14-12** | **Bit 11** | **Bits 10-9** | **Bits 8-0** |
| MRefValid | Reserved | RFU | Hour | TransitionID |

TransitionID, Hour, RFU: These parameters are defined in the Metadata Reference Block Structure of the BIC Metadata Reference Message in [2].

MRefValid: Indicates if the MRef is valid (i.e., whether or not the Module has received the corresponding MRef via the SiriusXM broadcast). The valid values are:

0 = invalid

1 = valid

## Lookahead Metadata Reference Dispatch

The Module sends the Lookahead Metadata Reference Dispatch to the Host to report MRef’s associated with an upcoming Live content transition. The Module will send the Lookahead Metadata Reference Dispatch for any available audio channel when new Lookahead MRef’s are detected for that channel’s Live content.

This message is automatically enabled and cannot be disabled by the Host.

The Host sends the Generic Response to the Module to confirm receipt of the Metadata Reference Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 12

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 60 |
| 7-8 | SID | uint16 | See below |
| 9-10 | LAMRef | uint16 | See below |
| 11 | Offset | uint8 | See below |

**SID**: SID associated with this MRef. See Section 4.8.

**LAMRef**: Lookahead Metadata Reference. Contains the following parameters defined in the Lookahead Metadata Reference Block Structure of the BIC Lookahead Metadata Reference Message in [2]: Transition ID, Hour, RFU, and Type. The parameters are ordered as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Bits 15-14** | **Bits 13-11** | **Bits 10-9** | **Bits 8-0** |
| Type | RFU | Hour | TransitionID |

**Offset**: Offset in seconds from this Lookahead Metadata Reference Dispatch to the expected delivery time of the associated Metadata Reference Dispatch. Valid values are 1 thru 108.

# Firmware Update Messages

Firmware Update (FWU) is the process of installing new firmware (i.e., executable code or data) into the Module. The Host manages FWU by breaking a binary file into packets and sending them sequentially to the Module. The binary file contains all of the information needed to identify what it is, what its version is, etc., and define its interdependencies with other Gen8 firmware. The Module verifies the validity of the individual packets and the overall binary file, then programs its nonvolatile memory with this new firmware. A Module reset is required before the new firmware will take effect.

## FWU Start Dispatch

The Host sends the FWU Start Dipatch to the Module to begin the FWU process. Only one FWU can be in progress at any time, so if the Module receives this message and FWU is in progress then it is terminated and FWU is restarted using the new command data. Once the Host receives a successful response it can immediately begin sending FWU Packet Dispatches to the Module to download the new binary.

The FWU process must be followed for each binary (i.e., COP or CAP) to be flashed. The Host has the following options for updating the Module's firmware on any given power cycle:

* Only update the COP firmware.
* Only update the CAP firmware.
* Update the COP and CAP firmware as a non-coupled update.
* Update the COP and CAP firmware as a coupled update.

If a coupled COP and CAP are to be flashed, then both must be specified as being flashed as coupled, and only if both are successfully flashed prior to a power cycle will the Module use them on the next power cycle.

If a non-coupled COP and CAP are to be loaded, then whatever binary(ies) are successfully flashed will be used on the next power cycle.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: FWU Start Response or FWU Start Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 11

The **Control** field was a late addition to this message. To retain backward compatibility, the Module must allow for FWU Start Dispatch to contain only 11 bytes, and use a default value for **Control** (i.e., 0).

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 44 |
| 7-10 | NumBinaryBytes | uint32 | See below |
| 11 | Control | uint8 | See below |

**NumBinaryBytes**: Total number of bytes in the binary to be downloaded. The valid values are 1024 - 1,638,400.

**Control**: Bitfield of options for controlling FWU download and processing.

|  |  |
| --- | --- |
| **Bits 7-1** | **Bit 0** |
| Reserved | Coupled |

Coupled: Indicates whether or not new COP and CAP firmware must both be programmed successfully before either can be used. E.g., this would be done in the case where a change to the COP/CAP interface requires the firmware for both processors to be updated at the same time to ensure compatibility. The value values are 0=not coupled, 1=coupled.

## FWU Start Response

When the Module receives a FWU Start Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the FWU Start Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 32

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 45 |
| 7 | Status | uint8 | See below |

**Status**: Provides status on the FWU process. The valid values are:

0 = FWU process successfully started.

1 = Previous FWU was in progress and has been aborted. FWU process successfully started.

## FWU Start Error Response

When the Module receives a FWU Start Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the FWU Start Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 46 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = too few bytes specified for the binary length

1 = too many bytes specified for the binary length

2 = too few bytes specified for the binary length; FWU that was in progress has been terminated

3 = too many bytes specified for the binary length; FWU that was in progress has been terminated

4 = a binary has already been flashed as coupled, cannot load a non-coupled binary

5 = a binary has already been flashed as non-coupled, cannot load a coupled binary

6 = a binary has already been flashed as coupled, cannot load a non-coupled binary; FWU that was in progress has been terminated

7 = a binary has already been flashed as non-coupled, cannot load a coupled binary; FWU that was in progress has been terminated

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## FWU Packet Dispatch

The binary file being downloaded is broken into 1024-byte snippets by the Host. The Host sends the FWU Packet Dispatch containing the first of those snippets to the Module. The Module responds with a FWU Packet Response confirming that snippet has been received. The Host then sends the next snippet, and so on, until the entire file has been transferred. The Host must send the snippets in sequential order.

Message Usage: Production Infotainment System

Message Direction: Sent to Module

Response: FWU Packet Response or FWU Packet Error Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 1035

Maximum **NumMsgBytes**: 1400

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 1035 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 47 |
| 7-10 | PacketID | uint32 | See below |
| 11-1034 | PacketData[1024] | uint8 | See below |

**PacketID**: Sequence number of the current packet. The starting value of this field is 0 and it must increment by 1 for each subsequent packet. The valid range is all unsigned integer values, though in reality it will be limited by the size of the largest allowable binary file.

**PacketData**: Binary data (snippet). This field will always be 1024 bytes long. For the last packet of the download, the last (1024 – (*NumBinaryBytes* from the FWU Start Dispatch modulo 1024)) bytes are invalid and will be ignored by the Module. *PacketData* must contain at least one byte of valid data.

## FWU Packet Response

When the Module receives a FWU Packet Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the FWU Packet Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 12

Maximum **NumMsgBytes**: 32

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 48 |
| 7 | Status | uint8 | See below |
| 8-11 | PacketID | uint32 | See below |

**Status**: Provides status on the FWU packet. The valid values are:

0 = Packet accepted. Ready for next packet.

1 = Packet accepted. No more packets are expected (i.e., this was the last packet).

**PacketID**: Sequence number of the last packet that was successfully processed.

## FWU Packet Error Response

When the Module receives a FWU Packet Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the FWU Packet Error Response to the Host.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 60

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 60 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 49 |
| 7 | ErrorCode | uint8 | See below |
| 8-11 | PacketID | uint32 | See below |
| 12-59 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = FWU is not in progress so this command is invalid.

1 = Packet ignored. The first *PacketID* after the start of FWU was not 0.

2 = Packet ignored. The *PacketID* was out of sequence. *PacketID’s* must be monotonically increasing.

3 = Packet ignored. A packet was received after all expected packets had been processed.

4 = FWU aborted. Invalid binary header start bytes.

5 = FWU aborted. Invalid binary header end bytes.

6 = FWU aborted. Unknown binary type specified in binary header.

7 = FWU aborted. Invalid binary header number of bytes.

8 = FWU aborted. Binary header number of bytes does not agree with FWU Start Dispatch.

9 = FWU aborted. Invalid binary header checksum.

10 = FWU aborted. Invalid Firmware checksum.

11 = FWU aborted. Attempt to load the same FW without resetting Module.

12 = FWU aborted. Invalid binary header version.

13 = FWU aborted. Invalid binary header variable field.

14 = FWU aborted. Digital signature is missing from binary header or is invalid.

16 = FWU aborted. Invalid number of header bytes specified in binary header.

17 = FWU aborted. Attempt to revert to a previous version of COP FW.

18 = FWU aborted. Maximum number of attempts for this binary has been exceeded.

**PacketID**: Sequence number of the last packet that was successfully processed. If no packet has been successfully processed then PacketID is set to 0xFFFFFFFF.

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## FWU Status Dispatch

When FWU is in progress the Module sends the FWU Status Dispatch to the Host to report the status of the FWU process. The Module will send this indication whenever any of the FWU statuses contained in the message changes, but will send it no more often than once per second.

The Host sends the Generic Response to the Module to confirm receipt of the FWU Status Dispatch.

Message Usage: Production Infotainment System

Message Direction: Sent from Module

Response: Generic Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 28

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 28 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 50 |
| 7 | State | uint8 | See below |
| 8-11 | NumBytesRcvd | uint32 | See below |
| 12-15 | NumBytesProcessed | uint32 | See below |
| 16-17 | EstSecondsRemaining | uint16 | See below |
| 18-19 | NumNVMEraseFailures | uint16 | See below |
| 20-21 | NumNVMWriteFailures | uint16 | See below |
| 22-23 | NumNVMReadFailures | uint16 | See below |
| 24-25 | NumReadbackBytesMismatch | uint16 | See below |
| 26-27 | NumOutOfOrderPackets | uint16 | See below |

**State**: Indicates the state and status of the FWU process. The valid values are:

0 = FWU is in progress

1 = FWU completed successfully

2 = FWU aborted: NVM erase failed

3 = FWU aborted: NVM write failed

4 = FWU aborted: NVM read failed

5 = FWU aborted: NVM readback verification failed

6 = FWU aborted: Digital signature verification failed

**NumBytesRcvd**: Number of valid FWU data bytes the Module has received. When this number is equal to the number of bytes specified in the FWU Start Dispatch all data will have been received by the Module and the download phase of FWU is complete. The valid values are 0 thru *NumBinaryBytes* specified in the most recently received FWU Start Dispatch.

**NumBytesProcessed**: Number of FWU data bytes the Module has processed and written to its nonvolatile memory, if necessary. When this number is equal to the number of data bytes specified in the FWU Start Dispatch the binary file will have been completely stored in the Module’s nonvolatile memory. However, since the Module has to take other actions to complete the FWU this does not mean FWU is complete. The valid values are 0 thru *NumBytesRcvd*.

**EstSecondsRemaining**: Estimated number of seconds remaining until FWU is complete, assuming the Host provides the binary data to the Module faster than programming occurs. This number will take into account flash erase time, programming time, binary verification time, etc. The valid range is all unsigned integer values, though in reality it will be limited by the size of the largest allowable binary file. LSB = 1 sec.

**NumNVMEraseFailures**: Number of NVM erase commands that failed during the current FWU attempt.

**NumNVMWriteFailures**: Number of NVM write commands that failed during the current FWU attempt.

**NumNVMReadFailures**: Number of NVM read commands that failed during the current FWU attempt.

**NumReadbackBytesMismatch**: Number of bytes read back from NVM that did not match the binary that was sent to the Module for programming for the current FWU attempt.

**NumOutOfOrderPackets**: Number of FWU Packet Dispatches received that did not have the *PacketID* that the Module expected.

# Production Support Messages

This set of SXi8 messages is used to support the Manufacturing, Factory Test, Service, and code-update management of SiriusXM Modules.

## Set Production Data Dispatch

Prior to the Module being built into a product, a set of production data will be written to the Module. The Module will store this data in its NVM and provide it to the Host in the Module Configure Response at every subsequent startup. The format and content of the data is unknown to the Module, but the Host can use this data to identify the Module for test/debug purposes and for code update purposes.

When the Module receives a Set Production Data Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command (including writing the data to NVM). If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Set Production Data Error Response to the Host. Once the response is sent, power to the Module can be safely removed.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: Generic Response or Set Production Data Error Response

Response Timeout: 3000 msec

Minimum **NumMsgBytes**: 28

Maximum **NumMsgBytes**: 128

Test: Full validation by Test Team

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 28 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 94 |
| 7-11 | IVSMID[5] | uint8 | See below |
| 12-27 | ProductionData[16] | uint8 | See below |

**IVSMID**: This is the IVSMID reported to the Host in the Module Configure Response. Because of the criticality of the production data, this field is included to help prevent a Host from accidentally sending this message to the Module.

**ProductionData**: Data to be stored in the Module’s NVM during production that can later potentially be used to identify date of manufacture, manufacturer, OEM, etc. The format and content of this data is described in the SXM Client and Firmware Update Operations Concept document.

## Set Production Data Error Response

When the Module receives a Set Production Data Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Set Production Data Error Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 98 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = invalid IVSMID – data not written

1 = NVM erase failed – data not written

2 = NVM write failed – data not written

3 = NVM read failed – data not written

4 = NVM verification failed – data not written

5 = Maximum number of allowed Production Data writes to NVM has been exceeded.

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Set Service Data Dispatch

During production and factory test, a set of service data will be written to the Module. The Module will store this data in its NVM and provide it to the Host in the Get Service Data Response upon request. The format and content of the data is unknown to the Module, but the Host can use this data to identify the Module and its characteristics for test/debug purposes.

The service data will likely contain data such as the factory model number, hardware configuration, FA package ID, bill-of-materials used for that Module, RF sensitivity data, etc. By having this information available electronically, a service center could analyze a field failure much faster since they don’t have access to the factory database, which also contains this information. This data would also allow identification of a Module with an unreadable or damaged product label.

When the Module receives a Set Service Data Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command (including writing the data to NVM). If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Set Service Data Error Response to the Host. Once the response is sent, power to the Module can be safely removed.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: Generic Response or Set Service Data Error Response

Response Timeout: 3000 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 268

Maximum **NumMsgBytes**: 512

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 268 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 95 |
| 7-11 | IVSMID[5] | uint8 | See below |
| 12:267 | ServiceData[256] | uint8 | See below |

**IVSMID**: This is the IVSMID reported to the Host in the Module Configure Response. Because of the criticality of the service data, this field is included to help prevent a Host from accidentally sending this message to the Module.

**ServiceData**: Data to be stored in the Module’s NVM during production and factory test that can later be used to identify the Module and its capabilities. The format and content of this data is described in the TBD document.

## Set Service Data Error Response

When the Module receives a Set Service Data Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Set Service Data Error Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 99 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = invalid IVSMID – data not written

1 = NVM erase failed – data not written

2 = NVM write failed – data not written

3 = NVM read failed – data not written

4 = NVM verification failed – data not written

5 = Maximum number of allowed Service Data writes to NVM has been exceeded.

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Get Service Data Dispatch

During production and factory test, a set of service data will be written to the Module. The Host will send the Get Service Data Dispatch to the Module to request the service data be read from NVM and returned to the Host in the Get Service Data Response. Typically this will be done at a test or service center.

When the Module receives a Get Service Data Dispatch, it performs any processing associated with the message and sends the Get Service Data Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: Get Service Data Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 7

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 7 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 96 |

## Get Service Data Response

When the Module receives a Get Service Data Dispatch, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the Get Service Data Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 263

Maximum **NumMsgBytes**: 512

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 263 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 97 |
| 7-262 | ServiceData[256] | uint8 | See below |

**ServiceData**: Data stored in the Module’s NVM during production and factory test to identify the Module and its capabilities. The format and content of this data is described in the TBD document. Until the Host sets this data using the Set Service Data Dispatch, all bytes of this data item will be set to 0xFF.

## Tuner Calibration Dispatch

This command allows the Host to initiate the Tuner calibration procedure. Depending on which Tuner is installed, the Module will perform the appropriate calibration procedure and return a Tuner-specific response.

When the Module receives a Tuner Calibration Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, performs any processing associated with that command, and sends the ADI Tuner Calibration Response, NXP Tuner Calibration Response, or Broadcom Tuner Calibration Response to the Host depending on which Tuner is currently active.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: ADI Tuner Calibration Response, NXP Tuner Calibration Response, or Broadcom Tuner Calibration Response

Response Timeout: 2000 msec

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 127 |
| 7 | Command | uint8 | See below |

**Command**: Controls the operation the Module is to perform. The valid values are:

0 = Perform Tuner calibration and return results

1 = Do not perform Tuner calibration – just return the results

2 = Reset Tuner calibration results to their defaults

3 = Run cordic error sweep

## ADI Tuner Calibration Response

When the Module receives a Tuner Calibration Dispatch and the ADI Tuner is the active Tuner, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the ADI Tuner Calibration Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 512

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 140 |
| 7 | Status | uint8 | See below |

**Status**: Provides status on the Tuner calibration command. The valid values are:

0 = Calibration command successful

1 = Calibration command failed

2 = Invalid **Command**

3 = Calibration has not been performed, default values in use

## Broadcom Tuner Calibration Response

When the Module receives a Tuner Calibration Dispatch and the Broadcom Tuner is the active Tuner, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the Broadcom Tuner Calibration Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 512

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 141 |
| 7 | Status | uint8 | See below |

**Status**: Provides status on the Tuner calibration command. The valid values are:

0 = Calibration command successful

1 = Calibration command failed

2 = Invalid **Command**

3 = Calibration has not been performed, default values in use

## NXP Tuner Calibration Response

During production and factory test, the NXP Tuner will need to have a digitally-controlled crystal oscillator (DCXO) calibration performed. The calibration procedure requires a High Band or Low Band test signal to be input to the Module, and the Tuner determines a set of 128 frequency offsets. These offsets are stored in the Module’s NVM and are thereafter used within the AFC algorithm of the Module.

When the Module receives a Tuner Calibration Dispatch and the NXP Tuner is the active Tuner, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the NXP Tuner Calibration Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 796

A number of fields were added to this message for SXi8 v1.25. Per the Factory Team’s request, the new fields were inserted where they made logical sense and no attempt was made to maintain backward compatibility.

Maximum **NumMsgBytes**: 1024

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 803 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 128 |
| 7 | Status | uint8 | See below |
| 8-9 | Temperature | int16 | See below |
| 10-11 | InitialFrequencyOffset | int16 | See below |
| 12-13 | InitialDCXOOffset | int16 | See below |
| 14-15 | InitialCordicOffset | int16 | See below |
| 16-17 | DCXOCtrlCalInit | uint16 | See below |
| 18-29 | PolynomialParam[3] | int32 | See below |
| 30-33 | FigureOfMerit[2] | uint16 | See below |
| 34-545 | CordicError[128] | int32 | See below |
| 546-801 | FrequencyOffset[128] | int16 | See below |
| 802 | CordicSweepStatus | uint8 | See below |

**Status**: Provides status on the NXP Tuner calibration command. The valid values are:

0 = Calibration command successful

1 = Calibration command failed

2 = Invalid **Command**

3 = Calibration has not been performed, default values in use

4 = Calibration has not been performed. Maximum number of allowed Tuner Calibration Data writes to NVM has been exceeded.

5 = Could not update DCXO data in nonvolatile memory.

**Temperature**: Tuner temperature at which the calibration was conducted read from baseband IC. If calibration was not performed, then the value is set to 0. LSB=1 dec C. Valid values are -40 thru 120.

**InitialFrequencyOffset**: Initial frequency offset for the calibration. If calibration was not performed, then the value is set to 0. LSB=1 ppm \* 2-8. Valid values are -15360 thru 15360.

**InitialDCXOOffset**: Initial DCXO offset for the calibration. If calibration was not performed, then the value is set to 0. LSB=1 ppm \* 2-8. Valid values are -15360 thru 15360.

**InitialCordicOffset**: Initial cordic offset for the calibration. If calibration was not performed, then the value is set to 0. LSB=1 ppm \* 2-8. Valid values are -15360 thru 15360.

**DCXOCtrlCalInit**: Initial DCXO calibration control value. If calibration was not performed, then the value is set to 0. Valid values are -64 thru 63.

**PolynomialParm**: Polynomial parameters (coefficients?). If calibration was not performed or failed, then the values are set to 0. LSB=2-16. Valid values are -32768 thru 32767.

**FigureOfMerit**: Curve-fitting statistics. If calibration was not performed or failed, then the values are set to 0. LSB=2-16. Valid values are 0 thru 65535.

**CordicError**: Calculated cordic error for the corresponding frequency bin when running the cordic error sweep command. LSB=1 PPM \* 2-16. Valid values are -2147483648 thru 2147483647.

**FrequencyOffset**: Frequency offsets correction values to be applied to the default DCXO step table to compute a working step table to be used by AFC algorithm for the NXP Tuner on subsequent Module power-ons. If calibration was not performed or failed, then the values are set to 0. LSB=1 ppm \* 2-8. Valid values are -15360 thru 15360.

**CordicSweepStatus**: Provides status running the cordic error sweep command. This field is set to 0 when running any other **Command**. The valid values are:

0 = No error

1 = No CHDec Front End bulk frequency lock

2 = CHDec Front End bulk frequency offset must be 0

3 = CHDec fails demod lock test, carrier frequency estimate not reliable

4 = Tuner SPI (write, read, or verify) error

5 = CHDec cannot make repeatable frequency estimate on a carrier

6 = CHDec cannot make repeatable frequency estimate across carriers

7 = Invalid temperature reading

8 = Other error

## GPIO Control Dispatch

The Module can have one test GPIO pin exposed for external use. During production and factory test this pin will be toggled.

When the Module receives a GPIO Control Dispatch, it performs any processing associated with the message and sends the GPIO Control Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: GPIO Control Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 129 |
| 7 | Command | uint8 | See below |

**Command**: Specifies whether the test GPIO should be turned on or off. The valid values are:

0 = Turn GPIO off

1 = Turn GPIO on

## GPIO Control Response

When the Module receives a GPIO Control Dispatch, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the GPIO Control Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 130 |
| 7 | Status | uint8 | See below |

**Status**: Provides status on the test GPIO setting. The valid values are:

0 = GPIO setting succeeded

1 = Dispatch ignored; invalid value for **Command**

2 = GPIO setting failed

## Set Operating Mode Dispatch

The Module can be put into a high current drain mode to support stress testing. Other Module settings can also be adjusted with this message.

When the Module receives a Set Operating Mode Dispatch, it performs any processing associated with the message and sends the Set Operating Mode Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: Set Operating Mode Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

The **HBDemodControl** and **LBDemodControl** fields were late additions to this message. To retain backward compatibility, the Module must allow for this dispatch to contain only 8 bytes, and then use a default value for **HBDemodControl** (i.e., 0xFF) and **LBDemodControl** (i.e., 0xFF).Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 11 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 131 |
| 7 | Command | uint8 | See below |
| 8 | HBDemodControl | uint8 | See below |
| 9 | LBDemodControl | uint8 | See below |
| 10 | CNNumReads | uint8 | See below |

**Command**: Specifies the operating mode to put the Module in. The valid values are:

0 = Normal operating mode

1 = High current stress operating mode

**HBDemodControl**: Controls the High Band demodulators, with bit fields defined as:

| **Bits 7-6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- | --- | --- | --- |
| Reserved | TERRB | SAT2B | SAT1B | TERRA | SAT2A | SAT1A |

0 = Turn indicated demodulator off

1 = Turn indicated demodulator on

**LBDemodControl**: Controls the Low Band demodulators, with bit fields defined as:

| **Bits 7-3** | **Bit 2** | **Bit 1** | **Bit 0** |
| --- | --- | --- | --- |
| Reserved | TDM2 | COFDM | TDM1 |

0 = Turn indicated demodulator off

1 = Turn indicated demodulator on

**CNNumReads**: The number of C/N reads per frame used in determining the averaged C/N values contained in the C/N Status Dispatch. The valid values are 0, which disables the C/N averaring, and 1-255 which specifies the number of samples per averaged C/N.

## Set Operating Mode Response

When the Module receives a Set Operating Mode Dispatch, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the Set Operating Mode Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 132 |
| 7 | Status | uint8 | See below |

**Status**: Provides status on the operating mode setting. The valid values are:

0 = Operating mode setting succeeded

1 = Dispatch ignored; invalid value for **Command**

2 = Operating mode setting failed

## UART1 Control Dispatch

UART0 on the Module is used for SXi8 communication with the Host. In some Modules, the pins for UART1 are also brought out and need to be tested. By putting UART1 in loopback mode, its Rx and Tx lines can be tested. Typically, the test would be run by putting UART1 in loopback mode, transmitting data to UART1, receiving and verifying the echoed data from UART1, and putting UART1 back into its normal (i.e., disabled) mode.

When the Module receives a UART1 Control Dispatch, it performs any processing associated with the message and sends the UART1 Control Response to the Host.

When in loopback mode, UART1 is configured to 460800 bps, 8 data bits, 1 stop bit, and no parity.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: UART1 Control Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 133 |
| 7 | Command | uint8 | See below |

**Command**: Specifies the UART1 mode to put the Module in. The valid values are:

0 = Disable UART1 Rx and Tx (i.e., normal operation)

1 = Enable loopback mode (i.e., enable UART1 Rx and Tx, and echo any Rx data on Tx)

## UART1 Control Response

When the Module receives a UART1 Control Dispatch, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the UART1 Control Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 8

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 8 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 134 |
| 7 | Status | uint8 | See below |

**Status**: Provides status on the UART1 mode setting. The valid values are:

0 = UART1 mode setting succeeded

1 = Dispatch ignored; invalid value for **Command**

2 = UART1 mode setting failed

## DDR3 Test Control Dispatch

The DDR3 Test Control Dispatch is used to initiate a test of the DDR3 RAM in the Module. The test covers the DDR3 that is allocated for audio storage, which is the majority of the COP’s available DDR3.

Once the test is successfully initiated, all other Module features will cease to operate. The Module may not play audio and will not execute received Dispatches. The test is destructive, so that at the conclusion of the test the Module must be reset to restore operation.

The test time depends on the size of the memory available for testing, and the number of test loops requested.

When the Module receives a DDR3 Test Control Dispatch, it performs any processing associated with the message and sends the DDR3 Test Control Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: DDR3 Test Control Response

Response Timeout: TBD sec

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 9

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 9 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 135 |
| 7 | TestControl | uint8 | See below |
| 8 | NumLoops | uint8 | See below |

**TestControl**: Controls how the DDR3 test is to be performed. The valid values are:

0 = Run all loops of the test to completion, regardless of any errors found.

1 = Stop the test immediately on detection of an error.

**NumLoops**: The number of DDR3 test loops to perform. Valid values are 1-10.

## DDR3 Test Control Response

When the Module receives a DDR3 Test Control Dispatch, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the DDR3 Test Control Response to the Host.

Note that no information about the error is given (e.g., address in error, data written vs. data read, etc). Only the fact that an error occurred is provided.

After sending the DDR3 Test Control Response to the Host, the Module will no longer communicate on the SXi8 interface and must be reset.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 29

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 29 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 136 |
| 7 | Status | uint8 | See below |
| 8 | NumLoops | uint8 | See below |
| 9-28 | Errors[10] | uint16 | See below |

**Status**: Provides status on the DDR3 test. The valid values are:

0 = All requested tests were completed

1 = Dispatch ignored; invalid value for **TestControl**

2 = Dispatch ignored; invalid value for **NumLoops**

3 = Requested tests were halted on detected error

**NumLoops**: The number of DDR3 test loops that were started. Valid values are 0-10.

**Errors**: The number of errors that were detected for each loop of the DDR3 test. Valid values are 0-65535. If more than 65535 errors are detected for any loop, then the corresponding **Errors** entry is set to 65535.

## Maintenance Status Dispatch

During the Module’s lifetime there is maintenance status that the Module FW accumulates in order to aid in detection, analysis, and repair of Module hardware issues. This data is initialized at production time, then accumulated thereafter and stored in NVM. It is never reset. The Host will send the Maintenance Status Dispatch to the Module to request the maintenance status be read from NVM and returned to the Host in the Maintenance Status Response. Typically this will be done at a test or service center.

When the Module receives a Maintenance Status Dispatch, it performs any processing associated with the message and sends the Maintanance Status Response to the Host.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: Maintenance Status Response

Response Timeout: 500 msec

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 7

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 7 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 143 |

## Maintenance Status Response

When the Module receives a Maintnance Status Dispatch, it verifies the Module is in the proper state to receive the command, performs any processing associated with the command, and sends the Maintenance Status Response to the Host.

The Module HW can suffer permanent damage if operated beyond its rated operating temperature. It is important for Field Service to know if the maximum operating temperature has been exceeded, and if so, how often.

The Serial Flash parts being used in the Module have a typical erase/write limit of 100,000 cycles. The NVM is stored in two flash sectors in ping-pong style, effectively doubling the number of allowed NVM rewrites to 200,000. The Module requirements are written to reduce erase/write cycles, but the rewrite status will ensure these measures are effective and determine whether or not a wear-leveling algorithm is needed.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: Full validation by Test Team

Minimum **NumMsgBytes**: 117

The Serial Flash fields were late addition to this message. Module FW using v1.29 and earlier will transmit this message without these fields and the message will have a length of 117 bytes.

Maximum **NumMsgBytes**: 1400

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 149 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 144 |
| 7 | MaxTemp | uint8 | See below |
| 8-11 | NumOverTemps | uint32 | See below |
| 12 | YearOfLastOverTemp | uint8 | See below |
| 13 | MonthOfLastOverTemp | uint8 | See below |
| 14 | DayOfLastOverTemp | uint8 | See below |
| 15 | HourOfLastOverTemp | uint8 | See below |
| 16 | MinuteOfLastOverTemp | uint8 | See below |
| 17-20 | NVMRewrites | uint32 | See below |
| 21-116 | NVMRewriteTrigger[24] | uint32 | See below |
| 117-120 | SerialFlashProductID | uint32 | See below |
| 121-128 | SFDPHeader[2] | uint32 | See below |
| 129-136 | SFDPParamHeader[2] | uint32 | See below |
| 137-148 | SFDPParamTable[3] | uint32 | See below |

**MaxTemp**: Maximum chip temperature that has been read. LSB = 1o C. Valid values are 0-255.

**NumOverTemps**: The number of times the maximum allowable chip temperature has been reached or exceeded. All values are valid.

**YearOfLastOverTemp**: The year when the last chip overtemp was recorded, minus 2018. If **NumOverTemps** is 0 then this field is invalid. Valid values are 0-255.

**MonthOfLastOverTemp**: The month when the last chip overtemp was recorded. If **NumOverTemps** is 0 then this field is invalid. Valid values are 1-12.

**DayOfLastOverTemp**: The day when the last chip overtemp was recorded. If **NumOverTemps** is 0 then this field is invalid. Valid values are 1-31.

**HourOfLastOverTemp**: The hour when the last chip overtemp was recorded. If **NumOverTemps** is 0 then this field is invalid. Valid values are 0-23.

**MinuteOfLastOverTemp**: The minute when the last chip overtemp was recorded. If **NumOverTemps** is 0 then this field is invalid. Valid values are 0-59.

**NVMRewrites**: The number of times the NVM database has been written to serial flash. Each write requires the sector be erased, then the values written. All values are valid.

**NVMRewriteTrigger[24]**: A count of each action that could trigger the NVM database to be rewritten to serial flash. All values are valid. Entries in the data array are for:

0 = NVM data determined to be invalid or a new NVM database version caused an NVM reset

1 = Host-initiated NVM reset

2 = CHDEC Bulk Delay change

3 = Service Data change

4 = Production Data change

5 = AFC data (e.g., DCXO calibration data) change

6 = New DSMT received

7 = Module subscription change

8 = Channel Data change

9 = BDC channel change

10 = Low Band Operational Mode change

11 = New TDS Channel Event Table Definition

12 = New IMS Channel Table Definition

13 = New IMS Cut Table Definition

14-23 = Reserved

**SerialFlashProductID**: Serial flash manufacturer ID, device ID, memory type, and capacity.

**SFDPHeader[2], SFDPParamHeader[2], SFDPParamTable[3]**: Serial flash discoverable paramters (SFDP) for the installed serial flash. SFDPHeader[0] contains the SFDP signature. SFDPHeader[1] contains the number of parameter headers and the SFDP revision. SFDPParamHeader[0] contains the parameter table length, parameter table revision, and parameter ID LSB. SFDPParamHeader[1] contains the parameter ID MSB and the parameter table pointer. SFDPParamTable contains the first three elements of the SFDP parameter table.

## C/N Status Dispatch

The Module sends the C/N Status Dispatch to report the High Band and Low Band carrier/noise values. The Module shall send this message once per second whenever the C/N Status Dispatch is enabled, regardless of whether or not any data in the message changes. The Set Operating Mode Dispatch is used to configure the number of C/N samples to average at 1 msec intervals from the start of the High Band or Low Band frame.

The sending of this message is controlled by the Status Monitor Dispatch.

The Module ignores any Host response to this message, and the Module will not attempt resends of this message.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Response: None

Response Timeout: NA

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 25

Maximum **NumMsgBytes**: 512

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 25 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 145 |
| 7-8 | HBCNSat1A | uint16 | See below |
| 9-10 | HBCNSat2A | uint16 | See below |
| 11-12 | HBCNTerrA | uint16 | See below |
| 13-14 | HBCNSat1B | uint16 | See below |
| 15-16 | HBCNSat2B | uint16 | See below |
| 17-18 | HBCNTerrB | uint16 | See below |
| 19-20 | LBCNSat1 | uint16 | See below |
| 21-22 | LBCNSat2 | uint16 | See below |
| 23-24 | LBCNTerr | uint16 | See below |

**HBCNSat1A, HBCNSat2A, HBCNTerrA, HBCNSat1B, HBCNSat2B, HBCNTerrB**: The High Band satellite 1, satellite 2, and terrestrial carrier/noise ratios for ensembles A and B. LSB=0.01 dB. The valid values are 0-65535.

**LBCNSat1, LBCNSat2, LBCNTerr**: The Low Band satellite 1, satellite 2, and terrestrial carrier/noise ratios. LSB=0.01 dB. The valid values are 0-65535.

## Messaging Status Dispatch

The Module sends the Messaging Status Dispatch to report various statuses for the SXi8 messaging interface. The Module shall send this message once when the Messaging Status Dispatch becomes enabled, and thereafter, the message is sent once every 5 seconds while it remains enabled, regardless of whether or not any data in the message changes.

The sending of this message is controlled by the Status Monitor Dispatch.

The Module ignores any Host response to this message, and the Module will not attempt resends of this message.

Message Usage: Infotainment System Development

Message Direction: Sent from Module

Response: None

Response Timeout: NA

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 73

The **ResponsePendingRecoveries** field was a late additions to this message. Modules using SXi8 versions earlier than v2.9 will transmit this message without this field and the message will have a length of 73 bytes.

The **DataPacketsOutOfOrder** field was a late additions to this message. Modules using SXi8 versions earlier than v2.20 will transmit this message without this field and the message will have a length of 77 bytes.

Maximum **NumMsgBytes**: 256

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 81 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 17 |
| 7-10 | NumBytesRcvd | uint32 | See below |
| 11-14 | NumBytesXmtd | uint32 | See below |
| 15-18 | NumMsgsRcvd | uint32 | See below |
| 19-22 | NumMsgsXmtd | uint32 | See below |
| 23-26 | DispatchTimeouts | uint32 | See below |
| 27-30 | DataPacketsDropped | uint32 | See below |
| 31-34 | NumStrayBytes | uint32 | See below |
| 35-38 | NumNonsequentialTransactionIDs | uint32 | See below |
| 39-42 | NumIgnoredMsgs | uint32 | See below |
| 43-46 | DataPacketsProcessed | uint32 | See below |
| 47-50 | CorruptedDataPackets | uint32 | See below |
| 51-54 | DataPacketsLost | uint32 | See below |
| 55-58 | LowPriorityBumps | uint32 | See below |
| 59-62 | Reserved | uint32 | See Section 4.5 |
| 63-66 | InvalidDSMTs | uint32 | See below |
| 67-70 | MRefEntriesNotFound | uint32 | See below |
| 71-72 | DSMTVersion | uint16 | See below |
| 73-76 | ResponsePendingRecoveries | uint32 | See below |
| 77-80 | DataPacketsOutOfOrder | uint32 | See below |

**NumBytesRcvd** – This is the number of bytes received by the Module since it was last reset.

**NumBytesXmtd** – This is the number of bytes transmitted by the Module since it was last reset.

**NumMsgsRcvd** – This is the number of properly formed SXi8 messages received by the Module since it was last reset.

**NumMsgsXmtd** – This is the number of SXi8 messages transmitted by the Module since it was last reset.

**DispatchTimeouts** – This is the number of Dispatches that the Module did not receive a Response to within the timeout specified for that Dispatch since the Module was last reset.

**DataPacketsDropped** – This is the number of data packets that were available to be sent to the Host but were dropped to make room for newer data packets of the same priority before they could be sent..

**NumStrayBytes** – This is the number of bytes the Module received that were not part of a properly formed SXi8 message. Typically stray bytes will only occur on startup, or on a noisy or faulty SXi8 interface.

**NumNonsequentialTransactionIDs** – The number of dispatches the Module received with a TransactionID that did not meet the requirements of section 4.3.

**NumIgnoredMsgs** – The number of dispatches the Module received and ignored while a prior dispatch was still being processed.

**DataPacketsProcessed** – The number of data packets the Data Manager (DM) Task received from the Service Layer (SL) Task for processing.

**CorruptedDataPackets** – The number of data packets that DM determined were corrupted. DM checks for Sync in SDTP and XMAPP packets, and if the Sync is not correct then the packet is declared to be corrupted and is discarded.

**DataPacketsLost** – SL passes data packets to DM via a shared buffer. If DM does not process the data packets before SL needs to reuse the buffer, then those packets are lost. This is the number of data packets that are lost in this manner.

**LowPriorityBumps** – The number of low priority data packets that were discarded to make room for high priority data packets.

**DataPacketTimeouts** – The number of data packet Dispatches that the Module did not receive a Response to within the timeout specified for that data Dispatch. This is a subset of the **DispatchTimeouts** count.

**InvalidDSMTs** – The number of DSMT’s the Module received but discarded because it detected that one or more fields within the DSMT had a value outside its allowed limits.

**MRefEntriesNotFound** – The number of Cut MRefs for which a corresponding Integrated Metadata Service entry could not be found.

**DSMTVersion** – The version of the DSMT currently in use in the Module.

**ResponsePendingRecoveries** – The number of times a dispatch was received from the Host and the previously received dispatch had timed out without its corresponding response having been sent to the Host.

**DataPacketsOutOfOrder** –The number SDTP data packets received out of order, as determined by inspection of the data services’ SDTP packet counter (PLPC). These data packets are dropped and not sent to the host.

## NVM Test Control Dispatch

The NVM Test Control Dispatch is used to initiate a NVM factory test of unused flash sectors.

This dispatch should only be used during factory testing. It should not be performed during normal Module operation to guarantee that the NVM is not being used by other processes.

Test execution time depends on the number of unused NVM sectors and the number of test loops requested. As a benchmark, test duration is approximately 21 seconds when there are 31 unused sectors of NVM with 1 test loop.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent to Module

Response: NVM Test Control Response

Response Timeout: TBD sec

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 9

Maximum **NumMsgBytes**: 64

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 9 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 166 |
| 7 | TestControl | uint8 | See below |
| 8 | NumLoops | uint8 | See below |

**TestControl**: Controls how the NVM test is to be performed. The valid values are:

0 = Stop the test immediately on detection of an error.

**NumLoops**: The number of NVM test loops to perform. Valid values are 1-10.

## NVM Test Control Response

When the Module receives a NVM Test Control Dispatch, it verifies the command and performs any processing associated with the command and sends the NVM Test Control Response to the Host.

The Errors field contains information about the test result of each of the 64 sectors of NVM.

Message Usage: Manufacturing and Service Depot

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Manufacturing Team

Minimum **NumMsgBytes**: 73

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 73 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 167 |
| 7 | Status | uint8 | See below |
| 8-71 | Errors[64] | uint8 | See below |
| 72 | NumLoops | uint8 | See below |

**Status**: Provides status on the NVM test. The valid values are:

0 = Test Completed Successfully

1 = Test Failed and Halted for Errors

**Errors**: Each byte in Errors array represent one of the 64 sectors available in the NVM. The valid values are:

0 = Sector test successful.

1 = Sector test failed on compare

2 = Sector test failed on erase

3 = Sector test failed on write

4 = Sector test failed on read

255 = Sector not tested

**NumLoops**: The number of NVM test loops that were started. Valid values are 0-10.

# Engineering Messages – Memory

The messages in the chapter are intended for use in an Engineering and/or Test environment. These messages will be disabled in production Module Firmware and should not be used by a production Host.

If the definition of an Engineering message changes, then an attempt will be made to make it backward and forward compatible with the message’s previous definition. However, if this is not doable without unnecessarily complicating the message, then compatibility will not be preserved.

## Write Memory Dispatch

The Host sends the Write Memory Dispatch to the Module to write a value to a specific memory address in the COP’s address space. This could include DDR, ITCM , DTCM, or memory mapped hardware registers.

There is nothing to prevent this command from overwriting code, data, vital hardware registers, etc. It is up to the Host to know what it is writing and how it will affect Module Firmware operation. Given the danger inherent in this message, it is recommended that only Engineering tools implement this message and that Production Hosts do not have access to this message.

When the Module receives a Write Memory Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Write Memory Error Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent to Module

Response: Generic Response or Write Memory Error Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 20

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 20 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 108 |
| 7-11 | IVSMID[5] | uint8 | See below |
| 12-15 | Address | uint32 | See below |
| 16-19 | Data | uint32 | See below |

**IVSMID**: This is the IVSMID reported to the Host in the Module Configure Response. Because of the criticality of this message, this field is included to help prevent a Host from accidentally sending this message to the Module.

**Address**: Address of memory to write to. The valid range is 0x00000000 through 0xFFFFFFFC, and must be evenly divisible by 4. The sender of the message is responsible for verifying that the address is a valid, writable memory location and will not cause any undesirable side effects.

**Data**: Value to write to the specified address.

## Write Memory Error Response

When the Module receives a Write Memory Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Write Memory Error Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 109 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = invalid **IVSMID** – data not written

1 = **Address** is not evenly divisible by 4 – data not written

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Read Memory Dispatch

The Host sends the Read Memory Dispatch to the Module to read a value from a specific memory address in the COP’s address space. This could include DDR, ITCM , DTCM, or memory mapped hardware registers.

There is nothing to prevent this command from reading from an invalid address, or cause fatal effects if reading from certain hardware registers. It is up to the Host to know what it is reading and how it will affect Module Firmware operation. Given the danger inherent in this message, it is recommended that only Engineering tools implement this message and that Production Hosts do not have access to this message.

When the Module receives a Read Memory Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, performs any processing associated with that command, and sends the Read Memory Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent to Module

Response: Read Memory Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 11

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 11 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 112 |
| 7-10 | Address | uint32 | See below |

**Address**: Address of memory to read from. The valid range is 0x00000000 through 0xFFFFFFFC, and must be evenly divisible by 4. The sender of the message is responsible for verifying that the address is a valid, readable memory location and will not cause any undesirable side effects.

## Read Memory Response

When the Module receives a Read Memory Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, performs any processing associated with that command, and sends the Read Memory Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 12

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 113 |
| 7 | Status | uint8 | See below |
| 8-11 | Data | uint32 | See below |

**Status**: Indicates if the read request was successful or not, and why. The valid values are:

0 = successful, data is valid

1 = failed, **Address** is not evenly divisible by 4, data is not valid

**Data**: uint32 read from **Address**.

## Write Serial Flash Dispatch

The Host sends the Write Serial Flash Dispatch to the Module to write the specified number of values to the specified sequential addresses in the COP’s serial flash.

There is nothing to prevent this command from overwriting code, NVM data, etc. It is up to the Host to know what it is writing and how it will affect Module Firmware operation. Given the danger inherent in this message, it is recommended that only Engineering tools implement this message and that Production Hosts do not have access to this message.

When the Module receives a Write Serial Flash Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are successful then the Module sends the Generic Response to the Host. If these operations are not successful then the Module sends the Write Serial Flash Error Response to the Host.

Message Usage: SiriusXM Engineering Debug releases only

Message Direction: Sent to Module

Response: Generic Response or Write Serial Flash Error Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 82

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 82 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 110 |
| 7-11 | IVSMID[5] | uint8 | See below |
| 12 | Command | uint8 | See below |
| 13-16 | Address | uint32 | See below |
| 17 | NData | uint8 | See below |
| 18-81 | Data[64] | uint8 | See below |

**IVSMID**: This is the IVSMID reported to the Host in the Module Configure Response. Because of the criticality of this message, this field is included to help prevent a Host from accidentally sending this message to the Module.

**Command**: Controls the erase/write operation to be performed by the Module. The valid values are:

0 = do not erase the serial flash sector before writing the values

1 = erase the enclosing serial flash sector before writing the data

**Address**: Beginning address of serial flash to write. The valid range is 0 thru [size of serial flash – 1]. The Module increments the address accordingly for each data value written.

**NData**: Number of uint8 data items in **Data** to write to the serial flash. The valid values are 1-64.

**Data**: Data to write to the serial flash. Unused entries are ignored by the Module.

## Write Serial Flash Error Response

When the Module receives a Write Serial Flash Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, and performs any processing associated with that command. If these operations are not successful then the Module sends the Write Serial Flash Error Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 56

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 56 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 111 |
| 7 | ErrorCode | uint8 | See below |
| 8-55 | ErrorText[48] | uint8 | See below |

**ErrorCode**: Indicates why the command failed. The valid values are:

0 = invalid **IVSMID** – data not written

1 = invalid **Command** – data not written

2 = invalid **Address** – data not written

3 = invalid **NData** – data not written

4 = **Address**/**NData** traverses a serial flash sector – data not written

5 = NVM erase failed

6 = NVM write failed

7 = readback failed

8 = this feature is disabled in the currently running code

**ErrorText**: ASCII characters describing the error in human-readable form. Any unused bytes at the end of the text will be set to 0. The Host should ignore the content in this field as it is for Engineering Use Only and could be changed at any time.

## Read Serial Flash Dispatch

The Host sends the Read Serial Flash Dispatch to the Module to read the specified number of values from the specified sequential addresses in the COP’s serial flash.

When the Module receives a Read Serial Flash Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, performs any processing associated with that command, and sends the Read Serial Flash Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent to Module

Response: Read Serial Flash Response

Response Timeout: 500 msec

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 12

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 12 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 114 |
| 7-10 | Address | uint32 | See below |
| 11 | NData | uint8 | See below |

**Address**: Beginning address of serial flash to read. The valid range is 0 thru [size of serial flash – 1]. The Module increments the address accordingly for each data value read.

**NData**: Number of uint8 data items to read from the serial flash. The valid values are 1-64.

## Read Serial Flash Response

When the Module receives a Read Serial Flash Dispatch it validates the command parameters, verifies the Module is in the proper state to receive the command, performs any processing associated with that command, and sends the Read Serial Flash Response to the Host.

Message Usage: SiriusXM Engineering

Message Direction: Sent from Module

Test: General message validation by Test Team; data verification by Development Team

Minimum **NumMsgBytes**: 72

Maximum **NumMsgBytes**: 128

Message Format:

| **Byte #** | **Field Name** | **Data Type** | **Value** |
| --- | --- | --- | --- |
| 0 | Sync | uint8 | See Section 4.1 |
| 1-2 | NumMsgBytes | uint16 | 72 |
| 3-4 | Checksum | uint16 | See Section 4.6 |
| 5 | TransactionID | uint8 | See Section 4.3 |
| 6 | MessageID | uint8 | 115 |
| 7 | Status | uint8 | See below |
| 8-71 | Data[64] | uint8 | See below |

**Status**: Indicates if the read request was successful or not, and why. The valid values are:

0 = successful, data is valid

1 = failed, invalid **Address**, data is not valid

2 = failed, invalid **NData**, data is not valid

**Data**: uint8 values read from the specified serial flash **Address**. The first **NData** values are valid, and any remaining entries are set to 0.

1. All of the Pellets have the same “Prefix” (a0012345678900001234b032f305) which is the portion of the Pellet prior to the Selection byte [↑](#footnote-ref-1)