### # Title:

EINJv2 Changes

### # Status:

Draft

### # Document:

ACPI Specification Version 6.5

### # Submitter:

• Leo Duran (AMD)

• Harb Abdulhamid (Ampere)

• Samer El-Mahmoud (Arm)

### # Summary of the change

This ECR addresses a set of issues found in EINJv2 as defined in ACPI spec v6.5:

• Is not evident how to determine the maximum number of Component Syndrome Array entries supported by the platform.

• EINJV2\_GET\_ERROR\_TYPE does not return severities per supported error type.

• EINJV2\_SET\_ERROR\_TYPE does not have a mechanism to identify the platform for support of vendor-specific data.

• Adding another “SET\_ERROR\_TYPE” action increases arbitration complexity for both the OS and platform firmware.

• The “Error Injection Version 2 Operation" section is missing steps around the trigger error action table.

### # Benefits of the change

Enables support for EINJv2.

### # Impact of the change

Platform firmware and OSPM would have to implement support for the proposed changes.

### # Detailed description of the change

For the most part, the proposed changes remove the EINJv2\_SET\_ERROR\_TYPE action in favor of extending the existing SET\_ERROR\_TYPE\_WITH\_ADDRESS action for support of EINJv2.

**Format for Markups:**

Entries may include “**…**” to indicate unchanged text.

inserted text

~~deleted text~~

**Table 18.25: Error Injection Actions**

|  |  |  |
| --- | --- | --- |
| Value | Name | Description |
| … | … | … |
| ~~0x10~~ | ~~EINJV2\_SET\_ERROR\_TYPE~~ | ~~New Error Injection action introduced in EINJv2 for the~~  ~~purpose of injecting an EINJv2 error type, with address,~~  ~~severity, and detailed syndrome. Only one Error type can be~~  ~~injected at any given time. For multiple injections at the~~  ~~same time, then the platform will return an error condition.~~  ~~The RegisterRegion field (see Table 18.26) in~~  ~~EINJV2\_SET\_ERROR\_TYPE points to a data structure~~  ~~whose format is defined in Table 18.34~~  ~~See Section 18.6.4.1, for the EINJv2 error type definition~~ |
| ~~0x11~~ | ~~EINJV2\_GET\_ERROR\_TYPE~~ | ~~Returns the extended error injection capabilities (EINJv2~~  ~~features) of the platform:~~  ~~Bit 0 - EINJv2 Processor Error Type supported~~  ~~Bit 1 - EINJv2 Memory Error Type supported~~  ~~Bit 2 - EINJv2 PCIe Error Type supported~~  ~~Bit 3-30 - Reserved (Must be zero)~~  ~~Bit 31 - EINJv2 Vendor specific error codes~~ |
| 0x11 | EINJV2\_GET\_ERROR\_TYPE | Returns the EINJv2 injection capabilities of the platform.  See Table 18.33**.** |
| … | … | … |

**Table 18.30: Error Type Definition**

|  |  |
| --- | --- |
| **Bit** | **Description** |
| … | … |
| 30 | EINJv2 Error Type. ~~If this bit is set, then the support for EINJV2\_SET\_ERROR\_TYPE and EINJV2\_GET\_ERROR\_TYPE actions are supported.~~ If this bit is set, the SET\_ERROR\_TYPE\_WITH\_ADDRESS data structure includes the EINJv2 Extension Structure defined in Table 18.34. **NOTE:** This may only be used with the action GET\_ERROR\_TYPE and it is not permitted to set this bit with SET\_ERROR\_TYPE or SET\_ERROR\_TYPE\_WITH\_ADDRESS. |
| 31 | Vendor Defined Error Type … |

**Table 18.31: SET\_ERROR\_TYPE\_WITH\_ADDRESS Data Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Error Type | 4 | 0 | Bitmap of error types to inject. ~~See~~ **~~Table 18.30~~**~~.~~ If the EINJv2 Error Type bit is set by the GET\_ERROR\_TYPE action, the encoding of this field depends on Bit [3] of the Flags field below:   * (Flags [3] == 0), see Table 18.30 for Standard Error. * (Flags [3] == 1), see Table 18.33 for EINJv2 Error.   Otherwise, see Table 18.30 for Standard Error.  This field is cleared by the platform once it is consumed. |
| Vendor Error Type Extension Structure Offset | 4 | 4 | Specifies the offset from the beginning of ~~the table to the vendor error type extension structure. If no vendor error type~~  ~~extension is present, bit31 in error type must be clear and this field must be set to 0.~~ this structure to the Vendor Error Type Structure (see Table 18.32). This field is only valid if Bit [31] (Vendor Defined Error Type) or Bit [30] (EINJv2 Error Type) are set by the GET\_ERROR\_TYPE action. A value of 0 implies that the Vendor Error Type Extension Structure is not present.  **NOTE:** This field is Read-Only to software. |
| Flags | 4 | 8 | Bit [0] - Processor Identification Field Valid  Bit [1]- Memory Address and Memory ~~a~~Address ~~Mask~~ Range Field Valid  **NOTE:** For CXL errors, the Memory Address points to a CXL 1.1 compliant memory-mapped Downstream port  Bit [2] - PCIe SBDF field valid  **NOTE:** For CXL errors, the SBDF points to a CXL 2.0 compliant Root port.  Bit [3] – EINJv2 Extension Structure Valid, See Table 18.34. **NOTE:** If the EINJv2 Error Type bit is not set by the GET\_ERROR\_TYPE action, this bit is RESERVED and the EINJv2 Extension Structure is not present in this structure.  Bit [31:~~3~~4] - RESERVED  This field is cleared by the platform once it is consumed. |
| … | … | … | … |
| EINJv2 Extension Structure | 6 + (N \*32) | 36 | EINJv2 Extension Structure. See Table 18.34. |

**Table 18.32: Vendor Error Type Extension Structure**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Attribute** | **Description** |
| Length | 4 | ~~0x0~~0 | Set by platform. RO for Software. | Length, in bytes, of the entire Vendor Error  Type Extension Structure. |
| SBDF | 4 | ~~0x04~~4 | Set by platform. RO for Software | This provides a PCIe Segment, Bus, Device and Function number which can be used to read the Vendor ID, Device ID, and Rev ID, so that software can identify the system for error injection purposes. The platform sets this field and is RO for Software. |
| Vendor ID | 2 | ~~0x08~~8 | Set by platform. RO for Software | Vendor ID which identifies the device manufacturer. This is the same as the PCI SIG defined Vendor ID. The platform sets this field and is RO for Software. |
| Device ID | 2 | ~~0x0A~~10 | Set by platform. RO for Software | This 16-bit ID is assigned by the manufacturer that identifies this device. The platform sets this field and is RO for Software. |
| Rev ID | 1 | ~~0x0C~~12 | Set by platform. RO for Software | This 8-bit value is assigned by the manufacturer and identifies the revision number of the device. The platform sets this field and is RO for Software. |
| *Reserved* | 3 | ~~0x0D~~13 | Set by platform. RO for Software | *Reserved* |
| OEM Defined Structure | N | ~~0x10~~16 |  | The rest of the fields are defined by the OEM. NOTE: This OEM Defined Structure is only valid if Bit [31] (Vendor Defined Error Type) is set by the GET\_ERROR\_TYPE action. |

**18.6.4.1 EINJv2 Error Types**

~~EINJ version 2 (EINJv2) introduces a new error injection action The table below defines the error type codes returned from EINJV2\_GET\_ERROR\_TYPE, as well as the error type set by EINJV2\_SET\_ERROR\_TYPE~~.

~~EINJV2\_SET\_ERROR\_TYPE actions must be present as part of the EINJ Action Table. OSPM is free to choose this action for advanced error injection options. The platform will give precedence to EINJV2\_SET\_ERROR\_TYPE. That is, if a non-zero Error Type value is set by EINJV2\_SET\_ERROR\_TYPE, then any Error Type value set by SET\_ERROR\_TYPE\_WITH\_ADDRESS and/or SET\_ERROR\_TYPE will be ignored.~~

~~Also, EINJv2 breaks out the Error Type from the severity. The following table describes the new Error Type encoding.~~

If the GET\_ERROR\_TYPE action returns the DWORD with Bit [30] set, it means that EINJv2 error types are supported, and as a result the EINJV2\_~~SET~~GET\_ERROR\_TYPE action must be present in the EINJ Action Table (see Table 18.25). The following table defines the error type bitmap returned by the EINJV2\_GET\_ERROR\_TYPE action.

**Table 18.33: EINJv2 Error Type**

|  |  |
| --- | --- |
| **Bit** | **Description** |
| 0 | Processor Error |
| 1 | Memory Error |
| 2 | PCIe Error |
| ~~3-30~~ 3-31 | *Reserved* |
| ~~31~~ | ~~Vendor Defined Error~~ |

**NOTE**: Severity is not explicitly specified by the EINJv2 Error Type encoding, but rather implied by the payload in the EINJv2 Extension Structure (see Table 18.34**).**

**Table 18.34: EINJv2 Extension Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Length | ~~3~~4 | 0 | Length of the entire EINJv2 Extension Structure, in bytes.  **NOTE:** This field is Read-Only to software. |
| Revision | ~~1~~2 | ~~3~~4 | 1 – Initial Revision.  **NOTE:** This field is Read-Only to software. |
| Component Array Count (N) | 2 | 46 | This represents the number of entries in the Component Array, where 0 means no entries. The intent is to support error injection into multiple components simultaneously, where each entry represents a unique component. **NOTE:** The maximum number of entries supported by the platform can be calculated as follows:  Max Count = (EINJv2 Length – 6) / (32) |
| Component Array [] | N \* 32 | ~~6~~8 | Array of EINJv2 Component Entry Structures. See Table 18.35. |

**Table 18.35: EINJv2 Component Entry Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Component ID | 16 | 0 | Component ID definition depends on the EINJv2 Error Type.   * Processor Error (0x1):   The lower 32 bits represent the ACPI UID of the processor, as represented in MADT. The remaining bits are vendor specific.   * Memory Error (0x2):   This represents the Device ID within the memory module (e.g., DDR DIMM) for a particular system physical address. For example: 18 x 4 DIMMs support up to 18 devices (0-17) per address. 9 x 8 DIMMs support up to 9 devices (0-8) per address. It is possible to inject error syndrome into multiple devices.   * PCIe Error (0x4):   The lower 32 bits represent the SBDF, encoded like the PCIe SBDF field in Table 18.31.The remaining bits are vendor specific. |
| Component Syndrome | 16 | 16 | Component Syndrome definition depends on the EINJv2 Error Type.   * Processor Error (0x1):   The usage of these bits is vendor specific.   * Memory Error (0x2):   This indicates the bit mask of data bits to flip within a memory device. (e.g., If the set syndrome bit value is zero, it is flipped to one. And if the set syndrome bit value is one, it is flipped to zero). The range of valid bits depends on the device specified by Component ID.  **Example 1:** For a DDR4 18x4 memory device topology with a  burst length of 8 (e.g., 64-byte cache line in a single burst), there will be up to 32 valid bits per device that may be modified per burst. If bit 3 in this mask is set, then bit offset 3 in that burst will be flipped.  **Example 2:** For a DDR5 5x8 memory device topology with a burst length of 16 (e.g., 64-byte cache line in a single burst), there will be up to 128 valid bits per device that may be modified per burst.   * PCIe Error (0x4):   The usage of these bits is vendor specific. |

**NOTES:**

1) For support of vendor specific data, the “Vendor Error Type Extension Structure” must be present so that software can identify the platform (see Table 18.32).

2) If any Component ID or Component Syndrome value is not supported by the platform, the EXECUTE\_OPERATION action will fail, and the GET\_COMMAND\_STATUS action will return Invalid Access (0x2).

~~Table 18.34:~~ **~~EINJV2\_SET\_ERROR\_TYPE Data Structure~~**

|  |  |  |  |
| --- | --- | --- | --- |
| **~~Field~~** | **~~Byte~~**  **~~Length~~** | **~~Byte~~**  **~~Offset~~** | **~~Description~~** |
| ~~Error Type~~ | ~~4~~ | ~~0~~ | ~~Bit map of error types to inject (see~~ [~~Table~~](#_bookmark1489)[~~18.30~~](#_bookmark1489)~~). Only one Error~~  ~~Type bit may be set at a time. This field is cleared by the platform once it is consumed.~~ |
| ~~Error Type Code~~ | ~~1~~ | ~~4~~ | ~~Vendor specific code for each error type used to indicate error in-~~  ~~jection behavior. Value of zero is default error injection behavior as defined by EINJv2. Non-zero value indicates vendor specific be- havior.~~ |
| ~~Flags~~ | ~~3~~ | ~~5~~ | ~~This field indicates which of the remaining fields are valid: Bit 0 - Address Valid~~  ~~Bit 1 - Address Range Valid Bit 2 - Severity Valid~~  ~~Bit 3 - Component Syndrome Count and Array is valid All other bits are reserved~~ |
| ~~Length~~ | ~~4~~ | ~~8~~ | ~~This specifies the length of the entire structure including the com-~~  ~~ponent syndrome array.~~ |
| ~~Severity~~ | ~~4~~ | ~~12~~ | ~~Optional field specifying the severity of the injected error:~~   1. ~~- Corrected Error~~ 2. ~~- Uncorrected Non-Fatal~~ 3. ~~- Uncorrected Fatal~~   ~~All other values are reserved~~ |
| ~~Address~~ | ~~8~~ | ~~16~~ | ~~Optional field specifying the physical address of the memory that is~~  ~~the target for the injection. Valid if Bit [0] of the Flags field is set.~~ |
| ~~Address Range~~ | ~~8~~ | ~~24~~ | ~~Optional field specifying the physical address range mask of the~~  ~~memory that is the target for the injection. Valid if Bit [1] of the Flags field is set.~~ |
| ~~Component Syn-~~  ~~drome Count (N)~~ | ~~4~~ | ~~28~~ | ~~This represents the maximum number of components supported in~~  ~~the Component Syndrome Array. This is intended to support in- jecting an error into multiple components / devices simultaneously. For Example: If Component Syndrome Count is valid per the Flags field and the value Count (N) is 4, this structure contains the error Component Syndrome Array for 4 unique components.~~ |
| ~~Component Syn-~~  ~~drome Array~~ | ~~N \* 32~~ | ~~32~~ | ~~This is an array based on the Component Syndrome Entry Structure,~~  ~~each entry being 32-bytes as described in~~ [~~Table~~](#_bookmark1495)[~~18.35~~](#_bookmark1495)~~. The length of this structure is 32 x Component Syndrome Count.~~ |

~~Table 18.35:~~ **~~EINJV2\_SET\_ERROR\_TYPE Component Syndrome Structure~~**

|  |  |  |  |
| --- | --- | --- | --- |
| **~~Field~~** | **~~Byte~~**  **~~Length~~** | **~~Byte~~**  **~~Offset~~** | **~~Description~~** |
| ~~Component~~  ~~ID~~ | ~~16~~ | ~~0~~ | ~~Component ID definition depends on the Error Type Value. Note that because this is a union structure, the byte length is 16 bytes to accommodate the largest possible ID (i.e. FRU ID for vendor specific errors).~~  ~~Processor Error (0x1):~~  ~~The bottom 32-bit represents the ACPI ID (represented in MADT) of the processor.~~  ~~The remaining bits are vendor specific.~~  ~~Memory Error (0x2):~~  ~~This represents the Device ID within the memory module (e.g. DDR DIMM) for a particular system physical address. For example: 18 x 4 DIMMs support up to 18 devices (0-17) per~~  ~~address. 9 x 8 DIMMs support up to 9 devices (0-8) per address. It is possible to inject error syndrome into multiple device instances up to the Component Syndrome Count.~~  ~~PCIe Error (0x4):~~  ~~This represents the SBDF. Vendor Specific Error (0x80000000)~~  ~~This represents some platform specific identifier (e.g. FRU ID or other vendor specific error format).~~ |
| ~~Component~~  ~~Syndrome~~ | ~~16~~ | ~~16~~ | ~~The Component Syndrome definition depends on the Error Type Value. Processor Error (0x1):~~  ~~The usage of the syndrome bits will be vendor specific.~~  ~~Memory Error (0x2):~~  ~~This indicates bit mask of data bits that must be flipped within a memory device. (e.g. If the set syndrome bit value is zero, the bit value will be changed to one. if the set syndrome bit value is one, the bit value will be changed to zero). The range of valid bits depends on the component error injection granularity.~~  ~~Example 1: For a DDR4 18x4 memory device topology with a burst length of 8 (e.g., 64-byte cache line in a single burst), there will be up to 32 valid bits per device that may be modified per burst. If bit 3 in this mask is set, bit offset 3 of the device in that burst will be flipped.~~  ~~Example 2: For a DDR5 5x8 memory device topology with a burst length of 16 (e.g., 64-byte cache line in a single burst), there will be up to 128 valid bits per device that may be modified per burst.~~  ~~PCIe Error (0x4):~~  ~~The usage of the syndrome bits will be vendor specific.~~  ~~Vendor Specific Error (0x80000000)~~  ~~The usage of the syndrome bits will be vendor specific.~~ |

* + 1. **Error Injection Operation**

Before OSPM can use this mechanism to inject errors, it must discover the error injection capabilities of the platform by executing a GET\_ERROR\_TYPE. See [Table](#_bookmark1489) [18.30](#_bookmark1489) for a definition of error types.

After discovering the error injection capabilities, OSPM can inject and trigger an error according to the sequence described below.

Note that injecting an error into the platform does not automatically consume the error. In response to an error injection, the platform returns a trigger error action table. The software that injected the error must execute the actions in the trigger error action table ~~in order~~ to consume the error. If a specific error type is such that it is automatically consumed on injection, the platform will return a trigger error action table consisting of NO\_OP.

1. Executes a BEGIN\_INJECTION\_OPERATION action to notify the platform that an error injection operation is beginning.
2. Executes a GET\_ERROR\_TYPE action to determine the error injection capabilities of the system. This action returns a DWORD bit map of the error types supported by the platform (see [Table](#_bookmark1489) [18.30](#_bookmark1489)).
3. If GET\_ERROR\_TYPE returns the DWORD with Bit [31] set, it means that vendor defined error types are present, apart from the standard error types (see [Table](#_bookmark1489) [18.30](#_bookmark1489)).
4. If GET\_ERROR\_TYPE returns the DWORD with Bit [30] set, it means that EINJv2 error types are present, apart from the standard error types (see [Table](#_bookmark1489) [18.3](#_bookmark1489)0). In this case, OSPM executes the EINJv2\_GET\_ERROR\_TYPE action to determine the EINJv2 error injection capabilities of the system. This action returns a DWORD bit map of the error types supported by the platform (see Table 18.33).
5. OSPM chooses the type of error to inject by executing a SET\_ERROR\_TYPE or a SET\_ERROR\_TYPE\_WITH\_ADDRESS \_WITH\_ADDRESS action (see [Section](#_bookmark1488) [18.6.4](#_bookmark1488)).
   1. If the OSPM chooses to inject one of the supported standard error types, then it sets the corresponding bit in the error type bitmap. For example, if OSPM chooses to inject a “Memory Correctable” error, then the OSPM sets the value 0x0000\_0080 in the error type bitmap.
   2. If the OSPM chooses to inject one of the vendor-defined error types, then it sets bit [31] in the error type bitmap.
      * OSPM ~~exectures~~ executes the SET\_ERROR\_TYPE\_WITH\_ADDRESS\_WITH\_ADDRESS action to retrieve the location of the “SET\_ERROR\_TYPE\_WITH\_ADDRESS data structure”, to then get the location of the “Vendor Error Type Extension Structure” by reading the “Vendor Error Type Extension Structure Offset” (see [Table](#_bookmark1491) [18.32](#_bookmark1491)).
        + OSPM reads the Vendor ID, Device ID and Rev ID from the PCI config space whose path (PCIe Segment/Device/Function) is provided in the “SBDF” field of the Vendor Error Type Extension Structure.
        + If the Vendor ID/Device ID and Rev IDs match, then the OSPM can identify the platform it is running on and would know the Vendor error types that are supported by this platform.
        + The OSPM writes the vendor error type to inject in the “OEM Defined Structure” field (see [Table](#_bookmark1491) [18.32](#_bookmark1491)).
      * Optionally, for either standard or vendor-defined error types, the OSPM can choose the target of the injection, such as a memory range, PCIe Segment/Device/Function or Processor APIC ID, depending on the type of error. The OSPM does this by executing the SET\_ERROR\_TYPE\_WITH\_ADDRESS action to fill in the appropriate fields of the “SET\_ERROR\_TYPE\_WITH\_ADDRESS Data structure” (see [Table](#_bookmark1490) [18.31](#_bookmark1490)).
   3. If the OSPM chooses to inject one of the EINJv2 error types, it then executes the SET\_ERROR\_TYPE\_WITH\_ADDRESS action to fill in the appropriate fields of the “SET\_ERROR\_TYPE\_WITH\_ADDRESS Data structure” (see [Table](#_bookmark1491) [18.3](#_bookmark1491)1). The “Error Type” field is encoded according to the “EINJv2 Error Type” bit map (see [Table](#_bookmark1491) [18.3](#_bookmark1491)3), and Bit [3] of the “Flags” field is set to denote a valid “EINJv2 Extension Structure”.

For example, if OSPM chooses to inject a Memory error pattern into a device at a particular system physical address, then OSPM sets:

* + - * Error Type = 0x2 (EINJv2 Memory Error)
      * Memory Address = 0000FFFFFFF0000
      * Memory Address Range = 0x0 (No Address Range)
      * Flags = 0xA

Bit [1] – Memory Address and Memory Address Range Field Valid

Bit [3] – EINJv2 Extension Structure Valid

* + - * Component Array Count = 1
      * Component ID [0] = {00000000000000000000000000000004}
      * Component Syndrome [0] = {000000000000000000000000A5A5A5A5}

In this example software is trying to inject a 32-bit bit-flip pattern into a single device and across single burst at a particular system physical address.

1. Executes an EXECUTE\_OPERATION action to instruct the platform to begin the injection operation.
2. Busy waits by continually executing CHECK\_BUSY\_STATUS action until the platform indicates that the operation is complete by clearing the abstracted Busy bit.
3. Executes a GET\_COMMAND\_STATUS action to determine the status of the completed operation.
4. If the status indicates that the platform cannot inject errors, stop.
5. Executes a GET\_TRIGGER\_ERROR\_ACTION\_TABLE operation to get the physical pointer to the TRIG- GER\_ERROR action table. This provides the flexibility in systems where injecting an error is a two (or more) step process.
6. Executes the actions specified in the TRIGGER\_ERROR action table.
7. Execute an END\_OPERATION to notify the platform that the error injection operation is complete.

**~~18.6.7 Error Injection Version 2 Operation~~**

~~Before OSPM can use this mechanism to inject errors, it must discover the error injection capabilities of the platform by executing a EINJV2\_GET\_ERROR\_TYPE. See~~ [~~Table~~](#_bookmark1489)[~~18.30~~](#_bookmark1489) ~~for a definition of error types.~~

~~After discovering the EINJv2 error injection capabilities, OSPM can inject and trigger an error according to the sequence described below.~~

~~Note that injecting an error into the platform does not automatically consume the error. In response to an error injection, the platform returns a trigger error action table. The software that injected the error must execute the actions in the trigger error action table in order to consume the error. If a specific error type is such that it is automatically consumed on injection, the platform will return a trigger error action table consisting of NO\_OP:~~

1. ~~Executes a BEGIN\_INJECTION\_OPERATION action to notify the platform that an error injection operation is beginning.~~
2. ~~Executes a EINJ\_GET\_ERROR\_TYPE action to determine the error injection capabilities of the system. This action returns a DWORD bit map of the error types supported by the platform (see~~ [~~Table~~](#_bookmark1489)[~~18.30~~](#_bookmark1489)~~).~~
3. ~~If EINJV2\_GET\_ERROR\_TYPE returns the DWORD with Bit [30] set, it means that EINJv2 error types are present, apart from the standard error types (see~~ [~~Table~~](#_bookmark1489)[~~18.30~~](#_bookmark1489)~~).~~
4. ~~Executes a EINJV2\_GET\_ERROR\_TYPE action to determine the error injection capabilities of the system. This action returns a DWORD bit map of the error types supported by the platform (see~~ [~~Table~~](#_bookmark1493)[~~18.33~~](#_bookmark1493)~~).~~
5. ~~If EINJV2\_GET\_ERROR\_TYPE returns the DWORD with Bit [1] set, it means that memory error types are present.~~
6. ~~OSPM chooses the type of error to inject by executing a EINJV2\_SET\_ERROR\_TYPE action.~~
   1. ~~If the OSPM chooses to inject one of the supported standard error types, then it sets the corresponding bit in the error type bitmap.~~

~~For example: if OSPM chooses to inject a Memory error, then the OSPM sets the value 0x0000\_0002 in the error type bitmap. If OSPM chooses to inject a memory error pattern into a device at a particular DRAM address, the Flags will be set to 0x9 to indicate component syndrome and address are valid. OSPM will populate the Address field with the system physical address and the component syndrome count and array syndrome.~~

~~Error Type = 0x2 (Memory Error)~~

~~Error Code = 0x0 (no vendor specific behavior) Length=0x44~~

~~Severity=0 (not used) Address=0000FFFFFFF00000~~

~~Address Range=0x0~~

~~Flags = 0x9 (Address and Component Syndrome Array is Valid) Component Syndrome Count = 1~~

~~Component Syndrome Array [0] = { 00000000000000000000000000000004 , 000000000000000000000000A5A5A5A5 }~~

~~So in this example software is trying to inject a 32-bit bit flip pattern on one device across a single burst for a particular system physical address.~~

~~EINJV2\_SET\_ERROR\_TYPE=000000000000000000000000A5A5A5A5000000000000000000000000000000040000000 00000000000000FFFFFFF000000000000000000044000009000000002~~

1. ~~Executes an EXECUTE\_OPERATION action to instruct the platform to begin the injection operation.~~
2. ~~Busy waits by continually executing CHECK\_BUSY\_STATUS action until the platform indicates that the operation is complete by clearing the abstracted Busy bit.~~
3. ~~Executes a GET\_COMMAND\_STATUS action to determine the status of the completed operation.~~
4. ~~If the status indicates that the platform cannot inject errors, stop.~~