

## TCG PC Client Device Driver Design Principles for TPM 2.0

---

Version 1.0  
Revision 0.24  
February 8, 2019

Contact: [admin@trustedcomputinggroup.org](mailto:admin@trustedcomputinggroup.org)

Public Review

### **Work in Progress**

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

## **DISCLAIMERS, NOTICES, AND LICENSE TERMS**

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at [www.trustedcomputinggroup.org](http://www.trustedcomputinggroup.org) for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

## CHANGE HISTORY

REVISION	DATE	DESCRIPTION
1.00/0.22	January 03, 2019	<ul style="list-style-type: none"><li>Move of revision 0.21 to new TCG template</li></ul>
1.00/0.23	February 01, 2019	<ul style="list-style-type: none"><li>Include review feedback</li></ul>
1.00/0.24	February 08, 2019	<ul style="list-style-type: none"><li>Modified Figure 1, Figure 4, Figure 5 and Figure 6 after review feedback</li></ul>

## CONTENTS

DISCLAIMERS, NOTICES, AND LICENSE TERMS .....	1
CHANGE HISTORY .....	2
1 SCOPE .....	5
2 Terms and Definitions .....	6
2.1 Acronyms and Abbreviations.....	6
3 References .....	7
4 Introduction .....	8
4.1 The TPM Register Interface .....	8
5 Interface Selection .....	9
6 Locality and Access Protocols .....	11
7 Code Flow for Using TPM.....	13
7.1 FIFO Protocol.....	13
7.1.1 Explanations and Definitions.....	13
7.1.2 Eliminating the read of TPM.burstCount .....	14
7.2 CRB Protocol .....	17
7.2.1 Explanations and Definitions.....	17
8 Interrupt usage.....	20
8.1 Setup.....	20
8.2 Enabling Interrupts .....	20
8.3 Handling Interrupts.....	20
9 Addressing, Power Management and TPM Operational Modes .....	22
9.1 TPM Addressing.....	22
9.2 Power Management .....	22
9.2.1 Reminder .....	22
9.2.2 Power State Transitions.....	22
9.2.3 Driver Power Management .....	23
10 TPM Timeouts .....	24
11 Error Management.....	25
11.1 Error Handling Specific to Platform Firmware .....	25
11.2 Error handling in the OS DD .....	26

## FIGURES

Figure 1 Interface Selection.....	10
Figure 2 Locality access protocol - DD Point of View .....	12
Figure 3 Send Command using FIFO Protocol.....	15
Figure 4 Receive Response using FIFO Protocol .....	16
Figure 5 Send Command using CRB Protocol .....	18
Figure 6 Receive Response using CRB Protocol .....	19

## 1 SCOPE

Even though the target platform for this reference document is a PC Client platform as defined by the PC Client Specification, this document may also be used as a device driver design reference for other platform architectures.

This document describes the anticipated behavior of either a BIOS or an OS-present Device Driver (DD). It is likely, however, that the more restricted environment provided by a BIOS may lead to a DD that utilizes fewer options than a full featured OS-present DD. For example, a BIOS DD would likely not utilize the interrupt features of a TPM and rely on polling instead; while the OS-present DD would setup the platform so that the TPM notifies the OS-present DD about state changes via interrupt.

The term OS-present DD applies to the DD within a Static OS and a Dynamic OS.

## 2 Terms and Definitions

For the purposes of this guidance document, a DD is assumed to be just the DD that interfaces with the TPM and does not include access control and resource management.

### 2.1 Acronyms and Abbreviations

CRB	Command Response Buffer
DD	Device Driver
FIFO	First In First Out data buffer structure
ISR	Interrupt Service Routine
OS	Operating System
PFP	Platform Firmware Profile (see [PFP])
PTP	Platform TPM Profile (see [PTP])
TPM	Trusted Platform Module

### 3 References

[TPM-Lib]	TCG Trusted Platform Module Library Specification Family 2.0 Revision 1.16 or later
[PTP]	TCG PC Client Platform TPM Profile Level 00, Revision 01.03 v22 or later
[PFP]	TCG PC Client Platform Firmware Profile Level 00, Revision 01.03 v51 or later



## 4 Introduction

This document supplements the TCG PC Client Platform TPM Profile for TPM 2.0 Specification [PTP]; in particular this document provides guidance for DD writers interested in developing a DD to talk to a TPM designed to meet the requirements of the [PTP].

Before using this document, the reader should be familiar with the TPM architecture as described in Part 1 of the TPM Library Specification and be thoroughly familiar with Section 5.4, “System Interaction and Flows,” and Section 5.6, “Interrupts,” of the [PTP] specification.

The intent of the [PTP] authors was to provide sufficient standardization, guidance, and restrictions to allow a common DD within each Operating System or each Operating System environment to access and control TPMs from various TPM manufacturers, while still providing TPM manufacturers sufficient flexibility for product and feature differentiation. The [PTP] is, however, written from a TPM developer’s perspective and contains little in the way of informative comments assisting DD designers.

The audience for this document is TPM manufacturers and TPM DD design, development, and test engineers. TPM manufacturers will find this document valuable as it explains the behavior of the TPM from the DD’s perspective. Additionally, this document provides guidance that may prove useful for Independent BIOS Vendors.

### 4.1 The TPM Register Interface

The [PTP] defines the TPM registers and the method of access by register interface type, as well as any physical interface dependencies.

## 5 Interface Selection

A TPM according to the [PTP] may provide up to three different logical interfaces on a physical SPI interface:

- FIFO according to TIS 1.3
- FIFO according to [PTP]
- CRB

*Note: FIFO interface may be either FIFO according to TIS 1.3 or FIFO according to [PTP].*

A TPM's LPC interface only supports the FIFO interface that is leveraged with FIFO on SPI. A TPM's I2C interface supports only the FIFO interface as well, but that is specific (and unique) for I2C. Depending on platform implementation, the default interface of the TPM must be switched according to Figure 1 Interface Selection.

*Note: The switching is only relevant for BIOS DDs.*

February 08, 2019

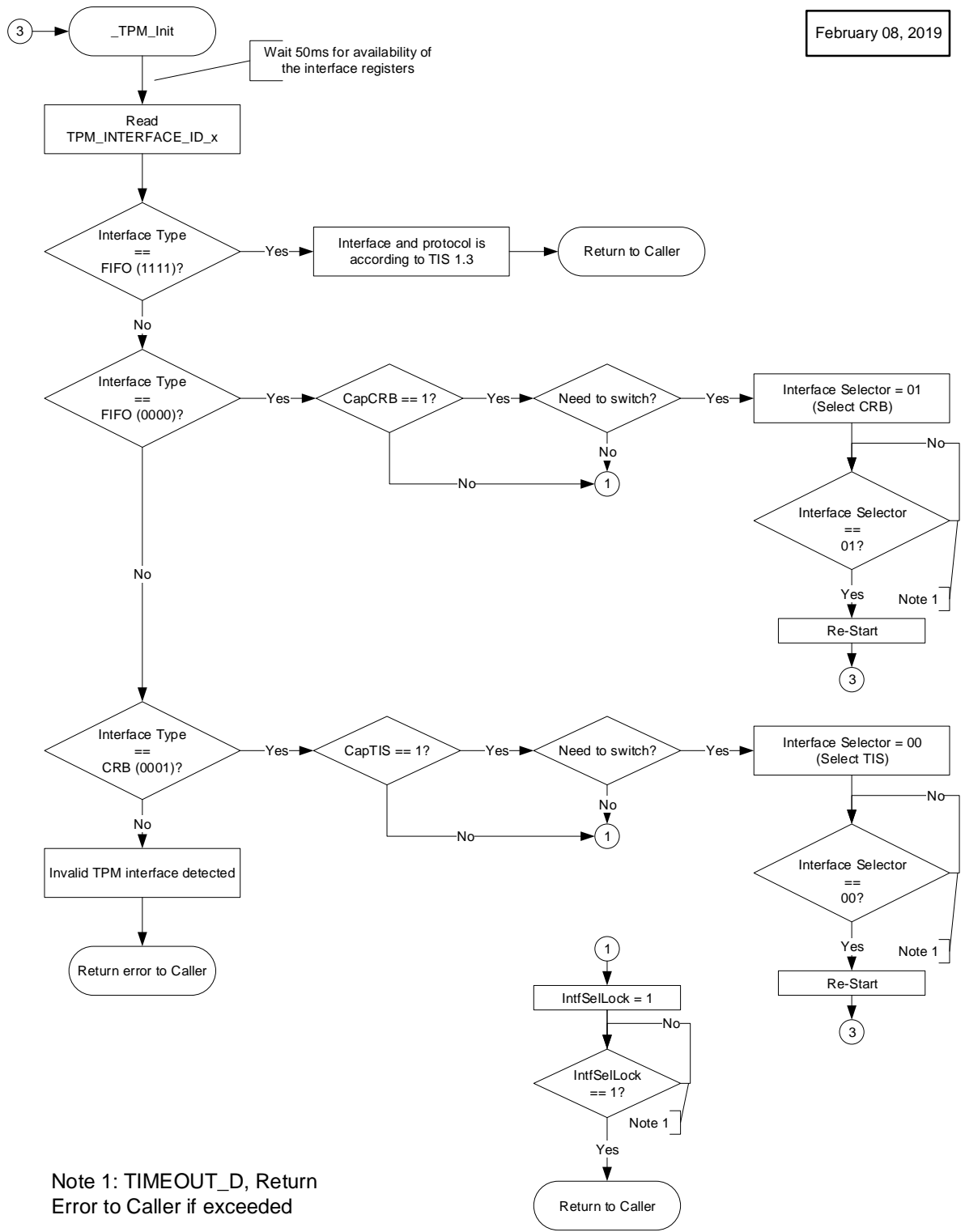


Figure 1 Interface Selection

## 6 Locality and Access Protocols

Locality, as defined in the [PTP], is the mechanism by which the TPM provides different platform components with segregated access to the TPM. The TPM accomplishes this by providing different register address ranges for each locality. For additional information, see the [PTP] Section 5.2.

Locality is an assertion to the TPM that a command's source is associated with a particular platform component. Locality can be thought of as hardware-based authorization. The TPM is not actually aware of the nature of the relationship between the locality and the platform component. The protection and separation of the localities (and therefore the association with the corresponding platform components) is defined in the [PTP].

The design of the TPM provides single threaded access for commands: i.e., each command must complete before another is requested. Locality does not mean that different platform components can access the TPM at the same time. The scheme used is that of a request / grant scheme at the TPM level. Typically, access to a locality is non-preemptive, i.e., if a locality has access to the TPM, then that locality must release its locality before another locality can be granted access. An exception is provided via the seize mechanism. This mechanism is intended to be used only as a "fail-safe", not a normally executing protocol, preventing what may be a deadlocked system.

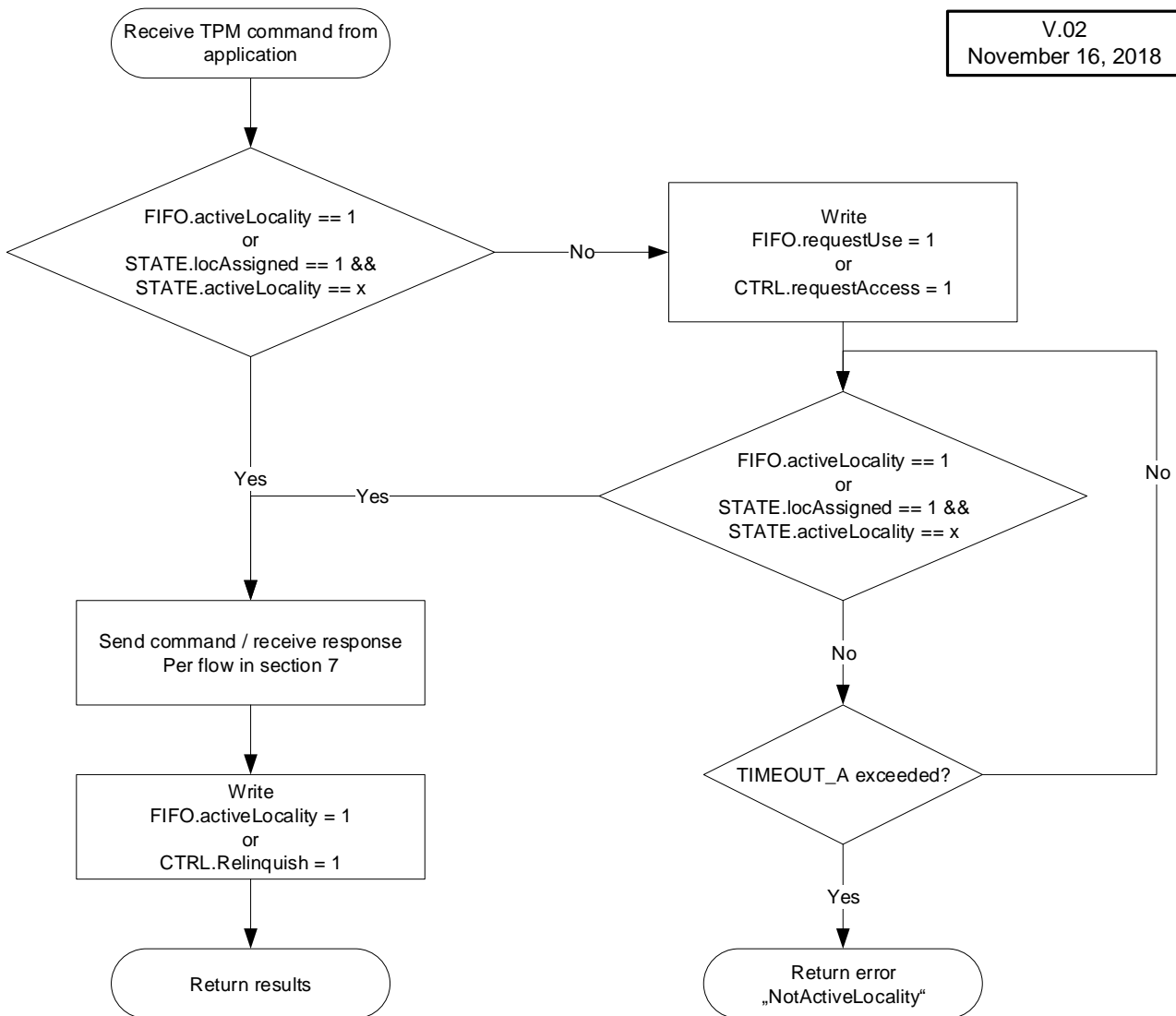
For the FIFO and CRB interfaces, assertion of locality is done by interacting with the TPM at specified blocks of address ranges. Each locality is assigned an address range, and, when a command is received at the address range associated with a locality, the TPM sets an internal flag (localityModifier) to record which locality has been granted access.

All processes associated with TPM Localities have access to the TPM\_ACCESS\_x (FIFO) or TPM\_LOC\_STATE\_x (CRB) register which is used to check and/or request use of their associated Locality. If the currently executing process's Locality is not the TPM's currently active Locality a request can be made using the TPM\_ACCESS\_x/TPM\_LOC\_STATE\_x assigned to the process's Locality. If there is either no TPM Locality set or the Locality that currently has access to the TPM releases it, the requesting Locality is granted access.

Name or Abbreviation	Description
x	The Locality Modifier
FIFO	The short form for TPM_ACCESS_x, the access register for a specific locality. Applies to FIFO and I2C interface.
STATE	The short form for TPM_LOC_STATE_x, the register indicating the current locality state of the TPM. Applies to CRB interface.
CTRL	The short form for TPM_LOC_CTRL_x, the register to control the current locality state of the TPM. Applies to CRB interface.

**Table 1 Legend for Figure 2 Locality access protocol - DD Point of View**

V.02  
November 16, 2018



**Figure 2 Locality access protocol - DD Point of View**

## 7 Code Flow for Using TPM

The following flow charts (Figure 3, Figure 4, Figure 5 and Figure 6) provide an informative sequence of code flows that a DD would take by following both the normative and informative sections of the [PTP] regarding the FIFO or CRB protocol. These flow charts are provided for reference only and are informative in nature.

It is normal practice to have a DD that checks register consistency during communication with the TPM. This ensures that the actual TPM status is aligned with the TPM status assumed by the DD. It is recommended to add a check for TPM\_CRB\_STS\_X.tpmSts and TPM\_CRB\_STS\_X.tpmIdle (CRB interface) during command write, execute and read, to make sure no FATAL error occurred and that the TPM is not in the idle state.

### 7.1 FIFO Protocol

The code flow for the FIFO Protocol is designed for common use cases. There are a few optimization possibilities, which can be applied in specific platforms. The developer of the DD has to consider the capabilities of the platform the DD is designed for and has to determine whether there is any functional risk when applying optimizations.

#### 7.1.1 Explanations and Definitions

The DD variables and abbreviations defined in Table 2 below are used in the following flow charts for the FIFO protocol (Figure 3 and Figure 4):

Name or Abbreviation	Description
BCNT	DD variable holding the value read from the TPM_STS_x.burstCount register
TxSize	DD variable indicating number of bytes remaining in the DD's transmit buffer. TxSize may be derived from the commandSize of the command or from the DD's interface parameter.
TPM.	The short form for TPM_STS_x.
e_	Event
RxSize	DD variable indicating the number of bytes read into the DD's receive buffer
bytes2Read	Total number of bytes to read for the current response, derived from the responseSize parameter. Because responseSize is initially unknown, this variable is initialized with the value 10 (tag + responseSize + responseCode). This allows optimization for the case where the entire response is only 10 bytes and is read in one burst. After having received the first 6 bytes bytes2Read can be updated with the real responseSize.
paramSizeFlag	Indicates whether bytes2Read contains the initial default value of 10 or the real value from responseSize
fCancelRequested	DD variable to store a cancel request from the calling application
TIMEOUT_*	Interface timeout, see section 10 TPM Timeouts

**Table 2 Legend for Figure 3 Send Command using FIFO Protocol and Figure 4 Receive Response using FIFO Protocol**

*Notes: Any timeout condition indicates a malfunction of the TPM which needs proper error handling. If such a condition cannot be corrected the TPM is defective.*

*All expressions follow C style syntax*

### 7.1.2 Eliminating the read of TPM.burstCount

If the DD has the indication that the TPM is ready to receive a command (TPM.commandReady == 1), the DD may read TPM.burstCount only once and subsequently write all bytes (initial value of TxSize) contained in the DD's transmit buffer to the TPM FIFO.

*Note: This optimization requires that the hardware handshake between the host and the TPM (for LPC: SYNC = Long Wait, for SPI: Wait states per [PTP], 6.4.5 Flow Control) works absolutely correctly. Otherwise, the platform may hang if the handshake has issues.*

V.12  
February 08, 2019

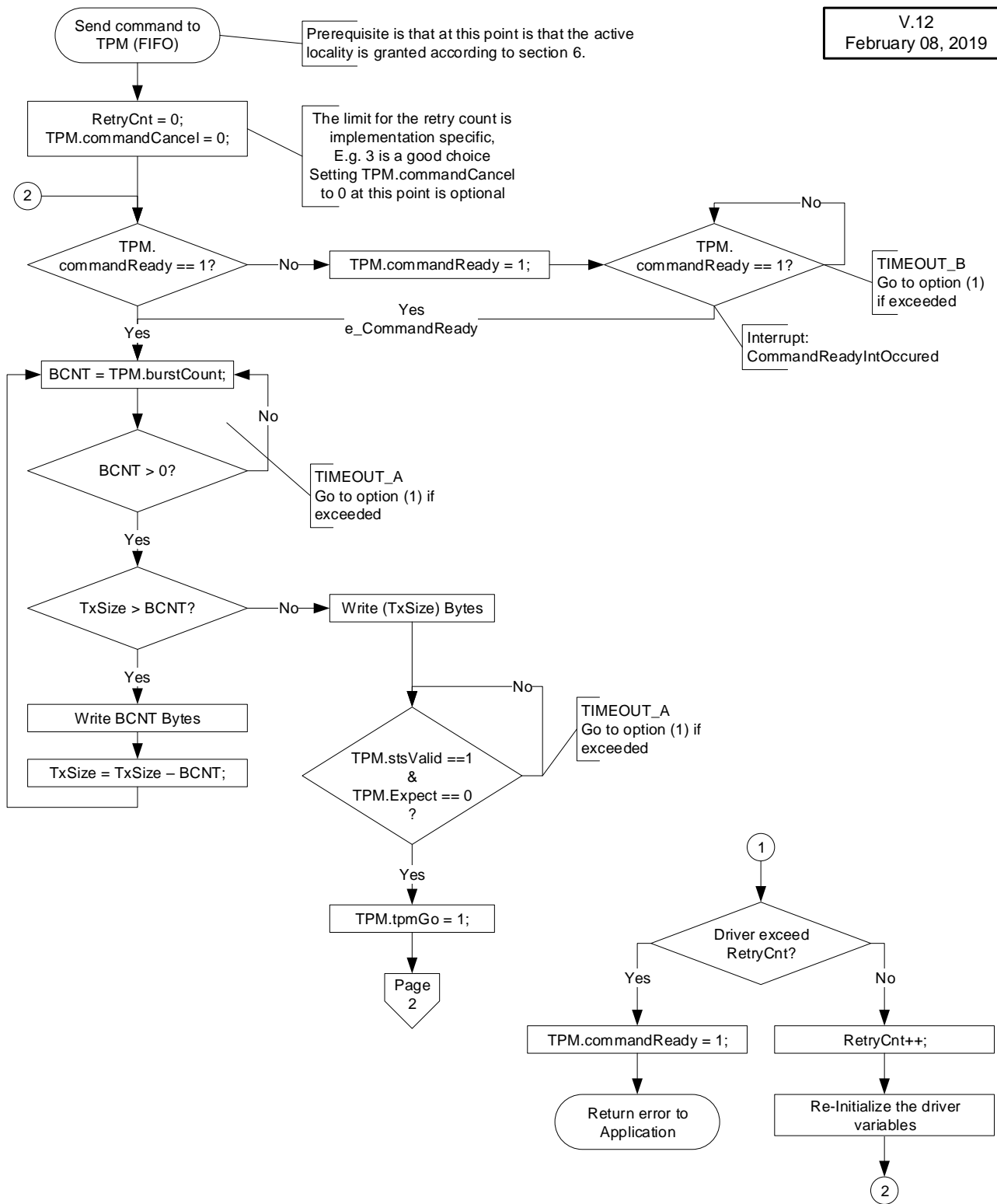


Figure 3 Send Command using FIFO Protocol



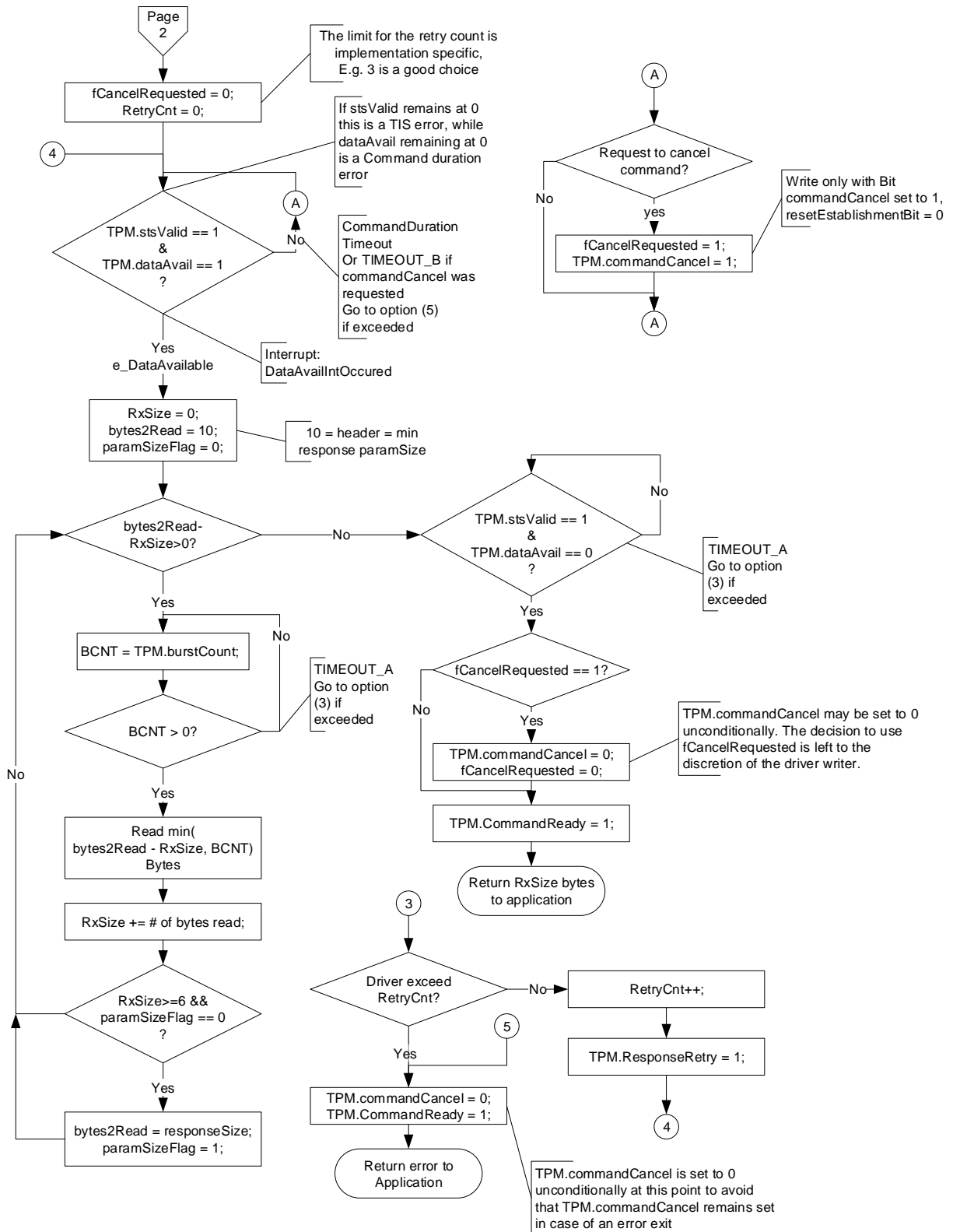


Figure 4 Receive Response using FIFO Protocol

## 7.2 CRB Protocol

### 7.2.1 Explanations and Definitions

The DD variables and abbreviations defined in Table 3 below are used in the following flow charts for the CRB protocol (Figure 5 and Figure 6):

Name or Abbreviation	Description
TPM_CRB_DATA_BUFFER_x	On command transmission this is the command buffer as determined by TPM_CRB_CTRL_CMD_LADDR_x and TPM_CRB_CTRL_CMD_HADDR_x. On response reception this is the response buffer as determined by TPM_CRB_CTRL_RSP_ADDR_x
TPM_STS.	The short form for TPM_CRB_CTRL_STS_x
TPM_REQ.	The short form for TPM_CRB_CTRL_REQ_x
TPM_START	The short form for TPM_CRB_CTRL_START_x.Start
TPM_INTF_ID	The short form for TPM_CRB_INTF_ID_x
e_	Event
RxSize	DD variable indicating the total number of bytes read into the DD's receive buffer
fCancelRequested	DD variable to store a cancel request from the calling application
TIMEOUT_*	Interface timeout, see section 10 TPM Timeouts

Table 3 Legend for Figure 5 Send Command using CRB Protocol Figure 3 Send Command using FIFO Protocol and Figure 6 Receive Response using CRB Protocol

*Notes: Any timeout condition indicates a malfunction of the TPM which needs proper error handling. If such a condition cannot be corrected the TPM is defective.*

*All expressions follow C style syntax*

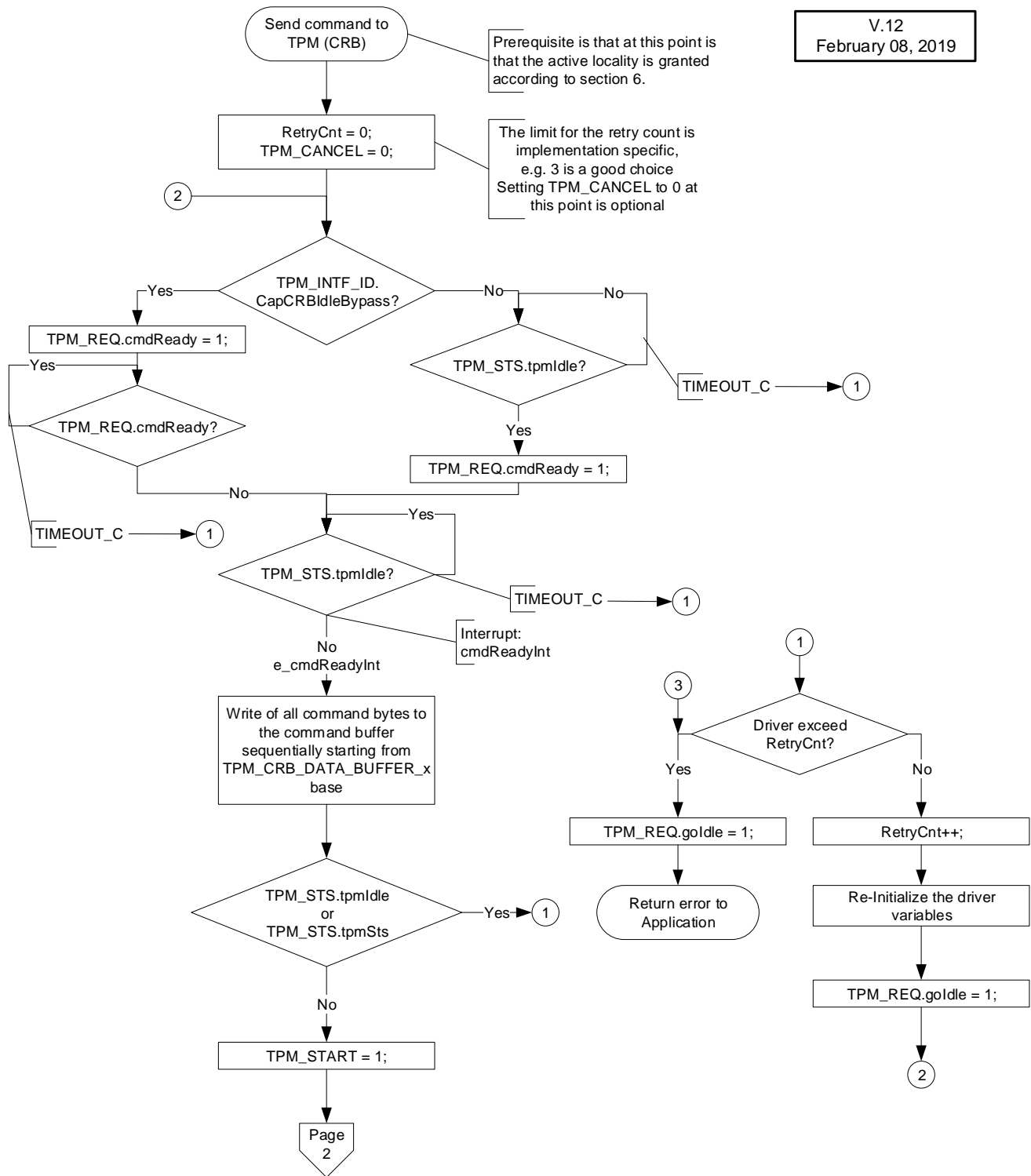


Figure 5 Send Command using CRB Protocol

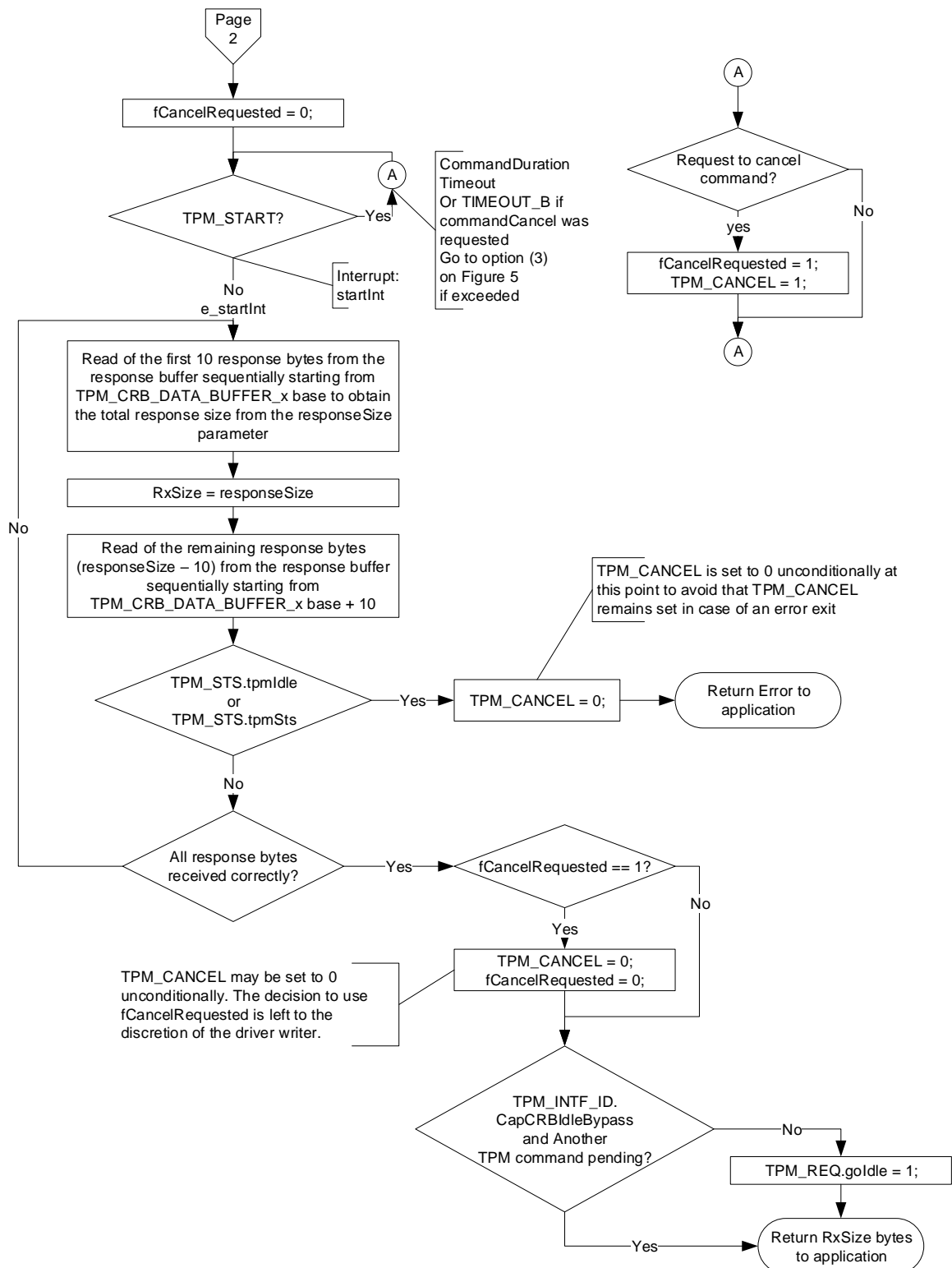


Figure 6 Receive Response using CRB Protocol

## 8 Interrupt usage

This section repeats some of the information contained in the TCG PC Client Platform TPM Profile [PTP] Specification, Section 5.6 “Interrupts”.

### 8.1 Setup

An OS DD for TPM will register an ISR with the OS to indicate that it can handle interrupts. This is usually done during initialization of the TPM DD. The OS will then make sure that whenever the TPM device fires an interrupt, the TPM DD's ISR is invoked to handle the interrupt.

OS DD design guides suggest a split approach to handle interrupts: have an ISR that takes care of quick tasks, such as acknowledging an interrupt. The ISR would then schedule another handler, sometimes referred to as Deferred Procedure Call (DPC), which does more time intensive work.

### 8.2 Enabling Interrupts

Enabling TPM interrupts is as simple as setting the bit for the respective interrupt in the interrupt enablement register (TPM\_INT\_ENABLE\_x or TPM\_CRB\_INT\_ENABLE\_x).

There are different approaches on how to use interrupts. One is to enable interrupts when the OS TPM DD starts, another is to enable interrupts selectively. If the OS TPM DD follows the first approach, it must synchronize between the interrupt handler and the code that sends TPM commands to determine whether the interrupt is expected or not. Enabling interrupts only after a command has been sent and disabling the interrupt after the interrupt has been received will not require such synchronization. The OS TPM DD writer must be aware that there are performance costs associated with enabling and disabling interrupts, namely additional register reads and writes.

The OS may also incur an overhead for handling interrupts. It must dispatch the interrupt to the correct interrupt service routine. While an interrupt is dispatched, the OS usually masks other interrupts until the interrupt has been acknowledged. To address these penalties, the [PTP] contains the description of a possible optimization: Many commands respond immediately during normal operation, so the DD, after sending a command, should poll the TPM for a response, while keeping the data available interrupt disabled. If the DD determines that the TPM will not be able to respond immediately, it will stop polling the TPM and enable the data available interrupt. If the DD does this, there is a possibility of a race condition between the time the last read occurred during polling and when the interrupt is enabled. Therefore, after enabling the interrupt, the DD should poll the TPM one more time.

The same holds true for changes affecting other interrupts.

For LPC devices, the TPM\_INT\_ENABLE\_x register also contains the polarity of the interrupt. The TPM DD should take care not to change the polarity during normal operation.

Some TPM implementations require the OS TPM DD to set the interrupt vector that has been assigned by firmware/OS to the TPM device. The OS TPM DD queries the device information provided by the OS and writes the interrupt vector to the TPM\_INT\_VECTOR\_x register.

### 8.3 Handling Interrupts

When an event occurs that causes the TPM to signal an interrupt, the TPM must set the appropriate fields in the TPM\_INT\_STATUS\_x or TPM\_CRB\_INT\_STS\_x register. If the TPM has not already sent an interrupt, the 0 to 1 transition of a field in the TPM\_INT\_STATUS\_x or the TPM\_CRB\_INT\_STS\_x register must cause the TPM to assert the appropriate interrupt per the SIRQ or PIRQ protocol. The interrupt service routine will read the TPM\_INT\_STATUS\_x or TPM\_CRB\_INT\_STS\_x register to determine the cause and take appropriate action. When the interrupt has been serviced, the interrupt service routine must do an End-of-Interrupt (EOI) to the I/O APIC to re-

arm the TPM's interrupt in the I/O APIC. Then the interrupt service routine must also send a TPM\_EOI to the TPM to allow the TPM to send new interrupts.

The TPM must not issue another interrupt until it has received its TPM\_EOI message, even if new events occur that should cause an interrupt. The TPM should set the appropriate field in TPM\_INT\_STATUS\_x or TPM\_CRB\_INT\_STS\_x register, but the actual assertion of the interrupt will only occur until after the TPM\_EOI was received. If the interrupt handler detects multiple fields set, it may handle all the causes and clear multiple status fields. This means that an interrupt may be handled without causing a new interrupt. The interrupt handler may continue to check if the TPM\_INT\_STATUS\_x or TPM\_CRB\_INT\_STS\_x field has new fields set before sending the TPM\_EOI. Note that there can only be one outstanding notification per interrupt.

If the TPM DD implements a strategy that enables and disables the data available interrupt for each command, based, for instance, on the duration of the command, the data available interrupt should be disabled when acknowledging the interrupt. This must happen before the next command is sent. Due to the serial nature of a TPM, the TPM's communication interface should not trigger another data available interrupt before the next command has been sent.

Note that interrupts may be shared between devices. That means that the ISR for the TPM DD can be invoked when another device triggers an interrupt. The TPM DD must check the TPM\_INT\_STATUS\_x or TPM\_CRB\_INT\_STS\_x register to see if the TPM device generated an interrupt.

If the TPM DD implements a deferred procedure call (DPC), the DPC for the data available interrupt could read the response or signal another thread to continue processing the response.

## 9 Addressing, Power Management and TPM Operational Modes

### 9.1 TPM Addressing

The TPM, along with the other platform components, is initialized during platform reset. Upon platform reset, the S-CRTM, by definition, is the first code to execute. By necessity, this environment tends not to have the ability to discover the logical location of a TPM. The D-CRTM has the same limitations. Therefore, to provide the best security and performance properties for the PC Client, the PTP specification defines location of the TPM as a fixed location with no allowance for variance.

An ACPI based Operating System uses the ACPI table provided by the platform's firmware to allocate and reserve TPM resources. The DD uses the Plug-n-Play manager's interface to discover the resources that have been allocated for current platform's boot cycle. (e.g., has the TPM DD been allocated for this boot cycle, which IRQ was allocated, etc?) However, if there is no ACPI table entry for the TPM, the DD will not be loaded by the operating system and therefore can do nothing.

### 9.2 Power Management

#### 9.2.1 Reminder

For PCs, ACPI defines a set of power states that each have different performance and power requirements.

#### System States

System states describe the power state of the entire system. The defined system states are enumerated below:

- S0 – Working state (faster response times, high power usage)
- S1, S2, S3 – Sleep (some components may be powered off; volatile memory is powered)
- S4 – Hibernate (system appears off; volatile memory content is save to non-volatile memory)
- S5 – Off

#### Device States

The device states D0–D3 are device dependent:

- D0 or Fully On is the operating state.
- D1 and D2 are intermediate power-states whose definition varies by device.
- D3: The D3 state is further divided into D3 Hot (has auxiliary power), and D3 Cold (no power provided):  
 D3 Hot: A device can assert power management requests to transition to higher power states.  
 D3 Cold: means Off has the device powered off and unresponsive to its bus.

The only transitions allowed for system states and devices states are those defined in the ACPI specification. There is no direct correspondence between device states and system states; for example, a device may be state D3 while the system is in state S0.

#### 9.2.2 Power State Transitions

The TPM is designed to draw power only during the platform “operational states” i.e., specifically during the S0, S1 and S2 ACPI power states. The TPM draws no power during the platform’s “non-operational” states – S4 and S5. The TPM may draw power from the auxiliary or standby power source during S3 but may only do so to maintain the required state upon issuance of a TPM2\_Shutdown (STATE) command.

A TPM is allowed to enter a lower power state during the platform’s “operational states” under specific conditions. The TPM, if in the idle state, can reduce its power consumption by shutting down internal functional blocks as long as the TPM device driver (i.e. the flowchart Figure 3 Send Command using FIFO Protocol) and TPM functionality are not impacted. When the TPM receives a transaction on its interface that would cause it to move from Idle to

Ready, the TPM exits the low power mode within TIMEOUT\_B. The TPM is not allowed to enter a low power state prior to the receipt of a TPM2\_Startup command.

The design of a PC Client platform resuming from either S5 or S4 requires a complete and full initialization and startup of the platform's S-RTM. Therefore, the TPM's internal state that maintains the S-RTM is reset when resuming from either S4 or S5.

However, when resuming from S3, the platform, at least from a user experience point of view, appears as if the platform was in a state of suspension and no reset should occur. It is assumed that the Operating System maintains the S-CRTM chain during this suspended state. Therefore, no remeasurement should be necessary. In fact, the protected storage mechanisms provided by the TPM will be difficult to manage if the platform re-measures the Static CRTM upon resumption from S3.

Resuming from S3 without resetting the S-RTM is accomplished using a state preservation mechanism. When the platform is about to enter an S3 state, the DD is instructed to send a TPM2\_Shutdown (STATE) command to the TPM. This command instructs the TPM to save its state pertaining to the S-CRTM (see the specific command in the TPM 2.0 Library Specification for details of what must be saved and what must not be saved.)

Upon platform reset, the S-CRTM detects from which state the platform is resuming. If the platform is resuming from S3, the saved state is to be restored and the S-CRTM issues a TPM2\_Startup (STATE) otherwise the S-CRTM issues a TPM2\_Startup (CLEAR). The TPM2\_Startup (CLEAR) causes the TPM to discard the contents of PCR that represented the previous S-RTM chain.

Because the determination of the way the TPM starts impacts the security properties of the S-CRTM chain, the S-CRTM never allows the TPM to be reset outside the S-CRTM. For this reason, the TPM2\_Startup command is always issued by the S-CRTM and the DD never gains access to the TPM unless the TPM2\_Startup command has been issued. If the S-CRTM fails to issue the TPM2\_Startup command the S-RTM chain is left open. An open S-RTM chain is considered either a flaw or an attack. Lacking the proper connection back to the S-RTM, there is nothing, from a security perspective, the DD can do. If no TPM2\_Startup command is sent following platform reset, subsequent TPM commands will return with the return code TPM\_RC\_INITIALIZE. The DD should take no action and return the return code up to the application that issued the originating command. It is up to the application to decide what action to take.

### 9.2.3 Driver Power Management

The TPM provides mechanisms to change the operational state of the TPM but doesn't support any device states other than D0 (On) and D3 Cold. The DD is responsible for ensuring the TPM preserves any TPM state information required to survive a suspend (S3) or hibernate (S4) cycle.

- An ACPI enabled Operating System uses the ACPI interface to determine the TPM's resources.
- If the platform is transitioning from a powered-on State (S0) to a suspend (S3) or Hibernate (S4) AND the DD was designed to handle power management, it is critical that the DD be designed to send the TPM2\_Shutdown (STATE) command before entering suspend or hibernate, regardless of the current power state of the TPM.
- The platform firmware is responsible for issuing a TPM2\_Startup command. If the platform is resuming from suspend the platform firmware issues a TPM2\_Startup (STATE) command. If the platform is resuming from hibernate or powering on from an off state, the platform firmware issues a TPM2\_Startup (CLEAR). For detailed requirements regarding platform firmware, see the TCG PC Client Platform Firmware Profile for TPM 2.0 Systems [PFP].
- The DD should not track the power state of the TPM.



## 10 TPM Timeouts

Several timeout values were defined in the [PTP] to avoid situations where the DD would have to wait for a response from an unresponsive TPM.

Section 7 Code Flow for Using TPM explains in which situation during the communication or control flow which particular timeout value has to be applied.

The following timeouts have been defined:

General control timeouts, denoted as TIMEOUT\_A (A), TIMEOUT\_B (B), TIMEOUT\_C (C) and TIMEOUT\_D (D) are the maximum waiting time from a certain control operation from the DD until the TPM shows the expected status change. In practice, the response time of the TPM might be significantly shorter than the indicated timeout values.

In addition to the general control timeouts, the [PTP] has defined Command Timing Durations and Timeouts. The command timeouts defined by the [PTP] are intended to inform the DD when to issue a command cancel to attempt to recover the TPM. The durations defined in the [PTP] for commands requiring access to nonvolatile memory are for pre-OS environments only. Some TPM implementations may rely on OS DDs to access non-volatile memory in OS-present environments and, as such, may not be able to comply with the command timing requirements specified in the [PTP].

The DD should utilize the default timeouts specified in the [PTP] and program them as early as possible after its startup initialization.

In the event that the DD wishes to abort a command, it should issue a command cancel, as described in the [PTP]. Just in case the DD seizes the TPM from a lower locality, it has to assume the longest possible timeout value.

For all other commands not listed in the Command Timing table (see section Command Duration in the [PTP]) the following rule for maximum command duration applies:

- For TPM2\_Create, TPM2\_CreatePrimary and TPM2\_CreateLoaded the DD should utilize a timeout of 180 seconds.
- For all other remaining commands, the DD should utilize a timeout of 10 seconds.

## 11 Error Management

Consistent error management for TPMs is important because of the restricted environment in which TPM communications typically occur. In addition, security often depends on consistent and well thought out messages.

Section 5.5.1.8 of the [PTP] describes the errors which a TPM may encounter as:

- Errors that the TPM detects and force it into Failure Mode
- Errors that seem, to the TPM, to be attacks
- Transmission or protocol errors
- Errors that the TPM does not detect

From the perspective of the DD, failures can be grouped into two categories: TPM command failures and TPM functional failures. TPM command failures are described in the TPM Library Specification [TPM-Lib] and result in a TPM return code that describes the error. TPM functional failures may result from a functional failure within the TPM or from the TPM detecting an attack. In the first case, it is likely that the TPM will fail self-test and return an error indicating such a failure. The PTP defines this as “Failure Mode”. The TPM will respond to all commands with an error code defined in the Library specification. The TPM2\_GetTestResult command will return manufacturer-specific information regarding the nature of the failure.

A TPM that has detected an attack is indistinguishable from a TPM with a fatal error that prevents responses to commands. The TPM response to either condition is to enter what the [PTP] calls Shutdown Mode. When in Shutdown Mode, the TPM will not respond to any access attempt on the interface.

Any error that results in a return code can be interpreted by the calling application. The return code is submitted as part of the response packet from the DD to the calling application (see section 7 Code Flow for Using TPM). In the event that the TPM enters Shutdown Mode, the only means to return the TPM to a functional state is to restart the platform and, thus the TPM. If the TPM encountered a fatal error, or the detected attack condition remains following the restart, the TPM will remain in Shutdown Mode. In this case, it will appear to the DD that no TPM is present or is defective.

### 11.1 Error Handling Specific to Platform Firmware

The first code that interacts with the TPM resides either within microcode or within platform firmware. If CPU microcode with the capability to interact with the TPM exists, this code will send the TPM2\_Startup command. The microcode, as it is resource-constrained, likely will not have the capability to handle an error from the TPM in such a way that it can notify any subsequent operational code in the boot path that an error occurred. As such, it is recommended that platform firmware will send a subsequent TPM2\_Startup, and the firmware needs the capability to handle the error. The DD should always send the TPM return code to the caller.

Error handling is basically performed by the platform firmware. Therefore the DD generally returns the error code received from the TPM as described in section 7 Code Flow for Using TPM. The [PFP] describes in section 2.3.2 how TPM errors should be handled by the platform firmware.

If the TPM returns TPM\_RC\_INITIALIZE from TPM2\_Startup the DD may ignore this return code and treat the command as being executed successfully as the CPU microcode has already executed the TPM2\_Startup.

In addition to the conditions described in the [PFP] Section 2.3.2, the following error conditions from TPM2\_Startup need to be handled by the platform firmware:

- TPM\_RC\_UPGRADE

This error code indicates that the TPM currently is in field upgrade mode. The platform firmware should continue booting of the platform into a fail-safe mode (see [PFP] Section 2.3.2.2). No matter what, the platform firmware should not interfere with the field upgrade process.

*Note: To allow the continuation of the field upgrade process when the TPM is in field upgrade mode, it is necessary that the platform allows access to the TPM and loading of an OS TPM DD.*

- TPM\_RC\_REBOOT

This error indicates that the TPM needs a \_TPM\_INIT followed by a TPM2\_Startup (CLEAR) before the TPM can resume operation.

- TPM\_RC\_BAD\_TAG

This error code indicates that the TPM is a TPM 1.2 and all commands per the TCG TPM 1.2 specification apply. Handling of TPM 1.2 commands is not in the scope of this document.

*Note: All the errors listed above as well as other errors potentially returned from other TPM commands need to be handled by the platform firmware (e.g. TPM\_RC\_LOCALITY returned by TPM2\_PCR\_Extend).*

## 11.2 Error handling in the OS DD

As noted in the introduction to this section, the TPM device can be in different error states. For the OS it is important to distinguish whether the TPM device is fully functional, in a recoverable error mode, or not functional at all.

A TPM can be considered nonfunctional if the TPM device does not adhere to the protocols defined in the TCG specification. If the OS cannot communicate with the TPM using these protocols, it must assume that the TPM device is defective.

A TPM device in a recoverable error mode could be, for instance, processing a command that takes longer than expected and that can be aborted using the appropriate mechanism (see command abort or command cancellation in section 7 Code Flow for Using TPM). The OS can change the state of the TPM through a set of well-defined steps into a known good state.

A TPM in failure mode should also be considered nonfunctional because the OS cannot recover the TPM into a workable state. The only option available to the calling application is to reboot the platform to attempt a restart of the TPM (i.e. \_TPM\_INIT followed by a TPM2\_Startup (CLEAR)).

If the TPM returns a return code in response to a command, the DD does not take any action except returning the error code to the calling application. It is the responsibility of the calling application to handle errors.

*Note: This may be handled differently in DD designs that incorporate access control and resource management.*