Windows Hardware Error Architecture ACPI Table Specification

Version 1.0 — November 5, 2007

Abstract

This specification contains details of four Advanced Configuration and Power Interface (ACPI) tables created for use with the Windows Hardware Error Architecture (WHEA) feature introduced in the Windows Server® 2008 operating system.

This specification does not include details on how to program or use the tables. That information can be found in the WHEA Design Guide document, which can be requested from Microsoft by sending email to [wheafb@microsoft.com](http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/wheafb%40microsoft.com).

|  |
| --- |
| © 2007 Microsoft Corporation. All rights reserved. By downloading, printing, copying, or otherwise using this document or any of the ACPI tables contained herein, you agree to be bound by the terms of the license agreement included here.  |

**WINDOWS HARDWARE ERROR ARCHITECTURE (WHEA) ACPI TABLE SPECIFICATION LICENSE**

**IMPORTANT—READ CAREFULLY: This Microsoft License Agreement ("Agreement") is a legal agreement between you (either an individual or a single entity) and Microsoft Corporation for the Microsoft specification identified above ("Specification"). BY DOWNLOADING, COPYING OR OTHERWISE USING THE SPECIFICATION, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT DOWNLOAD, COPY, OR USE THE SPECIFICATION.**

The Specification is owned by Microsoft or its suppliers and is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. **THE SPECIFICATION IS LICENSED, NOT SOLD.**

1.         **GRANT OF LICENSE**.

(a)   Provided that you comply with all terms and conditions of this Agreement, Microsoft grants to you the following non-exclusive, worldwide, royalty-free, perpetual, non-transferable, non-sublicensable, limited license under any copyrights or patents that cover the table values described in the Specification and that are owned or licensable by Microsoft without payment of consideration to third parties,

(i)     to reproduce copies of the Specification for your and your contractor’s internal use for the sole purpose of (1) modifying your firmware and/or BIOS for computing devices (**"Firmware"**) so that it writes to memory the appropriate table values in the Specification or (2) modifying your software so that it may read from memory the appropriate table values (the **"Purpose"**),

(ii)    to implement the table values in your firmware and/or BIOS,

(iii)   to license to third parties directly and indirectly the table values as part of your Firmware (and any related documentation),

The foregoing license is granted only to the extent necessary to accomplish the Purpose and to license and/or distribute your Firmware containing the table values to third parties. The foregoing license shall not extend to any features of your Firmware that (i) are not required to comply with the Specification or (ii) to which there was a practicable alternative to infringing a patent.

(b)    Microsoft reserves all other rights it may have in the Specification, its implementation and any intellectual property therein.  The furnishing of this document does not give you or any other entity any license to any other Microsoft patents, trademarks, copyrights or other intellectual property rights.  Microsoft does not grant to you or any other entity any implied licenses or rights whatsoever under this Agreement.  Specifically, this Agreement does not grant any express or implied licenses or rights to any enabling technologies that may be necessary to fully utilize the tables or the table values described in the Specification.

2.    **ADDITIONAL LIMITATIONS AND OBLIGATIONS**.

(a)    You may use all or some of the tables provided, however for each table that you use, You must implement such table in its entirety (i.e. all fields) and without modification (e.g., byte length, offset, and permissible values as described in the Specification).

(b)    Your license rights to the Specification are conditioned upon you not creating, modify, or distributing your Firmware in a way that such creation, modification, or distribution may (a) create, or purport to create, obligations for Microsoft with respect to the Specification (or intellectual property therein) or (b) grant, or purport to grant, to any third party any rights or immunities to Microsoft’s intellectual property or proprietary rights in the Specification.

(c)    The foregoing license is applicable only to the version of the Specification which you are about to download. It does not apply to any additional versions of or extensions to the Specification.

(d)    Without prejudice to any other rights, Microsoft may terminate this Agreement if you fail to comply with the terms and conditions of this Agreement. In such event you must destroy all copies of the Specification and must not further distribute the table values.

3.     **INTELLECTUAL PROPERTY RIGHTS**. All ownership, title and intellectual property rights in and to the Specification, and any copies you are permitted to make herein, are owned by Microsoft or its suppliers

4.**DISCLAIMER OF WARRANTIES. To the maximum extent permitted by applicable law, Microsoft and its suppliers provide the Specification (and all intellectual property therein) and any (if any) support services related to the Specification ("Support Services") *AS IS AND WITH ALL FAULTS*, and hereby disclaim all warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties or conditions of merchantability, of fitness for a particular purpose, of lack of viruses, of accuracy or completeness of responses, of results, and of lack of negligence or lack of workmanlike effort, all with regard to the Specification, any intellectual property therein and the provision of or failure to provide Support Services. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT, WITH REGARD TO THE SPECIFICATION AND ANY INTELLECTUAL PROPERTY THEREIN. THE ENTIRE RISK AS TO THE QUALITY OF OR ARISING OUT OF USE OR PERFORMANCE OF THE SPECIFICATION, ANY INTELLECTUAL PROPERTY THEREIN, AND SUPPORT SERVICES, IF ANY, REMAINS WITH YOU.**

5.    **EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. To the maximum extent permitted by applicable law, in no event shall Microsoft or its suppliers be liable for any special, incidental, indirect, or consequential damages whatsoever (including, but not limited to, damages for loss of profits or confidential or other information, for business interruption, for personal injury, for loss of privacy, for failure to meet any duty including of good faith or of reasonable care, for negligence, and for any other pecuniary or other loss whatsoever) arising out of or in any way related to the use of or inability to use the SPECIFICATION, ANY INTELLECTUAL PROPERTY THEREIN, the provision of or failure to provide Support Services, or otherwise under or in connection with any provision of this AGREEMENT, even in the event of the fault, tort (including negligence), strict liability, breach of contract or breach of warranty of Microsoft or any supplier, and even if Microsoft or any supplier has been advised of the possibility of such damages.**

6.    **LIMITATION OF LIABILITY AND REMEDIES. Notwithstanding any damages that you might incur for any reason whatsoever (including, without limitation, all damages referenced above and all direct or general damages), the entire liability of Microsoft and any of its suppliers under any provision of this Agreement and your exclusive remedy for all of the foregoing shall be limited to the greater of the amount actually paid by you for the Specification or U.S.$5.00. The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails its essential purpose.**

7.    **APPLICABLE LAW**. If you acquired this Specification in the United States, this Agreement is governed by the laws of the State of Washington. In respect of any dispute which may arise hereunder, you consent to the jurisdiction of the state and federal courts sitting in King County, Washington.

8. **ENTIRE AGREEMENT. This Agreement is the entire agreement between you and Microsoft relating to the Specification supersedes all prior or contemporaneous oral or written communications, proposals and representations with respect to the Specification or any other subject matter covered by this Agreement. To the extent the terms of any Microsoft policies or programs for Support Services conflict with the terms of this Agreement, the terms of this Agreement shall control.**

Document History

| **Date** | **Change** |  |  |  |
| --- | --- | --- | --- | --- |
| November 5, 2007 | Version 1.0 - First publication |

Contents

[Introduction 1](#_Toc182018507)

[1. Error Record Serialization Table (ERST) 1](#_Toc182018508)

[2. BOOT Error Record Table (BERT) 4](#_Toc182018509)

[3. Hardware Error Source Table (HEST) 6](#_Toc182018510)

[4. Error Injection Table (EINJ) 6](#_Toc182018511)

List of Tables

[Table 1-1 Error Record Serialization Table (ERST) 1](#_Toc182018515)

[Table 1-2 Error Record Serialization Actions 2](#_Toc182018516)

[Table 1-3 Error Record Serialization Instructions 3](#_Toc182018517)

[Table 2‑1 BOOT Error Record Table (BERT) 4](#_Toc182018518)

[Table 2‑2 BOOT Error Region 5](#_Toc182018519)

[Table 3‑1 Hardware Error Source Table (HEST) 6](#_Toc182018520)

[Table 4‑1 Error Injection Table (EINJ) 6](#_Toc182018521)

[Table 4‑2 Error Injection Actions 7](#_Toc182018522)

[Table 4-3 Injection Instruction Entry 8](#_Toc182018523)

[Table 4-4 Injection Instructions 8](#_Toc182018524)

# Introduction

This specification contains details of four Advanced Configuration and Power Interface (ACPI) tables created for use with the Windows Hardware Error Architecture (WHEA) feature introduced in the Windows Server® 2008 operating system.

This specification does not include details on how to program or use the tables. That information can be found in the WHEA Design Guide document, which can be requested from Microsoft by sending email to [wheafb@microsoft.com](http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/wheafb%40microsoft.com).

Four tables are included in this specification:

1. Error Record Serialization Table (ERST)

2. BOOT Error Record Table (BERT)

3. Hardware Error Source Table (HEST)

4. Error Injection Table (EINJ)

# 1. Error Record Serialization Table (ERST)

This table is read and processed by the operating system (OS) during boot. The information in the ERST supplies the operating system with the information necessary to communicate with the platform’s error record serialization hardware.

Table 1-1 details the layout of the ERST which system firmware is responsible for building.

Table 1-1 Error Record Serialization Table (ERST)

| **Field** | **Byte length** | **Byte offset** | **Description** |
| --- | --- | --- | --- |
| ***ACPI Standard Header*** |  |  |  |
| Header Signature | 4 | 0x0 | ERST. Signature for the Error Record Serialization Table. |
| Length | 4 | 0x4 | Length, in bytes, of entire ERST. Entire table must be contiguous. |
| Revision | 1 | 0x8 | 1 |
| Checksum | 1 | 0x9 | Entire table must sum to zero. |
| OEMID | 6 | 0xA | OEM ID. |
| OEM Table ID | 8 | 0x10 | The manufacturer model ID. |
| OEM Revision | 4 | 0x18 | OEM revision of the ERST for the supplied OEM table ID. |
| Creator ID | 4 | 0x1C | Vendor ID of the utility that created the table.  |
| Creator Revision | 4 | 0x20 | Revision of the utility that created the table.  |
| ***Serialization Header*** |  |  |  |
| Serialization Header Size | 4 | 0x24 | Length in bytes of the serialization header. |
| Reserved | 4 | 0x28 | Must be zero. |
| Instruction Entry Count | 4 | 0x2c | The number of Serialization Instruction Entries in the Serialization Action Table. |
| ***Serialization Action Table*** |  |  |  |
| Serialization Instruction Entries |  | 0x30 | A series of error logging instruction entries. |

A Serialization Action is defined as a series of Serialization Instructions on registers that result in a well known action. A Serialization Instruction is a Serialization Action primitive and consists of either reading or writing an abstracted hardware register.

Table 1-2 identifies the supported error record Serialization Actions. Table 1-3 identifies the supported error record Serialization Instructions.

Table 1-2 Error Record Serialization Actions

| **Value** | **Name** | **Description** |
| --- | --- | --- |
| 0x0 | BEGIN\_WRITE\_OPERATION | Indicates to the platform that an error record write operation is beginning. This allows the platform to set its operational context. |
| 0x1 | BEGIN\_READ\_OPERATION | Indicates to the platform that an error record read operation is beginning. This allows the platform to set its operational context. |
| 0x2 | BEGIN\_CLEAR\_OPERATION | Indicates to the platform that an error record clear operation is beginning. This allows the platform to set its operation context. |
| 0x3 | END\_OPERATION | Indicates to the platform that the current error record operation has ended. This allows the platform to clear its operational context. |
| 0x4 | SET\_RECORD\_OFFSET | Sets the offset from the base of the *Error Log Address Range* to or from which the platform is to transfer an error record. |
| 0x5 | EXECUTE\_OPERATION | Instructs the platform to carry out the current operation based on the current operational context. |
| 0x6 | CHECK\_BUSY\_STATUS | Returns the state of the current operation. Once an operation has been executed through the EXECUTE\_OPERATION action, the platform is required to return an indication that the operation is busy until the operation is completed. This allows the platform to poll for completion by repeatedly executing the CHECK\_BUSY\_STATUS action until the platform indicates that the operation not busy. |
| 0x7 | GET\_COMMAND\_STATUS | Returns the status of the current operation. The platform is expected to maintain a status code for each operation. |
| 0x8 | GET\_RECORD\_IDENTIFIER | Returns the record identifier of an existing error record on the persistent store. The error record identifier is a 64-bit unsigned value as defined in Appendix N of version 2.1 of the UEFI specification. If the record store is empty, this action must return 0xffffffffffffffff. |
| 0x9 | SET\_RECORD\_IDENTIFIER | Sets the record identifier. The error record identifier is a 64-bit unsigned value as defined in Appendix N of version 2.1 of the UEFI specification. |
| 0xA | GET\_RECORD\_COUNT | Retrieves the number of error records currently stored on the platforms persistent store. The platform is expected to maintain a count of the number of error records resident in its persistent store. |
| 0xB | BEGIN\_DUMMY\_WRITE\_OPERATION | Indicates to the platform that a dummy error record write operation is beginning. This allows the platform to set its operational context. A dummy error record write operation performs no actual transfer of information from the *Error Log Address Range* to the persistent store. |
| 0xC | RESERVED | Reserved. |
| 0xD | GET\_ERROR\_LOG\_ADDRESS\_RANGE | Returns the 64-bit physical address the OS is to use as the buffer for reading/writing error records. |
| 0xE | GET\_ERROR\_LOG\_ADDRESS\_RANGE\_LENGTH | Returns the length in bytes of the Error Log Address Range. |
| 0xF | GET\_ERROR\_LOG\_ADDRESS\_RANGE\_ATTRIBUTES | Returns attributes that describe the behavior of the error log address range:Bit 0 (0x1) – Reserved.Bit 1 (0x2) – Non-Volatile: Indicates that the error log address range is in non-volatile RAM.Bit 2 (0x4) – Slow: Indicates that the memory in which the error log address range is locates has slow access times.All other bits reserved. |

Table 1-3 Error Record Serialization Instructions

| **Value** | **Name** | **Description** |
| --- | --- | --- |
| 0x00 | READ\_REGISTER | A READ\_REGISTER instruction reads the designated information from the specified Register Region. |
| 0x01 | READ\_REGISTER\_VALUE | A READ\_REGISTER\_VALUE instruction reads the designated information from the specified Register Region and compares the results with the contents of the Value field. If the information read matches the contents of the Value field, TRUE is returned, else FALSE is returned. |
| 0x02 | WRITE\_REGISTER | A WRITE\_REGISTER instruction writes a value to the specified Register Region. The Value field is ignored. |
| 0x03 | WRITE\_REGISTER\_VALUE | A WRITE\_REGISTER\_VALUE instruction writes the contents of the Value field to the specified Register Region. |
| 0x04 | NOOP | This instruction is a NOOP. |
| 0x05 | LOAD\_VAR1 | Loads the VAR1 variable from the register region. |
| 0x06 | LOAD\_VAR2 | Loads the VAR2 variable from the register region. |
| 0x07 | STORE\_VAR1 | Stores the value in VAR2 to the indicate register region. |
| 0x08 | ADD | Adds VAR1 and VAR2 and stores the result in VAR1. |
| 0x09 | SUBTRACT | Subtracts VAR1 from VAR2 and stores the result in VAR1. |
| 0x0A | ADD\_VALUE | Adds the contents of the specified register region to Value and stores the result in the register region. |
| 0x0B | SUBTRACT\_VALUE | Subtracts Value from the contents of the specified register region and stores the result in the register region. |
| 0x0C | STALL | Stall for the number of microseconds specified in Value. |
| 0x0D | STALL\_WHILE\_TRUE | The OS will continually compare the contents of the specified register region to Value until the values are not equal. The OS will terminate the stall if the elapsed time of the stall exceeds the value stored in VAR1. The value in VAR1 is specified in microseconds. |
| 0x0E | SKIP\_NEXT\_INSTRUCTION\_IF\_TRUE | This is a control instruction which compares the contents of the register region with Value. If the values match, the OS skips the next instruction in the sequence for the current action. |
| 0x0F | GOTO | The OS will go to the instruction specified by Value. The instruction is specified as the zero-based index. Each instruction for a given action has an index based on its relative position in the array of instructions for the action. |
| 0x10 | SET\_SRC\_ADDRESS\_BASE | Sets the SRC\_BASE variable used by the MOVE\_DATA instruction to the contents of the register region. |
| 0x11 | SET\_DST\_ADDRESS\_BASE | Sets the DST\_BASE variable used by the MOVE\_DATA instruction to the contents of the register region. |
| 0x12 | MOVE\_DATA | Moves VAR2 bytes of data from SRC\_BASE + Offset to DST\_BASE + Offset, where Offset is the contents of the register region. |

# 2. BOOT Error Record Table (BERT)

The BOOT Error Record Table (BERT) is used by the firmware to notify the OS during boot that the system previously experienced an error condition and either crashed or was shutdown. This is used in cases where the OS is unable to process the error at the time it occurs, e.g. where the BMC decided to reset the system. The OS is unable to create an error record describing this class of condition. The BOOT Error Record Table (BERT) format is shown in Table 2-1.

Table 2‑1 BOOT Error Record Table (BERT)

| **Field** | **Byte length** | **Byte offset** | **Description** |
| --- | --- | --- | --- |
| Header Signature | 4 | 0 | ‘BERT’. Signature for the BOOT Error Record Table. |
| Length | 4 | 4 | Length, in bytes, of BERT. |
| Revision | 1 | 8 | 1 |
| Checksum | 1 | 9 | Entire table must sum to zero. |
| OEMID | 6 | 10 | OEM ID. |
| OEM Table ID | 8 | 16 | The manufacturer model ID. |
| OEM Revision | 4 | 24 | OEM revision of the BERT for the supplied OEM table ID. |
| Creator ID | 4 | 28 | Vendor ID of the utility that created the table.  |
| Creator Revision | 4 | 32 | Revision of the utility that created the table.  |
| BOOT Error Region Length | 4 | 36 | The length in bytes of the boot error region. |
| BOOT Error Region | 8 | 40 | 64-bit physical address of the BOOT Error Region. |

The BOOT Error Region is a range of addressable memory the OS can access during initialization to determine if an unhandled error condition occurred. The format of the Boot Error Region is shown in Table 2-2.

Table 2‑2 BOOT Error Region

| **Field** | **Byte length** | **Byte offset** | **Description** |
| --- | --- | --- | --- |
| Block Status | 4 | 0 | Indicates to the OS what kind of error information is reported in the error packet:Bit 0 – Uncorrectable Error Valid: If set to one, indicates that an uncorrectable error condition exists.Bit 1 – Correctable Error Valid: If set to one, indicates that a correctable error condition exists.Bit 2 – Multiple Uncorrectable Errors: If set to one, indicates that more than one uncorrectable errors have been detected.Bit 3 – Multiple Correctable Errors: If set to one, indicates that more than one correctable errors have been detected.Bit 4–13 – Error Data Entry Count: This value indicates the number of Error Data Entries found in the *Data* section.Bit 14–31 – Reserved. |
| Raw Data Offset | 4 | 4 | Offset in bytes from the beginning of the Error Status Block to raw error data. The raw data must follow any *Generic Error Data Entries*. |
| Raw Data Length | 4 | 8 | Length in bytes of the raw data. |
| Data Length | 4 | 12 | Length in bytes of the generic error data. |
| Error Severity | 4 | 16 | Identifies the error severity of the reported error:0 – Recoverable1 – Fatal2 – Corrected3 – NoneNote: This is the error severity of the entire event. Each Generic Error Data Entry also includes its own Error Severity field. |
| Generic Error Data | Data Length | 20 | The information contained in this field is a collection of zero or more Generic Error Data Entries. |

# 3. Hardware Error Source Table (HEST)

The Hardware Error Source Table provides the platform firmware a way to describe a system’s hardware error sources to the OS.

The format of the Hardware Error Source Table is shown in Table 3-1.

Table 3‑1 Hardware Error Source Table (HEST)

| **Field** | **Byte length** | **Byte offset** | **Description** |
| --- | --- | --- | --- |
| Header Signature | 4 | 0 | ‘HEST’. Signature for the Error Record Serialization Table. |
| Length | 4 | 4 | Length, in bytes, of entire HEST. Entire table must be contiguous. |
| Revision | 1 | 8 | 1 |
| Checksum | 1 | 9 | Entire table must sum to zero. |
| OEMID | 6 | 10 | OEM ID. |
| OEM Table ID | 8 | 16 | The manufacturer model ID. |
| OEM Revision | 4 | 24 | OEM revision of the HEST for the supplied OEM table ID. |
| Creator ID | 4 | 28 | Vendor ID of the utility that created the table.  |
| Creator Revision | 4 | 32 | Revision of the utility that created the table.  |
| Error Source Count | 4 | 36 | The number of error source descriptors. |
| Error Source Structure[n] | - | 40 | A series of Error Source Descriptor Entries. |

The layout of the supported Error Source Descriptor Entry types can be found in the WHEA Design Guide document.

# 4. Error Injection Table (EINJ)

The Error Injection Table provides a generic interface mechanism through which the OS can inject hardware errors to the platform without requiring platform specific OS level software. System firmware is responsible for building this table, which is made up of Injection Instruction entries. Table 4‑1 details the layout of the table.

Table 4‑1 Error Injection Table (EINJ)

| **Field** | **Byte length** | **Byte offset** | **Description** |
| --- | --- | --- | --- |
| ***ACPI Standard Header*** |  |  |  |
| Header Signature | 4 | 0x0 | EINJ. Signature for the Error Record Injection Table. |
| Length | 4 | 0x4 | Length, in bytes, of entire EINJ. Entire table must be contiguous. |
| Revision | 1 | 0x8 | 1 |
| Checksum | 1 | 0x9 | Entire table must sum to zero. |
| OEMID | 6 | 0xA | OEM ID. |
| OEM Table ID | 8 | 0x10 | The manufacturer model ID. |
| OEM Revision | 4 | 0x18 | OEM revision of EINJ. |
| Creator ID | 4 | 0x1C | Vendor ID of the utility that created the table.  |
| Creator Revision | 4 | 0x20 | Revision of the utility that created the table.  |
| ***Injection Header*** |  |  |  |
| Injection Header Size | 4 | 0x24 | Length in bytes of the Injection Interface header. |
| Reserved | 1 | 0x28 |  |
| Injection Entry Count | 4 | 0x2c | The number of Instruction Entries in the Injection Action Table |
| ***Injection Action Table*** |  |  |  |
| Injection Instruction Entries |  | 0x30 | A series of error injection instruction entries |

Table 4‑2 identifies the supported error injection actions.

Table 4‑2 Error Injection Actions

| **Value** | **Name** | **Description** |
| --- | --- | --- |
| 0x0 | BEGIN\_INJECTION\_OPERATION | Indicates to the platform that an error injection is beginning. This allows the platform to set its operational context. |
| 0x1 | GET\_TRIGGER\_ERROR\_ACTION\_TABLE | Returns a 64-bit physical memory pointer to the TRIGGER\_ERROR action table. The TRIGGER\_ERROR action instructions when executed by software trigger the error that was injected by the immediately prior SET\_ERROR\_TYPE action. |
| 0x2 | SET\_ERROR\_TYPE | Type of error to Inject. Only one ERROR\_TYPE can be injected at any given time. If there is request for multiple injections at the same time, then the platform will return an error condition. |
| 0x3 | GET\_ERROR\_TYPE | Returns the error injection capabilities of the platform. |
| 0x4 | END\_OPERATION | Indicates to the platform that the current injection operation has ended. This allows the platform to clear its operational context. |
| 0x5 | EXECUTE\_OPERATION | Instructs the platform to carry out the current operation based on the current operational context. |
| 0x6 | CHECK\_BUSY\_STATUS | Returns the state of the current operation. Once an operation has been executed through the EXECUTE\_OPERATION action, the platform is required to return an indication that the operation is busy until the operation is completed. This allows software to poll for completion by repeatedly executing the CHECK\_BUSY\_STATUS action until the platform indicates that the operation is complete by setting not busy. The lower most bit (bit0) of the returned value indicates the busy status by setting it to 1 and not busy status by setting it to 0. |
| 0x7 | GET\_COMMAND\_STATUS | Returns the status of the current operation. The platform is expected to maintain a status code for each operation. Bits 1:8 of the returned value indicate the command status. |
| 0xFF | TRIGGER\_ERROR | This is not a true error injection action. In response to error injection, the platform returns a trigger error action table. This table consists of a series of injection instruction entries where the injection action is set to TRIGGER\_ERROR to distinguish such entries. |

An Injection action consists of a series of one or more Injection Instructions. Table 4-3 details the layout of an Injection Instruction Entry.

Table 4-3 Injection Instruction Entry

| **Field** | **Byte length** | **Byte offset** | **Description** |
| --- | --- | --- | --- |
| Injection Action | 1 | N | The injection action that this instruction is a part of. See Table 4‑2 for supported injection actions. |
| Instruction | 1 | N+0x1 | Identifies the instruction to execute. See Table 4-4 for a list of valid instructions. |
| Flags | 1 | N+0x2 | Flags that qualify the instruction. |
| Reserved | 1 | N+0x3 |  |
| Register Region | 12 | N+0x4 | Generic address structure as defined in section 5.2.3.1 of the ACPI specification to describe the address and bit.Address\_Space\_ID must be 0 (System Memory) or 1 (System IO). This constraint is an attempt to ensure that the registers are accessible in the presence of hardware error conditions. |
| Value | 8 | N+0x10 | This is the value field that is used by the instruction READ or WRITE\_REGISTER\_VALUE. |
| Mask | 8 | N+0x18 | The bit mask required to obtain the bits corresponding to the injection instruction in a given bit range defined by the register region. |

Table 4-4 lists the supported Injection Instructions.

Table 4-4 Injection Instructions

| **Op code** | **Instruction name** | **Description** |
| --- | --- | --- |
| 0x00 | READ\_REGISTER | A READ\_REGISTER instruction reads the designated instruction from the specified register region. |
| 0x01 | READ\_REGISTER\_VALUE | A READ\_REGISTER\_VALUE instruction reads the designated information from the specified Register Region and compares the results with the contents of the Value field. If the information read matches the contents of the Value field, TRUE is returned, else FALSE is returned. |
| 0x02 | WRITE\_REGISTER | A WRITE\_REGISTER instruction writes a value to eh specified Register Region. The Value field is ignored. |
| 0x03 | WRITE\_REGISTER\_VALUE | A WRITE\_REGISTER\_VALUE instruction writes the contents of the Value field to the specified Register Region. |
| 0x04 | NOOP | No operation. |