Unified Extensible Firmware Interface  
Engineering Change Request (ECR)

*Draft for Review*

Title: Add new interface to retrieve device-specific platform policy in the EFI\_PCI\_PLATFORM\_PROTOCOL & EFI\_PCI\_OVERRIDE\_PROTOCOL, and introduce EFI encodings for the other PCI features and its attributes

Document:

*Ashraf Javeed*

*Submission for Review Date: 05/28/2019*

Review Approval Date: *x/xx/200x*

Submission for Technical Editing Date: *x/xx/200x*

Submission for Draft Review Date: *x/xx/200x*

Verification Date: *x/xx/200x*

Verifier: *firstname lastname, company*

Summary

## Summary of Change

The PI Spec 1.7 Vol 5 [[1](#PI_spec_volume)] defines the EFI\_PCI\_PLATFORM\_PROTOCOL & EFI\_PCI\_OVERRIDE\_PROTOCOL, and both Protocols define the interface *GetPlatformPolicy()* function to retrieve platform policies, which are global to the system being booted. However, these Protocols do not have per component based platform policies which can be used during PCI enumeration.

The proposal is to define new interface to these PCI Platform Protocol and PCI Override Protocol which would have the capability to retrieve component specific platform policies, which could be utilized to configure during PCI enumeration.

## Benefits of the Change

This change would help the PCI Bus driver and the PCI Host Bridge Resource Allocation Protocol driver to retrieve platform policies specific to any integrated PCI device / Root Port components in the chipset, which could be utilized during PCI enumeration.

This introduction of a new interface would greatly help extending the PCI Bus driver to support additional PCI-compliant features. For example:

1. Maximum Payload size (MPS): Programming common data packet size in the entire PCI hierarchy originating from the PCI Root Port of the chipset, or it can be reduced to common value based on the platform policy for that chipset’s integrated Root Port and its downstream devices in the hierarchy.
2. Maximum Read Request Size (MRRS): programming this value if supported by the component based on its platform policy.

## Impact of Change

The introduction of this new interface will require new GUID to the PCI Platform Protocol and PCI Override Protocol, although it maintains backward compatibility for existing four interfaces.

The UEFI-based platform firmware would require to use this new GUID and its definition header to produce these PCI protocols.

The EDK2 kernel code driver, the PCI Bus driver is expected to adopt this new PCI protocol definition to configure the new PCI complaint features (MPS/MRRS, etc.).

This ECR shall define new GUID for the PCI Platform Protocol and the PCI Override Protocol, along with additional member field to track the protocol version for future definition extensions purpose.

This ECR defines new data type to collect device-specific platform policies (like the EFI\_PCI\_PLATFORM\_POLICY [[2](#PI_spec_volume_chapter)]) to support various PCI-compliant features, like that of MPS and MRRS, to enable the scoping of PCI Bus driver and PCI Host Bridge Resource Allocation protocol drivers to support these features in future.

## Backward Compatibility

This definition does not alter anything in the existing legacy interfaces definition, and its functionality will remain same as defined in the UEFI PI Specification.

Since the new definition of the protocol is required to be published in place of existing legacy definition; the proposal is to move the existing definition of the PCI Platform Protocol / PCI Override Protocol in to the Addendum section of the UEFI PI specification, marked for deprecation in near future. This is primarily to ensure the producers of the protocol have enough time to migrate to this new definition.

The Platform Firmware would be the producer of the new PCI Platform Protocol; and they can choose to produce only the legacy definition of this protocol until it is defined in UEFI PI specification.

The consumer of this protocol is the EDK2 kernel core PCI Bus driver; and the proposal for the consumer is to maintain backward compatibility by checking for the protocol published in platform, in the order as stated - first locate this new PCI Platform Protocol definition, if found continue to use only the new definition. If not found, then locate the legacy protocol definition for use (as it is implemented today).

If the platform firmware does not have the need to use the new PCI device policy interface of this new definition, as it does not want the EDK2 core PCI Bus driver to configure the new PCI complaint features whose device policy is acquired through this new interface; even in this situation the platform firmware can migrate to produce the legacy interfaces for this new protocol definition. The proposal is for the consumer of this protocol (who would eventually own the configuration of the new PCI complaint features) to provide feature separation facility, like through a PCD [[RFC](#RFC_PCI_features)] which the platform firmware can use to suppress the configuration of the new PCI features, thereby, it can avoid producing the device-specific policies for the new interface of this protocol.

## References

* **Industry Specifications**:-
* [UEFI Platform Initialization Specification Version 1.7](https://uefi.org/sites/default/files/resources/PI_Spec_1_7_final_Jan_2019.pdf)

[1] UEFI PI 1.7 volume 5

[2] UEFI PI 1.7 volume 5 chapter 11.6.1

* [UEFI Specification Version 2.8](https://uefi.org/sites/default/files/resources/UEFI_Spec_2_8_final.pdf)
* [PCI Express Base Specification Revision 4.0 Version 1.0](https://members.pcisig.com/wg/PCI-SIG/document/10912?downloadRevision=active)
* **EDK-2 GitHub References**:-
  + Feature Request: [BZ-1954](https://bugzilla.tianocore.org/show_bug.cgi?id=1954)
  + Edk2-staging branch: :- [UEFI\_PCI\_ENHANCE-2](https://github.com/tianocore/edk2-staging/tree/UEFI_PCI_ENHANCE-2)
  + Code changes Patch submissions: [Patch V3](https://github.com/tianocore/edk2-staging/commit/7b44b9b7d551ec4ffc9f4b2c2029eb36f99b1753), [V4](https://github.com/tianocore/edk2-staging/commit/144d2d3d0a4306266de12a521d617c2cf975bec9)
  + RFC for the new PCI feature configuration: [MPS, MRRS](https://bugzilla.tianocore.org/show_bug.cgi?id=2194), [RO, NS, CTO](https://bugzilla.tianocore.org/show_bug.cgi?id=2313).

Detailed Description of the Change  
and Special Instructions

## Redefinition of EFI\_PCI\_PLATFORM\_PROTOCOL / EFI\_PCI\_OVERRIDE\_PROTOCOL

Add new GUIDs and rename the PCI Platform Protocol to EFI\_PCI\_PLATFORM\_PROTOCOL2 and rename the PCI Override Protocol to EFI\_PCI\_OVERRIDE\_PROTOCOL2.

The new members are highlighted in yellow.

**EFI\_PCI\_PLATFORM\_PROTOCOL2**

**Summary**

This protocol provides the interface between the PCI bus driver/PCI Host Bridge Resource Allocation driver and a platform-specific driver to describe the unique features of a platform. This protocol is optional.

**GUID**

**#define EFI\_PCI\_PLATFORM\_PROTOCOL2\_GUID \**

**{ 0x787b0367, 0xa945, 0x4d60, 0x8d, 0x34, 0xb9, 0xd1, 0x88, \**

**0xd2, 0xd0, 0xb6)**

**Protocol Interface Structure**

**typedef struct \_EFI\_PCI\_PLATFORM\_PROTOCOL2 {**

**EFI\_PCI\_PLATFORM\_PHASE\_NOTIFY** *PlatformNotify;*

**EFI\_PCI\_PLATFORM\_PREPROCESS\_CONTROLLER** *PlatformPrepController***;**

**EFI\_PCI\_PLATFORM\_GET\_PLATFORM\_POLICY** *GetPlatformPolicy***;**

**EFI\_PCI\_PLATFORM\_GET\_PCI\_ROM** *GetPciRom***;**

**EFI\_PCI\_PLATFORM\_GET\_DEVICE\_POLICY** *GetDevicePolicy;*

**UINT8** *MajorVersion*;

**UINT8** *MinorVersion*;

**} EFI\_PCI\_PLATFORM\_PROTOCOL2;**

**Parameters**

*PlatformNotify*

The notification from the PCI bus enumerator to the platform that it is about to enter a certain phase during the enumeration process. See the **PlatformNotify()** function description (in PI Specification version 1.7).

*PlatformPrepController*

The notification from the PCI bus enumerator to the platform for each PCI controller at several predefined points during PCI controller initialization. See the **PlatformPrepController()** function description (in PI Specification version 1.7).

*GetPlatformPolicy*

Retrieves the platform policy regarding enumeration. See the **GetPlatformPolicy()** function description (in PI Specification version 1.7).

*GetPciRom*

Gets the PCI device’s option ROM from a platform-specific location. See the **GetPciRom()** function description (in PI Specification version 1.7).

*GetDevicePolicy*

Retrieves the device policy regarding enumeration. See the **GetDevicePolicy()** function description

*MajorVersion*

The major version of this PCI Platform Protocol

*MinorVersion*

The minor version of this PCI Platform Protocol

**Description**

The **EFI\_PCI\_PLATFORM\_PROTOCOL2** is published by a platform-aware driver. This protocol is optional; see PCI Platform Protocol Overview in Design Discussion (in PI Specification version 1.7) for scenarios in which this protocol is required. There cannot be more than one instance of this protocol in the system. If the PCI bus driver detects the presence of this protocol before enumeration, it will use the PCI Platform Protocol to obtain information about the platform policy. The PCI bus driver will use this protocol to get the PCI device's option ROM from a platform-specific location in storage. It will also call the various member functions of this protocol at predefined points during PCI bus enumeration. The member functions can be used for performing any platform-specific initialization that is appropriate during the enumeration phase.

The PCI Bus driver shall use the new interface ***GetDevicePolicy()*** to retrieve the device-specific platform policy, and it shall do as the last stage of the PCI enumeration, after installing the PCI IO Protocol to obtain the EFI handle for each of the PCI devices; thus the new interface can use the EFI handle as the input parameter to the platform to identify the device/controller.

As per the present definition and specification of this protocol, the major version is 1, and minor version is 1. Any driver utilizing this protocol shall use these versions number to maintain the backward compatibility as per its specification changes in future.

**EFI\_PCI\_OVERRIDE\_PROTOCOL2**

**Summary**

This protocol provides the interface between the PCI bus driver/PCI Host Bridge Resource Allocation driver and an implementation's driver to describe the unique features of a platform. This protocol is optional.

**GUID**

**#define EFI\_PCI\_OVERRIDE2\_GUID \**

**{ 0xb9d5ea1, 0x66cb, 0x4546, { 0xb0, 0xbb, 0x5c, 0x6d, \**

**0xae, 0xd9, 0x42, 0x47 } }**

**Protocol Interface Structure**

**typedef EFI\_PCI\_PLATFORM\_PROTOCOL2 EFI\_PCI\_OVERRIDE\_PROTOCOL2;**

**Description**

The PCI Override Protocol is published by an implementation aware driver. This protocol is optional. But it must be called, if present, during PCI enumeration. There cannot be more than one instance of this protocol in the system.

If the PCI bus driver detects the presence of this protocol before bus enumeration, it will use the PCI Override Protocol to obtain information about the platform policy. If the PCI Platform Protocol does not exist or returns an error, then this protocol is called.

The PCI bus driver will use this protocol to get the PCI device's option ROM from an implementation-specific location in storage. If the PCI Platform Protocol does not exist or returns an error, then this function is called.

It will also call the various member functions of this protocol at predefined points during PCI bus enumeration. The member functions can be used for performing any implementation-specific initialization that is appropriate during the particular phase.

## Add new interface to the EFI\_PCI\_PLATFORM\_PROTOCOL2 / EFI\_PCI\_OVERRIDE\_PROTOCOL2

The name of the new interface would be GetDevicePolicy, and it will take the EFI handle of the PCI IO Protocol as the input parameter and gives out the EFI\_PCI\_PLATFORM\_EXTENDED\_POLICY as the output parameter for the PCI Bus driver and the PCI Root Bridge Resource Allocation Protocol drivers to utilize during PCI enumeration to configure the given component.

**EFI\_PCI\_PLATFORM\_PROTOCOL2.GetDevicePolicy()**

Summary

The PCI Bus driver and PCI Host Bridge Resource Allocation Protocol drivers can call this member function to retrieve the platform policies for the PCI component, regarding the PCI enumeration.

Prototype

**typedef**

**EFI\_STATUS**

**(EFIAPI \* EFI\_PCI\_PLATFORM\_GET\_DEVICE\_POLICY) (**

**IN CONST EFI\_PCI\_PLATFORM\_PROTOCOL2 \****This*,

**IN EFI\_HANDLE** *PciDevice*,

**OUT EFI\_PCI\_PLATFORM\_EXTENDED\_POLICY \****PciExtPolicy*

**);**

Parameters

This

Pointer to the **EFI\_PCI\_PLATFORM\_PROTOCOL2** instance.

PciDevice

The associated PCI IO Protocol handle of the PCI device. Type **EFI\_HANDLE** is defined in **InstallProtocolInterface()** in the UEFI 2.1 Specification

*PciExtPolicy*

The pointer to platform policy with respect to other PCI features like, the MPS, MRRS, etc. Type **EFI\_PCI\_PLATFORM\_EXTENDED\_POLICY** is defined in "Related Definitions" below.

Description

The **GetDevicePolicy()** function retrieves the platform policy for a particular component regarding PCI enumeration. The PCI bus driver and the PCI Host Bridge Resource Allocation Protocol driver can call this member function to retrieve the policy.

The existing **GetPlatformPolicy()** member function is used by the PCI Bus driver to program the legacy ranges, the data that is returned by that member function determines the supported attributes that are returned by the **EFI\_PCI\_IO\_PROTOCOL.Attributes()** function.

The **GetDevicePolicy()** memberfunction is meant to return data about other PCI compliant features which would be supported by the PCI Bus driver in future; like for example the MPS, MRRS, Extended Tag, ASPM, etc. The details about this PCI features can be obtained from the PCI Base Specification 4.x. The EFI encodings for these features are defined in the **EFI\_PCI\_PLATFORM\_EXTENDED\_POLICY**, see the Related Definition section for this.

This member function will use the associated EFI handle of the PCI IO Protocol to determine the physical PCI device within the chipset, to return its device-specific platform policies. The caller is responsible to allocate buffer and pass its pointer to this member function, to get its device-specific policy data.

**Status Codes Returned**

|  |  |
| --- | --- |
| EFI\_SUCCESS | The function completed successfully, may returns platform policy data for the given PCI component |
| EFI\_UNSUPPORTED | PCI component belongs to PCI topology but not part of chipset to provide the platform policy |
| EFI\_INVALID\_PARAMETER | If any of the input parameters are passed with invalid data |

**Related Definitions**

**typedef struct {**

**EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE** *DeviceCtlMPS***;**

**EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE** *DeviceCtlMRRS***;**

**EFI\_PCI\_CONF\_EXTENDED\_TAG** *DeviceCtlExtTag***;**

**EFI\_PCI\_CONF\_RELAX\_ORDER** *DeviceCtlRelaxOrder***;**

**EFI\_PCI\_CONF\_NO\_SNOOP** *DeviceCtlNoSnoop***;**

**EFI\_PCI\_CONF\_ASPM\_SUPPORT** *LinkCtlASPMState***;**

**EFI\_PCI\_CONF\_COMMON\_CLOCK\_CFG** *LinkCtlCommonClkCfg***;**

**EFI\_PCI\_CONF\_EXTENDED\_SYNCH** *LinkCtlExtSynch***;**

**EFI\_PCI\_CONF\_ATOMIC\_OP** *DeviceCtl2AtomicOp***;**

**EFI\_PCI\_CONF\_LTR** *DeviceCtl2LTR***;**

**EFI\_PCI\_CONF\_PTM** *PTMControl***;**

**EFI\_PCI\_CONF\_CTO\_SUPPORT** *CTOsupport;*

**EFI\_PCI\_CONF\_RESERVES** *Reserves[116]***;**

**} EFI\_PCI\_PLATFORM\_EXTENDED\_POLICY;**

**EFI\_PCI\_PLATYFORM\_EXTENDED\_POLICY** is altogether 128-byte size, with each byte field representing one PCI feature and its bitmask with the following legal combinations for each data member, which is representing the valid combinations of PCI attributes, defined in the PCI Express Base Specification 4.0, version 1.0.

*DeviceCtlMPS*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature Maximum Payload Size (MPS). Refer to PCI Base Specification 4, (chapter 7.3.5.3) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities. Below is it data type and the macro definitions which the driver uses for interpreting the platform policy.

**typedef UINT8 EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE;**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_128B  0x01 //set to default 128B**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_256B  0x02 //set to 256B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_512B  0x03 //set to 512B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_1024B 0x04 //set to 1024B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_2048B 0x05 //set to 2048B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_PAYLOAD\_SIZE\_4096B 0x06 //set to 4096B if applicable**

*DeviceCtlMRRS*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature Maximum Read Request Size (MRRS). Refer to PCI Base Specification 4, (chapter 7.3.5.4) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities. Below is it data type and the macro definitions which the driver uses for interpreting the platform policy.

**typedef UINT8 EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE;**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_128B 0x01 //set to default 128B**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_256B  0x02 //set to 256B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_512B  0x03 //set to 512B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_1024B 0x04 //set to 1024B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_2048B 0x05 //set to 2048B if applicable**

**#define EFI\_PCI\_CONF\_MAX\_READ\_REQ\_SIZE\_4096B 0x06 //set to 4096B if applicable**

*DeviceCtlExtTag*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature Extended Tag field enable and 10-bit Tag Requester enable. Refer to PCI Base Specification 4, (chapter 7.3.5.4) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_EXTENDED\_TAG;**

**#define EFI\_PCI\_CONF\_EXTENDED\_TAG\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_EXTENDED\_TAG\_5BIT 0x01 //set to default 5-bit**

**#define EFI\_PCI\_CONF\_EXTENDED\_TAG\_8BIT 0x02 //set to 8-bit if applicable**

**#define EFI\_PCI\_CONF\_EXTENDED\_TAG\_10BIT 0x03 //set to 10-bit if applicable**

*LinkCtlASPMState*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe link’s Active State Power Management (ASPM). Refer to PCI Base Specification 4, (chapter 7.3.5.7) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_ASPM\_SUPPORT;**

**#define EFI\_PCI\_CONF\_ASPM\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_ASPM\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_ASPM\_L0s\_SUPPORT 0x02 //set to L0s state**

**#define EFI\_PCI\_CONF\_ASPM\_L1\_SUPPORT 0x03 //set to L1 state**

**#define EFI\_PCI\_CONF\_ASPM\_L0S\_L1\_SUPPORT 0x04 //set to L0s and L1 state**

*DeviceCtlRelaxOrder*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Device’s Relax Ordering enable/disable. Refer to PCI Base Specification 4, (chapter 7.3.5.4) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_RELAX\_ORDER;**

**#define EFI\_PCI\_CONF\_RO\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_RO\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_RO\_ENABLE 0x02 //set to enable state**

*DeviceCtlNoSnoop*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Device’s No-Snoop enable/disable. Refer to PCI Base Specification 4, (chapter 7.3.5.4) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_NO\_SNOOP;**

**#define EFI\_PCI\_CONF\_NS\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_NS\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_NS\_ENABLE 0x02 //set to enable state**

*LinkCtlCommonClkCfg*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Link’s Clock configuration is common or discrete. Refer to PCI Base Specification 4, (chapter 7.3.5.7) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_COMMON\_CLOCK\_CFG;**

**#define EFI\_PCI\_CONF\_CLK\_CFG\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_CLK\_CFG\_ASYNCH 0x01 //set to default asynchronous clock**

**#define EFI\_PCI\_CONF\_CLK\_CFG\_COMMON 0x02 //set to common clock**

*LinkCtlExtSynch*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe link’s Extended Synch enable/disable. Refer to PCI Base Specification 4, (chapter 7.3.5.7) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_EXTENDED\_SYNCH;**

**#define EFI\_PCI\_CONF\_EXT\_SYNCH\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_EXT\_SYNCH\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_EXT\_SYNCH\_ENABLE 0x02 //set to enable state**

*DeviceCtl2AtomicOp*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Device’s AtomicOp Requester enable/disable. Refer to PCI Base Specification 4, (chapter 7.3.5.16) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_ATOMIC\_OP;**

**#define EFI\_PCI\_CONF\_ATOMIC\_OP\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_ATOMIC\_OP\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_ATOMIC\_OP\_ENABLE 0x02 //set to enable state**

*DeviceCtl2LTR*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Device’s Latency Tolerance Reporting Mechanism enable/disable. Refer to PCI Base Specification 4, (chapter 7.3.5.16) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_LTR;**

**#define EFI\_PCI\_CONF\_LTR\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_LTR\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_LTR\_ENABLE 0x02 //set to enable state**

*PTMControl*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Device’s Precision Time Measurement (PTM) support enable/disable. Refer to PCI Base Specification 4, (chapter 7.9.16) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_PTM;**

**#define EFI\_PCI\_CONF\_PTM\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_PTM\_DISABLE 0x01 //set to default disable state**

**#define EFI\_PCI\_CONF\_PTM\_ENABLE 0x02 //set to enable state only**

**#define EFI\_PCI\_CONF\_PTM\_ROOT\_SEL 0x03 //set to root select & enable**

*CTOsupport*

This data variable member is reserved to retrieve the PCI device platform policy for the PCI-compliant feature PCIe Device’s Completion Timeout (CTO) support disable or set to supported ranges. Refer to PCI Base Specification 4, (chapter 7.5.3.16) on how to translate the below EFI encodings as per the PCI hardware terminology. If this data member value is returned as 0 than there is no platform policy to override, this feature would be enabled as per its PCI specification based on the device capabilities.

**typedef UINT8 EFI\_PCI\_CONF\_CTO\_SUPPORT;**

**#define EFI\_PCI\_CONF\_CTO\_AUTO 0x00 //No request for override**

**#define EFI\_PCI\_CONF\_CTO\_DEFAULT 0x01 //set to default range of 5us to 50ms**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_A1 0x02 //set to range of 50us to 100us**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_A2 0x03 //set to range of 1ms to 10ms**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_B1 0x04 //set to range of 16ms to 55ms**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_B2 0x05 //set to range of 65ms to 210ms**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_C1 0x06 //set to range of 260ms to 900ms**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_C2 0x07 //set to range of 1s to 3.5s**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_D1 0x08 //set to range of 4s to 13s**

**#define EFI\_PCI\_CONF\_CTO\_RANGE\_D2 0x09 //set to range of 17s to 64s**

**#define EFI\_PCI\_CONF\_CTO\_DET\_DISABLE 0x10 //set to CTO detection disable**

*Reserves[116]*

Padded bytes under reserve data types to make up 128 bytes in total, to be used in future for defining the device-specific platform policy for a given PCIe feature.

**typedef UINT8 EFI\_PCI\_CONF\_RESERVES;**