UNIVERSAL STYLUS INITIATIVE ® (USI™)
STYLUS AND DEVICE TECHNICAL SPECIFICATION 1.0

June 30, 2016

Legal Notices and Disclaimers

THIS SPECIFICATION IS PROVIDED TO YOU "AS IS" WITH NO WARRANTIES WHATSOEVER, WHETHER EXPRESS, IMPLIED, STATUTORY, AT COMMON LAW, OR OTHERWISE. USER ASSUMES THE FULL RISK OF USING THIS SPECIFICATION. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, UNIVERSAL STYLUS INITIATIVE, INC. AND ALL AUTHORS OF THIS SPECIFICATION HEREBY DISCLAIM ANY AND ALL WARRANTIES AND CONDITIONS WITH RESPECT TO THIS SPECIFICATION AND ITS CONTENT AND THE TRADEMARKS APPEARING IN THIS SPECIFICATION, INCLUDING WITHOUT LIMITATION ANY AND ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SUITABILITY, RELIABILITY, ACCURACY, NON-INFRINGEMENT, VALIDITY OF RIGHTS, AND/OR TITLE. UNIVERSAL STYLUS INITIATIVE, INC. AND THE AUTHORS OF THIS SPECIFICATION HEREBY DISCLAIM ANY AND ALL LIABILITY OF ANY KIND, INCLUDING WITHOUT LIMITATION, LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS AND LIABILITY FOR PRODUCT DEFECTS AND PRODUCTS LIAIBILTY CLAIMS, RELATING TO USE AND/OR IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. IN NO EVENT SHALL UNIVERSAL STYLUS INITIATIVE, INC. AND/OR THE AUTHORS OF THIS SPECIFICATION BE LIABLE FOR ANY ACTUAL, DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL, PUNITIVE, EXEMPLARY, OR ENHANCED DAMAGES ARISING FROM YOUR USE AND/OR IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH .DAMAGES IN ADVANCE

THE PROVISION OF THIS SPECIFICATION TO YOU DOES NOT PROVIDE YOU, AND NOTHING IN THIS SPECIFICATION SHALL BE DEEMED AS GRANTING YOU, WITH ANY LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO USE ANY CONTENT OR TRADEMARKS DESCRIBED OR CONTAINED IN THIS SPECIFICATION OR TO ANY INTELLECTUAL PROPERTY RIGHT OWNED OR CONTROLLED BY UNIVERSAL STYLUS INITIATIVE, INC. OR THE AUTHORS OF THIS SPECIFICATION. YOU MAY HAVE BEEN GIVEN PERMISSION IN A SEPARATE WRITTEN AGREEMENT WITH UNIVERSAL STYLUS INITIATIVE, INC. TO USE THIS SPECIFICATION, AND YOUR USE IS SUBJECT TO SUCH AGREEMENT.

UNIVERSAL STYLUS INITIATIVE®, USI™, USI UNIVERSAL STYLUS INITIATIVE & Pen Design™, Pen Design™, and USI CERTIFIED™ are registered and unregistered trademarks, service marks, and certification marks of Universal Stylus Initiative, Inc. in the United States and other countries. Unauthorized use strictly prohibited.

Revision History

Rev.	Date	Revision Scope
0.3	June 4, 2015	First version published for member companies review
0.5	Aug 21, 2015	Complete arcitecture version for member companies review. This version has all major design concepts. Some detailes (such as message/register definitions, error handling etc.) may be missing, but all architectural choices are made by this version.
0.65	Oct 15, 2015	Complete architecture trial version for company review. Contains almost all details.
0.7	Nov 17, 2015	Complete version to be submitted to USI Board for approval as the USI Technical Specificaton version 0.7.
0.7-update1	Nov 20, 2015	Updates, including some renaming and addition of data missing from 0.7 revision.
0.71	Jan 11, 2016	General cleanup; Consistency across chapters addressed – Changed Bit diagrams across chapters so MSB is on left hand side consistently; Reference issues fixed;
0.76	Jan 20, 2016	Skipping in-between revision numbers to reduce conflicts with older version that may have those numbers. Incorporated proposed changes of Sticky bits (Stylus Color, Type and Width), UUID and Eraser Ready bit instead of Invert bit; General references cleanup where found. Clean (all changes accepted) on 02/03/2016
0.77	Feb 08, 2016	Updated with change requests CR002, CR003, CR004, CR005, CR006, CR008, CR009, CR010, CR012, CR013.
0.78	Feb 11, 2016	Updated CR006, CR009 and CR010, CR015, CR016, CR018, CR019, CR020, CR021, CR024, CR026, CR027 . Clarifications and cleanup across the document
0.80	Feb 16, 2016	Skipping 0.79 to indicate major changes. GID table bit numbering fix;
0.85	Feb 20, 2016	Clarifications on IMU data, USB Charger Noise, SEP Tyep 0 profile addition, general cleanup across document. Once the three TBD entries are filled, the rev will be 0.90, after which only minor clean up and any mistake-fixes are allowed.
0.86	Feb 26, 2016	All TBD values filled in. Fully complete version.
0.90	Mar 07, 2016	Changes based on the Mar 2-4, F2F discussions at the USI plugfest event – clarifications, addition of C.GetTiltTwist(…) and C.SetVendorExtension(…), fixing broken references, Appendicies K and L, etc.
1.0 (Draft)	Apr 25, 2016	CR40 (CRC fix), CR41 (HID Report) and removal of Vendor IDs from the table in Appendix F.
1.0	June 30, 2016	Release version. No content changes from 1.0 Draft.

Acknowledgements

Atmel (a subsidiary of Microchip Technology)
Chris Hill
Eivind Holsen
Richard Kerslake-Smith

Cirque Corporation
Jared Bytheway

Dell Inc.
Tom Lanzoni

Hanvon Technology Co., Ltd.
Jiangli Wei

HP Inc.
Jeremy Meyer

Intel Corporation
Shiva Aditham
Ajay Bhatt
Alex Erdman
Arvind Kumar
Pete Mueller

Lenovo Group Ltd.
Koji Kawakita

Sharp Corporation
Scott Bowden
Masayuki Miyamoto

Synaptics Incorporated
Kirk Hargreaves
Jeff Lukanc
David Sobel

Wacom Co., Ltd.
Dave Fleck
Sadao Yamamoto

Waltop International Corp.
Curtis Kuan

Table of Contents

Table of Figures

§ §

Table of Tables

§ §

1 Introduction

A stylus is an input device that works with a computer/tablet/phone screen to provide selection, pointing, drawing and writing capabilities.

The passive stylus is the simplest stylus design, with a tip made of a conductive material. It simply acts as a finger touch.

An active stylus, also known as an 'electronic pen', contains circuitry which allows it to produce a more natural and engaging user experience. Active styluses enable fine-grain writing and precision pointing by providing the touch controller a specific means to determine its location, responding to pressure, enhancing navigation while hovering above the screen and manipulating on-screen menus and controls through button presses.

In this document the general term 'stylus' refers to the active stylus and includes pen stylus designs.

The 'USI Stylus' is an interoperable active stylus that works seamlessly with multiple different touch capable devices whose touch controller (called "device" in this specification) conforms to the requirements stated in this document.

This specification defines the USI Stylus and the ways it interacts with touch capable devices.

1.1 Objectives of the Specification

This specification provides a complete technical definition of the interoperability of the USI Stylus with touch-capable devices, and a functional specification of the device components that interoperate with the USI Stylus.

1.2 Document Scope

This document is intended for implementers who are developing a controller, stylus, smartphone, tablet, notebook or even a completely new kind of device that will meet the interoperability requirements defined by this specification.

Some informative annexes are provided for readers to further understand the USI specification, and to provide more context about the design choices that were made.

This document is written for the technical representatives of the USI partner companies.

1.3 Goals of the Universal Stylus Initiative (USI)

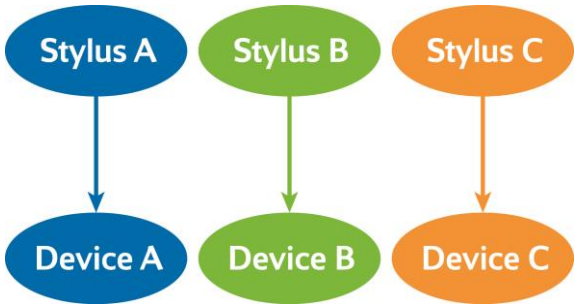
1.3.1 Universal Protocol

The Universal Stylus Initiative (USI) is an industry organization comprised of input controller and stylus engineering experts from over 30 member companies.

The primary goal of the USI is to satisfy the electronics customer's desire to have a single stylus that is capable of operating with all the devices that the customer owns or works with.

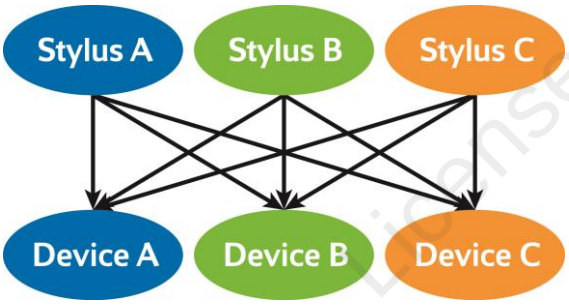
In a typical household, a family owns multiple devices – smartphones, tablets, laptops, PCs, TVs and even purpose-specific products that now provide screens, such as refrigerators, thermostats and automobile dashboards. Each of these devices has touch capabilities (operation using its screen rather than a keyboard) and more and more of these devices now support a stylus. Every user of a stylus would strongly prefer to carry a single stylus that works on all of these devices. It is USI's goal to make this happen.

Today the customer is stuck with each device's dedicated stylus or, in some rare occasions a customer might purchase an 'advanced' stylus that works with one or two devices from the same vendor. But even this stylus is not likely to work with any other devices that the person owns or encounters.



Today the user is forced to keep a specific stylus for each device they own because the signaling mechanisms and the communication protocols between the stylus and the device are proprietary. It is unlikely that a single stylus will operate with products from multiple vendors.

As the illustration on the right shows, the stylus A works only with Device A. It does not work on Device B or Device C.



The USI specification's primary goal is to define a universal protocol between stylus and device. This protocol is 'universal' because it provides interoperability between styluses and devices from multiple vendors. The user can use the USI Stylus across all of these devices, and the USI Styluses that come with the devices work with the devices from other manufacturers. This is illustrated in the diagram on the left.

1.3.2 Interoperability

Although passive styluses are inherently interoperable (since they work on any touchscreen that supports finger touch), they can't provide the precise control and broader feature-set afforded by active styluses. Passive styluses don't have hover, tilt and pressure sensing, nor do they provide pushbuttons for making multiple choices with a single touch.

The immediate goal of the USI is the development of new direct control peripherals and interfaces that interoperate across a broad spectrum of products. The extended goal is to enable new devices and applications that further stylus utility and provide benefits to users.

This specification defines the physical communication medium between USI Stylus and USI Device as well as the signaling and the communication protocol that runs on that medium. The protocol covers the discovery of a USI Stylus by a USI Device, the setup of the communication link between them, the packet formats of the commands and data that flow over that link, and the mechanism that negotiates the capabilities the device uses and provides.

This first USI specification focuses on capacitive touch systems. In these systems, styluses use the same capacitive sensors that are used for finger touch. The communication between the stylus and the controller happens through this capacitive sensor layer.

This USI specification is also designed to allow existing proprietary implementations of styluses and devices to coexist alongside USI Styluses and USI compatible devices. Some vendors might even implement dual-mode styluses (supporting both USI and proprietary functions) or dual-mode devices (e.g., tablets that accept both USI Styluses and proprietary styluses). Mechanisms are defined in this specification for the discovery of proprietary implementations and the support of switching between USI and proprietary functions when dual mode operation is implemented. These capabilities will enable a smoother transition of today's implementations to both this and future USI defined standards.

1.3.3 What This Specification is Not

This specification does not specify the hardware or firmware/software implementations of the controller, the stylus or the devices that incorporate those functionalities. It also does not define the physical design of the stylus, such as its shape, size, color, and button placement.

Furthermore, this document does not specify the method to be used to determine the stylus' location. That is left to the vendor-specific design of the USI controller.

This specification focuses on the 'over the air' behavior of the communication between two entities, and leaves ample room for implementers to add new capabilities on top of the specified functionality. This allows for innovation well beyond the basic capabilities defined by the specification.

A large number of other factors are part of an overall end-to-end solution. These include the primary design and application of each device as well as the application programming interface (API) that each application uses to access the functions of the stylus and controller.

The API is typically defined by the operating system or software vendor, and is out of scope for USI. However, USI defines the changes to the USB implementers' Forum's Device Class Definition for Human Interface Devices (HID) interface protocol that are required to support USI Stylus features. The HID interface is the base software protocol used by many operating system vendors to define their APIs. USI will partner with the developers, producers and vendors of operating systems, development tools, components and application ecosystems to realize the new, greatly expanded end-to-end solutions of the future.

1.4 Document Organization

Chapter 2 describes the underlying use cases that drive this specification.

Chapter 3 defines architecture requirements that stem from the use cases.

Chapter 4 provides a high level overview of the architecture.

Chapter 5 and 6 define the core of the USI specification. These focus on the communications physical and data link layers as well as the messaging protocol flow and error handling.

Chapter 7 documents changes to the HID interface protocol that are required to expose the capabilities of the USI Stylus to the host operating system

Chapter 8 provides an analysis of the power requirements.

2 Usage Scenarios

2.1 Introduction

Usage scenarios are real world examples of how users interact with styluses and devices.

The USI specification supports a number of usage scenarios, including the operation of single and multiple styluses with single and multiple devices, sequentially or concurrently.

One example: a creative writing/sketching session. When the author picks up a stylus and moves it to the device, the actual writing (inking) session starts when the stylus approaches a device. When the stylus is operating on the device, the session is in progress.

Of course, the author doesn't expect this progress to stop when he or she moves to another device. Instead, the immediate expectation is that progress will continue as the stylus moves to another device. The stylus needs to operate just as a pencil does – a pencil doesn't suddenly stop making marks when the author picks up a new piece of paper or starts a sketch on an art pad.

Once the author finishes writing and sketching, he or she might put the stylus away and end the creative session.

The exact definition of when an author's session ends is subjective and varies according to the author's preferences, intentions and general behavior. Does the author's working session stop when the author pulls the stylus away from the surface because he or she is thinking? Hopefully not.

Generally, a 10 to 15 minute timeout (no use of the stylus on the device surface) is the guideline of when the author's session ends. Sometimes an author has significant gaps in output at the beginning of the effort. However, once it starts, a 10 to 15 minute gap in usage is a viable guideline for the session having ended.

A detailed study of user behavior with a stylus and a screen is documented in Appendix D and provides a few key takeaways for product designers:

- The study defines an 'Approach Time', which is the average time it takes for the user to move a stylus from a position at a certain height (10 mm) to the point where it touches the screen. The optimal value of Approach Time was determined to be 35 ms.
- Also studied was the 'Dwell Time'. This is the interval between the stylus first physically touching the screen and it being lifted up again from the screen (the process of "tapping"). The average Dwell Time value was noted to be approximately 30 ms.
- Actual user timing measurements of user behavior are critical to the system designer's understanding of both the importance of the time it takes a mark to show on the screen and the acceptable versus unacceptable latencies of the actions of the overall stylus-device system.

2.2 Single Stylus with Single Device

A single stylus operating with a single device is the most common usage scenario. The single device is a touch capable device, such as a PC, tablet or smart phone.

In a writing session, when a stylus approaches a screen, it shall be detected within a latency period that allows a seamless and fluid user experience. A stylus is detected when it is in close proximity to the screen. The user expects to see some sort of feedback on the screen (usually in the form of a hover

cursor). If the stylus is used for writing or drawing, inking needs to start as soon as the stylus tip touches the screen. If the stylus is used for on-screen object selection, then inking needs to start immediately after an icon selection is successful.

A stylus can be used in a variety of ways. Some of the common uses include:

- Writing
  - A student writes a solution to a problem.
  - A business executive takes notes in a meeting.
  - A home user creates a to-do list, and periodically adds items to the list.
  - An author writes a chapter of a new book.
- Annotating documents
  - Documents are edited, transferred for comments, and transferred back for more work, all 'on the fly'.
- Drawing
  - Detailed drawings and paintings are created by an artist, with the same variety of brushstrokes, colors, mixture palettes and effects available with physical media – and all in real time (no programming, no extraneous setup).
- Pointing and user interface navigation
  - Manipulating user interfaces that contain millions of small identification points (maps, photos, design graphics), a stylus is capable of much more precision input than fingertips can provide.
- Signature capture
  - Forms are presented in software for clear (legible) user annotations, initials and signatures.
- Gaming
  - More precise inputs than typically possible with mice and game controllers enable new types of games.
  - Some games use forms and precise responses to enable interactive interfaces with the players – such as Sudoku and crossword puzzles

2.3 Simultaneous Stylus and Touch

Most devices that support stylus input also respond to finger touch – often supporting multiple fingers at the same time. In a photo album, for instance, a user might shuffle, organize, move and delete pictures – all using multi-touch gestures (e.g., pinch-to-zoom, hand rotation, flick left, flick right, etc.) by both hands and up to 10 fingers. On some larger screens, such as a lay-flat portable all-in-one, multiple users might use both hands to operate on the device's touch screen.

The user expects that the device's multi-touch capabilities continue to operate when the stylus is being used.

For instance, while an artist is drawing with one hand, he or she might use the other to change colors or to perform gestures to zoom in/out. And the user's latency expectations for the drawing actions remain the same as if the other actions were not taking place.

Of course, during stylus operations, the user also expects to be able to rest his or her palm on the screen without the system sensing the palm as a touch. All palm touches need to be rejected.

2.4 Single Stylus with Multiple Devices

Multiple device usage is normal today. Many users regularly use desktop PCs, laptops, tablets, smartphones, as well as more dedicated devices, including 'smartTVs'/DVRs and automotive navigation/entertainment/communications systems. Users want/need to be able to carry a single stylus and use it with all of their devices. All while, the user expects a consistent experience – performance, latency and otherwise across all of the devices.

When a stylus is in close proximity to a device screen, a hover cursor appears. When the stylus is in contact with the screen, it inks and when it physically leaves the screen, a hover cursor re-appears briefly while the stylus is still in close proximity to the screen. Once it is out of range of that screen and in close proximity to another, a hover cursor appears on the second screen. Finally, when it is in contact with the second screen, the stylus inks on that. If the stylus is in range of both the first and second screens, the stylus continues to work on the screen on which it was last inking as long as it still is in proximity of the screen (as indicated by the hover cursor). If the hover cursor disappears, the stylus can be used with the other screen.

An example of a single stylus being used with multiple devices:

- The user starts by taking notes in a meeting with a tablet. The notes are shared in real time on all of the user's devices.
- The user edits those notes on a PC for a report.
- During a commuter train ride, the user reviews the report on a smartphone and makes minor corrections with the stylus.
- The user does a final review of the report on another PC and adds a signature before submitting the final report.

2.5 Multiple Styluses with a Single Device

2.5.1 Introduction

Some devices are able to simultaneously support multiple styluses. While this capability might add complexity, it is needed to support expected near-term usages and advanced future scenarios. The latency and performance expectations remain the same as they are with a single stylus on a single device.

A few examples of the need for supporting multiple styluses:

- The user lost the stylus that came with the tablet, picks up another stylus and starts writing (this particular case does not require simultaneous operation).
- Two users, each with a stylus in hand, are collaborating on a presentation; they take turns marking and annotating the document on the same device
- Same as the two-user case above, except that the users want their annotations to be identified uniquely, so they can modify/erase their own content
- On a lay-flat all-in-one that has a multi-touch screen, two to four users play a game, simultaneously using their own styluses to write or draw on the screen.

Depending on its implementation, a device might support one or more styluses at the same time. The user needs a way to determine how many simultaneous styluses the device supports.

For example, if a device only supports four styluses at once, and a fifth stylus arrives, then the fifth will not be able to function. The device either gives an indication of why the fifth stylus does not work or supports user to queries about the device's capabilities.

When one of the four styluses goes out of range, the fifth stylus can be discovered and used.

If the device runs out of some of its other resources before it reaches the advertised limit of the number of styluses, then it provides a notification to the user that no more styluses can be supported. This might happen if one of the stylus is using advanced features that require the transfer of additional information / applications, and thus the device's resources become saturated. Example: an implementation advertises that four styluses are supported and four styluses are detected by the device, but one of the styluses uses special features that increase the communications payload so heavily that data link bandwidth runs out. In this case the implementation might choose only to support two or three styluses until the necessary resources are freed up. A warning message appears to notify the users.

When a device implementation supports multiple styluses, it needs to support at least two styluses, even if they are feature rich styluses that saturate the available resources. And, in that case, the devices needs to issue a warning message to notify the user(s) when additional styluses attempt to write on the device screen.

2.5.2 All Styluses are USI

While one user is writing on a device with a USI Stylus, another user brings in another USI Stylus and makes some annotations on the notes written by the first user. Both USI Styluses work simultaneously, and show the written text and annotations in different (application specific) colors.

USI Styluses can come and go seamlessly on the device and the user's experience with the stylus that is inking is not impacted. Additional styluses continue to work until the device reaches a limit of what its implementation can support. A new stylus is never able to 'bump off' a stylus that is currently being used on the device. And the user is always able to find out how many simultaneous styluses an implementation can support.

2.5.3 USI Stylus Followed by a Proprietary Stylus

While one user is writing on a device with a USI Stylus, another user brings a proprietary stylus near the device screen and attempts to write.

If the second stylus supports only proprietary mechanisms (that is, it has no USI support), then this new stylus does not work.

If the second stylus is dual mode capable (it supports both USI and proprietary protocols), then the second stylus works in the USI mode and the users are able to use both styluses simultaneously.

When the USI Stylus moves away from the screen, the proprietary stylus can approach and operate in proprietary mode.

2.5.4 Proprietary Stylus Followed by USI Stylus

In this case it is assumed that the device is dual mode capable (i.e., it supports both USI and proprietary protocols).

While the user is writing on a device with a proprietary stylus, another user brings a USI Stylus near the device screen and attempts to write. In this situation, the approaching USI Stylus does not work and the user receives a notification from the system.

When the proprietary stylus moves away from the screen, the USI Stylus can approach and operate normally.

2.6 Stylus Features and Extensions

USI Styluses can support many unique features, such as pre-assigned colors, multiple buttons, and other value-add features via globally unique stylus identifiers and vendor extensions. This allows stylus technology suppliers to provide a rich ecosystem for a wide variety of devices.

When a stylus is detected by a device, such features are also discovered by the device.

Any of the supported features can be used in any of the scenarios described in this chapter.

Additional features are exposed when both sides (stylus and device) are capable of supporting the same features that are not part of the USI required features. For example, a USI Stylus might have an extension feature of reporting the barrel pressure information as the user is holding the stylus and pressing it with his fingers. This 'squeeze' information is a feature extension on stylus A and is supported on device D1, but not on device D2. This feature needs to be exposed by device D1 to the applications. The stylus still operates with device D2 and provides all required inking features. . If the user's application supports the proprietary extensions from stylus A, the user expects that information to be made available by device D1.

2.7 Expected Usability of Common Stylus Features

Various features of the USI Stylus need to behave the same across all implementations.

For example, 'Erase' is a common feature for a stylus. The erasure feature is normally assigned to one of the buttons located on the stylus barrel. Another method is to use the tail end of the stylus as an eraser (flip the stylus and erase like a pencil). An erase feature can be activated when the stylus is within the detection range and either the assigned button is pressed or the tail end of the stylus is facing the screen. The erase function might be assigned to other buttons on the stylus, if desired. Still, the latency between the button press and the feature activation needs to remain the same as the inking latency.

Changing the color of the ink is another common example. Often, color can be changed by using a menu in an application. However, certain colors might be accessed directly from buttons or another interface that is available on the stylus. Or the user might cycle through certain colors by button presses. In addition, a stylus can be manufactured with a dedicated output color.

The inking style, width and pattern might be changed for a drawing application, either through application menus or a button press from the stylus. Some styluses provide persistent color and inking style configuration that can persist across devices. For example, when a classroom teacher is making corrections on several students' devices, the teacher could set the pen ink to red and expect to write in the same color on all of the devices. In other implementations a special dedicated stylus might be designed to always provide a particular inking style, such as a brush.

All of the latency requirements for inking described in chapter 3 are also applicable to changing configurations and modes, as much as they are for the simple button press.

2.8 Power and Battery Life

An USI Stylus needs to have a battery life that provides a good user experience. The actual battery life will vary with the technology used (see section 3.21 for the exact recommendation for each technology).

At any time the device needs to be able to report the stylus' battery status and remaining battery life to the user, so that the user can take timely action (such as changing the batteries or recharging) before the stylus loses power.

§ §

3 Requirements for the USI™ Technical Specification

This chapter defines the high level architectural requirements that apply to this USI Stylus and Device Technical Specification, itself. This chapter does not define the requirements either for a USI Stylus or for a USI Device.

Note: The functional and interoperability requirements that a USI Stylus and USI Device shall meet are all defined in Chapters 4 through 8 of this specification. And the testable details that shall be met for compliance with these requirements are all defined in the USI Certification Program.

This chapter defines only the requirements placed on the architecture defined by this specification.

Note: in the remainder of this chapter the noun “stylus”, by itself, refers to a USI Stylus, the noun “controller”, by itself, refers to a USI controller, and the noun “device”, by itself, refers to a USI Device. In addition, “this specification” and “specification” refer to the present document, the USI Stylus and Device Technical Specification.

3.1 Stylus Functionality

The USI Stylus is targeted for systems that use a common sensor to detect both finger touch and stylus, either hovering close-to or in-contact-with the sensor that is part of each device.

This specification shall provide a means for a controller to retrieve stylus position coordinates, the status of one or more buttons on the stylus, and information about the force (pressure) of the tip of the stylus on the device’s screen.

The stylus has a capacitive coupling to the sensor, and modulation of an e-field between the stylus and sensor conveys both the stylus’ current position and data. The specification shall provide a way for a stylus to receive control commands from the device. These commands are primarily used by the device to discover nearby styluses and tune the operation of each stylus.

3.2 Interoperability

The specification shall specify a way for a stylus to operate with, and provide at least a minimum of stylus functionality with, any device (containing a touchscreen) that is marked as ‘USI Certified’. Although voltage level, duty cycle, pulse shape (rise/fall time), pulse jitter, wander, drift, and harmonics might not be the same for all USI Styluses, the specification shall specify the e-field requirements for signal transmission that shall be met by all USI Styluses. The specification shall specify the reception of the stylus signal at a specified hover height and specified stylus tip angle.

Similarly, the specification shall define the e-field requirements for transmission of the signals that can be received by the USI Stylus. The specified stylus communication links should be compatible with all types of capacitive touch sensor technologies. The USI Stylus utilizes one or more frequencies that have less charger and display noise, and avoids frequencies occupied by other applications that might cause interference.

USI is defining the USI Compliance Test Plan, which describes the certification conditions and tests that are required for conformance to this specification. All USI conformant implementations are by definition those that are compliant with the USI Compliance Test Plan.

3.3 Touch Panel Technology

The specification shall specify the communication link for USI Stylus to work with capacitive touch capable devices. However, the USI communications protocol architecture specified by this specification shall not preclude other technologies (optical, ultrasonic, electro-magnetic resonance etc.) from being considered for support in future USI specifications.

3.4 System Noise Avoidance

The specification shall specify the protocol such that the USI Stylus operates under all reasonable noise conditions and interference that might be caused by chargers, displays and noise from other radios (WiFi, Bluetooth, broadband, etc.) on the device. This specification shall specify these noise and interference levels. The protocol specified by the specification should consider the interference from any other signal transmitting or interfering components in the system and any other noise caused by the external environment. In all cases the specification should specify best practices for the USI Stylus and the devices to avoid the interfering noise.

3.5 Scalability

The specification shall support one or more styluses operating on a USI Device. Some device implementations may only support a single stylus at one time, and may not support concurrent use of multiple styluses. But, for implementations that can simultaneously support multiple styluses, the specification shall provide the framework needed to support them. And the specification shall provide the user with a way to discover the capabilities of each device and to find out the maximum number of concurrent styluses it can support.

This specification shall supports the full range of screen sizes – including smartphones, tablets, laptops and all-in-ones. This range includes 4 inch to 32 inch screens.

The goal of this specification is to provide a foundation that will work with all current and future display and capacitive sensor technologies (i.e. OGS, ITO, single or multi-layer sensors, metal mesh, on-cell, in-cell, etc.).

3.6 Contact Accuracy

The specification shall allow a USI Stylus to meet the following contact accuracy requirements.

When the stylus is in contact with the screen, the physical contact and the contact position that the device reports shall be:

a) ≤ ± 0.5 mm while the stylus is touching the center portions of the screen

b) ≤ ± 1 mm when the stylus is within 3.5 mm of the device screen's edges.

3.7 Hover Accuracy

The specification shall allow a USI Stylus to meet the following hover accuracy requirements.

When the stylus is hovering, its physical position shall be reported by the device within ≤ ± 1 mm (center) of its real position at a height of 10 mm. For heights above 10 mm there shall be no specification, but the error should reduce as the stylus' height approaches 10 mm.

3.8 Position Jitter

The specification shall allow a USI Stylus to meet the following position jitter requirements.

Position jitter is defined as the temporary deviations of the reported position from the stylus' actual position.

Reported position jitter, whether stationary or moving, shall not exceed a maximum of both:

a) ≤ ± 0.3 mm while the stylus is in the center of the screen

b) ≤ ± 0.5 mm when the stylus is within 3.5 mm of the device screen's edges.

3.9 Hover Range

The specification shall allow a USI Stylus to meet the following hover range requirements.

The stylus' location is determined by the controller and is reported while it is hovering (≤ 10 mm). Above 10 mm thru 15 mm the stylus is 'in-range', where it may be detected by the controller, but no location coordinates are reported. In-range detection may be used for assistance with palm rejection and to provide a better overall user experience.

3.10 Hover Jitter

The specification shall allow a USI Stylus to meet the following hover jitter requirements.

Hover jitter is defined as the temporary deviations of the reported hover position from the actual hover position perpendicular to the screen. Reported hover position jitter shall not exceed a maximum of ± 0.5 mm when the stylus is hovering at the height specified in section 3.9, whether at the center or the edges of the sensor.

3.11 Time to Ink – Idle (at the start of a writing session)

Time to ink (TTI) latency is defined as the delay between the time the stylus makes its initial approach to the screen and the time the stylus' location is first ready to be reported to the device's operating system. Idle state is defined as the state when a writing session is not in progress. Chapter 2 describes the writing session from the user's point of view. The specification shall allow a USI Stylus to meet the TTI idle latency (at the start of a writing session) target of less than 150 ms.

3.12 Time to Ink – Active (during a writing session)

TTI active latency is defined as the delay between the time the stylus in active state makes its initial contact with the touch screen and the time the stylus' location is first reported to the device's operating system. The specification shall allow a USI Stylus to meet the TTI active latency target of less than 35 ms.

3.13 Move Latency

While the stylus is in contact with the screen, the move latency is the delay between the time the stylus arrives at a location and the time its location is reported to the device's operating system. The specification shall allow a USI Stylus to meet the move latency target of less than 10 ms.

3.14 Eraser

The specification shall enable a USI Stylus to implement a hardware mechanism to provide eraser functionality. This mechanism can operate via a specific button, or by the 'back of the pen' acting as an eraser. The specification shall allow this information to be reported to the device's operating system within the same latency requirements as the time to ink latency (idle or active).

3.15 Buttons

The specification shall enable a USI Stylus to support at least two buttons and sending the button state information to the device. . The specification shall allow this information to be reported to the device's operating system within the same latency requirements as the time to ink latency (idle or active).

3.16 Stylus Force/Pressure

The specification shall enable a USI Stylus to report multiple levels of tip force. Some implementations may support 256 levels of force while other implementations may support as high as 4096 levels of force. This specification shall allow the reporting of these ranges of force as chosen by the implementation.

The minimum value of the force below which a stylus is not required to report the tip force is referred to as the activation force and is typically about 10 g.

The maximum value of force above which a stylus is not required to report the accurate tip force value is 350 g. However, a stylus shall still report either true value or at least 350 g.

The specification shall allow a USI Stylus to report the full range of values between the Activation force and the Maximum force. A stylus shall measure and report pressure at least at the rate of 60 samples per second.

Note: stylus force is often referred to as 'pressure' in marketing descriptions. The value reported by the stylus is a force value in grams.

3.17 Stylus Report Rate

Stylus report rate is defined as the number of touch samples per second (in Hz) that are reported when the stylus is in contact with the screen. Stylus implementations today range from 60 Hz to 240 Hz

reporting rates. Future implementations are expected to range from 40 Hz to 300 Hz report rates. The USI specification shall define a protocol that allows the 40 Hz to 300 Hz range of report rates to be supported by specific implementations.

3.18 Resolution

The specification shall allow the resolution of the stylus' position reporting to be at least 150 dots per inch (dpi). For device native display resolutions larger than that, the specification shall allow the stylus position reporting resolution to be equal to or greater than the device's native display resolution.

3.19 USI Coexistence with Proprietary Styluses

The specification shall allow the device and stylus manufacturers to implement support for proprietary modes of operation, in addition to their fully supporting the specified USI functions. The details of operation in proprietary modes shall be left to the vendor, except for discovery and switching between USI and proprietary modes, both of which shall be defined by this specification.

Table 3-1 shows the various modes of operation for the USI Stylus or proprietary stylus, when it is operating with a USI controller or with a proprietary controller. In this table the heading 'Proprietary + USI' refers to a dual mode implementation that can switch between one USI and one proprietary mode. A dual mode implementation shall still be considered USI compliant as long as both the following conditions are met:

- It can switch to USI mode when it detects that it is communicating with an USI implementation.
- When it is in USI mode, the implementation meets all of the test requirements specified in the USI Compliance Test Plan.

Table 3-1 USI and Proprietary Modes of USI Stylus Operation

Stylus / Controller	Proprietary	Proprietary + USI	USI Only
Proprietary	Proprietary mode	Proprietary mode	Not discovered
Proprietary + USI	Proprietary mode	USI mode preferred	USI mode
USI Only	Not discovered	USI mode	USI mode

3.20 Vendor Extensibility

USI Stylus and controller vendors can use vendor specific features to provide additional or proprietary data of interest to the device and the applications. These data are in addition to the standard data, such as pressure, button pushes and tilt that shall be defined in this specification.

For example, the stylus can:

- Provide squeeze information as the stylus body is pressed
- Provide grip identification on how the stylus is held in the hand.

The USI specification shall allow such vendor specific information to be communicated from the stylus to the device.

3.21 Power and Battery Life

The stylus is in 'active use' when 50 percent of the time is spent in actual writing (stylus is in contact with the screen), even though the stylus is not in contact with the screen the rest of the time.

The specification shall define a protocol such that, with a high quality dry battery that has a rating of 500-600 mAH, the USI Stylus should be able to operate for 1 year with an average of 3 hours per day of active use. This assumes a 500-600 mAh alkaline AAA battery. The battery life might vary with the use of other battery types. For instance, for a super-capacitor USI Stylus the expectation is that, with 10 - 15 seconds of charge time, the charge should last for 2 hours of active use. (The stylus is in active use when it is in session.)

The specification shall define a mechanism for a user to be able to ask the stylus at any time about its battery status and remaining battery charge (at a minimum a low battery indication), so that the user can take timely action (such as changing the batteries or recharging) before the stylus loses power.

3.22 Performance Requirements Summary

Table 3-2 summarizes the system level performance stylus requirements that shall be implementable by following this specification. Detailed compliance tests and criteria are defined in the USI Compliance Specification.

Table 3-2 Performance Requirements Summary

Parameter	Value
Contact accuracy (center)	≤ ± 0.5 mm
Contact accuracy (edge)	≤ ± 1.0 mm
Hover accuracy	≤ ± 1.0 mm
Jitter (center)	≤ ± 0.3 mm
Jitter (edge)	≤ ± 0.5 mm
Jitter (hover up to 10 mm)	≤ ± 0.5 mm
Time to ink latency (idle)	≤ 150 ms
Time to ink latency (active)	≤ 35 ms
Move latency	≤ 10 ms
Force (reportable range)	10 g –350 g (Recommended interval)
Report Rate	Range between 60 Hz-300 Hz
Resolution	150 dpi or the native resolution, whichever is greater
Battery life	≥ 1 year with a 500 - 600 mAh AAAA battery

§ §

4 System Architecture Overview

4.1 Architecture Introduction

The core of the USI Stylus and Device Technical Specification is the definition of the communication link between the stylus and the controller. The USI specification defines the physical layer, data link layer and packet transfer protocols for this link. The chapters of this specification cover the details of each layer and define the architecture. This section provides an executive summary of the communication link.

Figure 4-1 shows the physical components in a USI Device that interact with a USI Stylus.

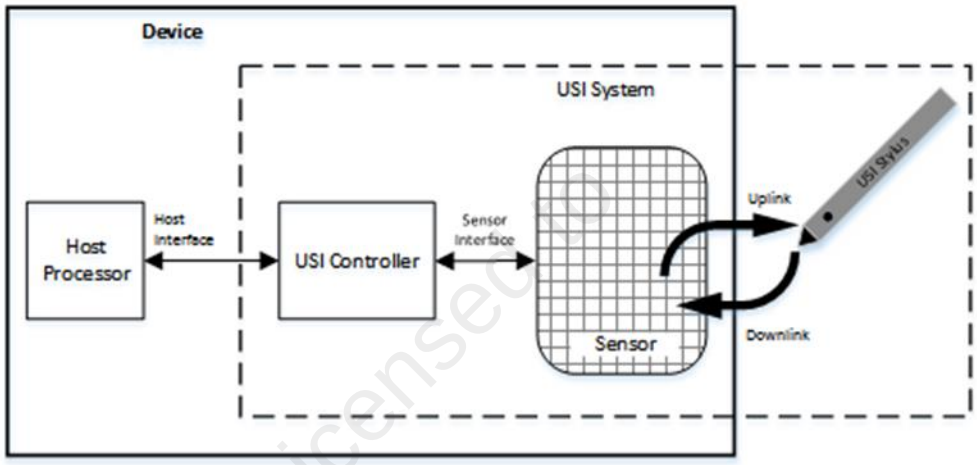


Figure 4-1 USI System Architecture

The term 'device' refers to a computing system (e.g., a mobile phone, tablet, laptop, 2-in-1, all-in-one, desktop). A USI compliant device supports a USI Stylus using capacitive touch technology.

The device has a capacitive touch sensor layer consisting of an array of electrodes, arranged in a vendor-specific grid pattern that can detect changes in capacitance. The touch detection mechanism might operate by using interaction with a passive conductive element (a passive stylus, or a finger), or operate through a signal sent by an active stylus. This specification is focused on the transmission of signals between an active stylus and the controller in the device.

The device typically contains a controller that drives the array and senses the signals and the changes in capacitance values from the touch sensor. The controller is required in the USI architecture. In addition, it is assumed that the same controller supports both touch and at least one active stylus. There may be implementations of devices that have a separate 'stylus controller', and this is not prevented by this specification, but such a scenario is an unlikely implementation. Therefore, in this specification the term 'controller' applies to the controller that communicates with the USI Stylus, whether or not it also supports touch by fingers or a proprietary stylus.

The host processor in the device is connected to the controller through a physical interface bus-such as I2C, SPI, or USB. In Figure 4-1 this bus is referred to as 'host interface'. The definition of the physical bus used in this interface is outside the scope of this specification.

The stylus includes a transmitter (tip or cone of the stylus) that transmits a signal down to the controller by modulating its tip voltage. These voltage modulations can then couple to the sensor electrodes. This is called 'downlink'.

The spatial profile of the downlink voltage signal can be used by the controller to locate the stylus. Additionally, voltage modulation can be used to communicate ancillary data (such as force and button presses). The communication between stylus and controller is inherently near field communications and is limited to a distance of a few centimeters.

The location and ancillary data from the stylus can be communicated from the controller to the host processor by using the processor interface.

In a typical touch enabled device the controller drives an array of X-lines in the touch sensor. After driving the lines, the controller senses a perpendicular array of Y-lines to measure the capacitance levels and capacitance changes.

The controller can also modulate the voltage to create a signal on the X-axis electrodes; this signal can be received by the stylus. This communications path is referred to as the 'uplink' and is used to transmit configuration and timing information from the controller through the sensor to the stylus.

The USI enabled device uses this architecture to provide and control communications both up from the controller to the stylus (uplink) and down from the stylus to the controller (downlink).

In all USI interactions the controller acts as the master in a master/slave relationship, and is responsible for:

1. Discovering a USI Stylus in its vicinity
2. Establishing the communication link with the stylus
3. Configuring the stylus for further communication
4. Disconnecting the communication link when the stylus moves out of range.

Depending on the capabilities of the controller, it can establish communications with more than one stylus simultaneously by assigning each stylus a unique numerical identifier (ID). In this 'multiple stylus' case, the controller is also responsible for assigning IDs, arbitrating between the styluses and avoiding collisions among the assigned IDs.

4.2 Physical Layer

The physical layer is the lowest layer of a set of communication layers. The physical layer is defined by an electrical specification for the communication of raw bits between the controller and the stylus. The electrical specification details the amplitudes, broadcast frequencies, guard bands, modulation and coding schemes, timing and bit synchronization, and noise budgets for both the uplink and the downlink.

The USI org, after a careful analysis, has determined the frequency spectrum to be used by all USI implementations. The key factors determining this selection are charger noise, display interference and the frequencies in use by other signal transmitting components in the device (including wireless local area networks and mobile wide area networks).

The 100 to 500 kHz range has been selected because it is a relatively quiet and does not carry significant sources of interference. The uplink and downlink communications operate on different frequencies.

Uplink communication employs a broader frequency range, using direct sequence spread spectrum (DSSS) modulation. This modulation and frequency are fixed and defined by this specification.

Downlink uses a more targeted, narrower band. The downlink band is dynamically selected from several possibilities by the controller, based on dynamic noise and whether the controller currently is initiating

the support of multiple styluses at multiple frequencies. The frequency selected by the controller is communicated to the stylus using the uplink communication link.

While the preferred downlink frequency is in the quiet 325 to 500 kHz range, an implementation can use a lower band (100 to 325 kHz) if it determines that there is no interference at that frequency at the time the controller-stylus link is operating.

4.3 Controller as the Master

In the USI architecture, the communication between the controller and the stylus is based on the fundamental premise that the controller is the master and configures the stylus during an association 'pairing' process.

The controller periodically transmits a Beacon that is used for a timing reference for communication between the controller and the stylus as well as data communication. The periodicity of the Beacon is decided by the controller's implementation, but is always in the range of 4 ms to 25 ms.

An unpaired stylus (one whose communications are not yet linked to the controller) is listening for the Beacon. As the stylus starts to move into the range of the sensor/controller, it receives the Beacon and transmits an ACK in response to the Beacon. Once the ACK is received by the controller, the stylus-controller pairing process is initiated.

Information identifying the period at which the Beacons are transmitted is communicated in the discovery Beacon, or any time this information changes. This allows a stylus to know when it is expected to listen for subsequent Beacons. The stylus then can use this information to go to low-power modes at times when it is not expecting a Beacon to be present.

During the pairing process the controller directs the stylus to use a specific downlink frequency, a specific set of timeslots, and the information to be communicated in those timeslots. This is described in more detail in section 4.5.

4.4 Uplink Communication

The controller shall broadcast a Beacon periodically, with a period of between 4 ms and 25 ms. The duration of the Beacon shall be approximately 1 ms. All unpaired styluses shall listen for Beacons in order to discover the controller in its proximity. This receipt of the Beacon by the stylus and the stylus' subsequent ACK to the controller mark the start of the pairing process.

Additionally, the Beacon is also used for communication from the controller to the stylus. This is the only means for the controller to transmit to the stylus. The time between the Beacons is reserved for downlink communication and finger touch detection.

During the pairing process the controller transmits setup information to the stylus. This includes the downlink frequency and downlink slot(s) to use, and the type of information that should be sent by the stylus in those timeslots.

Even paired/operational styluses shall listen to the Beacons and interpret the information in them. This allows:

- The stylus to obtain frame timing synchronization information by aligning its time reference to the last bit of the received Beacon
- The controller to dynamically request changes in the behavior of the stylus (or each stylus, if it is paired with more than one) by transmitting new configuration information.

4.5 Downlink Communication

The time period between two successive Beacon starts (T_{Beacon}) is referred to as a 'USI Frame'. A USI frame is evenly distributed into downlink timeslots. These timeslots are 250 µs long. The number of timeslots (up to a maximum of 64) is dictated by the available time between the Beacons. For example, a 16 ms Beacon transmit interval includes time for the 1 ms Beacon, 250 µs of turnaround gap, up to a 2 ms ACK, and 12.75 ms divided into 51 timeslots of 250 µs each. These 51 timeslots can be used by the stylus to send a position packet and the data it has available for the controller (button presses, force on the tip, etc.). If an ACK of 0.5ms is chosen, then this will overlap Slot 0 and this should not be allocated for the use of any paired stylus. Similarly ACK sizes of 1ms and 2ms overlap with Slots 0 to 2 and Slots 0 to 6 respectively.

The frequency for the stylus to use in its downlink communication is chosen by the controller, and the controller transmits its parameters inside the Beacon. The controller may allocate the same frequency to more than one stylus and manage the communications from the styluses by assigning them different timeslots (time division multiplexing). Or the controller may direct different styluses to communicate simultaneously on different frequencies bands (frequency division multiplexing). The controller may also use a combination in which it instructs two styluses to transmit on the same frequency and allocates different timeslots for them, and allows a third stylus to transmit on a different frequency and use all of the available slots in that frame.

Each stylus remains unaware of the existence of any other styluses that are paired with the controller. To fully communicate with the controller, each stylus needs to know the frequency and the timeslots on which it shall transmit.

The bits sent by the stylus in a timeslot are encoded using differential binary phase shift keying (D-BPSK).

4.6 Downlink Timeslot Allocation

The timeslots available for downlink communication are distributed between the styluses (one or more) that are paired with the controller. The controller may also reserve some timeslots for finger touch detection and others for communications housekeeping, as the implementation and circumstances demand. These decisions are left to the controller implementation.

The Beacon (uplink) shall be followed by 250 µs of turnaround gap to enable the stylus(es) to get ready to transmit. A period of 250 µs to 2 ms, immediately following the turnaround gap, shall be allocated for any ACK that may be received from an unpaired stylus. This period is set aside every USI Frame to assure that an unpaired stylus always has time available to respond to the controller. In this period the stylus has time to start communicating with the controller, to transmit its capability information and settings, to receive its timeslot assignment and to complete its part of the pairing process.

Once the stylus receives the schedule of allocated timeslots from the controller, it begins using those for all data communication. The controller may still use the beacon timeslot to transmit configuration information or a read request to the stylus. The stylus shall respond to these requests in the ACK timeslot.

The controller has the freedom to allocate the available timeslots to fit its needs. If it runs out of timeslots, it may choose among several alternatives, including- not pairing this stylus, using a different frequency for the new stylus, or reducing the report rates of various data fields.

4.7 Commands

The controller shall assign a stylus ID (3 bits) to every stylus that is communicating with the controller. A stylus ID value of 0 indicates an unpaired stylus; a value of 7 indicates all paired styluses (acting as a broadcast command). This stylus ID is only valid during the session. When a stylus unpairs, the stylus ID is not valid and the controller can reassign the ID number to a different stylus that gets connected.

All commands from the controller shall be transmitted in the Beacon. A command might be targeted to an unpaired stylus (for discovery), to all styluses (such as a request to move to a new frequency), or to a specific paired stylus (e.g., request it to transmit the stylus' globally unique stylus ID). The stylus shall transmit its responses to these commands in the ACK timeslot.

All required and optional commands are defined in this specification. In addition, several command IDs are reserved for vendor extensibility. Depending on the vendor ID of the controller, these reserved commands may have different meanings.

4.8 Support for Proprietary Implementations

Because a principal goal of the USI specification is to enable a smooth transition from today's proprietary implementations to USI standard implementations, the USI discovery process supports the discovery of proprietary implementations. This design is described in more detail in section 6.9.

The support of a vendor-specific extension mechanisms allows a vendor to create differentiated capabilities while interoperating fully with standard implementations from other vendors.

§ §

5 Physical and Data Link Layer

5.1 Overview

The USI controller communicates with the USI Stylus by transmitting a Beacon. The Beacon serves as a timing reference for all styluses and transmits commands to initiate two-way communication with one or more styluses. All configuration writes and status reads are initiated with Beacon commands.

An example of an USI Frame is shown in Figure 5-1 below. The first bit of the Beacon packet is the start of a USI frame. The commands from the controller are in the Beacon. The DSSS code match at the end of the last chip of the Beacon serves as the common timing reference point (T_{EOB}) for both the controller and the stylus. Together with the Beacon the ACK and Data packets sent by the stylus make up the USI frame.

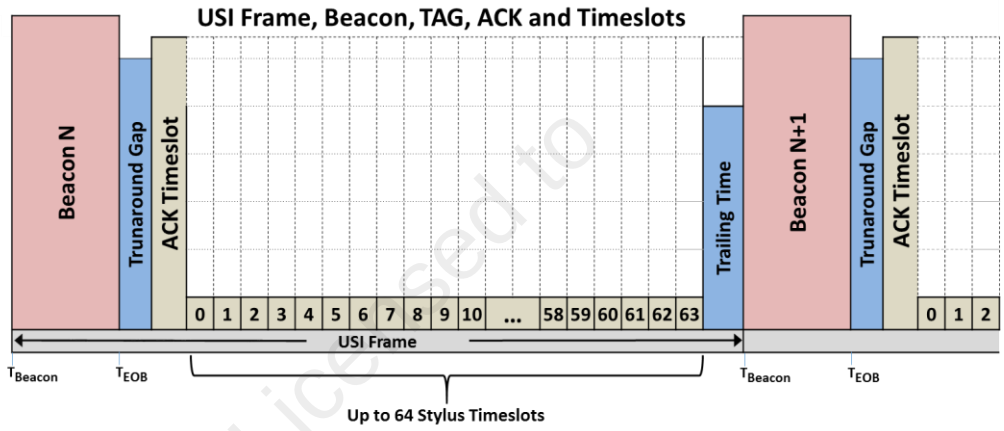


Figure 5-1 Example USI Frame Format

When an addressed stylus (paired or unpaired) detects a Beacon, it responds to Beacon commands in the Acknowledge (ACK) timeslot. Stylus responses are configured to be 1, 2, 4 or 8 timeslots long (250 µs, 500 µs, 1.0 ms or 2.0 ms). This response confirms the receipt of the command and returns any requested data. An addressed stylus responds to controller commands using the ACK timeslot and not its assigned timeslots, unless the Beacon from the controller is addressed all paired styluses. If the Beacon addresses all styluses, each stylus responds in its first configured data timeslot.

The controller configures the stylus to transmit one or more data or position packets. The controller assigns each packet to a specific stylus as well as a specific timeslot in which to start transmitting. The controller also assigns to each stylus a downlink frequency and a number of time slots to use for each packet. See section 5.5 for further stylus packet information.

Stylus Data packets provide data to the controller such as pressure and button information, or provide other requested read data of any of the data types.

The stylus Position packet contains a single frequency tone that can be used by the controller to detect the X-Y position of the stylus. The controller may or may not use Position packets.

Uplink transmission uses direct sequence spread spectrum (DSSS) modulation and downlink uses D-BPSK signaling.

DSSS is a spread spectrum modulation technique. The USI controller transmits the packet data using a bandwidth that is in excess of the bandwidth that is actually needed by the packet data. This spreads the transmitted signal over a large bandwidth, making the resulting wideband signal appear as a noise signal. This allows greater resistance to un-intentional interference with the transmitted signal.

The DSSS packet data is used to modulate a bit sequence known as the pseudo noise (PN) spreading code. USI uses a single PN spreading code. The PN spreading code consists of pulses of a much shorter duration (larger bandwidth) than the pulse duration of the packet data. Therefore the modulation by the packet data has the effect of chopping up (turning into 'chips') the pulses of the signal. The duration of each pulse of the PN code is referred to as the 'chip duration'. The PN code is a bit pattern with varying frequency components for each data bit that is transmitted. This further increases the signal's resistance to interference. If one or more chips in the PN code are damaged during transmission, the original data can be recovered by the stylus due to the stylus also using the same PN code to de-spread the transmission.

The stylus uses the same PN code to reduce the spread signal to its original bandwidth while removing any noise or interference. A DSSS spreading/de-spreading mechanism has a processing gain defined as the ratio of the spreading bandwidth over the packet data rate. See section 5.3 for more details.

A short preamble transmitted prior to the data ensures the detection of the first DSSS data bit. The preamble is a 3 bit Barker code, described in section 5.3.

The downlink uses D-BPSK (Differential Binary Phase Shift Keying), which is a form of phase modulation that conveys data at 1 bit per symbol by changing the phase of the carrier wave. It uses two phases to encode data; these phases are separated by 180 degrees.

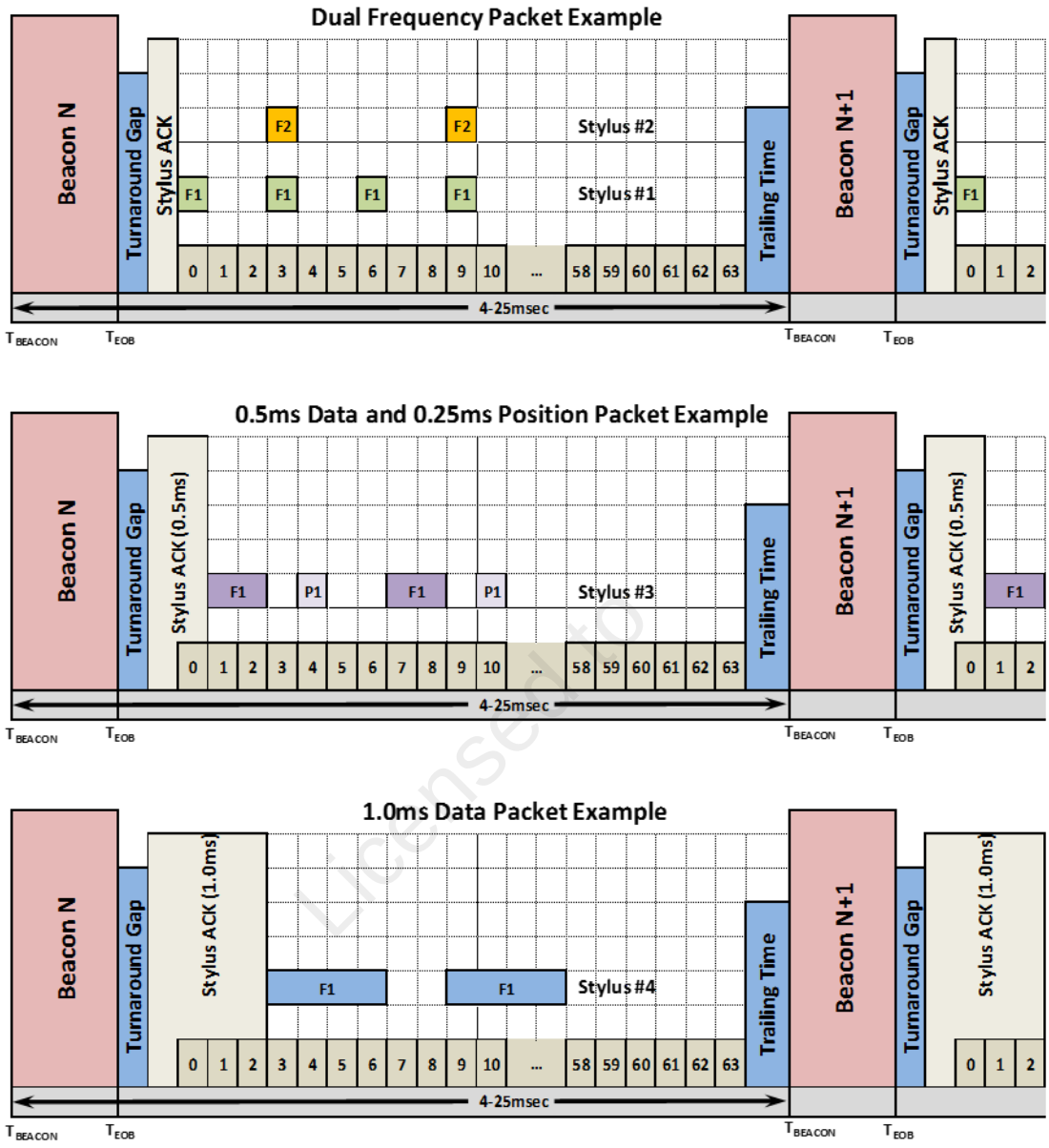


Figure 5-2 Example USI Frame Beacon, ACK and Packets

Figure 5-2 shows the signaling inside a USI frame. The correct decode of the last bit of the Beacon signals the common timing reference point to both the stylus and the controller.

A turnaround gap (TAG) of 250 µs follows the Beacon. The TAG allows the stylus sufficient time to process the beacon commands and to prepare a response. The stylus responds to beacon commands in the ACK timeslot.

The ACK packet is a data packet that always starts at 250 µs after the end of the Beacon. The ACK packet and the other data packets are configured to be 250 µs, 500 µs, 1 ms or 2 ms long.

The USI frame may be configured with up to 64 timeslots that are used for multiple styluses to transmit multiple data and position packets. All styluses reply to the controller commands using the ACK timeslot. In the example in Figure 5-2

- Stylus #1 is transmitting four packets each 250 µs long, using frequency 1 and Stylus #2 is transmitting two packets each 250 µs long, using frequency 2 (first figure in Figure 5-2).
- Stylus #3 is transmitting two packets each 500 µs long, using frequency 1, and two Position packets (P1) each 250 µs long. The ACK and Data packet durations match but the Position packet duration is unique (second figure in Figure 5-2).
- Stylus #4 is transmitting two packets each 1 ms long, using frequency 1. The ACK packet matches the Data packet (third figure in Figure 5-2).

Note that, in order to support two styluses in the same timeslot, the controller shall have a multi-frequency receiver.

5.2 Basic USI Communication Operation

The USI communication protocol is based on single master/multiple slave operation, with the controller being the master in the system and each stylus acting as a slave. Multiple styluses may be in communication with a controller at the same time. Each stylus in the system shall possess its own local identifier that is allocated by the controller (the stylus' local address is called its "StylusID").

The basic protocol starts with the controller sending a communication packet called a 'Beacon'. The Beacon is used as both a timing reference for communication in the current frame between the controller and each stylus, as well as a 'command' to configure the stylus and set up a dedicated communication link. The Beacon is a 33 bit packet. The period of the Beacon transmissions is controller dependent but should be in the range of 4 ms to 25 ms. The interval between the starts of two successive Beacons is referred to as a 'USI frame'.

When a stylus comes into range of a controller for the first time, the controller discovers the stylus and shall set up a communication link with it. When the stylus detects the beacon signal, it shall respond to each controller command in the ACK slot. This slot is of duration 250 µs and starts 250 µs after the end of the Beacon (T_{EOB}). As instructed by the controller, the stylus may use additional timeslots (TS0-TS6) after the allocated ACK timeslot to extend the ACK time. The ACK may be extended to a maximum of 2.0 ms.

Once a response has been detected by the controller, it may pair with the stylus and allocate a communication channel unique to that stylus. The channel consists of one or more starting timeslots for the stylus to send Data and Position packet to the controller, as configured by the controller. The controller also configures the number of time slots for all Data packets and all Position packets.

The timeslots are each 250 µs long; a maximum of 64 timeslots may be allocated. The actual number of timeslots available depends on the USI frame period. The number shall be a whole number of 250 µs timeslots (e.g., if the frame is 16.6 ms, then the number of slots is (16.6 - 1.5)/0.25 = 60 timeslots). Several timeslots can be combined to form a stylus response called a 'packet'. Each packet has a configurable duration and can be 250 µs, 500 µs, 1.0 ms or 2.0 ms long. The data contained in a packet are independent of the duration that is used and shall be fixed to 21 bits long.

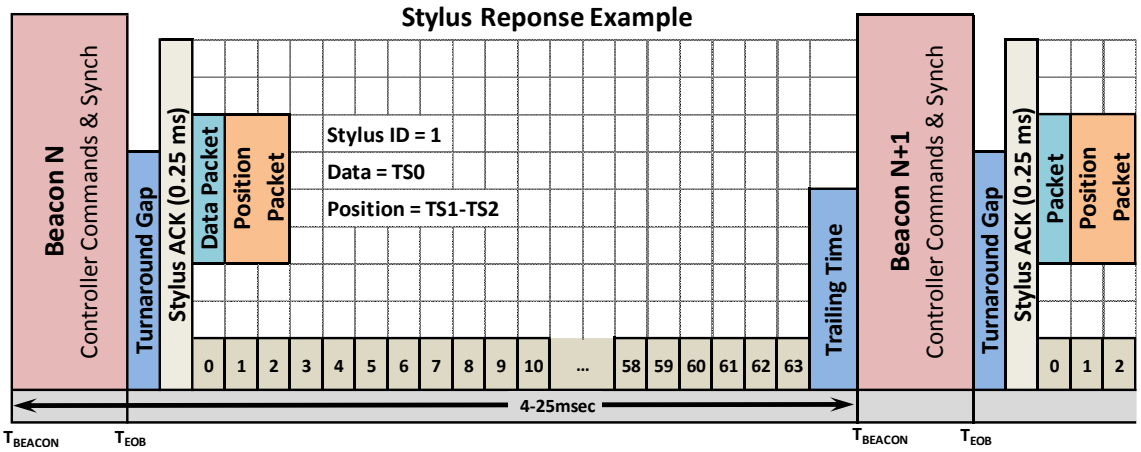


Figure 5-3 Example Stylus Data and Position Packet Response

In Figure 5-3 the stylus has been allocated a local Stylus ID (ID1) and a communication link has been allocated. The data packet duration and the ACK duration are both set to 1 timeslot, and the position packet is set to 2 timeslots. The single packet contains default information required for every USI frame, such as pressure and button status.

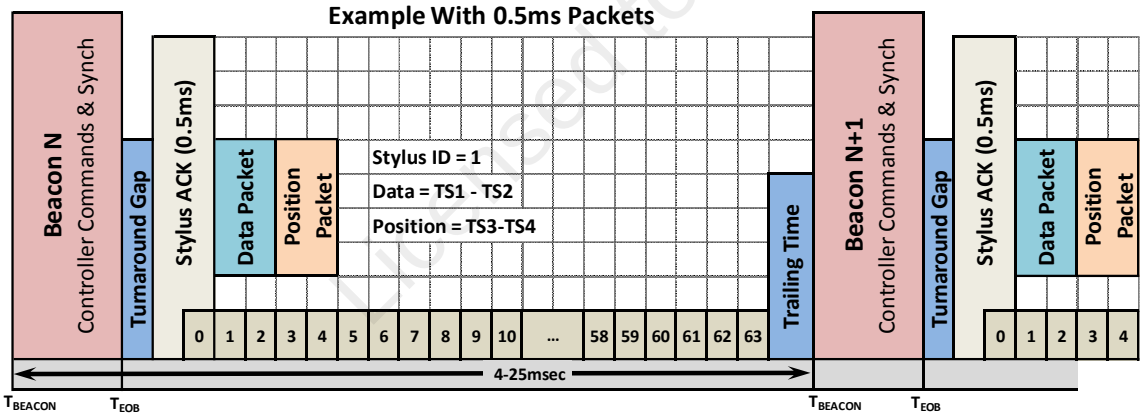


Figure 5-4 Example USI Frame Beacon, ACK Using 500usec Packets

In the example shown in Figure 5-4 the data packet duration and the ACK duration are both set to 2 timeslots, and the position packet is also set to 2 timeslots.
If additional information is required, it is requested via a command to the specific stylus; the stylus returns the data using the ACK timeslot.

If additional styluses come within range of the controller and become paired with the controller, the controller shall assign each a unique stylus ID and a unique communication link (with a set number of timeslots) to avoid collision with the controller's communications with the other styluses and within the timeslot capabilities of the controller.

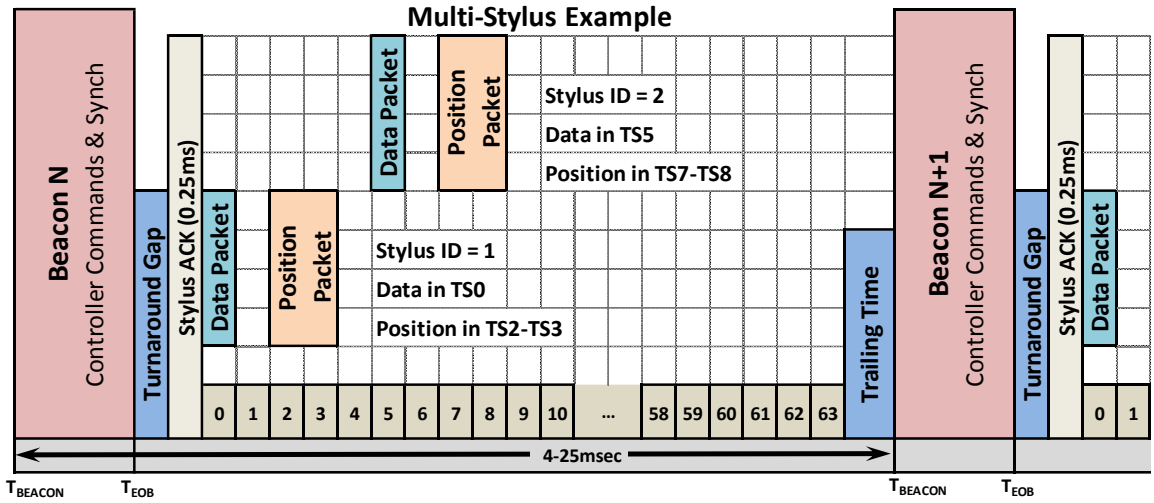


Figure 5-5 Example Multiple Stylus Frame Beacon, ACK and Packets

In Figure 5-5 two styluses have been paired with the controller and assigned different timeslot periods. Both styluses use an ACK packet of 250 µs. Stylus ID1 has been allocated a packet of duration 250 µs per beacon period and a 500 µs position packet. Stylus ID2 has been allocated a data packet of 250 µs duration and a position packet of 500 µs. In applications that support multiple styluses the ACK duration should be identical for all styluses.

The Beacon acts as a timing reference for the start of a new period; the end of the last chip of the Beacon is the 'zero' timing reference point (T_{EOB}).

5.3 USI Controller to Stylus Uplink

5.3.1 Controller Uplink

The controller nominally transmits the Beacon once every 16.6 ms but may transmit as slowly as once every 25 ms and as fast as once every 4 ms. The Beacon has a duration of 1023 ± 1.023 µs. In order to detect the controller, an unpaired stylus should listen for the Beacon for a 26 ms period. As part of the controller-stylus pairing negotiation the controller shall inform the stylus about the length of the fixed interval from the start of one Beacon to the start.

5.3.2 Uplink Beacon

The stylus shall not initiate any actions prior to a communication link being established with the controller. The controller searches for a stylus by transmitting an uplink Beacon that contains stylus configuration data. A stylus listens for a Beacon from a controller during a 26 ms window. An uplink Beacon is always 33 bits long and is comprised of 3 bits of preamble, 3 bits of stylus ID, a 6 bit command ID, 16 bits of data, and a 5 bit CRC. All unpaired styluses are addressed with stylus ID 0. Configured (fully paired) and partially configured (paired) styluses are assigned Stylus IDs 1-6; and Stylus ID 7 is used to issue a broadcast command to all fully paired styluses.

During the 1023 µs Beacon the controller shall reliably broadcast 33 bits of data to a stylus. The 33 bits of data are used for the commands to the stylus, specifying modes, reads, writes, and other functions. All

bits of the Beacon are transmitted LSB-first, starting with the Preamble[2:0] as shown in Figure 5-6 Uplink Beacon Fields and Bits

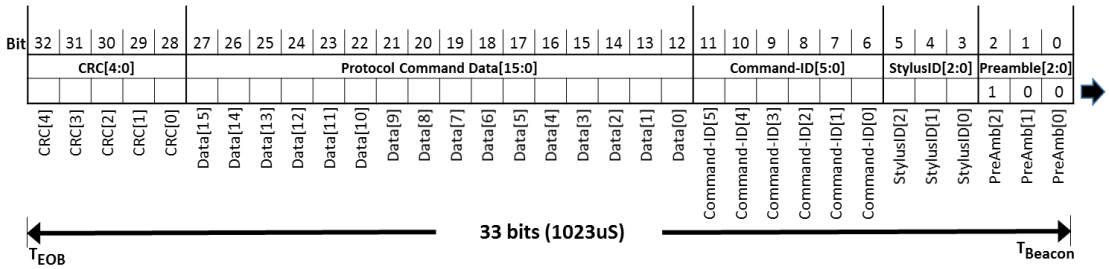


Figure 5-6 Uplink Beacon Fields and Bits

Table 5-1 Uplink Transmission Bit Order

Uplink Bit Index	Uplink Bit Transmitted
0	PreambleBit = 0
1	PreambleBit = 0
2	PreambleBit = 1
3	StylusID[0]
4	StylusID[1]
5	StylusID[2]
6	Command-ID[0]
7	Command-ID[1]
8	Command-ID[2]
9	Command-ID[3]
10	Command-ID[4]
11	Command-ID[5]
12	Data[0]
13	Data[1]
…	…
27	Data[15]
28	CRC[0]
29	CRC[1]
30	CRC[2]
31	CRC[3]
32	CRC[4]

5.3.3 Beacon Packet Error Detection

The Beacon command packet is comprised of a 3 bit Barker preamble, 25 protocol data bits and 5 bits for a cyclic redundancy check (CRC) to provide error detection. The 25 protocol data bits contain - 3 bit StylusID, 6 bit Command-ID and 16 bit Data fields as shown in Figure 5-7. The controller shall generate the 5 bit CRC on the 25 protocol data bits, and send it with each transmitted Beacon. All bits are sent LSB first

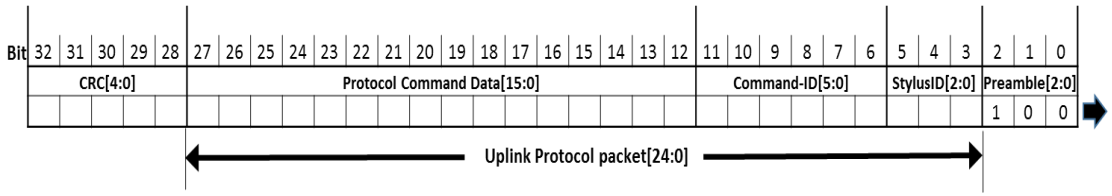


Figure 5-7 Beacon Command Structure

The CRC is calculated using the MSB as the first bit into the generator, the generator seeded value is all zeros, and the padding bits at the end are each '0'.

The generator polynomial is:

$$G(X) = X^5 + X^4 + X^3 + 1$$

The binary bit pattern that represents this polynomial is 111001B. On the receiver, the CRC is regenerated from the 25-bits of the Protocol Packet and compared to the received CRC to verify if the received protocol packet has any errors. The CRC polynomial is able to detect all Beacons that have 1 to 3 error bits, and is also able to detect all Beacons with an odd number of error bits. If the stylus receives a Beacon packet that contains a CRC error, it shall not process the command. The CRC pseudocode is in Appendix J.

5.3.4 Controller Transmitter Amplitude

The nominal amplitude of the uplink signal should be between 2.5 Vpp and 15 Vpp. The actual signal specification is in pC_{rms} and may be found in section 5.3.7

5.3.5 DSSS Spreading Code

A maximum length sequence (MLS) has been selected for the DSSS spreading code. Key attributes of a MLS are its periodic auto-correlation and flat fast Fourier transform (FFT) at all frequencies. It is a pseudo-random binary sequence and is generated with primitive polynomials. This provides maximum scrambling of single tone noise.

The USI DSSS coding sequence is a 31 bit MLS. In hexadecimal this code is 58F9A42B. As with the data bits, this DSSS code shall be transmitted LSB first.

The coding gain from this spreading code is calculated as $10\log_{10}$ (length 31) = 14.9 dB.

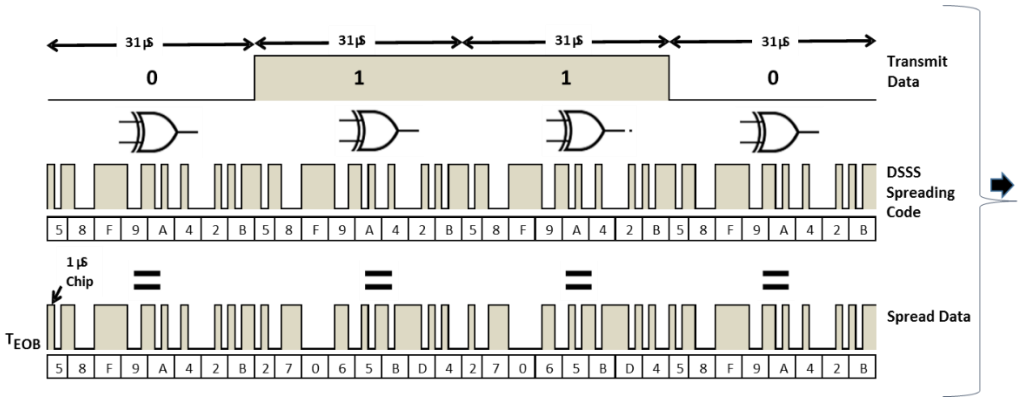


Figure 5-8 Example of DSSS Data Spreading Code

The example in Figure 5-8 shows how a '0110' data to be sent from the controller to a stylus is converted into a wideband signal using a PN spreading code using 1 µs chip signaling. The stylus reverses the process and de-spreads the signal back into the original data format using the same PN spreading code.

To ensure the detection of the first DSSS data bit, a 3 bit preamble is transmitted prior to the beacon data. The preamble is a 3 bit Barker code of 001. See Table 5-1.

5.3.6 Uplink Specifications

The uplink specifications are outlined in Table 5-2.

Table 5-2 Uplink Specification Summary

Parameter	Value	Description
Beacon broadcast rate	Min = 4 ms (250 Hz) Typical = 16.6 ms (60 Hz) Max = 25 ms (40 Hz)	Rate is set by the controller.
DSSS broadcast voltage	2.5-15 Vpp 3.3 Vpp nominal	Controller Tx level, assuming a copper sheet.
DSSS sequence length	31 chips	Each bit of uplink data is encoded as a 31 chip DSSS sequence.
DSSS '1' chip broadcast	High signal (nominally 3.3V)	Directly broadcast the output of the data bit XORed with the DSSS sequence.
DSSS '0' chip broadcast	Low signal (nominally 0.0V)	
DSSS code for binary data '0'	0x58F9A42B (LSB transmitted first)	Length 31 code is optimized to focus energy above 200 kHz.
DSSS code for binary data '1'	0x27065BD4 (LSB transmitted first)	This is the complement of the binary '0' code.
DSSS chip rise and fall times (20 % - 80 %)	< 300 ns	At the line driving the sensor. A slow sensor increases the rise and fall times.
Beacon preamble	Binary 100 (LSB transmitted first)	3 bit (93 µs) long preamble is used for synchronization. 31 bit DSSS spreading code is applied to the preamble.
Beacon data length	30 bits (930 µs)	31 bit DSSS spreading code is applied to data bits.
DSSS code chip rate	1 µs ± 0.1 %	
Sensor transfer function	Worst-case sensor gain shall be above that of a single pole at 144 kHz	

5.3.7 Controller Electrical Requirements

To ensure USI PHY layer signaling interoperability between USI Styluses and USI controllers, all controller uplink signaling shall meet the minimum electrical profile. See

Table 5-3.

To ensure that a stylus can receive a Beacon at a minimum hover height, the controller shall be able to transfer a minimum amount of charge to a test structure at specified conditions. A test fixture (see Figure 5-9) will measure the peak to peak AC charge transferred to a 4 mm diameter pad with a 150 micron (6 mil) gap in the middle of at least 50 mm diameter grounded plane on 1.5 mm (62 mil) FR-4. A solder mask may cover the sensor and plane. Detailed test fixture schematics and layout are available on the USI website.

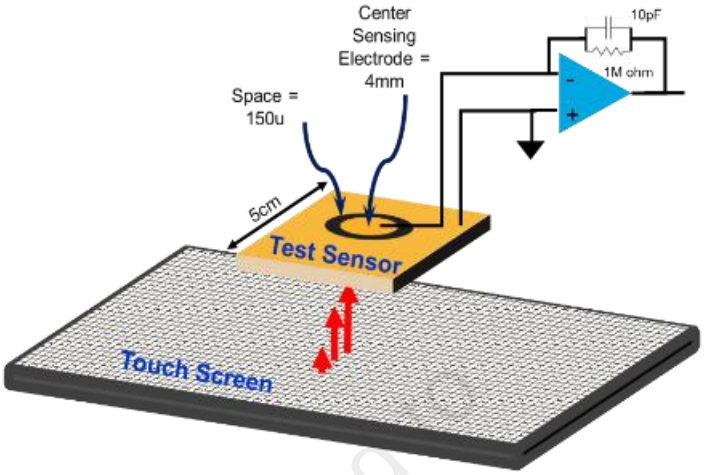


Figure 5-9 Test Fixture for Controller Electrical Measurements

Table 5-3 Controller Limits

<table>
<tr><th>Controller + Touch Sensor Parameter</th><th>Value</th><th>Description</th></tr>
<tr><td>Minimum Controller Charge Transfer at 100kHz</td><td>Touching > 100fC_{rms}
5mm Hover > 7.0 fC_{rms}
10mm Hover > 3.5 fC_{rms}</td><td></td></tr>
<tr><td>Controller transmitter charge transfer</td><td><table><tr><th>Frequency (kHz)</th><th>Signal Normalized to 100kHz Signal</th></tr><tr><td>100</td><td>1</td></tr><tr><td>200</td><td>> 0.71</td></tr><tr><td>300</td><td>> 0.53</td></tr><tr><td>400</td><td>> 0.41</td></tr><tr><td>500</td><td>> 0.34</td></tr><tr><td>600</td><td>> 0.28</td></tr><tr><td>700</td><td>> 0.25</td></tr><tr><td>800</td><td>> 0.22</td></tr><tr><td>900</td><td>> 0.19</td></tr><tr><td>1000</td><td>> 0.17</td></tr></table></td><td></td></tr>
<tr><td>Controller transmitter edge rate</td><td>Rise and fall time of ≤ 300 ns at the controller pin</td><td>20 % to 80 % of signal swing
(for full swing transitions)</td></tr>
<tr><td>Controller transmitter chip frequency accuracy</td><td>± 0.1 %</td><td>For DSSS data transmission</td></tr>
</table>

5.4 Stylus Tip

5.4.1 Sensor-to-Stylus Coupling

Empirically, as shown in Figure 5-10, the capacitive coupling from the entire surface of a sensor to a stylus that is touching the surface is between 100 fF (for a 5 mm long, 1 mm diameter tip) to 1.5 pF (for a cone type tip). At 10 mm hover height the signals drop to ~25 % of their original values and to ~7 % to ~10 % at 100 mm hover height.

The coupling of a stylus tip to a single sensor trace drops much more rapidly. This implies that the stylus can detect the sensor from much farther away than the sensor can detect the stylus.

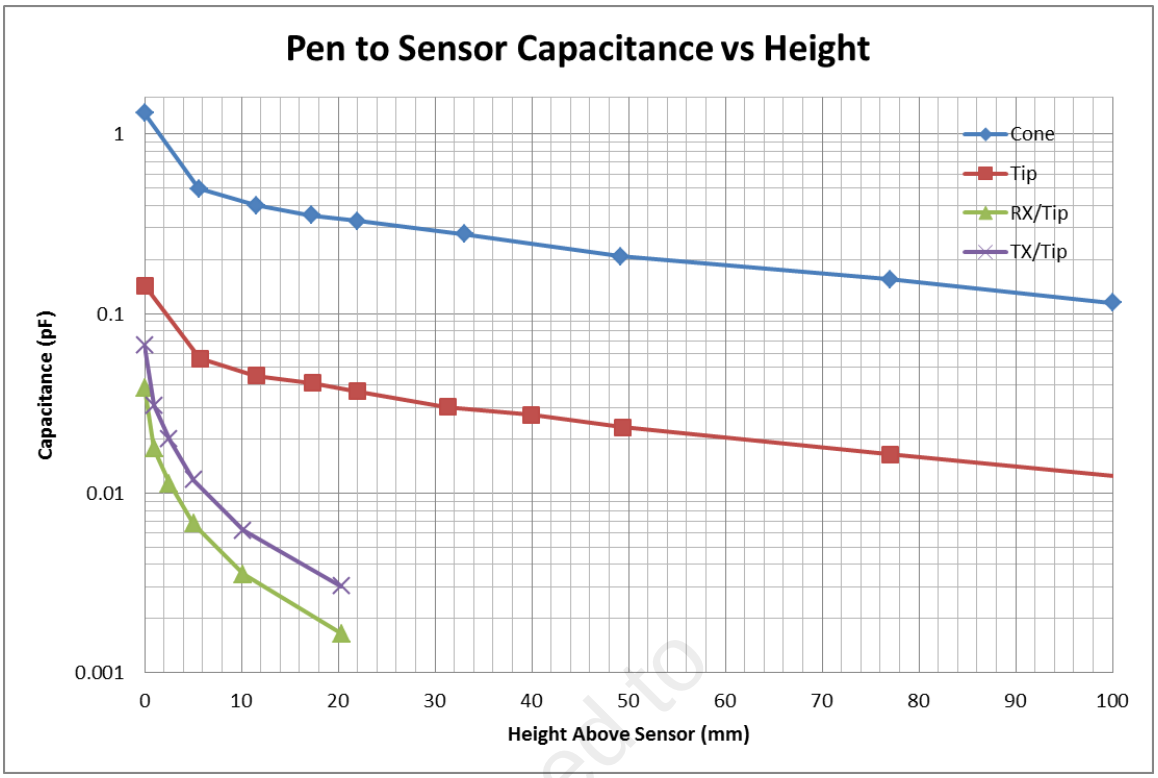


Figure 5-10 Coupling from a Sensor to a Stylus Tip/Cone

The stylus tip is modeled as a 1 mm diameter by 5 mm long tip. The cone narrows at the base of the tip as shown in Figure 5-11.

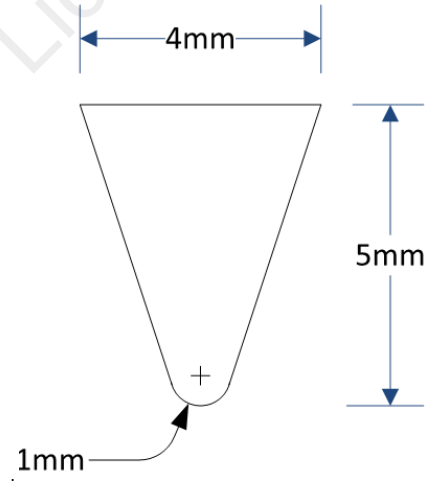


Figure 5-11 Model Stylus Tip dimensions

5.4.2 Stylus Electrical Profile – Type 0 (SEP Type 0)

Stylus Electrical Profile defines the electrical signal behavior of all USI Stylus compliant with this profile. [*Please note that 'Type 0' profile is the only profile defined by this version of the specification. Future versions of the specification may define additional profiles*.]

The Stylus Electrical Profile for all USI Stylus supporting Type 0 profile shall be within the template shown in Figure 5-12 for all stylus angles between -45° and +45°.

The USI compliance spec defines the test apparatus to be used for collection of this data. The collected data should then be

- normalized – to the scale of 0 to 1 for minimum and maximum signal
- and centered – positions which are 4 mm apart and have equal signal amplitude as shown in Figure 5-12 are marked as -2 mm and +2 mm The zero position refers to the point halfway between these two.

Normalized and Centered data shall then be compared to the profile curve.

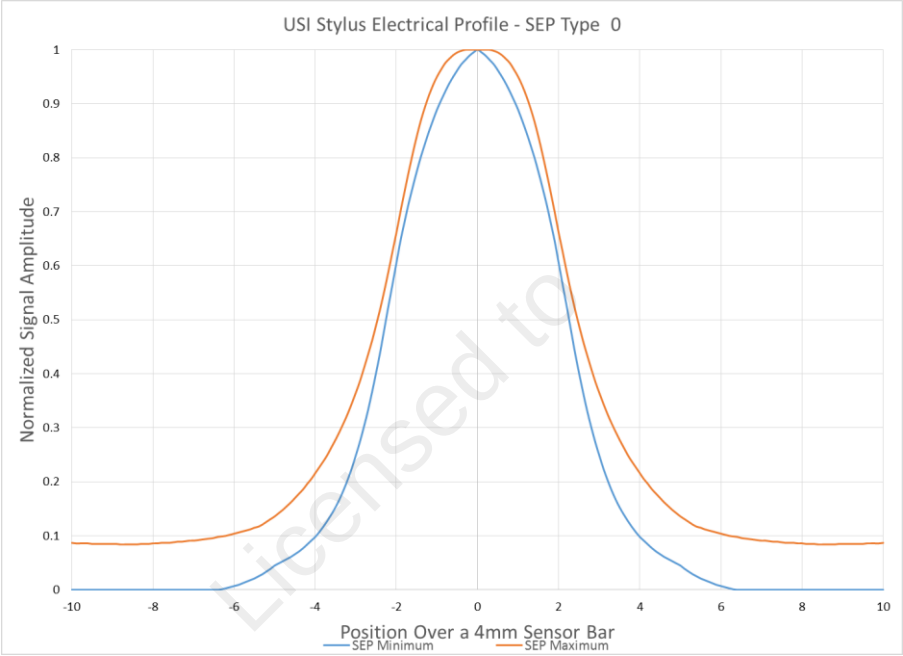


Figure 5-12 Type 0 Stylus Electrical Profile for 0 to 45 degrees tilt

5.5 USI Stylus to Controller Downlink

When a USI Stylus comes within range of a USI controller, both go through the Discovery and Pairing states. During the pairing process the controller configures the stylus to transmit the Position packet and/or Data packet to the controller.

Prior to the discovery and pairing process, the stylus does not transmit during any of the timeslots. During its Discovery state the stylus responds to the controller commands by transmitting only during the ACK timeslot.

During the Pairing state the controller configures the stylus to transmit the position packet and/or data in various timeslots. The ACK and Data packets are assigned to 1, 2, 4 or 8 timeslots in duration (both are set identically).

After pairing is completed, the stylus enters Operation state and continuously transmits Position packet and/or Data packets using the assigned timeslots. In the Operation state the controller transmits the Beacon to maintain a common timing reference between the controller and the stylus and to complete lower priority configuration and status functions.

The stylus shall never change the packet characteristics, unless it is configured to do so by the controller. The controller defines the frequency and duration of stylus responses at the start of the pairing process.

5.5.1 Stylus Transmit Timeslots

Every stylus transmit timeslot shall use a 250 µs period. The number of total stylus transmit timeslots per USI frame is set by the controller (up to 64) and may be changed by a Beacon command.

5.5.2 Stylus Downlink Frequency Accuracy

Stylus downlink frequency accuracy is defined as the difference between the stylus downlink frequency requested by the controller and the actual downlink frequency transmitted by the stylus during downlink transmission. The stylus frequency accuracy shall be better than or equal to ± 0.1 % accuracy.

5.5.3 Stylus Timeslot Accuracy

Stylus timeslot accuracy is defined as the difference between the end of the last chip of the controller Beacon and the first transition edge of the first bit of the stylus transmission:

- The accuracy of the start of each timeslot shall be better than or equal to ± 5 µs across the entire USI frame.
- The 10 µs packet gap of no transmission at the end of each packet ensures that no packets overlap.
- The stylus transmission square wave duty cycle may be off from 50 % by up to 126 ns (or one cycle of an 8 MHz clock plus 1 ns of uncertainty).

5.5.4 Down-Link Data Packet Coding

The downlink data bits are coded using D-BPSK as shown in Figure 5-13. The downlink consists of a start bit (logic 0) followed by 20 bits of data and CRC (LSB first). A binary 1 is transmitted with no phase shift from the previous bit and a binary 0 is transmitted with a 180 degree phase shift from the previous bit.

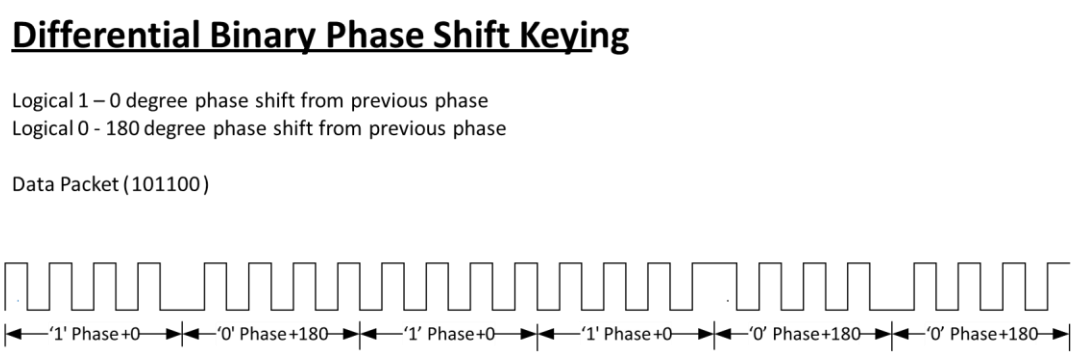


Figure 5-13 D-BPSK Coding

5.5.5 Stylus Timing Synchronization and Turnaround Gap

After detecting and properly decoding a Beacon, the stylus uses the end of the last chip of the DSSS code match (the end of the last chip of the 33rd bit) to establish a common timing reference between the

controller and the stylus. The stylus resets its internal timing reference to match the last Beacon match. It uses this timing reference to transmit both ACK responses and packet data.

A turnaround gap (TAG) timeslot follows the end of the Beacon. This allows the stylus time to complete its DSSS code detection, process the received data and prepare to reply. Neither the controller nor the stylus transmit during this turnaround period.

5.5.6 Stylus Acknowledge Packet

The controller may request a response from a stylus via a beacon command – either during Discovery and Pairing states or during other states. The addressed stylus responds to the controller during the ACK timeslot.

The ACK reply is similar to any other Data from the stylus (same frequency, coding and duration), except that it starts at the end of the TAG. This point in time is referenced as T_{ACK}. The ACK packet uses the same format as a data packet (1 bit start, 16 bit data, 4 bit CRC). The ACK packet is transmitted LSB first, as described in section 5.5.7.

Prior to the discovery and pairing process, the stylus does not transmit during any of the timeslots.

During its Discovery state the stylus responds to the controller commands as if paired, by transmitting Data packets starting at the ACK timeslot. The durations of these Data packets are configured by the controller as part of discovery. The ACK packet and the Data packet use the same number of timeslots, as configured by the controller. Timeslots TS0 – TS6 may be used with ACK packets longer than 250 µs. ACK responses are "Data packets" (same encoding, same duration) that occur in response to a beacon command and starting at T_{ACK}.

During the Pairing state the controller configures the stylus to transmit the Position packet and/or Data packet starting at various timeslots. The ACK and Data packets are assigned to 1, 2, 4 or 8 timeslots in duration (both are set identically). Position packets may have a duration different from Data packets.

The controller will configure the transmit frequency number of cycles per data bit during the Discovery and Pairing states.

The data transmitted during the ACK employs D-BPSK coding. The frequency and number of cycles per data bit ensures that at least the last 10 µs of the ACK packet are unused, in order to avoid collision with any packet transmitted using the first timeslot. This is referred to as “Gap” in Figure 5-14.

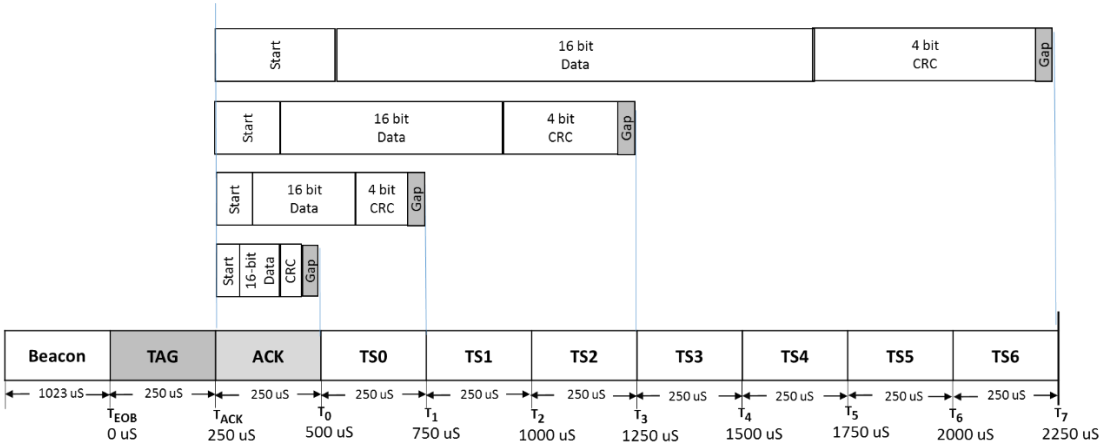


Figure 5-14 ACK Packet Timing

5.5.7 Stylus Downlink Packet

A controller configures the packets for each stylus based on the system capabilities; this configuration occurs after the controller discovers the stylus' capabilities. The controller configures the stylus downlink frequency, ACK and data packet duration, and the position packet duration.

The zero time of the stylus shall be set to be coincident with the end of the last chip sent in the Beacon. This time point is called T_{EOB}.

The first available timeslot (TS0) starts at 500 µs after T_{EOB}. The number of available timeslots shall be defined during the Pairing state (or during subsequent reconfigurations) by the controller via beacon commands. The number of available timeslots (up to a maximum of 64) is determined by the time between the uplink Beacons and shall be allocated by the controller. The controller shall ensure that no paired styluses attempt to transmit data during the ACK sequence, unless it is responding to a Beacon command.

Every downlink data packet shall be 21 bits long (1 start bit, 16 data bits, 4 bit CRC). The value of the start bit shall be a logic 0. All subfields in the Data packet are transmitted LSB first. The ordering of the bits transmitted in a downlink Data packet shall be as described in Table 5-4 and Figure 5-15.

Table 5-4 Downlink Transmission Bit Order

Downlink Bit Index	Downlink Bit Transmitted
0	StartBit = 0
1	Data[0]
2	Data[1]
3	Data[2]
…	…
16	Data[15]
17	CRC[0]
18	CRC[1]
19	CRC[2]
20	CRC[3]

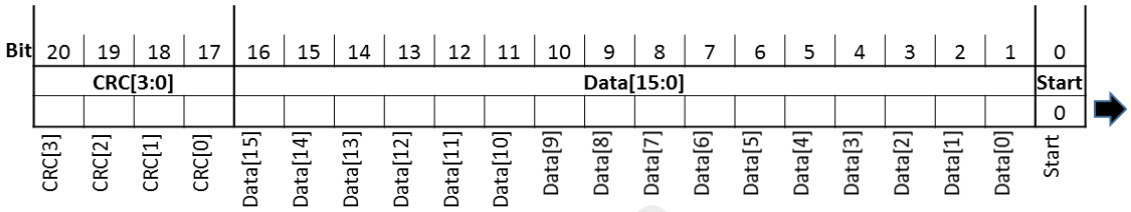


Figure 5-15 Downlink Packet Fields and Bits

A position packet shall only transmit a tone during the packet – no start bit and no CRC, but the 10 µs gap is still required. While an ACK packet and a Data packet use the same number of timeslots, the position packet may be configured to use a unique number of timeslots.

The controller configures each stylus to transmit a packet using either 1, 2, 4 or 8 concatenated timeslots (either 250 µs, 500 µs, 1 ms or 2 ms) as shown in Figure 5-16. The controller shall configure the starting timeslot for each packet.

A stylus may be configured to transmit using one or more packets per USI frame. More than one stylus may transmit packets in a given USI frame.

All packets shall allow at least the last 10 µs of the packet to be unused to provide an unused gap to the start of the next packet.

A packet that combines 2, 4, or 8 timeslots shall maintain a minimum 10 µs gap only at the end of the last timeslot used (single timeslot packet uses ≤ 240 µs, a 2-timeslot packet uses ≤ 490 µs, etc.).

All subfields in the Data packet are transmitted LSB first.

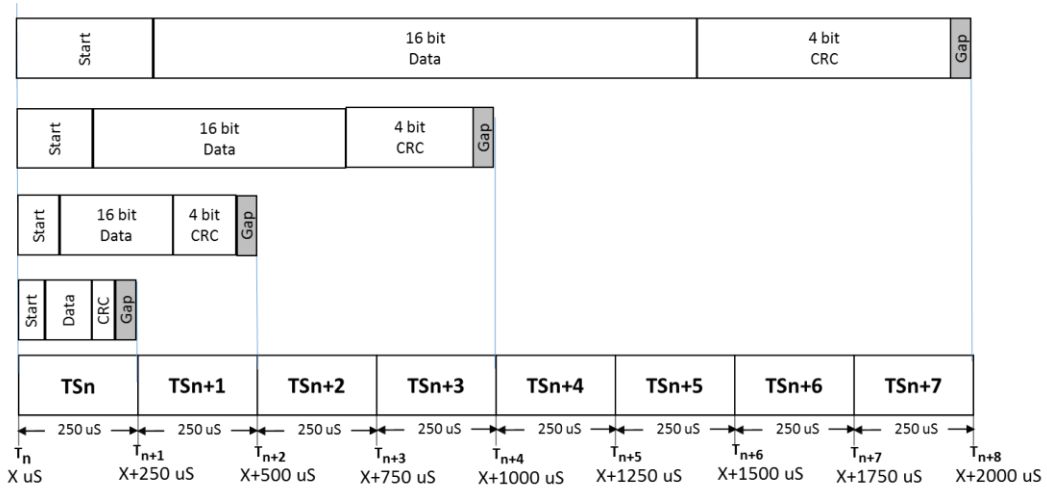


Figure 5-16 Packets using 1, 2, 4 or 8 Timeslots

5.5.8 Downlink D-BPSK Frequency

The controller:

- Sets the stylus downlink frequency by a beacon command, for optimal noise mitigation.
- Sets the stylus transmit frequency and packet duration during the initial link negotiation.
- Selects the appropriate number of D-BPSK pulses to ensure that the minimum 10 µs packet gap exists at the end of each downstream packet.

The stylus selects the number of D-BPSK cycles per data bit as a function of the stylus transmission frequency and packet duration, according to Table 5-5. The number of transmit cycles per bit in Table 5-5 are specified to allow at least 10 µs of 'gap time', the time at the end of each packet prior to the start of the next timeslot. This accommodates up to ± 5 µs of uncertainty in stylus timing.

Table 5-5 Data and Position Packet Configurations

Register Setting				Data Cycles/Bit				Position Cycles/Packet			
			Duration Setting:	0	1	2	3	0	1	2	3
			Duration (µsec):	250	500	1000	2000	250	500	1000	2000
N	Divider Formula	8MHz Divider	Nom Freq (KHz)								
0	N+16	16	500.000	5	11	23	47	120	245	495	995
1	N+16	17	470.588	5	10	22	44	112	230	465	936
2	N+16	18	444.444	5	10	20	42	106	217	440	884
3	N+16	19	421.053	4	9	19	39	101	206	416	837
4	N+16	20	400.000	4	9	18	37	96	196	396	796
5	N+16	21	380.952	4	8	17	36	91	186	377	758
6	N+16	22	363.636	4	8	17	34	87	178	360	723
7	N+16	23	347.826	3	8	16	32	83	170	344	692
8	N+16	24	333.333	3	7	15	31	80	163	330	663
9	N+16	25	320.000	3	7	15	30	76	156	316	636
10	N+16	26	307.692	3	7	14	29	73	150	304	612
11	N+16	27	296.296	3	6	13	28	71	145	293	589
12	N+16	28	285.714	3	6	13	27	68	140	282	568
13	N+16	29	275.862	3	6	13	26	66	135	273	548
14	N+16	30	266.667	3	6	12	25	64	130	264	530
15	N+16	31	258.065	2	6	12	24	61	126	255	513
16	N+16	32	250.000	2	5	11	23	60	122	247	497
17	N+16	33	242.424	2	5	11	22	58	118	240	482
18	N+16	34	235.294	2	5	11	22	56	115	232	468
19	N+16	35	228.571	2	5	10	21	54	112	226	454
20	Lookup	37	216.216	2	5	10	20	51	105	214	430
21	Lookup	39	205.128	2	4	9	19	49	100	203	408
22	Lookup	41	195.122	2	4	9	18	46	95	193	388
23	Lookup	43	186.047	2	4	8	17	44	91	184	370
24	Lookup	46	173.913	1	4	8	16	41	85	172	346
25	Lookup	49	163.265	1	3	7	15	39	80	161	324
26	Lookup	52	153.846	1	3	7	14	36	75	152	306
27	Lookup	56	142.857	1	3	6	13	34	70	141	284
28	Lookup	60	133.333	1	3	6	12	32	65	132	265
29	Lookup	65	123.077	1	2	5	11	29	60	121	244
30	Lookup	71	112.676	1	2	5	10	27	55	111	224
31	Lookup	78	102.564	1	2	4	9	24	50	101	204

Note dark shaded region are not recommended

If the controller supports a multi-frequency receiver, it may assign the same timeslot(s) to multiple styluses, each with a different D-BPSK downlink frequency.

The stylus downlink frequency is assumed to be based on an 8 MHz clock divided down. The clock divisor is specified in Table 5-5.

For data packets the number of cycles per bit (as specified in Table 5-5) is calculated from the assigned frequency (Hz), packet duration (ms), minimum gap (10 µs=0.01 ms), and bits per packet (21) using the following formula:

$$Cycles\_per\_bit = floor\left(freq_{nom} * \left(\frac{duration - gap}{bits\ per\ packet}\right)\right)$$

where

duration = time duration of the packet

gap = 10 µs

bits per packet = 21
$freq_{nom}$ = stylus Tx frequency
floor() is rounded down (e.g., towards zero) to the nearest integer.

For a given frequency setting (N) and duration setting Table 5-5 specifies the actual frequency (based on an 8 MHz clock) and the number of transmit cycles per bit for Data packets and the number of cycles per packet for Position packets.

5.5.9 Downlink Packet Settings

The controller shall configure the stylus downlink frequency, data packet duration and position packet duration. Table 5-5 shows the range of downlink frequencies and associated cycles per bit allowed for each packet size. This table is based on an 8 MHz master clock. The controller configures the stylus downlink frequency and cycles per data bit by selecting one of the register values. This table is to be used for both the ACK packet and for each Position packet. Since the ACK packet and each Position packet could have different packet durations (for data vs position packet), different values could be required. A stylus may transmit one or more position packets per frame.

5.5.10 ACK and Data Packet Error Detection

Both downlink ACK packets and Data packets consist of a 1 bit Start bit, a 16 bit Data field and a 4 bit CRC (applied to the 16 bit Data field) as shown in Figure 5-17. The CRC polynomial selected is able to detect 1 bit error. All bits are sent with LSB first.

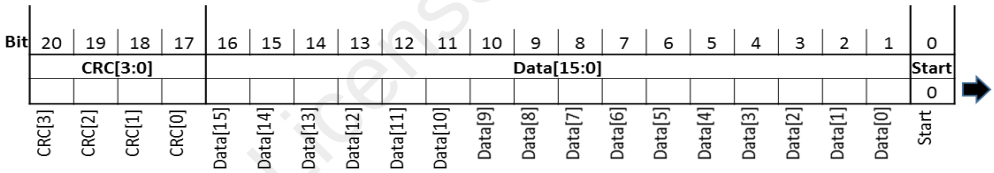


Figure 5-17 Downlink Packet Structure

The CRC is generated by the stylus to protect the ACK and Data Packets being transmitted to the controller. The stylus shall add the generated CRC to each packet that it transmits to the controller.

The CRC is calculated using the most significant bit (MSB) as the first bit into the generator, the generator seed value is all 0s, and the padding bits at the end are 0s. CRC algorithm uses a value of 0xF for final XOR operation. This ensures that a packet with all 0-data which is typical for a hover packet, does not also have an all 0-CRC value. This guarantees that no valid 20-bit downlink packet is all 0s or all 1s. The pseudocode for CRC calculation is in Appendix J.

The CRC generator polynomial is:

$$G(X) = X^4 + X^3 + 1$$

The binary bit pattern that represents this polynomial is 11001B. If all data are received without error, the CRC calculated from the 16 bits of data will match the 4-bits received CRC. There is an exception for the "S.REJECT" response whereupon the broadcast CRC is the complement of the actual CRC and the receive data is a fixed pattern – 0xAA55. This is the single out of band exception for the CRC verification. See section 6.2 for S.REJECT(…) response.

The controller shall re-generate the CRC in each ACK and data packet it receives. If a CRC error is detected, the controller shall not process the data and shall wait for a subsequent packet. If errors persist on

multiple packets, the controller may use the CRC error information to deploy noise mitigation techniques, such as changing the carrier frequency or the number of timeslots per packet.

5.5.11 Stylus Position Packet

A Position tone is a signal that can be used by the controller to detect the position of the stylus tip. The stylus transmits a fixed frequency for the entire packet duration.

As encoded in Table 5-5, a Position packet stops transmitting the tone for the last 10 µs ("Gap" in Figure 5-18) of the packet. This allows a timing margin between it and the start of the following packet.

The controller configures the starting timeslot for each Position packet. A stylus may be assigned to transmit one or more Position packets per USI frame.

The number of cycles/packet is calculated from the formula below:

$$Cycles\_per\_packet = floor\left(freq_{nom} * (duration - gap)\right)$$

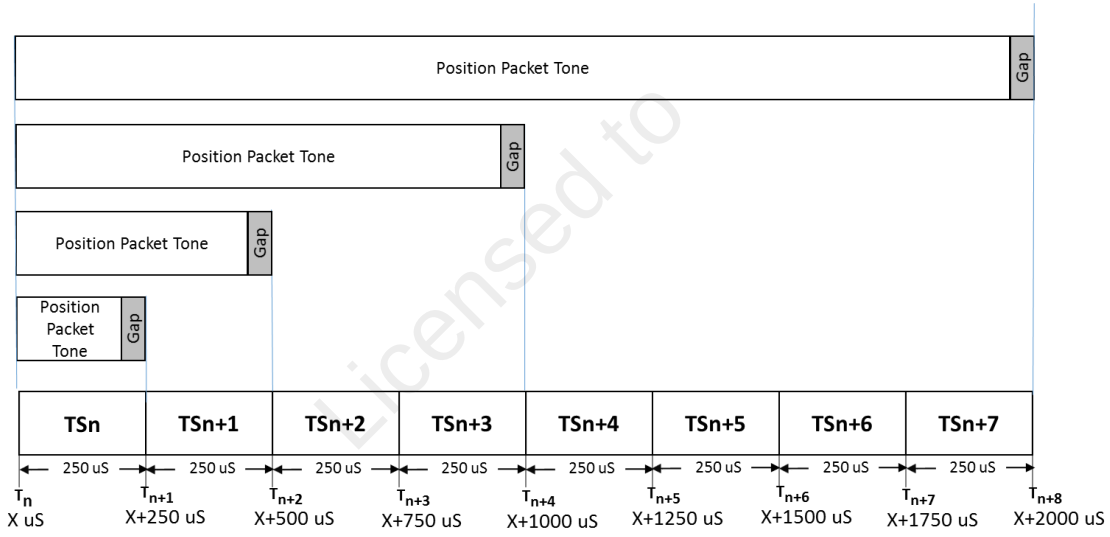


Figure 5-18 Position Packets

5.5.12 Stylus Input Full Scale Range

In order to avoid clipping, the stylus receiver circuit shall be able to handle 10 Vpp of interference on the sensor (the entire sensor being driven with 10 Vpp. A copper plate with 0.5mm of glass laminated to the surface is a good reference sensor), plus the maximum specified uplink signal's to tolerate any out of specification charger noise. Receivers should be able to decode the Beacon with reasonable error rates in the presence of this charger noise.

5.5.13 USI Stylus Electrical Requirements

To ensure USI PHY layer signaling interoperability between any stylus and any controller, all USI Stylus downlink signaling shall have a consistent electrical profile as listed in Table 5-6.

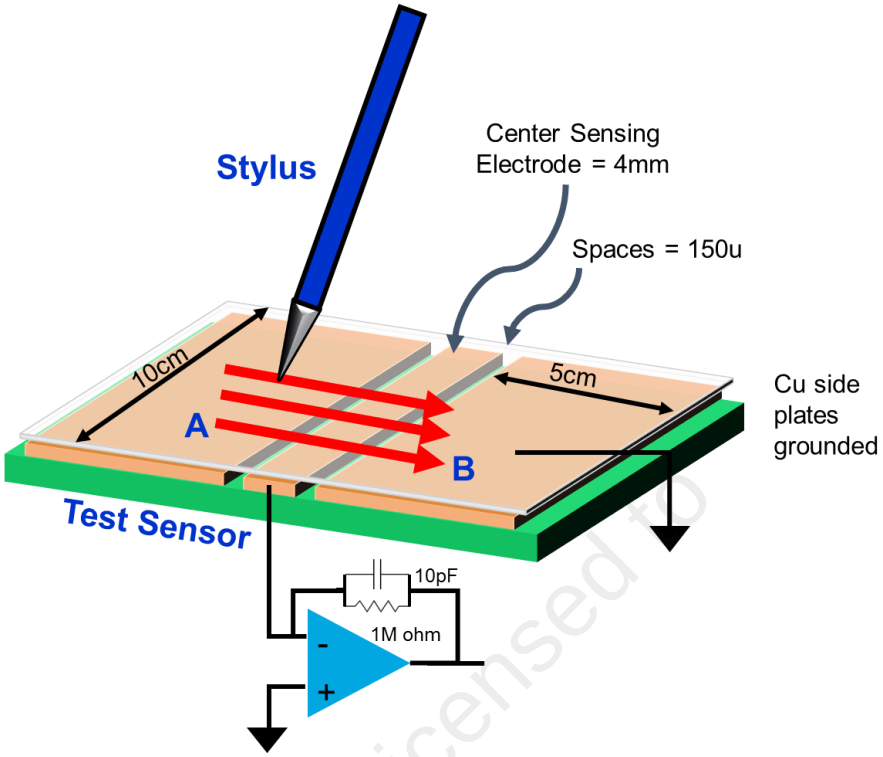


Figure 5-19 Test Fixture for Stylus Charge Measurements

The test fixture to measure the amount of charge provided by the stylus is shown in Figure 5-19.

Table 5-6 Stylus Requirements

USI Stylus Parameter	Value	Description
Input full scale range	0-15 V without noise 0-25 V with interference noise added	
Timeslot	250 µs	Timing unit for ACK, Data, or Position packets.
ACK and data packets	1, 2, 4, or 8 timeslots in duration	1 timeslot = 250 µs
USI Stylus transmitter peak-to-peak voltage at measurement point on test fixture	Minimum Vpp at stylus Tx electrode: > 20 Vpp.	
Hover amplitude of the Tx signal	Min: 0.08 pC $_{rms}$	Measured at 10 mm hover.
Contact peak amplitude of the Tx signal	Min: 0.5 pC $_{rms}$ Max: 2.5 pC $_{rms}$	Measured at 0 mm for either 0 or 45 degree tilt.
Stylus response curve at 0 degree tilt	All signal readings within ± 1 % of 0 degree reference curves	For normalized signal data.
Stylus response curve between 0 and 45 degrees tilt	All signal readings shall be between the 0 degree and the 45 degree reference curves with ± 1 % applied.	For normalized signal data.
USI Stylus transmitter edge rate	10 – 90 % rise/fall time requirements of ≤ 200 ns	Signal swing measured at the stylus tip
USI Stylus transmitter frequency range	All frequencies specified in Table 5-5.	The controller can configure the stylus to operate at any frequency within this table.
USI Stylus transmitter frequency accuracy	± 0.1 %	Used for ACK, data packets, and position packets
Downlink duty cycle accuracy	50% +/- 126 ns	Can be off by one 8 MHz clock cycle + 1 ns of uncertainty
Max packet timing error	± 5.0 µs	Relative to controller Teob
End of packet guardband to the start of the next packet (data packet or position packet)	≥ 10 µs minimum As specified in Table 5-5	There shall be at least 10 µs of unused time at the end of any packet (250 µs, 500 µs, 1.0 ms or 2.0 ms packets).

5.6 USB and PC Charger Noise Limits

To ensure USI PHY layer signaling interoperability, all USI systems powered by an external charger shall limit the noise emitted by the charger, when passed through a band pass filter with single poles at 290 kHz and 500 kHz, to no greater than 2 V peak to peak. This noise limit shall be met for fully charged, partially charged, and completely discharged batteries.

Additionally, the external charger shall provide at least one region of the spectrum between 100 kHz and 500 kHz in which the integrated power over a bandwidth of 100 kHz is less than 2mVpp.

In all cases, the noise from the external charger is measured between the charger output ground and true earth ground using a 10x probe with a nominal 10 MΩ input impedance in parallel with 5 - 20 pF. For details on the Charger interference with USI operations, see Appendix K.

6 Communications Protocol

6.1 Protocol Commands

A USI controller operates the USI Stylus with a fixed set of commands. This chapter describes the structure of the commands and the details of each command. In this specification, the reserved fields are denoted by either “Reserved” or “R” in commands and responses. All USI implementations shall ensure that the reserved fields are transmitted as “logical 0” and ignore (not interpret) them while receiving. However they should be included in CRC calculations in both directions.

6.1.1 Protocol Packet Structure

The Protocol packet is 25 bits long and consists of a 22 bit Command/Data field and a 3 bit StylusID field. The Protocol packet forms part of the Beacon uplink packet but excludes any CRC or preamble bits required by the physical layer. For details on the complete uplink packet, see section 5.3.2. The structure of the Protocol packet is shown in Table 6-1.

Table 6-1 Uplink Protocol Packet

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command/Data																						StylusID		

The protocol commands from the USI controller are divided into two main categories:

1. Read operation: the USI controller is requesting information from the stylus.
2. Write operation: the USI controller is configuring or updating the stylus operation.

The table below illustrates the structure of the Protocol packet for read and write messages.

Table 6-2 Uplink Protocol Packet Structure for Read/Write

Operation	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	READ-CTRL																Command-ID=63						StylusID		
Write	WRITE-DATA																Command-ID=0-62						StylusID		

6.1.1.1 StylusID (Stylus Identifier) Field

Each Protocol packet from the controller shall contain a StylusID field (see Table 6-3). This field is used to address a specific stylus, all unpaired styluses or all paired styluses.

Table 6-3 StylusID Field Values

StylusID	Description
0	USI controller addressing all unpaired styluses
1-6	USI controller addressing specific stylus
7	USI controller addressing all paired styluses (with StylusID set in range 1-6)

The stylus ID is a temporary value assigned to the stylus by the controller for the duration of the time it is paired with the controller. The stylus shall reset its StylusID (StylusID = 0) when it returns to the unpaired state.

6.1.1.2 Command Identifier (Command-ID)

The Command-ID is used to specify the type of command to be sent to the stylus. A specific Command-ID is used for a read operation.

Table 6-4 Command-ID Values

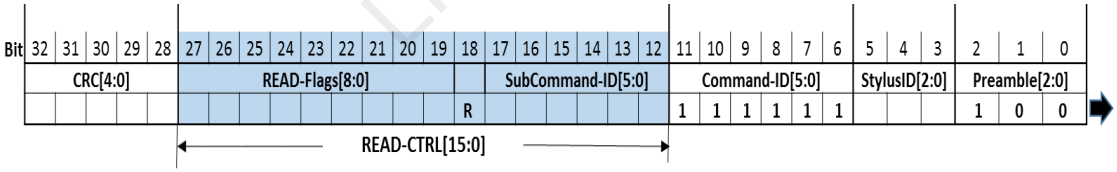
Command-ID	Description
0-62	Write command, in which Data is uploaded to the stylus either for configuration or information.
63	Read commands; values to read are identified by the READ-CTRL field.

6.1.1.3 Read From Stylus (READ-CTRL)

The READ-CTRL is only used for the read type of command (Command-ID=63). READ-CTRL is a subcommand describing which data field to read, and the method of reading that field.

Table 6-5 shows that READ-CTRL has two subfields, Subcommand-ID and READ-FLAGS.

Table 6-5 READ-CTRL Values



- **Subcommand-ID**: Specifies which data fields to read from the stylus.
- **READ-FLAGS**: Specify what type of information shall be output from the stylus and how it shall be delivered. Read flags are specified per command. If multiple READ-FLAGS are set, the read out is multi-packet, in which there is a 16 bit data value for each read flag. The output shall be sent within the same USI frame. The READ-FLAGS(n), where n is bit position, defines the read out of specific Data(n) from that command data table. For example, if READ-FLAGS(1)=1, then Data(1) from the commands data table is read out.

The Subcommand-ID indicates which fields to read from the stylus. These will typically be:

- Global ID
- Capabilities
- Sensor values
- Battery

The READ-FLAGS field describes how to read the data:

- Single flag set: Single read / single packet response for the USI frame

- Multiple flags set: Multi read / multiple packet response for the USI frame.

Each single Data packet from the stylus shall contain a 16 bit Data field.

Especially during pairing, it is desirable to read out more than one Data field at a time to get a quicker knowledge of the stylus' capabilities. For example, a first time pairing operation in which the USI controller is reading out all of the capabilities and the global ID from the stylus may be done in two ways:

- A series of single packet reads across several USI frames
- A multiple packet read in a single USI frame to reduce inking latency.

Note: For example, if a single read controller requires ~8 frames while a multi read can use 2, will result in a difference of ~100ms (assuming frame period of 16.67 ms), in response time.

Because the time to start inking is reduced in this situation, it is a benefit to utilize as much bandwidth as possible. The number of packets read out is controller-dependent: one controller might only have time to do a single read out while another might have the bandwidth to read out four Data packets in each USI frame. The USI controller that is reading out more than one packet needs to have enough available timeslots and bandwidth to use a multiple packet read.

The READ-FLAGS field supports up to nine Data packets being read out in one USI frame for a single command.

If the controller does a multiple packet read operation, the first Data packet starts in the ACK time slot. The next packet in the multiple read will starts in the next timeslot, after the previous packet is completed. For example, if two packets each with 250 µs data duration are read, the first packet starts in the ACK and the second packet in the TS0 slot. The order of the packet data will correspond to the order of the READ-FLAGS, starting from the LSB. For example, if READ-FLAGS indicates a read out of 3 data words (Data(n=0,1,3)) by setting read flags for these words, then the first packet contains Data(0), the second Data(1), and the third Data(3).

The next section gives examples of the data packet order.

6.1.1.3.1 Example: Multiple Packet Read with the C.GetCapability(…) Command

This capability of the stylus contains more than one Data packet. The example below shows how to read out the two first subfields using single read and multiple read.

For a single read the controller reads out the capability in two USI frames:

- USI Frame(N): C.GetCapability(READ-FLAGS=b00…01)
- USI Frame(N+1): C.GetCapability(READ-FLAGS=b00…10).

Figure 6-1 shows these single read commands and their Data packet order.

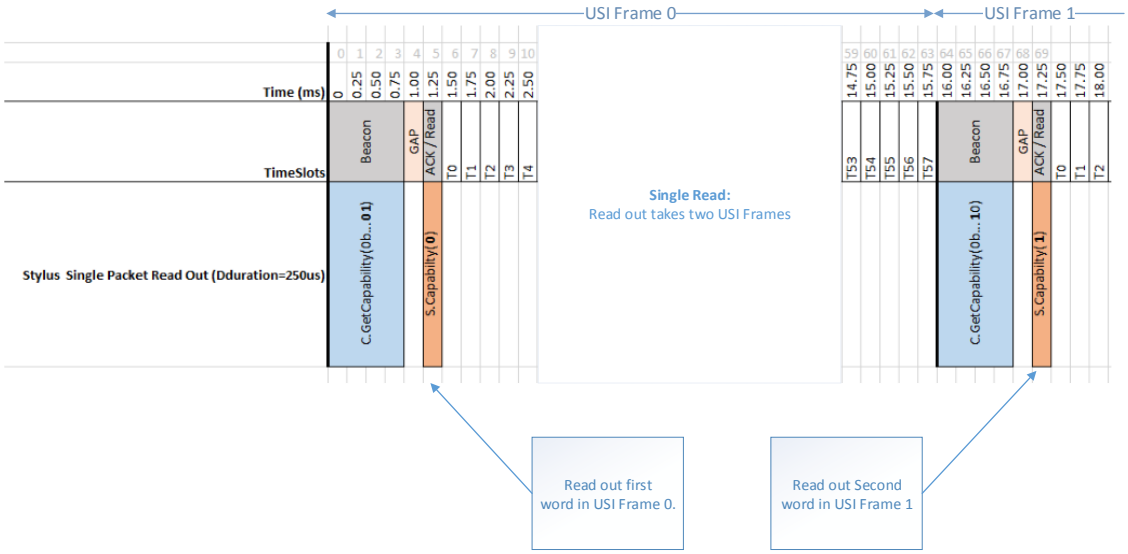


Figure 6-1 Example Single Read per Frame

If the controller sets two flags, it can read out the capability using one USI frame:

- USI Frame(N): C.GetCapability(READ-FLAGS=b00…11) .

Figure 6-2 shows this example's multiple read command and its Data packet order.

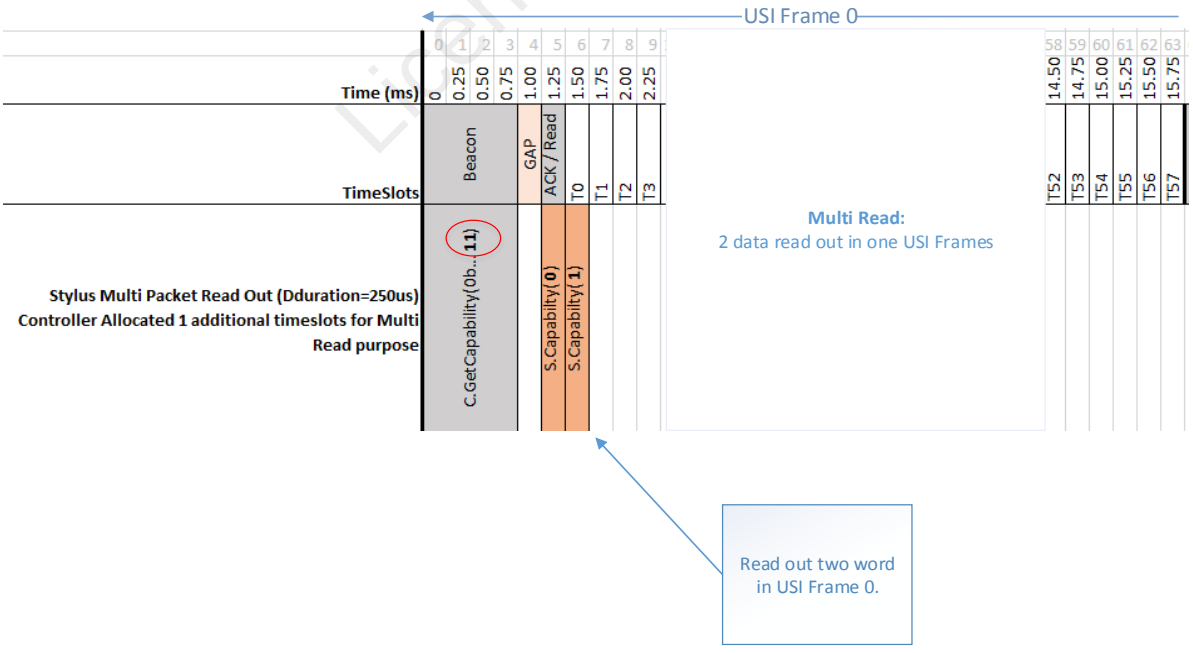


Figure 6-2 Example of Multiple Read in a Frame

Figure 6-3 shows three examples of multiple read out that illustrate how data is packed in time and the sending order from the stylus.

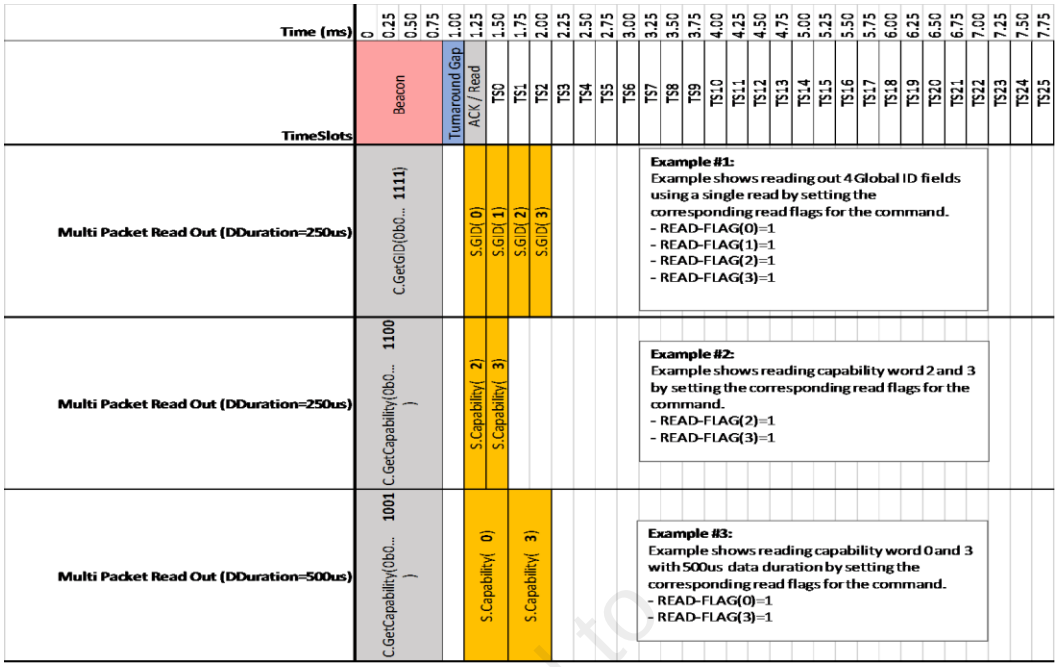


Figure 6-3 Three Examples of Multiple Read in a Frame

In these three examples the READ-FLAGS[8:0] lower flag bits will be read out first. Read order is from the READ-FLAGS LSB to its MSB.

The controller needs to set up all styluses in timeslots, so that collisions are avoided. Specifically, the stylus should never be set up to respond with downlink packets (Data or Position packets) in the time allocated for the multiple read.

On the other hand, if a multiple read operation is requested by the controller and the timeslot has been allocated for a downlink packet, the stylus should respond with the read out data instead of the downlink packet.

6.1.1.4 Write Commands to Stylus (WRITE-DATA)

The WRITE-DATA format is used for write Commands (Command-ID=0-62). WRITE-DATA uses a 16 bit Data field to set values in the USI Stylus.

The controller typically uses WRITE-DATA to set:

- Stylus ID
- Physical downlink configuration (such as downlink frequency and packet duration)
- Timeslot and packet delivery schemes.

6.1.2 Write Commands

6.1.2.1 Summary

Each write command can upload a 16 bit Data field to the stylus. A write command is used when the controller configures and manages the operation of the stylus.

Table 6-6 shows a summary of the write commands. Note that in the protocol packet also the StylusID field is included, but it is not included in Table 6-6 as it is not relevant for the command itself.

Table 6-6 Summary of Write Commands

Command	WRITE-DATA[15:0]	Command-ID[5:0]	Mandatory/ Optional Controller	Mandatory/ Optional Stylus
C.ConfigPhy(...)	Configure ID and physical setup of downlink	0	M	M
C.ConfigDownlink(...)	Configure downlink packet delivery setup	1	At least one of the three methods shall be supported	M
C.ConfigMenuSelect(...)	Configure downlink menu format	2		M
C.ConfigPacket(...)	Configure downlink packet delivery	3		M
C.DisablePacket(...)	Disable downlink packet delivery	4	O	M
C.SetColor(...)	Direct the stylus to set color values	5	M	O
C.SetButtons(...)	Direct the stylus to set buttons	6	M	O
C.SetType(...)	Configure the stylus type	7	M	O
C.VerifyHashLow(...)	Direct the stylus to check its hash ID 16 LSBits	8	M	M
C.UpdatePhy(...)	Direct the stylus to update the physical layer	9	O	M
C.ClearIntFlags(...)	Direct the stylus to clear one or more of its interrupt flags	10	M	M
C.TransmitPositionTone(..)	Direct the stylus to transmit a tone	11	O	M
C.SetWidth(…)	Configure the Stylus tip width	12	M	O
C.SendMyVendorID	Inform the stylus of the controller ID	13	M	O
C.VerifyHashHigh(…)	Direct the stylus to check its hash ID 16 MSBits	14	M	M
C.SetVendorExtension(…)	Device can set the vendor specific extension in a stylus from that vendor	15	M	O
Reserved	Reserved for future USI use	16-49		
Reserved	Reserved for USI debug and compliance	50-57		
Reserved	Reserved for vendor use	58-62		

In any row of the above table, mandatory (M) for either the controller or the stylus means they shall implement support for that command. Similarly, optional (O) for either of them means, they may choose to implement support for that command. When one side has "M" and another side has "O" it means the side with "M" shall implement the support for that command while the other side may decide not to. It shall be noted that some of the mandatory controller commands are triggered by an application and hence the reason for the controller to implement them. An example of such is C.SetWidth(…).

6.1.2.2 C.ConfigPhy(…)

This command is used to configure the physical downlink configuration. The Beacon and the data fields of the command are shown in the Figure 6-4 below.

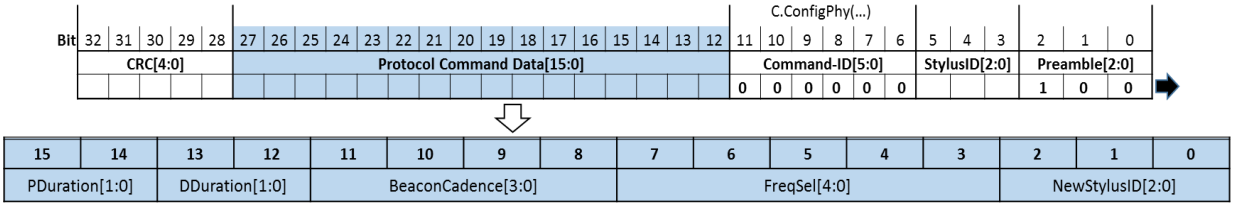


Figure 6-4 Beacon and Data Fields for C.ConfigPhy(…) command

- **StylusID**: Stylus ID used by the device and stylus as below:
  - 0 : Unpaired StylusID value, used by controller to discover a new stylus
  - 1-6: To address a specific paired stylus that has that ID
  - 7: Broadcast to all paired styluses
- **NewStylusID**: New stylus ID to be used by the stylus:
  - 0 and 7: Unpair the stylus.
  - 1-6: Initiate an attempt to allocate a new stylus ID.
- **FreqSel**: Frequency to use by the stylus transmitter. See Table 5-5 for details on frequency select. FreqSel[4:0] programs the Frequency setting to the desired value.
- **BeaconCadence**: These bits specify the approximate cadence of the Beacon from the USI controller. The stylus can utilize this value to sleep between Beacons and hence save power. Values are configurable from 4 ms to 25 ms. The value represents the earliest potential time for the Beacon to be sent and the stylus should search from the allocated time for a period defined by the search size. e.g., If the controller cadence is 16.66 ms, it would set BeaconCadence = 10 and it is expected that the stylus will search 16 ms to 18 ms for a Beacon signal.

Table 6-7 Beacon Cadence Configuration Values

Beacon Cadence [3:0]	Minimum Beacon Period (ms)	Search Window Size (ms)	Center Beacon Frequency (KHz)
0	0ms	Continuous Search	-
1	4	1	222.2
2	5	1	181.8
3	6	2	142.9
4	8	2	111.1
5	10	2	90.9
6	12	2	76.9
7	14	1	69.0
8	15	1	64.5
9	16	2	58.8
10	16	5	54.1
11	16	8	50.0

12	18	2	52.6
13	20	2	47.6
14	22	2	43.5
15	24	2	40.0

- **DDuration:** Duration of the Data packet
  0: 1 Timeslot
  1: 2 Timeslots
  2: 4 Timeslots
  3: 8 Timeslots.

Note that the duration will vary with the frequency selected. For actual numbers, see 5.5.8.

- **PDuration:** Duration of a Position packet (the tone used in the Position packet is specified separately):
  0: 250 µs packet duration
  1: 500 µs packet duration
  2: 1 ms packet duration
  3: 2 ms packet duration.

Note that the duration will vary with the frequency selected. For actual numbers, see Table 5-5.

6.1.2.3 C.ConfigDownlink(…)

This command is used to set up the deliver format for the stylus downlink format. The Beacon and the Data field of the command are shown in the Figure 6-5.

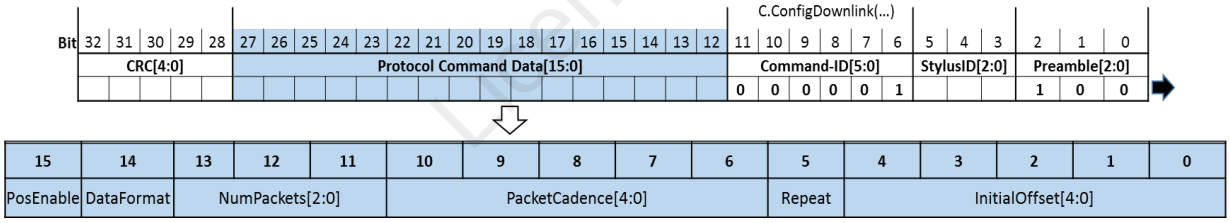


Figure 6-5 Beacon and Data Fields for C.ConfigDownlink(...) command

- **InitialOffset**: Timeslot during which the initial transmit packet starts. Timeslot0-31 (TS0-TS31) may be selected.
- **Repeat**: Determines whether the packet transmission will repeat 1 time. If this bit is set, the previous packet type should be repeated. Both Position packets and Data packets will repeat.
- **PacketCadence**: 'Start-to-start' cadence of the downlink packets. See Table 6-8.

Table 6-8 Packet Cadence Configuration

PacketCadence[4:0]	Packet Cadence in ms	Packet Cadence in Hz
0	2.0	500.00
1	2.25	444.44
2	2.5	400.00
3	2.75	363.64
4	3.0	333.33
5	3.25	307.69
6	3.5	285.71
7	3.75	266.67
8	4.0	250.00
9	4.25	235.29
10	4.5	222.22
11	4.75	210.53
12	5.0	200.00
13	5.25	190.48
14	5.5	181.82
15	5.75	173.91
16	6.0	166.67
17	6.25	160.00
18	6.5	153.85
19	6.75	148.15
20	7.0	142.86
21	7.25	137.93
22	7.5	133.33
23	7.75	129.03
24	8.0	125.00
25	8.25	121.21
26	8.5	117.65
27	8.75	114.29
28	9.0	111.11
29	9.25	108.11
30	9.5	105.26
31	9.75	102.56

- **NumPackets**: This field (see Table 6-9) specifies the number of packets sent between Beacons, not including the impact of the Repeat field or PosEnable field contents. If Repeat or PosEnable is set to 1, the number of packets will double.

Table 6-9 Number of Packets to Send

NumPackets[2:0]	Packets to send when (Repeat = 0) AND (PosEnable = 0)	Packets to send when (Repeat = 1) OR (PosEnable = 1)
0	No packets	No packets
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10
6	6	12
7	7	14

- **DataFormat**: Specifies the data delivery.
  0: Always send the USI standard data packets. See Table 6-11.
  1: Reserved
- **PosEnable**: When this bit is set, the transmitter shall insert a Position packet before each Data packet.

Note: Repeat and PosEnable shall never be both set to 1 by the Controller.

Section 6.1.4.1 contains examples of the usage of this command.

6.1.2.4 C.ConfigMenuSelect(…)

This command is used to quickly set up the data delivery format for the stylus. The Beacon and Data field for the command are shown in Figure 6-6

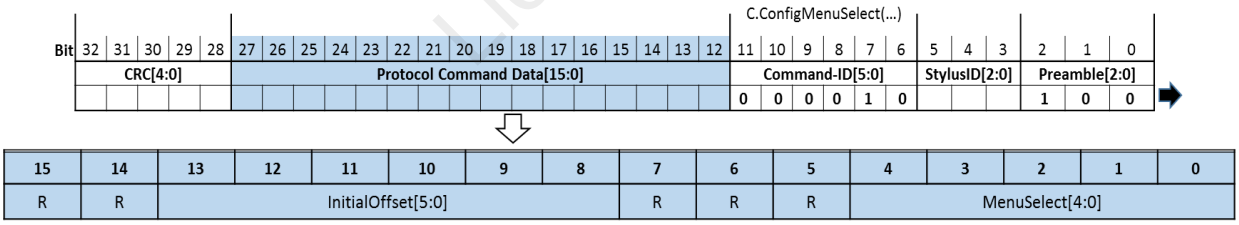


Figure 6-6 Beacon and Data Fields for C.ConfigMenuSelect(…) command

- **MenuSelect**: Select how the stylus should deliver data and there are currently six defined options as shown in Table 6-10. An example on how to use C.ConfigMenuSelect(…) is described in section 6.1.4.2.

Table 6-10 MenuSelect Options

MenuSelect	Configuration
0	No menu selection (reserved)
1	{TS0} : data packet-type 1 (Force/button) {TS10, TS41} : data packet-type 0 (Position)
2	{TS10} : data packet-type 1 (Force/button) {TS0, TS21, TS41} : data packet-type 0 (Position)
3	{TS0, TS32} : data packet-type 1 (Force/button) {TS7, TS23, TS38, TS53} : data packet-type 0 (Position)
4	{TS0, TS29} : data packet-type 1 (Force/button) {TS7, TS25, TS43} : data packet-type 0 (Position)
5	{TS0, TS11, TS36} : data packet-type 1 (Force/button) {TS5, TS17, TS29, TS41, TS53} : data packet-type 0 (Position)
6	{TS0, TS27} : data packet-type 1 (Force/button) {TS5, TS18, TS31, TS44} : data packet-type 0 (Position)
7-30	Reserved (for other entry submissions)
31	No menu selection (reserved)

- **InitialOffset**: Select start timeslot for delivery link:
  0: Timeslot0
  1: Timeslot1
  …
  63: Timeslot63.

6.1.2.5 C.ConfigPacket(…)

This command is used to add a single packet delivery. Packets already defined will still be delivered. If an added packet overlaps an existing packet then the existing packet will be replaced by the new packet. The Beacon and Data fields for the command are shown in Figure 6-7.

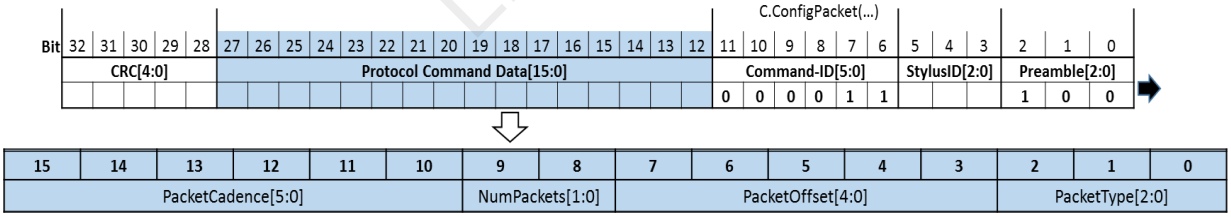


Figure 6-7 Beacon and Data Fields for C.ConfigPacket(...) command

PacketType: Defines which packet to insert to deliver.

Table 6-11 PacketType When Configuring Packet Delivery

PacketType[2:0]	Packet Type
0	Position
1	Default (IRQ, Buttons, Pressure)
2	Orientation (IMU, Tilt, Twist)
3	Extended Data (Barrel Pressure)
4	Vendor Data0
5	Vendor Data1
6	Reserved
7	Reserved

- **PacketOffset**: This defines the timeslot placement for the packet start:
  0: Timeslot0
  1: Timeslot1
  ….
  31: Timeslot31.
- **NumPackets**: This defines the number of packets to insert:
  0: 1 packet
  1: 2 packets
  2: 3 packets
  3: 4 packets.
- **PacketCadence**: This field is valid if the NumPackets > 0:
  Actual cadence = (PacketCadence + 1).
  This field indicates the cadence, in units of timeslots, of the packet in the frame.

Section 6.1.4.3 contains an example of the usage of this command.

6.1.2.6 C.DisablePacket(…)

This command is used to disable a single packet or all packets. The Beacon and Data fields for the command are shown in Figure 6-8.

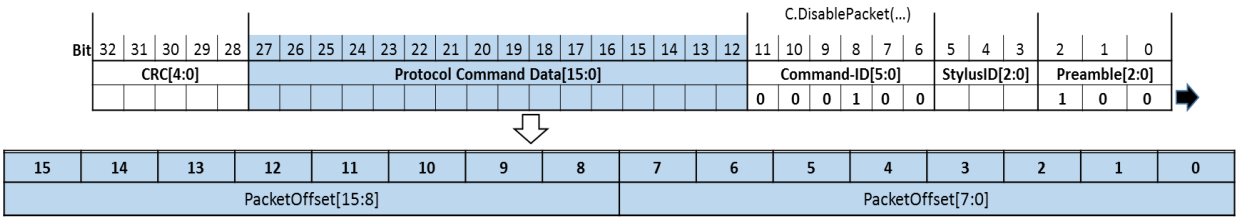


Figure 6-8 Beacon and Data fields for C.DisablePacket(…) command

- **PacketOffset:** This specifies where the packet that should be disabled starts. If data fields are all ones, all packets are disabled.

Table 6-12 PacketOffset and the Disable Command

PacketOffset[15:0]	Target Packet
0-63	Disable packet starting in the specific timeslot.
65535	Disable all packets
Other values	Reserved

6.1.2.7 C.SetColor(…)

Using this command the stylus user may be able to set the stylus' preferred inking color by setting sticky bits in the stylus. The Beacon and Data fields for the command are shown in Figure 6-9. Whether the PreferredColor is settable or not is indicated in the data of C.GetCapability(…).

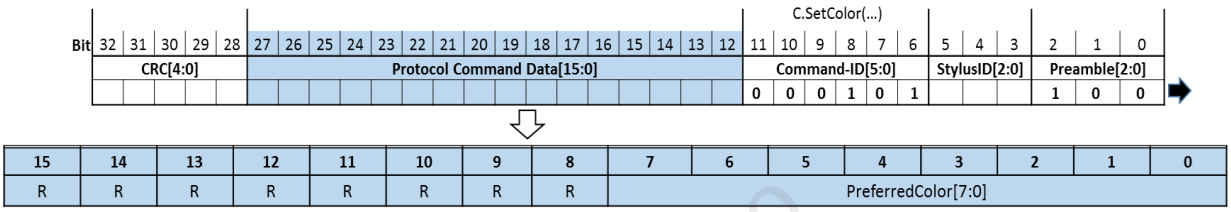


Figure 6-9 Beacon and Data fields for C.SetColor(...) command

- **PreferredColor**
  Defines the 8-bit index of the PreferredColor. The indices correspond to the keywords defined in https://www.w3.org/TR/SVG/types.html#ColorKeywords and cover major colors. The table is defined in Appendix H
  - PreferredColor can be a factory programmed value (e.g., a marker with red body color reports red as the Preferred Color), or
  - PreferredColor can be a value controlled by a switch on the stylus (e.g., a stylus with red, blue, green selection button, which changes the reported value of Preferred Color), or
  - PreferredColor can be a host-settable value that the stylus keeps in its persistent memory as long as it has the battery power.
  - Stylus shall indicate whether this value is settable by host or not (see section 6.1.3.3 for details on how stylus can indicate this using PrefColorRO bit).
  - Stylus that do not have a PreferredColor, shall set the value to 255, and if the stylus is unable to persist any change to this value, then it shall also indicate to the host that this value is NOT settable.

6.1.2.8 C.SetButtons(…)

This command is used to configure the stylus' barrel buttons. The controller may base on user preference its decision about which of the three barrel buttons to set up as the default for a barrel switch and/or eraser ready switch. The Beacon and Data field for the command are shown in Figure 6-10.

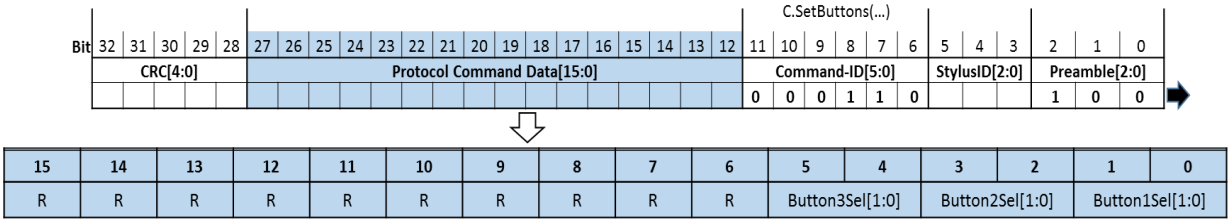


Figure 6-10 Beacon and Data fields for C.SetButtons(...) command

- **ButtonXSel**: For details on button selection, see section 6.1.3.3.

Button *closest* to the stylus tip is Button1 and the button numbering increases towards the top of the stylus. Button, *farthest* from the stylus tip is Button[N] where N depends on the number of buttons implemented.

6.1.2.9 C.SetType(…)

The stylus user can set the stylus' preferred brush type for inking. The Beacon and Data fields for the command are shown in Figure 6-11.

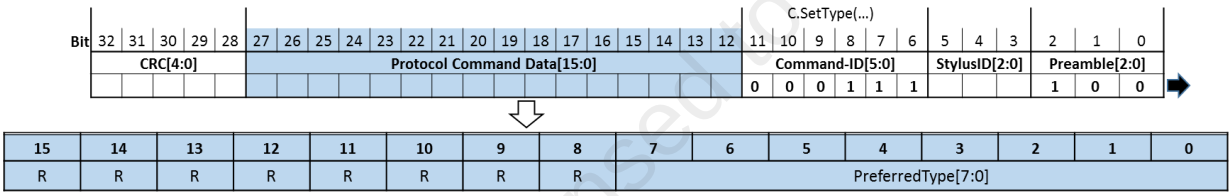


Figure 6-11 Beacon and Data fields for C.SetType(...) command

- **PreferredType :**
  - 0: Reserved
  - 1: Inking Stylus
  - 2: Pencil Stylus
  - 3: Chisel Marker
  - 4: Brush
  - 5: Highlighter
  - 6-254: Reserved
  - 255: No preference

This value in the stylus allows it to communicate its preference for the Inking Tip Type. This follows the similar behavior as PreferredColor. It can be set by the hardware, and can be set by the host and persisted by the stylus.

Stylus shall indicate whether this value is settable by host or not (see section 6.1.3.3 for details on how stylus can indicate this using PrefTypeRO bit)

Stylus that do not have a PreferredType, shall set the value to 255, and if the stylus is unable to persist any change to this value, then it shall also indicate to the host that this value is NOT settable

Stylus shall indicate whether this value is settable by host or not (see section 6.1.3.3 for details on how stylus can indicate this using PrefTypeRO bit)

Stylus that do not have a PreferredType, shall set the value to 255, and if the stylus is unable to persist any change to this value, then it shall also indicate to the host that this value is NOT settable.

6.1.2.10 C.VerifyHashLow(…)

This command requests the stylus to check that the stylus' hash ID matches the hash ID that the controller has for the stylus ID. If the stylus' local hash ID doesn't match the value that the controller has sent to the stylus, the stylus should return to its unpaired state. The Beacon and Data fields for the command are shown in Figure 6-12.

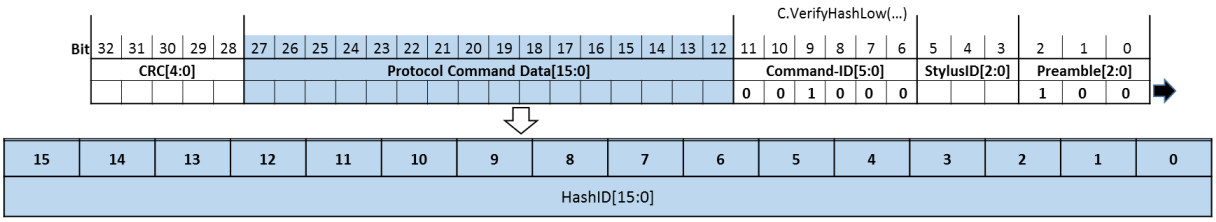


Figure 6-12 Beacon and Data fields for C.VerifyHashLow(...) command

- **HashID[15:0]:** The stylus should verify that the 16 LSBs of the local hash ID match the hash value uploaded by the controller.

6.1.2.11 C.VerifyHashHigh(…)

This command requests the stylus to check that the stylus' high order bits hash ID[31:16] matches the hash ID that the controller has for that stylus. If the stylus' local hash ID doesn't match the value that the controller has sent to it, the stylus should return to its unpaired state. The Beacon and Data fields for the command are shown in Figure 6-13.

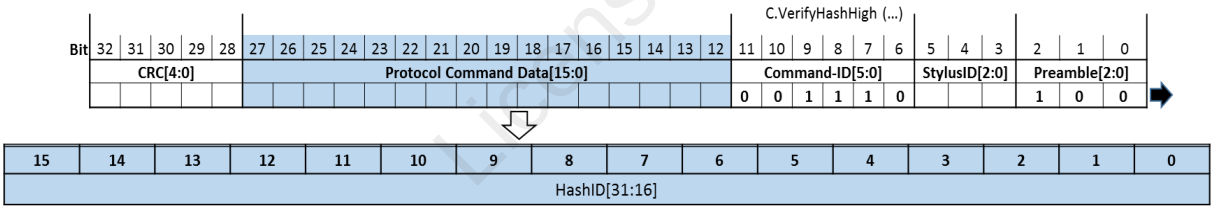


Figure 6-13 Beacon and Data fields for C.VerifyHashHigh(...) command

- **HashID[31:16]:** The stylus should verify that the 16 MSBits of the local hash ID match the hash value uploaded by the controller.

It should also be noted that the command following the C.VerifyHashLow() maybe C.VerifyHashHigh() to reduce probability of incorrect stylus identification.

6.1.2.12 C.UpdatePhy(…)

This command requests a stylus to update its existing physical configurations. The Beacon and Data fields for the command are shown in Figure 6-14.

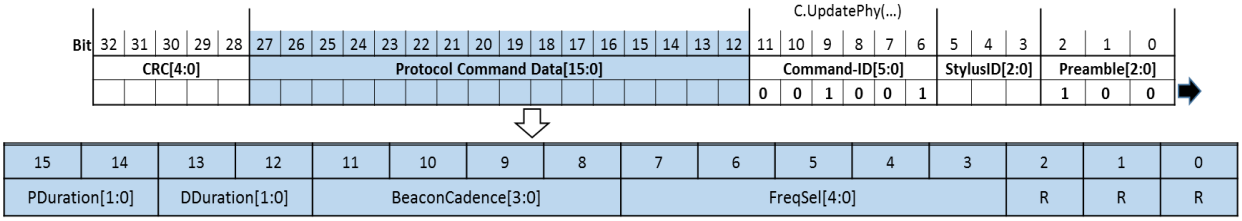


Figure 6-14 Beacon and Data fields for C.UpdatePhy(...) command

For details on data fields, see the specification of C.ConfigPhy(...). A stylus shall ensure that the changes take place in the same USI frame that this command has been received. If the controller doesn't receive an S.ACK(…) from the stylus, it could mean that the controller packet was lost and the stylus will be listening at the old cadence, or it could mean that the S.ACK(…) was lost and the stylus will be listening at the new cadence. The packet loss issue shall be dealt with the normal 250 ms timeout mechanism.

From a stylus' perspective the difference between this command and C.ConfigPhy(…) is that it shall respond with S.ACK(…) instead of S.HASH(…). In addition the bottom three reserved bits shall be ignored by the stylus as required.

6.1.2.13 C.ClearIntFlags(…)

This command clears one or more interrupt flags in the stylus. The Beacon and Data fields for the command are shown in Figure 6-15

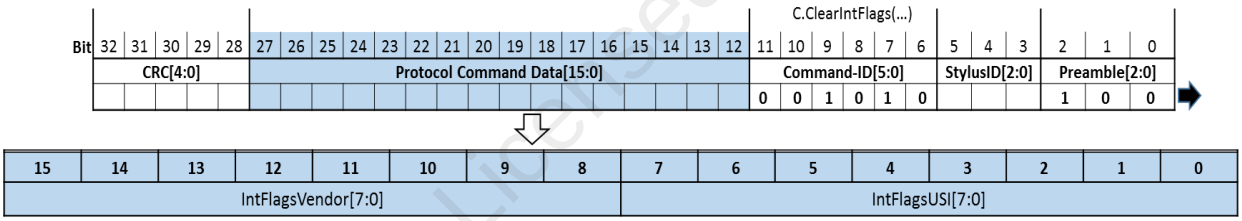


Figure 6-15 Beacon and Data fields for C.ClearIntFlag(...) command

A flag is cleared by writing 1 to the individual bit. If 0 is written to the bit, the bit retains its current value. See section 6.4 for decoding information.

6.1.2.14 C.TransmitPositionTone(…)

This command configures the stylus to output a single Position Tone response packet, starting in the ACK timeslot. The tone will only be sent once and only for the specific Beacon frame in which the command is written. The Beacon and Data fields for the command are shown in Figure 6-16

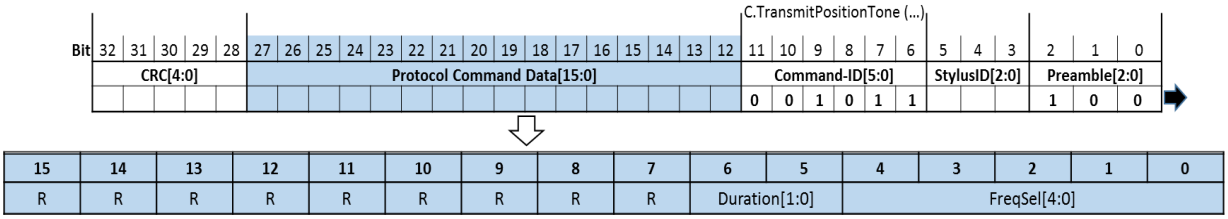


Figure 6-16 Beacon and Data fields for C.TransmitPositionTone(...) command

The parameters specified for this command are only valid for a single Position Tone packet. See section 6.1.2.2 for details of the frequency and duration parameter values.

6.1.2.15 C.SetWidth(…)

The stylus user can set the stylus' preferred brush width for inking. The Beacon and Data fields for the command are shown in Figure 6-17.

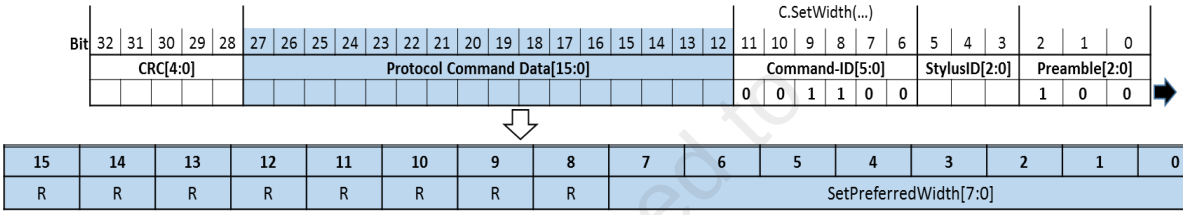


Figure 6-17 Beacon and Data fields for C.SetWidth(…) command

- **StylusPreferredWidth :**
  - 0: Thinnest possible
  - 1-254: Stroke Width in 0.1 mm increments (i.e. value of n = (0.1 * n) mm width)
  - 255: No preference

This value in the stylus allows it to communicate its preference for the Inking stroke width. This follows the similar behavior as PreferredColor. It can be set by the hardware, and can be set by the host and persisted by the stylus.

Stylus shall indicate whether this value is settable by host or not (see section 6.1.3.3 for details on how stylus can indicate this using PrefWidthRO bit).

Stylus that do not have a PreferredWidth, shall set the value to 255, and if the stylus is unable to persist any change to this value, then it shall also indicate to the host that this value is NOT settable.

6.1.2.16 C.SendMyVendorID(…)

Using this command the controller can send its own vendor ID to the stylus. A controller shall execute this command before it sends any command that is in vendor reserved space. The controller's vendor ID allows the stylus to interpret vendor specific commands correctly. The Beacon and Data fields for the command are shown in Figure 6-18.

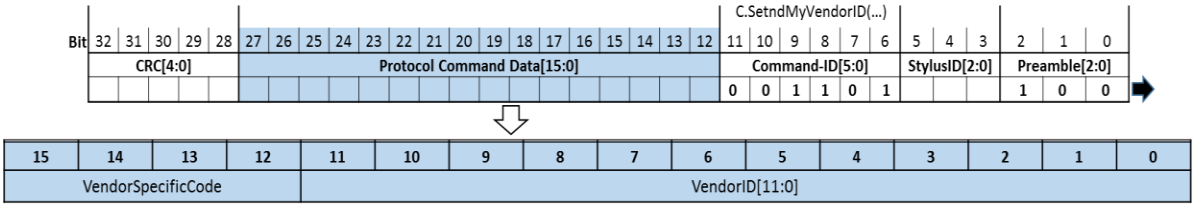


Figure 6-18 Beacon and Data fields for C.SendMyVendorID(…) command

- **VendorID[11:0]:** Each company that is developing USI specification based products is assigned an Unique ID and these IDs are listed in Appendix F . This ID will be common to all products developed by a company.

If the stylus supports vendor extensions that match the VendorID, then it shall return S.ACK().

If the stylus does not support any vendor extensions than the stylus shall return an S.REJECT() error to this command. The stylus should then set the error code to Unsupported Command.

If the stylus supports vendor extensions but it is for a different vendor than what is specified in VendorID in this command, then it shall return a S.REJCT(…) response. It should then set the error code to Invalid Parameters.

6.1.2.17 C.SetVendorExtension(…)

This command allows for vendor specific extension data in the stylus to be changed with the controller. The mechanism to perform the write is indicated here. The contents of the data are specific to each vendor and are not defined as part of this specification. The Beacon and Data fields for the command are shown in Figure 6-19.

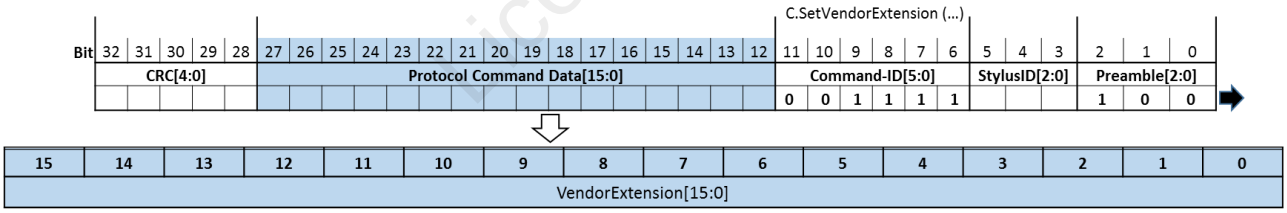


Figure 6-19 Beacon and Data fields for C.SetVendorExtension(…)

6.1.3 Read Commands

6.1.3.1 Summary

Each read command has 16 bits defining which value to read and how to read it.

Table 6-13 shows a summary of the read commands. The stylus ID is set in the protocol packet, but it is not included in the table below, as it is not relevant for the command itself. For all read commands Command-ID=63.

Table 6-13 Summary of Read Commands

Command	READ-FLAGS[8:0]	SubCommand-ID[5:0]	Command-ID[5:0]	Mandatory/ Optional Controller	Mandatory/ Optional Stylus
C.GetBattery(...)	Individual per command	0	63	M	M
C.GetCapability(...)		1	63	M	M
C.GetUSIVersion(...)		2	63	M	M
C.GetGID(...)		3	63	M	M
C.GetData(...)		4	63	M	M
C.GetVendorExtension(...)		5	63	M	O
C.GetPhy(...)		6	63	O	M
C.GetTimeslotSetup0(...)		7	63	O	M
C.GetTimeslotSetup1(...)		8	63	O	M
C.GetIntFlags(...)		9	63	M	M
C.GetHashID(...)		10	63	M	M
C.GetFirmwareVersion(...)		11	63	M	M
C.GetLastError(...)		12	63	M	M
C.GetIMU(...)		13	63	M	O
C.GetTiltTwist(...)		14	63	M	O
Reserved for USI use		15-49	63		
Reserved for USI Test		51-57	63	M	M
Reserved for Vendor		58-62	63		
Reserved for USI use		63	63		

In any row of the above table, mandatory (M) for either the controller or the stylus means they shall implement support for that command. Similarly, optional (O) for either of them means, they may choose to implement support for that command. When one side has "M" and another side has "O" it means the side with "M" shall implement the support for that command while the other side may decide not to.

6.1.3.2 C.GetBattery(..)

This command is used to read out the battery status from the stylus. The Beacon for this command is shown in Figure 6-20.

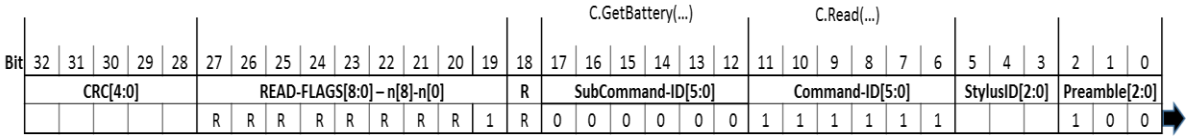


Figure 6-20 Beacon for C.GetBattery(...) command

Table 6-14 specifies the stylus response Data fields related to READ-FLAGS for the C.GetBattery(...) command.

Table 6-14 Stylus Data Fields for the C.GetBattery(...) response

Response Name:	S.GetBattery(...)															
	Stylus data fields															
READ-FLAGS[n]=1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n=0	-	-	-	-	-	-	-	-	Low	StateOfCharge						
n=1-8	Reserved															

- **StateOfCharge (SOC):** If available, this field should be used to report the state of charge in the stylus, as a percentage of full charge.
- **Low:**
  0: Stylus power source is ok
  1: Stylus power source is critically low and should be replaced/re-charged

Table 6-15 Battery State of Charge

Values	Description
0-100	Percentage full
127	State of charge not supported by the stylus.
Other values	Reserved.

6.1.3.3 C.GetCapability(…)

This command is used to read out the capability and sticky information of the stylus. The Beacon for this command is shown in Figure 6-21. In the Beacon n[x]= 1* means at least one or more of those bits shall be set if the data is to be returned in the current USI Frame.

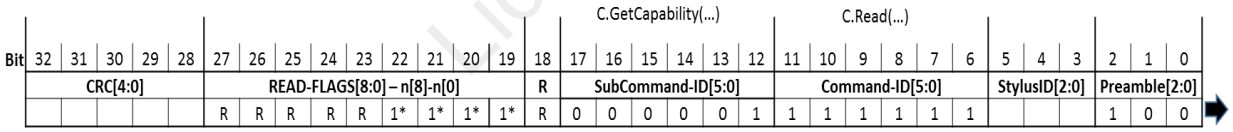


Figure 6-21 Beacon for C.GetCapability(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 16.

Table 6-16 Stylus Data Fields for the C.GetCapability(...) response

Response Name:	S.GetCapability(...)															
	Stylus data fields															
READ-FLAGS[n]=1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n=0	R	R	BTN3	BTN2	BTN1	HasMagnet	HasGyro	HasAccel	PrefWidthRO	PrefTypeRO	PrefColorRO	VendData1	VendData0	Twist	Tilt	BarrelPres
n=1	PreferredWidth[7:0]								PreferredType7:0]							
n=2	R	R	R	R	R	R	SEP[1:0]		PreferredColor[7:0]							
n=3	VendorExtension[7:0]								R	R	Button3Sel[1:0]		Button2Sel[1:0]		Button1Sel[1:0]	
n=4-8	Reserved															

The first word (n=0) contains the capability feature set. The next three octets contain sticky information; this information can, in some systems, be updated by the user.

- **BarrelPres:** Barrel pressure flag.
  0: Stylus doesn't have barrel pressure information.
  1: Stylus has barrel pressure information.
- **Tilt:** Tilt flag.
  0: Stylus doesn't have X-tilt and Y-tilt sensor information.
  1: Stylus has X-tilt and Y-tilt sensor information.

If the stylus measures orientation as azimuth and altitude instead of tilt, the stylus shall indicate it has tilt capability, and convert azimuth and altitude to X and Y tilt before reporting to the touch controller.

- **Twist:** Twist flag.
  0: Stylus doesn't have twist information.
  1: Stylus has twist information.
- **VendorData<N>**:
  0: Stylus doesn't have vendor specific data<N>.
  1: Stylus has vendor specific data<N>.
- **PrefColorRO**
  0: Stylus' PreferredColor value can be changed by a host application using C.SetColor(…) command
  1: Stylus' PreferredColor value is Read-Only and cannot be changed by the host.
- **PrefWidthRO**:
  0: Stylus' PreferredWidth value can be changed by a host application using C.SetWidth(…) command
  1: Stylus' PreferredColor value is Read-Only and cannot be changed by the host.
- **PrefTypeRO**:
  0: Stylus' PreferredType value can be changed by a host application using C.SetType(…) command
  1: Stylus' PreferredType value is Read-Only and cannot be changed by the host.
- **HasAccel:** IMU Accelerometer flag.
  0: Stylus doesn't have an accelerometer
  1: Stylus has an IMU with three-axis accelerometer.
- **HasGyro:** IMU Gyrloscope flag.
  0: Stylus doesn't have a gyroscope
  1: Stylus has an IMU with three-axis gyroscope.
- **HasMagnet:** IMU Magnetometer flag.
  0: Stylus doesn't have a magnetometer
  1: Stylus has an IMU with three-axis magnetometer.
- **BTN1, BTN2, BTN3:** Indicates whether the Stylus has Button1, Button2 and Button3.
  0: Stylus doesn't have the corresponding button
  1: Stylus the corresponding button.
- **PreferredWidth:**
  Defines the preferred stroke width. For details see section 6.1.2.15
- **PreferredType:**
  Defines the preferred tip type. For details see section 6.1.2.9
- **PreferredColor:**
  Defines the 8-bit index of the PreferredColor. For details see 6.1.2.7.
- **SEP[1:0]** -**Stylus Electrical Profile:**
  - 00: Type 0 - See Figure 5-12 for details
  - 01: Reserved
  - 10: Reserved

- 11: Reserved

- **Button<N>Sel**:
  The button select sticky bits represent the barrel button functionality. The three supported barrel buttons have the functions:
  0: No Function (Disabled)
  1: Barrel - the button closest to the stylus tip
  2: Secondary barrel switch – the button second closest to the stylus tip
  3: Eraser Ready (indicates the user intends to erase). Used by the UI to change screen cursor or tool selection. It is combined with the tip switch to generate and Erasing signal by the Touch controller.

- **VendorExtension:** Vendor extension code. Decoding is specific for each vendor.

6.1.3.4 C.GetUSIVersion(…)

This command is used to read out the USI version number of the stylus. The Beacon for this command is shown in Figure 6-22.

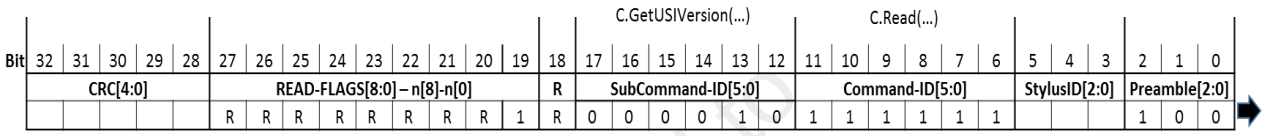


Figure 6-22 Beacon for C.GetUSIVersion(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 17.

Table 6-17 Stylus Data Fields for the C.GetUSIVersion(...) response

Response Name:	**S.GetUSIVersion(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	R	R	R	R	R	R	R	R	USIMajorVer[3:0]				USIMinVer[3:0]			
n=1-8	Reserved															

- **MajorVer:** USI Major Version

1: Major version is 1 [Stylus shall set the value to 1 for this version]
All other values are reserved.

- **MinorVer:** USI Minor Version

0: Minor version is 0 [Stylus shall set the value to 0 for this version]
All other values are reserved.

- MajorVer and MinorVer combination together imply what command set the controller and the stylus shall support as per this specification.

6.1.3.5 C.GetGID(…)

This command is used to read out the stylus' global ID. This ID should be unique for each stylus. The Beacon for this command is shown in Figure 6-23. In the Beacon n[x]= 1* means at least one or more of those bits shall be set if the data is to be returned in the current USI Frame.

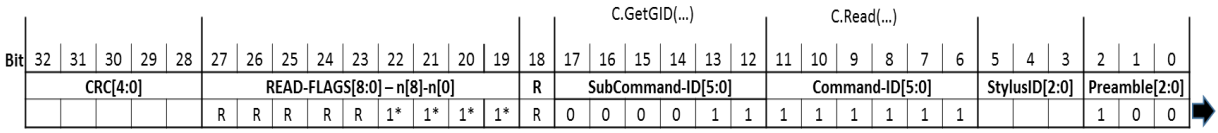


Figure 6-23 Beacon for C.GetGID(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 18.

Table 6-18 Stylus Data Fields for the C.GetGID(...) response

Response Name:	**S.GetGID(...)**															
	Global ID data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	GID0															
n=1	GID1															
n=2	GID2															
n=3	GID3															
n=4-8	Reserved															

- **GID3:0:**
  The global ID contains:
  - **Vendor-ID:** 1 code per USI vendor.
  - **Device-ID:** Unique ID for the stylus device. Could be a hardware and firmware specific identifier.

Table 6-19 –Global ID Data Fields

	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
GID0	Device-ID[15:0]															
GID1	Device-ID[31:16]															
GID2	Device-ID[47:32]															
GID3	USI Vendor-ID[11:0]												Device-ID[51:48]			

The USI vendor ID codes are defined in Appendix F.

6.1.3.6 C.GetData(…)

This command is used to read out data values from the stylus. These values are normally delivered from the stylus to the USI controller using the automatic downlink, but the controller can also command a manual read out of these registers. The Beacon for this command is shown in Figure 6-24. In the Beacon n[x]= 1* means at least one or more of those bits shall be set if the data is to be returned in the current USI Frame.

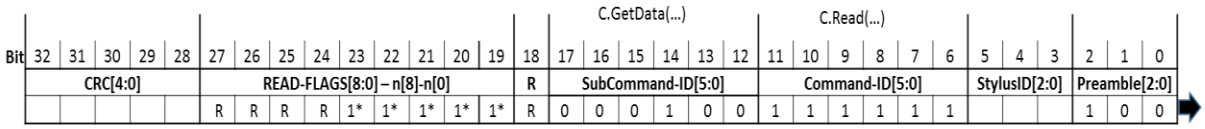


Figure 6-24 Beacon for C.GetData(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 20.

Table 6-20 Stylus Data Fields for the C.GetData(…) response

Response Name:	**S.GetData(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	Default data															
n=1	Orientation															
n=2	Extended Data															
n=3	Vendor Data0															
n=4	Vendor Data1															
n=5-8	Reserved															

For details on the Data packets and their data fields, see section6.5.

Note: For Orientation data,a stylus may return partial Tilt, Twist or IMU data. In order to get complete Orientation data, one should use C.GetIMU(…) or C.GetTiltTwist commands.

6.1.3.7 C.GetVendorExtension(…)

This command allows for Vendor specific data in the stylus to be read by the controller. The mechanism to perform the read is indicated here. The contents of the data are specific to each vendor and are not defined as part of this specification. The Beacon for this command is shown in Figure 6-25.

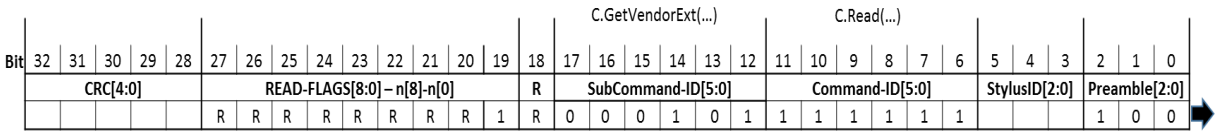


Figure 6-25 Beacon for C.GetVendorExtension(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 21.

Table 6-21 Stylus Data Fields for the C.GetVendorExtension(...) response

Response Name:	**S.GetVendorExt(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	VendorExtension[15:0]															
n=1-8	Reserved															

6.1.3.8 C.GetPhy(…)

This command is used to read out the configuration settings for the stylus' physical link. The Beacon for this command is shown in Figure 6-26.

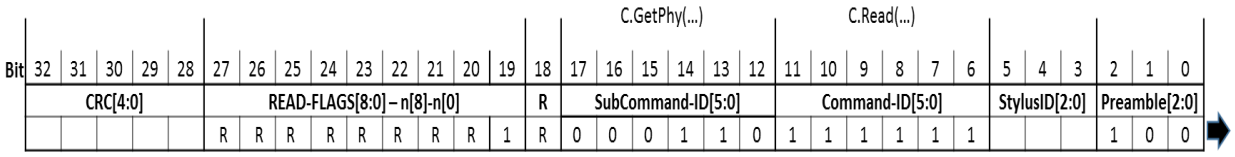


Figure 6-26 Beacon for C.GetPhy(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 22.

Table 6-22 Stylus Data Fields for the C.GetPhy(…) response

Response Name:	**S.GetPhy(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	PDuration[1:0]		DDuration[1:0]		BeaconCadence[3:0]				FreqSel[4:0]					StylusID[2:0]		
n=1-8	Reserved															

See section 6.1.2.2 for data field descriptions.

6.1.3.9 C.GetTimeslotSetup0(…)

This command is used to read out the timeslot setup for the stylus. The Beacon for this command is shown in Figure 6-27.

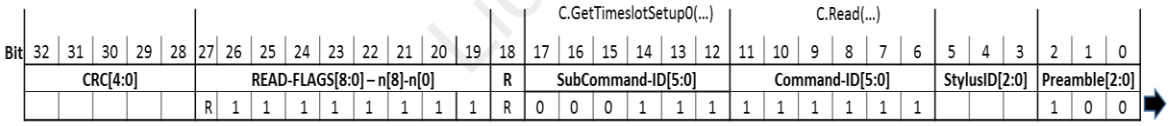


Figure 6-27 Beacon for C.GetTimeslotSetup0(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 23.

Table 6-23 Stylus Data Fields for the C.GetTimeslotSetup0(…) response

Response Name:	**S.GetTimeslotSetup0(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	Slot3				Slot2				Slot1				Slot0			
n=1	Slot7				Slot6				Slot5				Slot4			
n=2	Slot11				Slot10				Slot9				Slot8			
n=3	Slot15				Slot14				Slot13				Slot12			
n=4	Slot19				Slot18				Slot17				Slot16			
n=5	Slot23				Slot22				Slot21				Slot20			
n=6	Slot27				Slot26				Slot25				Slot24			
n=7	Slot31				Slot30				Slot29				Slot28			
n=8	R				R				R				R			

For each timeslot there is a 4 bit field whose contents specify whether there is a packet transmission occupying the specific timeslot, and which type of packet it is.

0: Position packet
1-7: Refers to data in the downlink table for USI (see section 6.5)
8-14: Reserved
15: No packets

6.1.3.10 C.GetTimeslotSetup1(…)

This command is used to read out the timeslot setup for the stylus. The Beacon for this command is shown in Figure 6-28.

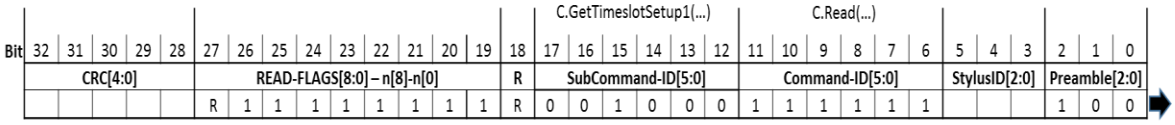


Figure 6-28 Beacon for C.GetTimeslotSetup1(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 24.

Table 6-24 Stylus Data Fields for the C.GetTimeslotSetup1(…) response

Response Name: S.GetTimeslotSetup1(...)																
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	Slot35				Slot34				Slot33				Slot32			
n=1	Slot39				Slot38				Slot37				Slot36			
n=2	Slot43				Slot42				Slot41				Slot40			
n=3	Slot47				Slot46				Slot45				Slot44			
n=4	Slot51				Slot50				Slot49				Slot48			
n=5	Slot55				Slot54				Slot53				Slot52			
n=6	Slot59				Slot58				Slot57				Slot56			
n=7	Slot63				Slot62				Slot61				Slot60			
n=8	R				R				R				R			

For each timeslot there is a 4 bit field whose contents specify whether there is a packet transmission occupying the specific timeslot, and which type of packet it is.

6.1.3.11 C.GetIntFlags(…)

This command is used to read out the interrupt flags from the stylus. The Beacon for this command is shown in Figure 6-29.

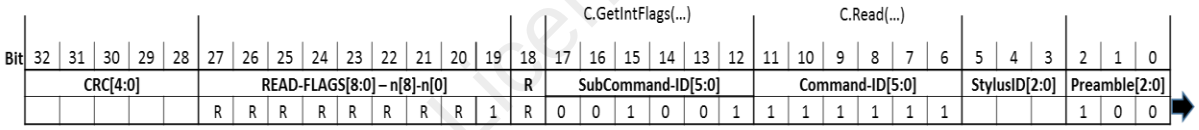


Figure 6-29 Beacon for C.GetIntFlags(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 25.

Table 6-25 Stylus Data Fields for the C.GetIntFlags(…) response

Response Name: S.GetIntFlags(...)																
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	IntFlagsVendor[7:0]								IntFlagsUSI[7:0]							
n=1-8	Reserved															

See section 6.4 for information on decoding,

6.1.3.12 C.GetHashID(…)

This command is used by the controller to read out the hash ID of the stylus. The Beacon for this command is shown in Figure 6-30.

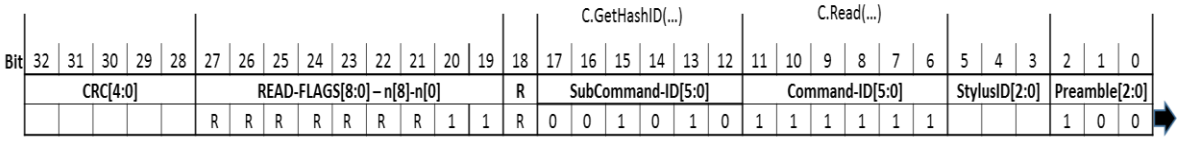


Figure 6-30 Beacon for C.GetHashID(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 26.

Table 6-26 Stylus Data fields for the C.GetHashID(...) response

Response Name:	**S.GetHashID(…)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	HashID[15:0]															
n=1	HashID[31:16]															
n=2-8	Reserved															

For details on stylus hashing, see section 6.7.

6.1.3.13 C.GetFirmwareVersion(…)

This command is used to read out the firmware version number of the stylus. The firmware version is specific to each vendor's implementation and not supplied by USI.org. The Beacon for this command is shown in Figure 6-31. The versions are 8-bits in size to match the sizes in HID report.

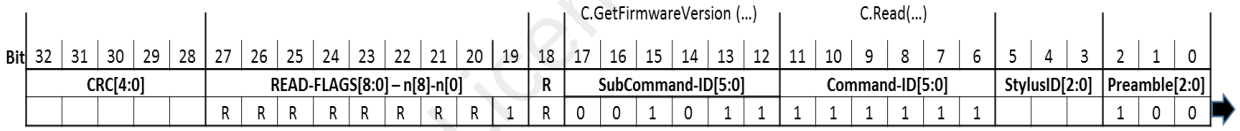


Figure 6-31 Beacon for C.GetFirmwareVersion(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 27.

Table 6-27 Stylus Data fields for the C.GetFirmwareVersion(...) response

Response Name:	**S.GetFirmwareVersion(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	FirmwareMajVer[7:0]								FirmwareMinVer[7:0]							
n=1-8	Reserved															

- **FirmwareMajVer:**
  0: Major version is 0
  1: Major version is 1
  ...
- **FirmwareMinVer:**
  0: Minor version is 0

1: Minor version is 1
...

6.1.3.14 C.GetLastError(…)

This command is used to read out the last error from the stylus. The stylus reports the error code from the last Read or Write command that resulted in the S.REJECT() response. If multiple commands result in S.REJECT(), then only the last error code is returned. Stylus shall always support this command and shall not respond with S.REJECT() for this command itself. If the stylus does not maintain the error codes, then the stylus shall return a value with the 'Error Information Available' bit set to zero, as described below. The Beacon for this command is shown in Figure 6-32.

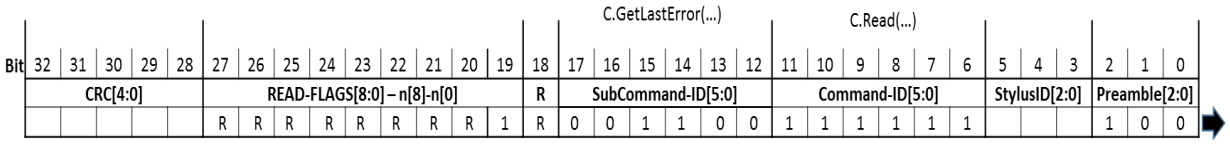


Figure 6-32 Beacon for C.GetLastError(...) command

The stylus response corresponding to those READ-FLAGS is shown in Table 6 28.

Table 6-28 Stylus Data fields for the C.GetLastError(...) response

Response Name:	**S.GetLastError(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	EIA	ErrorCode[5:0]						Rd/Wr	R	R	Command-ID[5:0]/SubCommand-ID[5:0]					
n=1-8	Reserved															

- Command/SubCommandID field includes the last command that caused the error
    - 6 bits of the Command-ID[5:0] in case of Write commands.
    - 6 bits of SubCommand-ID[5:0] in case of Read commands.
- Rd/Wr bit indicates whether it is Read command (0) or Write Command (1).
- ErrorCode indicates the code that is associated with the specified command/subcommand
    - 0: Unknown Error
    - 1: Unsupported Command/SubCommand
    - 2: Undefined Command-ID/SubCommand-ID
    - 3: Invalid parameters
    - 4-63 Reserved
- EIA – Error information available – If this bit is zero, than rest of the contents are invalid.

  Power on status of this shall be zero. Stylus shall set this bit if an error occurred and the stylus is capable of reporting errors on that last command. Stylus shall leave this bit as zero if no error has ever occurred.

6.1.3.15 C.GetIMU(…)

This command is used to read out the IMU data. Data is available for up to three sensors (Magnetometer, Gyroscope and Accelerometer) each with 3 axis (X,Y,Z). The Beacon for this command is shown in Figure 6 33.

Due to the asynchronous nature of the IMU with respect to the Controller read, a SampleID can be used. This just increments on every update of associated IMU value, and wraps around to 0 on overflow. If the SampleID is the same value for Magnetometer, Gyro and Accelerometer, it means the samples for them were taken at the same time.

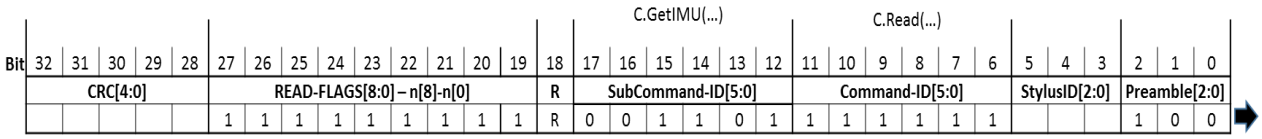


Figure 6-33 Beacon for C.GetIMU(...) command

The stylus response corresponding to the READ-FLAGS is shown in Table 6 29.

Table 6-29 Stylus Data fields for the C.GetIMU(...) response

Response Name:	**S.GetIMU(...)**															
	IMU data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	SampleID_MX				Magnetometer X-axis											
n=1	SampleID_MY				Magnetometer Y-Axis											
n=2	SampleID_MZ				Magnetometer Z-Axis											
n=3	SampleID_GX				Gyroscope X-axis											
n=4	SampleID_GY				Gyroscope Y-axis											
n=5	SampleID_GZ				Gyroscope Z-axis											
n=6	SampleID_AX				Accelerometer X-Axis											
n=7	SampleID_AY				Accelerometer Y-Axis											
n=8	SampleID_AZ				Accelerometer Z-Axis											

- **Magnetometer: X,Y,Z-axis**
  Provides the magnetometer axis value (See 7.3.1.11.3)
- **SampleID_MX, SampleID_MY, SampleID_MZ**
  Incremented every time a sample is updated, samples taken at the same time for each axis should have the same sample value.
- **Gyroscope: X,Y,Z-axis**
  Provides the Gyroscope axis value (See 7.3.1.11.2)
- **SampleID_GX, SampleID_GY, SampleID_GZ**
  Incremented every time a sample is updated, samples taken at the same time for each axis should have the same sample value.
- **Accelerometer: X,Y,Z-axis**

Provides the accelerometer axis value (See 7.3.1.11.1)

- **SampleID_AX, SampleID_AY, SampleID_AZ**
  Incremented every time a sample is updated, samples taken at the same time for each axis should have the same sample value.

6.1.3.16 C.GetTiltTwist(…)

This command is used to get Tilt and Twist data from the stylus, if supported by the stylus and declared in the capability flags. The Beacon for this command is shown in Figure 6 34. In the Beacon n[x] = 1* means at least one or more of those bits shall be set if the data is to be returned in the same USI Frame.

Due to the asynchronous nature of the Tilt/Twist data with respect to the Controller read, a sample field count can be used. This just increments on every update of associated sample values, and wraps around to 0 on overflow.

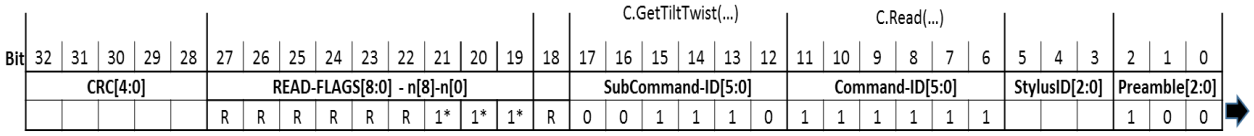


Figure 6-34 Beacon for C.GetTiltTwist(…) command

The stylus response corresponding to the READ-Flags is shown in Table 6 30.

Table 6-30 Stylus Response Fields for C.GetTiltTwist(…) response

Response Name:	**S.GetTiltTwist(…)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	SampleID_X				Tilt X-axis											
n=1	SampleID_Y				Tilt Y-Axis											
n=2	SampleID_T				Twist											
n=3-8	Reserved															

- **Tilt: X,Y-axis**
  Provides the Tilt axis value (See section 6.5.2 for more details)
- **Twist**
  Provides the Twist value (See section 6.5.2 for more details)
- **SampleID_X, SampleID_Y, SampleID_T**
  Incremented every time a sample is updated, samples taken at the same time for each value (X, Y and T) should have the same SampleID value.

6.1.4 Examples of Timeslots and Downstream Data Configuration

Before the setup configuration of the stylus is started, the physical downlink has to be configured. When this is done, the next operation is typically to configure in which timeslot and what data should be sent from the stylus to the controller.

For timeslot setup there are multiple commands that can be used. This chapter gives examples how the configuration settings will look in the time domain using the different setup commands:

- **C.ConfigDownlink(...):** Auto configuration allows for repeated data types per frame
- **C.ConfigPacket(...):** Manual configuration allows for single timeslots/data types definition per frame
- **C.ConfigMenuSelect(...):** Predefined configuration with fixed timeslot and data types per frame

6.1.4.1 Setup using C.ConfigDownlink(…)

This section gives examples of how downlink data is delivered using the C.ConfigDownlink(...) command.

6.1.4.1.1 Example1: Using a Position Packet

For the physical configuration the duration of the Position packet and the Data packet are configured to be different. The Position packet has a duration of 1 ms, while the data packet has a duration of 500 µs in the first example. The beacon cadence is 16 ms.

PosEnable is set and the stylus will therefore always send a Position packet between each pair of Data packets. For the timing setup the following are defined:

- NumPackets=2. (4 packets to be sent since PosEnable=1)
- InitialOffset=2
- PacketCadence=8 (4 ms)
- PosEnable=1
- DataFormat=0
- Repeat=0

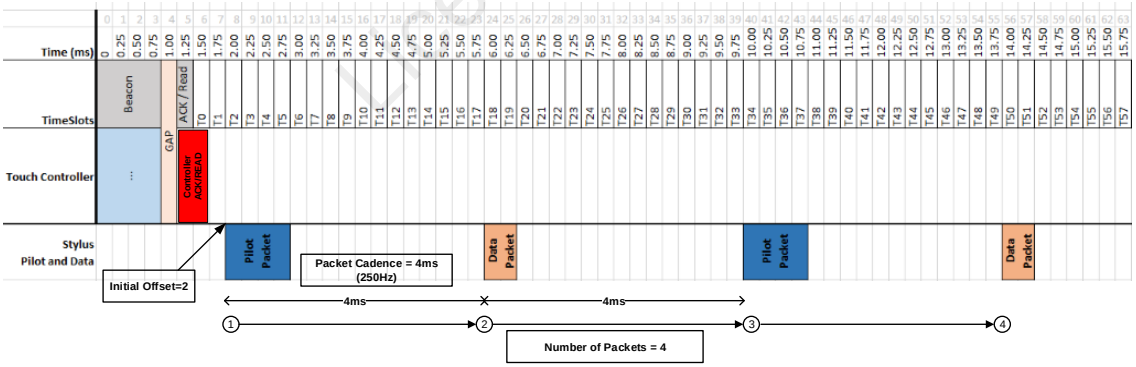


Figure 6-35 Example of C.ConfigDownlink(…) with a Position Packet

6.1.4.1.2 Example 2: Sending Data Packet Only

For the physical configuration the duration of the data packet are configured to 500 µs. The beacon cadence is 16 ms.

For the timing setup the following are defined:

- NumPackets=5.
- InitialOffset=2
- PacketCadence=4 (3 ms)
- PosEnable=0

- DataFormat=0 (Note that the same timing will be present if DataFormat=1.)
- Repeat=0

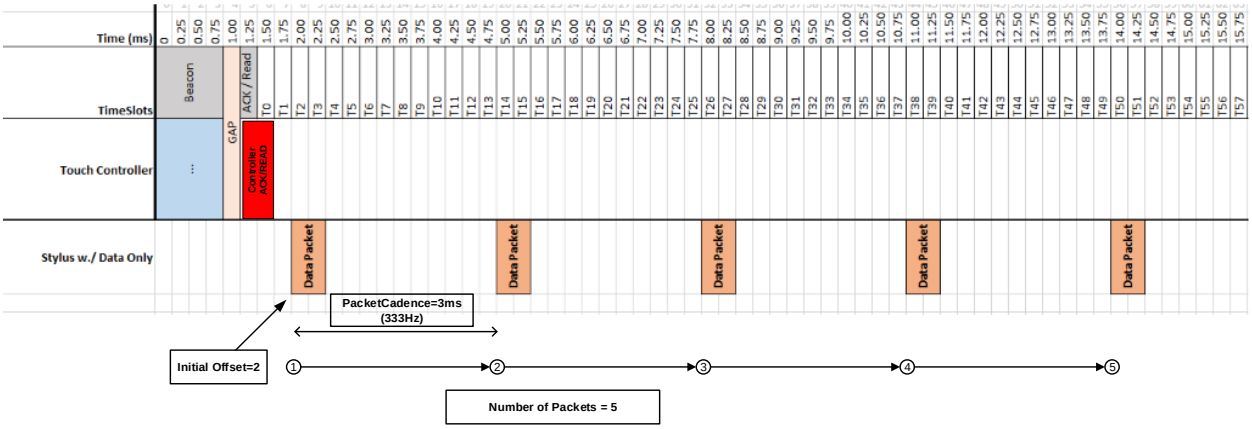


Figure 6-36 Example of C.ConfigDownlink(...) with only Data Packets

6.1.4.1.3 Example 3: Using a Data Packet with Repeat

For the physical configuration the duration of the data packet is configured to 500 µs. The beacon cadence is 16 ms.

For the timing setup the following are defined:

- NumPackets=5 (10 packets sent with Repeat=1).
- InitialOffset=4
- PacketCadence=4 (3 ms)
- PosEnable=0
- DataFormat=0 (Note that the same timing will be present if DataFormat=1.)
- Repeat=1

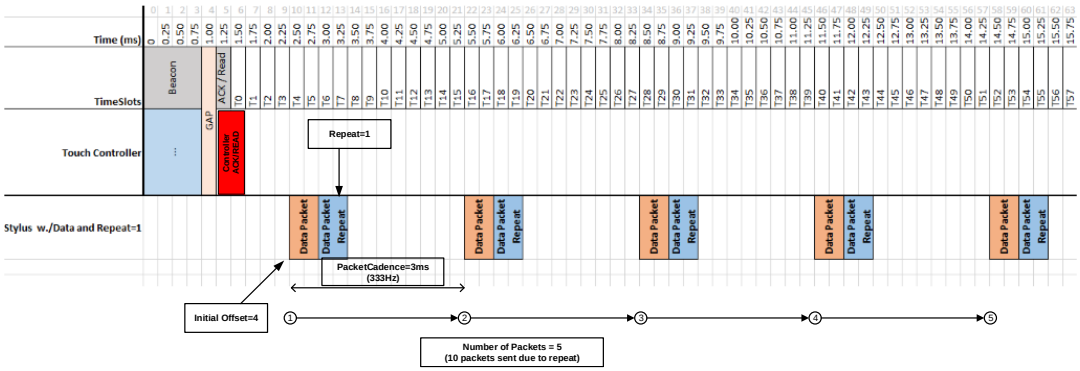


Figure 6-37 Example of C.ConfigDownlink(...) with Repeated Data Packets

6.1.4.2 Setup Using C.ConfigMenuSelect(…)

A number of predefined configurations (registered configurations with USI) may be used to enable fast setup of a series of timeslots and data types. A maximum of 30 predefined configurations may be used. The values 0 and 31 are currently reserved (see section 6.1.2.4 for details on the currently defined menu selection options). The packet types are defined in Table 6-11.

A stylus shall use MenuSelect index along with the IntialOffset programmed into it, to determine what type of packets need to be sent in what timeslots. The following are examples of how C.ConfigMenuSeelct works.

Example 1: MenuSelect#1: The controller configures the stylus with InitialOffset X (X=0); TS0 as the starting slot to send “default-data” type in TS0; Position packets during TS10 and TS41 timeslots. Then during TS0, the stylus shall send the data packet Type 1 and at timeslots TS10 and TS41 send Position packets.

Example 2: MenuSelect#1: The controller configures the stylus with InitialOffset X (X=3) and rest same as in the above example. Then, at timeslot TS3 (X+3) the stylus shall send data packet Type 1 and at timeslots TS13(X+10) and TS44(X+41), the Position packets.

6.1.4.3 Setup Using C.ConfigPacket(…)

C.ConfigPacket() sets up a packet to be repeated inside every USI frame.

For example, if the controller configures the stylus with the following values:

- PacketType=0
- PacketOffset=3
- NumPacket=5
- PacketCadence=10

the stylus will then transmit the six Position packets in the timeslots:
{TS3, TS13, TS23, TS33, TS43 and TS53}.

6.2 Stylus Beacon Response

All Read or Write commands shall always return a response. If the command is executed correctly, then the stylus shall return the data for Read command response, or return S.ACK() packet for Write command response. If the command is unsupported by the stylus or cannot be executed for some reason, the stylus shall respond with S.REJECT().

The only time Stylus shall not respond to a Read or Write command is the case when it receives a command with invalid CRC. In this case, the Stylus shall drop the packet and not respond.

The format of S.REJECT() shall be the following (for both Reads and Writes).

• 16 data bits shall have the fixed pattern 0xAA55.

• Given the fixed pattern for data, the inverted CRC (as required by S.REJECT(…) definition) is calculated to be 1000B.

The complete bit pattern (0x8AA55) for S.REJECT(…) response is shown below in Figure 6-38.

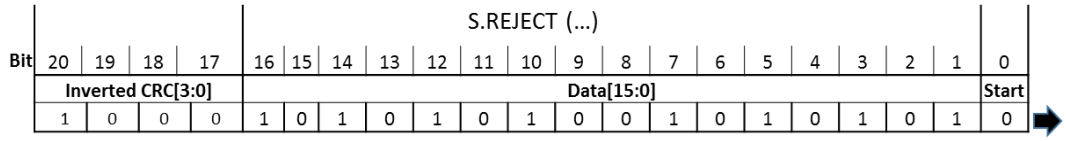


Figure 6-38 Stylus fields for S.REJECT(…) response

When the stylus sends an S.REJECT() response, it shall maintain the reason code for the error. This allows the controller to determine the cause of the failure. The controller can issue the C.GetLastError() command to retrieve the error code.

6.2.1 Response to Write Commands

Table 6-31 lists the various ACK / downlink packets sent by the stylus after the controller sends a write command.

Table 6-31 Stylus Responses to a Write Command

Type	Stylus Packet	Comment
C.ConfigPhy(…)Write		
	S.HASH(…)	Acknowledgment with a HashID is sent when the stylus has accepted a C.ConfigPhy(…) write command from the controller.
All other Write Commands		
	S.ACK(…)	Acknowledgement of write is sent when the stylus has accepted the write command.
	S.REJECT(…)	Rejection of write is sent when the stylus specifically rejects the write command (e.g., a single color stylus receives a command to change to a different color), or a command is not supported by the stylus

Table 6-32 shows the structure of the stylus response packet.

Table 6-32– Stylus Write-Response Packet Structures

Stylus Packet	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S.HASH(…)	HashID (16 LSB from 32 bit Hash ID)															
S.ACK(…)	Data from write command															
S.REJECT(…)	Inverted CRC and data pattern 0xAA55 (fixed pattern of 0x8AA55)															

6.2.2 Response to Read Commands

Table 6-33 shows the downlink packets from the stylus responding to a controller read command.

Table 6-33 Stylus Read Response Packet structures

Stylus Packet	Comment
Read Response	
S.'parameter'(…)	Returns the 16 bit data value requested by the 'C.Get'parameter' command.
S.REJECT(…)	Inverted CRC and data pattern 0xAA55 (fixed pattern of 0x8AA55)

6.2.3 No Response

The stylus shall send no response if it receives a packet with a CRC error.

6.3 Addressing All Styluses

When the controller is addressing all styluses (StylusID=7), the stylus shall return the READ/ACK data in the first timeslot (of TS0 to TS63) in which that stylus was assigned to transmit during normal operation.

Note that, if the stylus is configured to transmit Position Tone packets in (TS0 or TS63), the Position Tone slots will still be used to transmit a Tone. The first allocated Data packet timeslot shall be used for the READ/ACK data.

6.4 Stylus Interrupt

When the stylus is in its Operation state, it can send an interrupt to the controller to inform it that a change has occurred. The user changing the stylus color by pressing a button is an example of such a change.

The interrupt flag is part of the stylus' default data from the stylus. See the list in Table 6-35. This is a frequent message, which the Controller will need to decode at least once every frame in Operation.

The IRQ bit from the Stylus to the Controller gets set if one or more of the USI or Vendor Interrupt Flags for the Stylus is set, as shown in Figure 6-39. If no Interrupt Flags are set the IRQ bit will be logic 0.

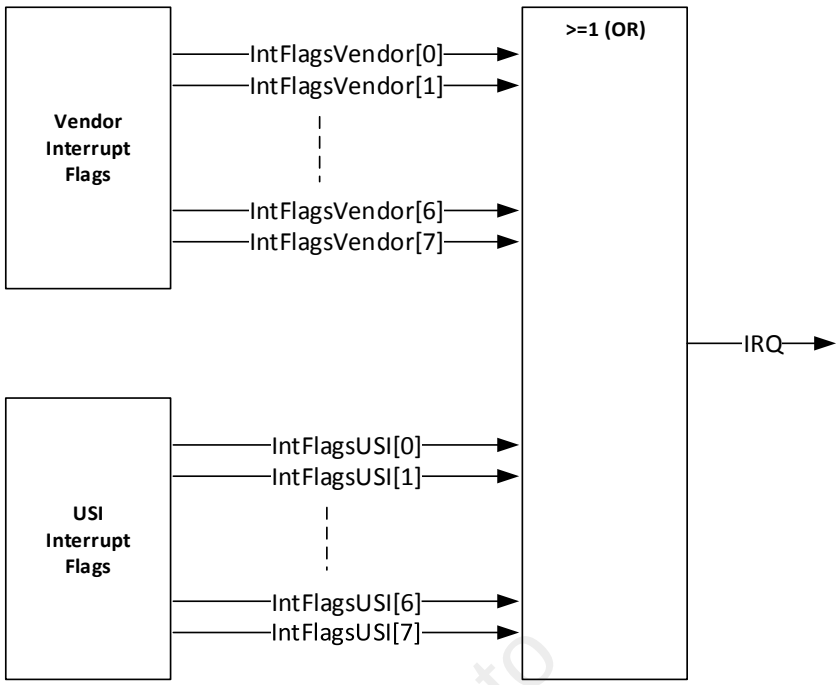


Figure 6-39 Stylus Interrupt Flags

The flags can be read by the C.GetIntFlags(…) command, and individually cleared by the C.ClearIntFlags(…) command.

Each flag will require individual handling, and it is specific for the flag which further operation that is required from the Controller.

The USI supports up to eight USI interrupt flags and eight Vendor Interrupt flags.

For the decoding of the USI interrupt flags, see Table 6-34 below.

Table 6-34 USI Interrupt Flags

IntFlagsUSI[n]	Interrupt flag
n=0	Stylus PreferredColor has changed
n=1	Stylus PreferredWidth has changed
n=2	Stylus PreferredType has changed
n=3	Stylus Tip Electrical Profile has changed
n=4	Stylus `Low Battery` warning
n=5-7	Reserved for future USI

NOTE: The decoding of vendor specific flags is vendor dependent. A Vendor specific interrupt flag should not be set if the Stylus is not running specific Vendor modes. A USI Controller not supporting Vendor mode should not be able to handle Vendor specific Interrupts.

Typical USI controller handling:

1. Controller detects the IRQ is set.
2. Controller invokes C.GetIntFlags(…) to read out the interrupt flags to detect which interrupt has occurred.

3. Controller runs its handler function for the interrupt flag that is set.
4. Controller invokes C.ClearIntFlags(…) to clear the interrupt flag.

6.5 Downlink Data Packet Types

The timeslot commands can set the stylus up to deliver data automatically. The stylus can then send either a Data packet or a Position packet. When it is sending a Data packet, the stylus may send any of five different data types. The data types are listed in Table 6-35. The packet delivery system allows for seven data types, but two values are reserved for future versions of this USI Technical Specification.

Table 6-35 Data Types for Downlink Data Delivery

Data Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Default data	IRQ	Button3	Button2	Button1	Pressure [11:0]											
Orientation	Sensor Flag [1:0]		Axis Flag [1:0]		Orientation Data [11:0]											
Extended Data	Reserved				Barrel pressure [11:0]											
Vendor Data0	Vendor specific data type															
Vendor Data1	Vendor specific data type															
Reserved	Future USI data type															
Reserved	Future USI data type															

6.5.1 Default data Packet Values

USI default data packet contains the following data:

6.5.1.1 IRQ

This bit is used by the Stylus to communicate any change in its capabilities. See section 6.4 for more details.

6.5.1.2 ButtonX

Buttons1, Button2, and Button3-these bits report the status of the stylus buttons. A value of 1 means that the corresponding button is pressed

6.5.1.3 Pressure

A 12 bit unsigned value that indicates the stylus tip pressure against the device surface. This value is reported by the stylus in a *log scale* that allows reported values from 5 gram force to 350 gram force or more. The exact scale and conversion factor is specified in Appendix L.

A stylus shall capture the pressure data at a minimum of 60 samples per second. It is recommended however that the stylus capture the data at a higher rate (up to the reporting rate the controller requests), to decrease inking latency, for better user experience. In any case, the pressure report rate sent to the Controller will still be what the Controller requested. If the pressure capture rate is lower than the report rate, the stylus shall either repeat the capture data to fill in until newer measurements take place or extrapolate appropriately. It is outside of the USI specification as to what extrapolation algorithm to employ.

A value of zero indicates that the tip is not touching the screen and indicates the 'hover' state.

***Note**: Current CRC scheme may allow certain two-bit errors to be propagated as valid values. A value of zero (hover) may get reported as 0x1 or 0x800 pressure values. To avoid this, the Stylus shall not report these two values and use the next higher value instead. These two values shall be rejected by the Controller as invalid pressure values.*

6.5.2 Orientation

The orientation data contains tilt, twist, and IMU data. The data is multiplexed and can occur more than once per frame. Ideally the touch controller would allocate as many IMU slots per frame as feasible up to the total number of IMU axis available (three, six, or nine slots) per coordinate report. If complete sensor data is not sent in a single frame the remaining data is sent in subsequent frames.

- **Sensor Flag:** The sensor flag indicates which orientation sensor originated the data:
  - Sensor Flag = 0: The sensor data is from an accelerometer
  - Sensor Flag = 1: The sensor data is from a gyroscope
  - Sensor Flag = 2: The sensor data is from a magnetometer
  - Sensor Flag = 3: The sensor data is from Tilt and/or Twist sensors
- **Axis Flag:** The axis flag indicates which sensor axis originated the data
  - Axis Flag = 0: The data is for the X axis or is X Tilt
  - Axis Flag = 1: The data is for the Y axis or is Y Tilt
  - Axis Flag = 2: The data is for the Z axis or Twist
  - Axis Flag = 3: Unused
- **Orientation Data:** The orientation data contains the sensor data indicated by the Sensor Flag and Axis Flag. Orientation data is only reported for the sensors indicated by the Capability flags. Orientation data is sent in a fixed order as follows:
  1. X Tilt then Y Tilt
  2. Twist
  3. Accelerometer X, then Y, then Z
  4. Gyroscope X, then Y, then Z
  5. Magnetometer X, then Y, then Z

Orientation data is represented as follows:

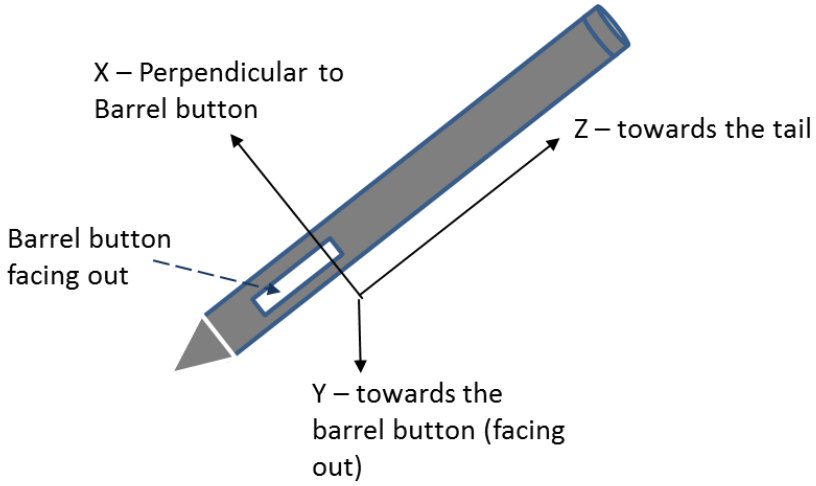
- **X tilt and Y tilt**:
  Tilt is reported as degrees relative to the device surface. Zero degrees is vertical to the device surface. -90 degrees X tilt is to the left. 90 degrees X tilt is to the right. -90 degrees Y tilt is away from the user. 90 degrees Y tilt is toward the user.

  Values are reported as signed 12 bit numbers. -2047 represents -90 degrees and +2047 represents +90 degrees.

  If the stylus naturally measures orientation as azimuth and altitude instead of tilt, the stylus shall indicate it has tilt capability, and convert azimuth and altitude to X and Y tilt for reporting back to the touch controller.
- **Twist:**
  Twist is reported as degrees.

  Values are reported as unsigned 12 bit numbers. 0 represents 0 degrees and 4096 represents +360 degrees.
- **IMU**:

Inertial measurement units report orientation relative to the earth. For consistency, positive Z is toward the tail, positive Y is toward the barrel buttons, and positive X is tangent to the barrel buttons in a clockwise direction.



Complete IMU data (all sensors for all axis) will be measured as simultaneously as possible and buffered in the stylus. This reduces temporal errors in the data. As soon as the last data item is sent from the buffer another sample is taken.

Values are reported as signed 12 bit numbers. The units are as follows

- Accelerometer: reported in mG (milli G)
- Gyroscope: reported in dps (degrees per second)
- Magnetometer: reported in milli Gauss

If both the stylus and touch controller have access to nine axis IMUs, the touch controller will:

- Report the Stylus IMU values to the host as IMU values.
- If the stylus does not report Tilt and Twist, the touch controller shall use the orientation information from the two IMUs, compute Tilt and Twist of the stylus relative to the device, and report these values to the host as Tilt and Twist above.

Since IMU data is reported to the host every packet, but a new (complete) measurement may not be available every packet, it is the responsibility of the touch controller to extrapolate intermediate IMU values for reporting to the host. So for example, if coordinates are available twice per frame, but a complete IMU measurement is only available once per frame, extrapolated IMU measurements are used for the missing IMU report.

6.5.3 Extended Data

This field contains the 12-bits of barrel pressure data. Stylus may have a specific side button or other grip points measuring the pressure on the barrel (body) of the stylus. The units of this are the same as described above for tip pressure.

6.5.4 Vendor data

Vendor Data0 and Vendor Data1 are allocated for vendor specific data.

6.6 Stylus Capabilities Information

The feature information bits of the USI Stylus are divided into two main groups:

- Capability information
- Sticky information

Stylus capabilities are parameters that never change, such as the type of sensors that are available (tilt, twist, barrel pressure).

Sticky Information, on the other hand, is information that the controller might be allowed to change by invoking the protocol commands, or it can be changed by the user by an explicit action, such as activating a special button on the stylus. The existence of sticky information allows the user to customize the stylus and store the settings in the stylus for use with multiple controllers. This information includes:

- Preferred Color
- Preferred Stylus brush type (PreferredType)
- Preferred Stylus width (PreferredWidth)
- Stylus Electrical Profile

How 'sticky' the stylus' information is, is up to implementation. The protocol may support updating certain information, but in some styluses that information is fixed in hardware. In other implementations the same sticky information might be configurable.

When sticky information is stored by a stylus, it might be stored only until the system loses power (so that information will be lost when the battery is replaced). Or in other implementations the same information might be preserved even when power is lost.

6.7 Stylus Hash ID

6.7.1 Why is Hashing Needed

Before inking can begin the controller needs to obtain certain information from the stylus. The fastest way to do this is for the controller to already have the information cached from a prior encounter with the stylus.

The hash provides a quick way to determine whether the information from the stylus is already in the controller's cache. If it is, the controller is able to bypass several steps in the pairing process when the stylus is approaching.

6.7.2 What Needs to be Hashed

The values that are needed to be hashed should create a 'unique' ID for the stylus. The information to be hashed is:

- Global ID
- Sticky information (information that might be changed by the user).

6.7.3 Hashing Algorithm

The USI hashing algorithm is MurmurHash3. For details about the algorithm, see Appendix G.

The USI hash algorithm produces a 32 bit hash digest:

- The 16 LSBs are used in the stylus response S.HASH(…) to the C.ConfigPhy(…) command

- The 16-LSBs are used by the controller when it sends C.VerifyHashLow(…) to ask the stylus to verify those bits
- The 16 MSBs are used by the controller when it sends the C.VerifyHashHigh(…) command to ask the stylus to verify those bits.

6.8 States of Operation

6.8.1 Overview

The following sections describe the operating states that the USI Stylus can occupy. These sections describe which protocol commands are expected during each of the states together with the expected protocol responses from the stylus. An illustration of state flow and transitions between states is given for reference.

The stylus has three main operating states:

- Discovery
- Pairing
- Operation.

Note that in some implementations the controller supports more than one stylus. The controller shall in this case hold an independent record of the state of each stylus.

6.8.2 Stylus Approach

When a USI Stylus approaches a USI controller, the controller shall go through a sequence of operations (write and read commands) to get the stylus from Discovery state into Operation state. In Figure 6-40 it can be seen that initially, when the stylus and controller are out of range, both are in their Discovery states. When the stylus approaches, a communication link opens and the stylus and controller can start to pair; this is the Pairing state. Once they are paired, the controller can start to convey the stylus information to the device host processor, including the stylus position, pressure and button status.

When the stylus is approaching a controller, its stylus ID shall be set to 0; this indicates the stylus has not been paired. The controller will be looking for the stylus and shall send out a C.ConfigPhy(…) command within the Beacon with NewStylusID value. As the stylus gets within range to detect the controller's Uplink packets, it will preliminarily (store but not use yet) set its new stylus ID assigned by the controller. The stylus shall not fully accept the new stylus ID until the controller addresses it using that NewStylusID. Until that happens, the stylus shall continue to accept new configuration commands and new stylus ID commands.

The stylus will respond with an ACK packet starting at time T_{ACK}. The communication link between the controller and the stylus is asymmetric. The controller will in some cases not see the first ACK issued by the stylus when the controller detection height is lower than the stylus detection height, as shown in Figure 6-40. The controller will continue to send the Discovery command until the controller sees an acknowledgement in an ACK packet from a stylus.

When the controller sees an acknowledgement in an ACK packet, it shall then address the new stylus to inform the stylus that it has been detected. The stylus will then fully accept its NewStylusID; this value which will remain in use as the stylus' StylusID until it moves out of range or the controller unpairs it.

In cases when two (or more styluses) approach the device at exactly the same time, there is a possibility that both styluses will send the ACK packet at the same time. It is possible that both ACKs collide and none is successfully received by the Controller. If such a collision is detected by the Controller, then the

Controller shall send a C.VerifyHashLow() command with a HASH of all zeros with the StylusID field set to 0. Upon receiving this command the Stylus shall return to unpaired state, and set a random back-off variable between 0-16. This back-off variable shall then be used to skip the number of beacons while responding to the C.ConfigPhy() requests. For example, if two unpaired styluses receive the C.VerifyHashLow() command, one of them may set its random back-off variable to 1 while other may set it to 9. The first stylus shall skip responding to the next C.ConfigPhy() command, while the second stylus shall skip responding to next 9 commands.

If the controller has not seen the stylus before, it will run through the Pairing process with the stylus. In the Pairing state the controller will read out the capability and configuration of the stylus. When the controller has learned the capabilities of the stylus, it shall put the stylus into the Operation state. In this state the stylus starts to deliver position and data messages in its allocated timeslots.

If the controller has previously seen the stylus hash ID before, it will put the stylus directly into Operation state by configuring its timeslot and packet delivery format.

6.8.3 Stylus Retreat

When the stylus starts to retreat from the stylus hover range, the controller loses the communication link with the stylus. If after a defined timeout period the communication is not restored, then the controller shall unpair (see section 6.8.8.2) from the stylus by invoking an unpair command. If the stylus is within range of the controller, then the stylus should receive the unpair command and disconnect from the communication link. If the stylus does not receive an unpair command or receives that command in error, then it shall wait until it is outside the controller's beacon range before halting communications and finally disconnecting.

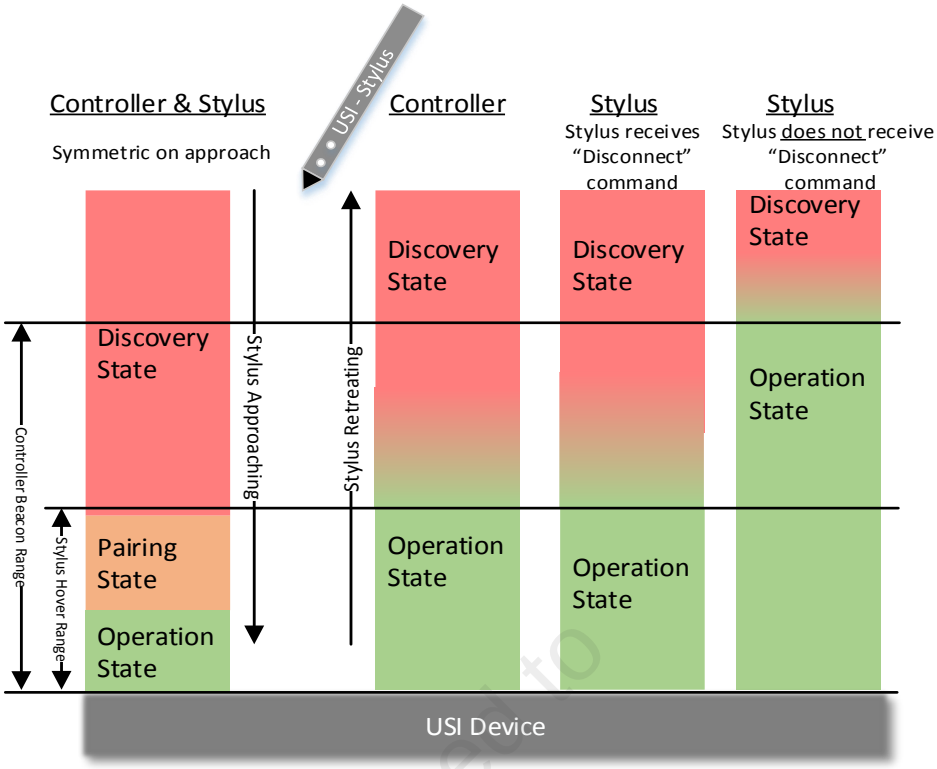


Figure 6-40 States during Stylus Approach and Retreat

For single stylus operation the controller and stylus shall only ever be in one state at a time. The three possible states are Discovery, Pairing and Operation.

When a single controller is capable of supporting multiple styluses, each link between the controller and an individual stylus may be in a different state from the controller's link with another stylus.

The USI controller shall be able to track multiple styluses that are in different states within a USI frame period. For example, when a single stylus is being used and a second stylus approaches the controller, the first stylus will be in its Operation state, while the second stylus needs to be discovered and then paired.

Figure 6-41 indicates the basic functionality in each state. Full details of each state and an illustration of the associated flow are given in the following sections.

The state descriptions are relevant for a single USI Stylus connection.

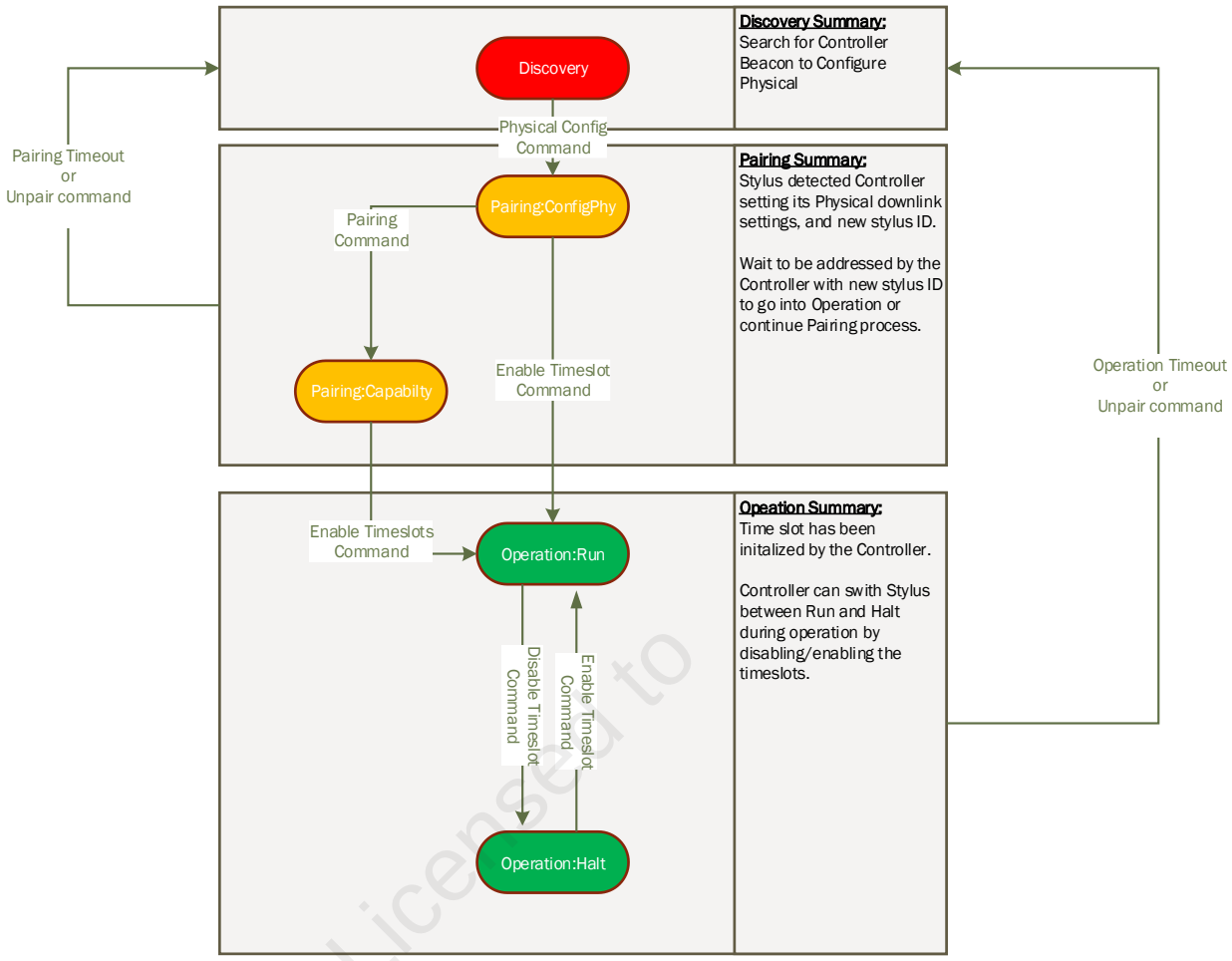


Figure 6-41 State Summary

6.8.4 Protocol Command and Relation to State

As described in section 6.1, the controller operates the stylus using a predefined command set. The commands are related to one or more operating states. Table 6-36 summarizes the commands and the state in which each is intended to be used.

Table 6-36 Overview of Commands and Primary State Usage for Pairing and Operation

	States					
Commands	**Discovery**		**Pairing**		**Operation**	
Write	**Controller**	**Stylus**	**Controller**	**Stylus**	**Controller**	**Stylus**
C.ConfigPhy(...)	M	M	O	T	O	
C.ConfigDownlink(...)			M (at least one)	M	O	T
C.ConfigMenuSelect(...)				M	O	T
C.ConfigPacket(...)				M	O	T
C.DisablePacket(...)					O	M
C.SetColor(...)					M	O
C.SetButtons(...)					M	O
C.SetType(...)					M	O
C.VerifyHashLow(...)			O	M	M	M
C.VerifyHashHigh(...)			O	M	M	M
C.UpdatePhy(...)					O	M
C.ClearIntFlags(...)					M	M
C.TransmitPositionTone(..)					O	M
C.SetWidth(…)					M	O
C.SendMyVendorID(…)					O	M
C.SetVendorExtension(…)					M	O
Read						
C.GetBattery(…)					O	M
C.GetCapability(...)			M	M	M	M
C.GetUSIVersion(...)			O	M	M	M
C.GetGID(...)			O	M	M	M
C.GetData(...)					O	M
C.GetVendorExtension(...)					O	M
C.GetPhy(...)					O	M
C.GetTimeslotSetup0(...)					O	M
C.GetTimeslotSetup1(...)					O	M
C.GetIntFlags(...)					M	M
C.GetHashID(…)					O	M
C.GetFirmwareVersion(…)					O	M
C.GetLastError(…)					O	M
C.GetTiltTwist(…)					M	O
Timeout(*)	M	T	M	M	M	M
Unpair command(*)		T	M	M	M	M

Notes: T: Trigger commands, used by Stylus to enter the state, implies it is Mandatory.

M: Mandatory support; used primarily in this state.
O: Optional support; could be used in this state.

(*): Timeout from pairing or operation or a controller command that unpairs the stylus.

6.8.5 Timeouts

The stylus and controller will both require timeouts when the communication link is lost, as shown in Figure 6-41, to ensure transition to the correct state. The timeout could be triggered due to the stylus' being out of range, a missed expected timeslot or incorrectly received packets. How fast the controller times out depends on which state it is in. The timeout is required to prevent the system from entering deadlock when packets are lost.

Table 6-37 shows the controller's timeout parameters.

Table 6-37 Controller Recommended Timeout Values

State	Timeout	Description
Discovery	N/A	The controller shall not have any timeout in this state.
Pairing	1 USI frame period	The controller can only lose the stylus response for a single USI frame. It shall repeat the command immediately on the next USI frame. If no response is seen, it unpairs the stylus.
Operation	> 250 ms	If the controller can't see the stylus packets, it unpairs the stylus.

The controller should count events or time between events if it doesn't see a valid stylus ACK or Read response while it is in its Pairing state and if it doesn't see valid packets from the stylus in the Operation state.

Table 6-38 shows the timeout parameters for the stylus.

Table 6-38 Stylus Timeout Values

State	Timeout	Description
Discovery	N/A	Stylus shall not have timeouts in this state.
Pairing	2 USI frame periods	Stylus times out if it isn't addressed in two subsequent USI frame periods.
Operation	> 250 ms	Stylus doesn't receive uplink synchronization. Stylus uses a look up table based on the beacon cadence setting.

6.8.6 Discovery State

The Discovery state occurs when either the controller or stylus is trying to find the other using the communication link of Beacon command and ACK response.

In the Discovery state the controller and stylus shall use the following command and responses.

Table 6-39 Commands Running in Discovery State

Commands	Discovery	Pairing	Operation
C.ConfigPhy(...)	X		

The basic discovery operation consists of:

1. The controller transmits the discovery command C.ConfigPhy(…) in the Beacon packet.
2. The stylus receives the discovery command and sends out a S.HASH(…) response in its ACK timeslot.

6.8.6.1 Controller Operation

When the controller is in Discovery state it is trying to discover the stylus. It shall send the C.ConfigPhy(...) command in its transmission of the Beacon. This command sets the new stylus ID and configures the physical interface for the downlink. When the stylus has received the command, it sets up its downlink and responds with an S.HASH(...) in its ACK timeslot. If the controller sees the correct S.HASH(…) response, it sets the stylus in its records as 'discovered' and transitions the stylus and itself to the Pairing state. If the controller doesn't see any ACK, it shall remain in the Discovery state, regularly sending the C.ConfigPhy(...) command.

The controller can use the Discovery state, or lack of it, to limit the number of styluses supported. If the controller has reached the limit of its capabilities, then it can decide to not transmit the discovery command or not pair with a new stylus.

6.8.6.1.1 Error Conditions for the Controller

If the controller misses the S.HASH(...) response from the stylus, the condition is handled by repeating the previous message in the next USI frame.

6.8.6.2 Stylus Operation

Until it is linked to the controller, the stylus shall keep looking for Beacon commands. If it sees the C.ConfigPhy(...) packet from the controller it shall set up the physical downlink and respond with the S.HASH(...) response which includes its own hash ID. The stylus then expects the controller in the next USI frame to start to continue the pairing process. But the controller might miss the packet. To avoid deadlock, a timeout counter is started. The counter is reset/started each time the stylus sees a C.ConfigPhy(...) command. The stylus needs to be addressed before the timeout in order to proceed to Pairing. Otherwise it shall forget its settings with the controller and shall return to searching for Beacon commands that contain C.ConfigPhy(...).

6.8.6.2.1 Error Conditions for the Stylus

If the stylus sends out the S.HASH(...) packet, there is no explicit acknowledgement from the controller that it has received the response. There is an implied acknowledgement from the controller by addressing the stylus with its new stylus ID. The stylus will use its timeout to avoid getting stuck waiting to be addressed with its new stylus ID before it begins a transition to its Pairing state. For details on timeouts during the Discovery state, see Table 6-37 and Table 6-38.

6.8.7 Pairing State

When the stylus is in Discovery state, the next state triggered is Pairing. When the stylus has discovered the controller by successfully receiving a C.ConfigPhy(…) packet, it will enter Pairing:ConfigPhy state, as shown in Figure 6-42.

Figure 6-42 Controller entering Pairing

When the controller has discovered a stylus, it performs a quick pairing if it already knows the stylus, or it conducts a full pairing process if the stylus is unknown. The quick pairing just sets up the data packets and timeslots, because the capabilities of the stylus are already known. The full pairing requires the controller to discover the capabilities of the stylus, read the stylus' global ID value and the USI version number, then set up the appropriate data packets and timeslots. Whether the stylus is known or not depends on whether the controller has previously seen the hash ID contained in the S.HASH() response from the stylus.

As a minimum for pairing with an unknown stylus, the controller shall read out (from the stylus):

- Stylus Capabilities – (e.g., number of buttons, tilt capable, etc.)
- Sticky information – (e.g., color, tip style, etc.).

This is done using the C.GetCapability(…) command.

The controller may read out the IDs and vendor information from the stylus while it is in the Pairing state. These may be read either during the Pairing process or in the Operation state. To minimize inking latency for an unpaired stylus, it is recommended to do the additional read out when the stylus is in the Operation state. Table 6-40 shows the read-out commands and their intended usage state.

Table 6-40 Commands for Pairing

Commands from Controller	States			Responses from Stylus
	Discovery	Pairing	Operation	
C.GetCapability(...)		**P**		S.Capability(…)
C.GetUSIVersion(...)		S	P	S.USIVersion(…)
C.GetGID(...)		S	P	S.GID(…)

Notes: P: Primary state; used primarily in this state.
S: Secondary state; could be used in this state.

For details on the commands listed in Table 6-40 see the command descriptions in section 6.1.2.

Figure 6-43 illustrates the actions and next state decision of the pairing operation for the stylus.

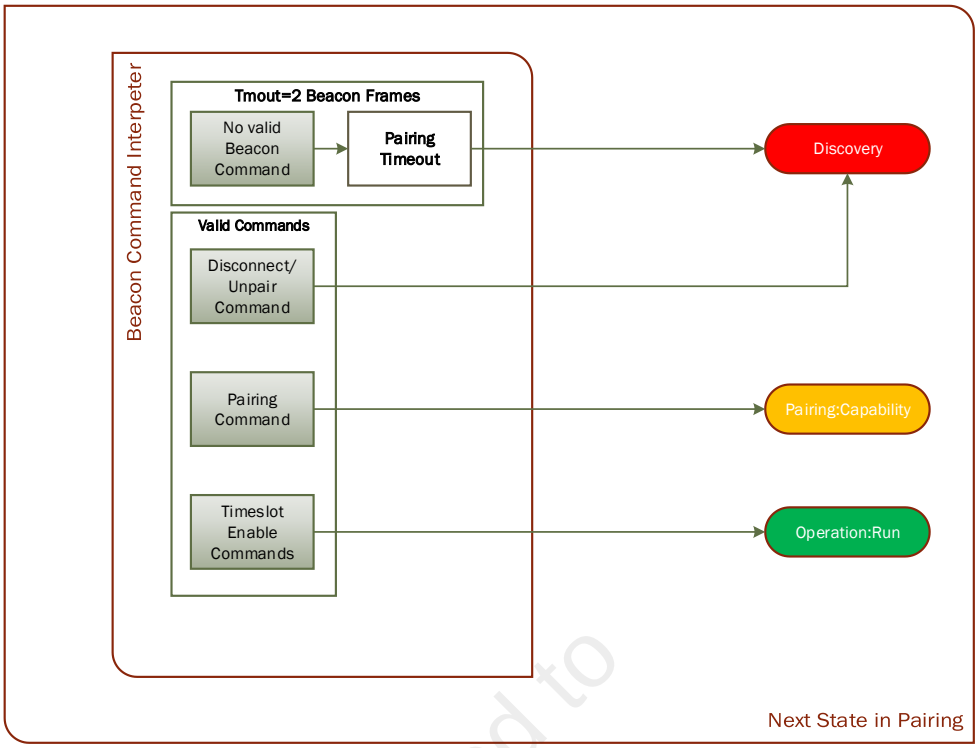


Figure 6-43 Basic Pairing State Flow Diagram

6.8.7.1 Controller Operation

When the controller first enters the Pairing state, it checks the value of the hash ID received from the stylus. If the hash ID is known, the controller can transition directly to Operation state. If the hash ID is unknown, then the controller uses the C.GetCapability command to acquire information about the capabilities of the stylus. Based on the responses, the controller can identify how many timeslots are needed to support the full functionality of the stylus. Given the available resources in the controller it may enable all or some of the capabilities of the stylus.

When the hash ID is unknown, the stylus will start to read out its information; this is part of the Pairing state. The controller has two types of iteration loop:

1. Loop through the command list to get capabilities
2. Based on capabilities write commands, repeat the command until the procedure is complete or until a timeout occurs.

If the controller does not see the appropriate responses from the stylus it shall eventually time out and put the stylus back into its Discovery state.

When all pairing commands are completed, the controller shall start to set up the stylus for its Operation state.

6.8.7.1.1 Error Conditions for the Controller

If during the pairing process initiated by the controller the stylus does not acknowledge the individual commands, then there are options to retry the same command until a response is received. If no response is seen within a defined timeout period, the controller shall force a change back to Discovery state.

6.8.7.2 Stylus Operation

After the stylus receives a new stylus ID and communications link physical configuration it is in the Pairing:ConfigPhy state. The stylus then looks for Beacon commands to start the further pairing process, the Pairing:Capability state, or to transition to the Operation (Operation:Run) state. The commands can be of the following types:

1) **Pairing command:** a command to read out data from the stylus (e.g., C.GetCapability(...) addressing it with the new stylus ID)
2) **Operation command:** a command to set timeslot configuration parameters in the stylus
3) **Unpair command:** a command disabling the stylus' link with the controller by clearing its stylus ID and forcing it back to Discovery state.
4) **Beacon Sync command:** General Beacon command not specifically addressed to the stylus being paired.

6.8.7.2.1 Error conditions for Stylus

If during the pairing process initiated by the controller the stylus does not receive packets addressed to the new stylus ID, then a timeout mechanism should be implemented to ensure the stylus does not remain in the Pairing state but transitions back to the Discovery state.

6.8.8 Operation State

At the start of the Operation state the controller issues a command to define the timeslots and packet types to be used during the Operation state. The command issued by the controller is one of the commands in Table 6-41. Once the stylus has acknowledged receipt of the command using the S.ACK(…) response, the controller and stylus are both in Operation state. This allows both the stylus to transmit the required data packets and the controller to receive them in the same USI frame period as the command was sent.

The timeslot and packet type configuration can be modified at any time during the Operation state using the same commands or the additional C.DisablePacket(...) command.

Table 6-41 Commands to start Operation

Commands	State		
	Discovery	Pairing	Operation
C.ConfigPacket(...)			T
C.ConfigDownlink(...)			T
C.ConfigMenuSelect(...)			T

Notes: T: Trigger commands, used to enter the state.

These commands have the following capabilities:

C.ConfigPacket(...): Manual configuration that allows for single timeslot/packet type definition per frame
C.ConfigDownlink(...): Auto configuration that allows for repeated data types per frame
C.ConfigMenuSelect(...): Predefined configuration with fixed timeslot and data types per frame.

When the Stylus is in Operation state and is transmitting packets in its timeslots, it is in the Operation:Run state. If the controller is in an exception situation and needs to disable all stylus packets in a USI frame, it can use the C.DisablePacket(all) command. When this happens, the stylus is in the Operation:Halt state.

This state will have the same timeouts and will accept the same commands as the Operation:Run state does.

During the Operation state the predefined packet types are allocated to timeslots, and additional data can be read as required using one of the commands listed in Table 6-42. The response is provided by the stylus in the ACK timeslot.

Table 6-42 Commands during Operation

Commands	State		
	Discovery	Pairing	Operation
C.DisablePacket(...)			P
C.SetColor(...)			P
C.SetButtons(...)			P
C.SetType(...)			P
C.GetHashID (...)			P
C.VerifyHashHigh(...)			P
C.UpdatePhy(...)			P
C.GetBattery(...)			P
C.GetUSIVersion(...)		S	P
C.GetGID(...)		S	P
C.GetData(...)			P
C.GetVendorExtension(...)			P
C.GetPhy(...)			P
C.GetTimeslotSetup0 or 1(...)			P
C.GetIntFlags(...)			P
C.ClearIntFlags(...)			p

Notes: P: Primary state; used primarily in this state.
S: Secondary state; could be used in this state.

Figure 6-44 illustrates the actions and decision of the stylus operation.

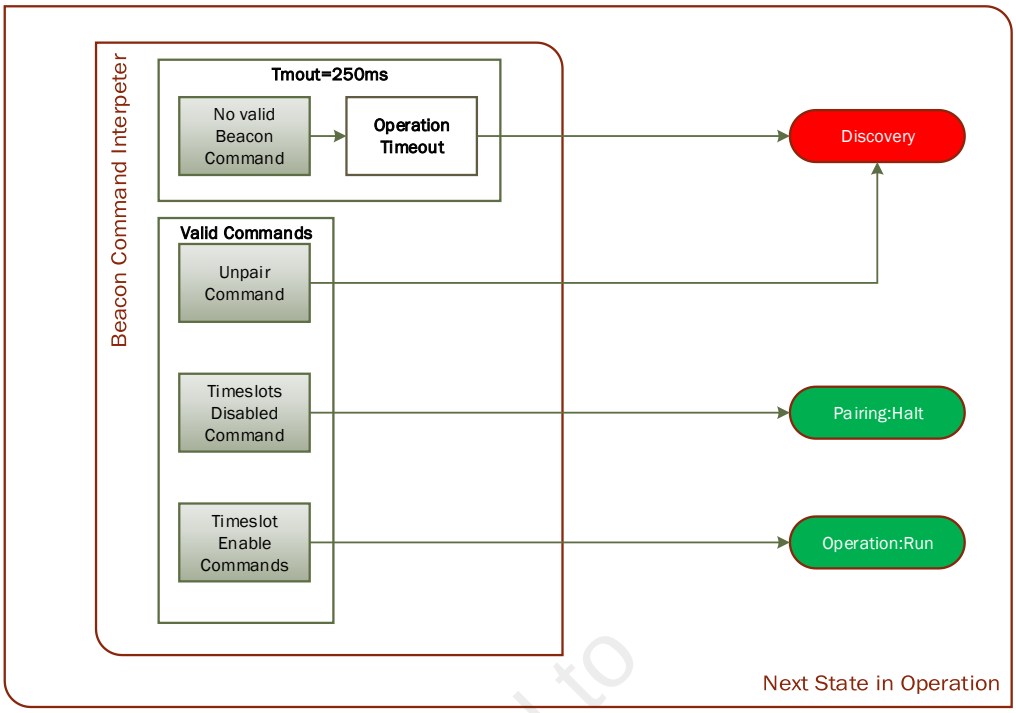


Figure 6-44 Basic Operation State Flow Diagram

6.8.8.1 Controller Operation

At the start of Operation state the controller issues a command (e.g., C.ConfigMenuSelect(...)) to define the timeslots and packet types to be used while the controller and stylus are in the Operation state. If the controller receives the S.ACK(…) response within the required timeout, from that point on it can start to receive data and calculate the position of the stylus. These data are then processed by the controller and passed to the host processor using the mechanism defined in Chapter 7.

Additional end user information or applications on the host can request data from the stylus (such as global ID or battery status). This is requested by the controller using specific Beacon commands; the responses are provided by the stylus in the ACK timeslot.

6.8.8.1.1 Error Conditions for the Controller

If during the start of operation the controller does not receive the S.ACK(…) in the ACK timeslot following the command, the controller can retry until either it receives an S.ACK(…) or the connection times out. If time out occurs, then the controller should issue an unpair command.

If during additional data reads the stylus response is not received or is received incorrectly, then the controller can retry the same command. It is up to the controller how many retries it supports.

Additionally if the controller is not requesting data from the stylus and is just receiving the stylus packet data, if this data is not received, due to stylus going out of range, then a timeout is used to move back to the Discovery state.

6.8.8.2 Stylus Operation

The stylus changes from the Pairing state to the Operation state on receiving a configuration command and subsequently responding to it. The stylus transmits a response S.ACK(…) followed by a request Data packet in the configured timeslots.

In subsequent USI frames the stylus shall receive the Beacon and checks whether the command is addressed to it by comparing the value of the StylusID field in the command against its own local value. If the command is addressed to the stylus, then, depending on whether the command is a read or write, the stylus either responds with the required read data in the ACK timeslot following the Beacon or updates its internal values.

Whether or not the command is directed to it, the stylus shall always respond in the configured timeslots with the required packet type.

The stylus continues to send Data packets in timeslots and respond to commands whenever it receives a Beacon. If a Beacon is not received, then it transmits no Data packet, if no Beacon is received for a defined timeout period, then the stylus moves to the Discovery state.

The controller may explicitly unpair the stylus by transmitting the command C.ConfigPhy(NewStylusID=0) to it.

6.8.8.2.1 Error Conditions for the Stylus

If the Beacon command is not received at the expected time, then a timeout is used to ensure that the stylus does not hang up waiting for a Beacon. The timeout moves the stylus from its Operation state to the Discovery state.

6.8.9 Sequence Diagram for Stylus Operation

This section contains examples of controller commands and stylus responses during typical operation scenarios.

6.8.9.1 Stylus with Known Hash ID

Figure 6-45 shows how a stylus with a known hash ID is paired and put into operation by the controller.

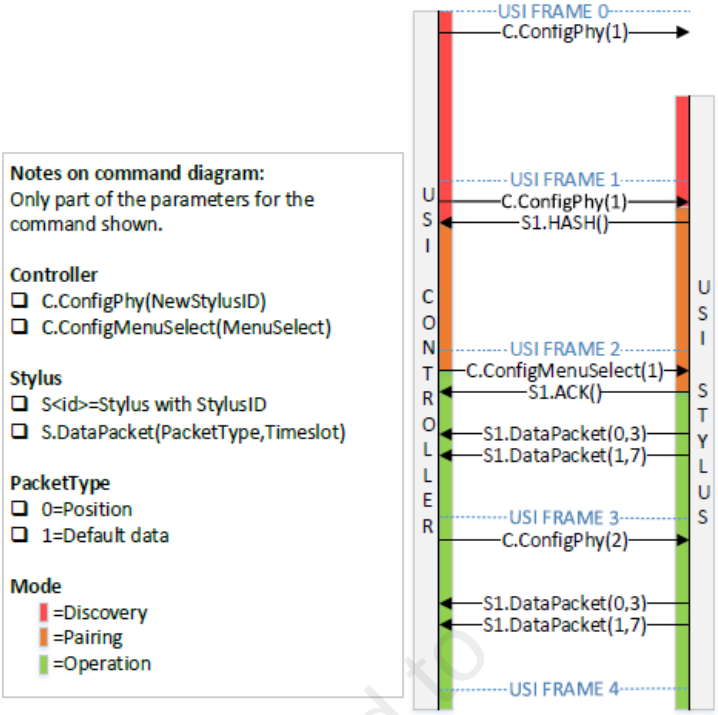


Figure 6-45 Controller Detecting a Stylus with a Known Hash ID (Single Packet Read)

6.8.9.2 Stylus with Unknown Hash ID

Figure 6-46 shows how a stylus with an unknown hash ID is paired using a capability read out and is put into operation by the controller.

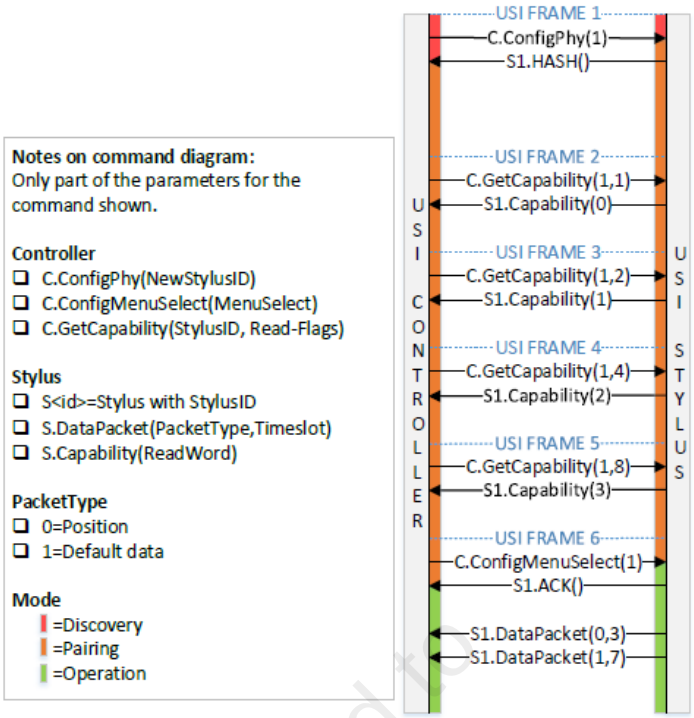


Figure 6-46 Controller Detecting a Stylus with an Unknown Hash ID (Single Packet Read)

6.8.9.3 Stylus with Unknown Hash ID (Multiple Packet Read)

Figure 6-47 shows how a stylus with an unknown hash ID is paired using a capability read out (multiple packet read) and is put into operation by the controller.

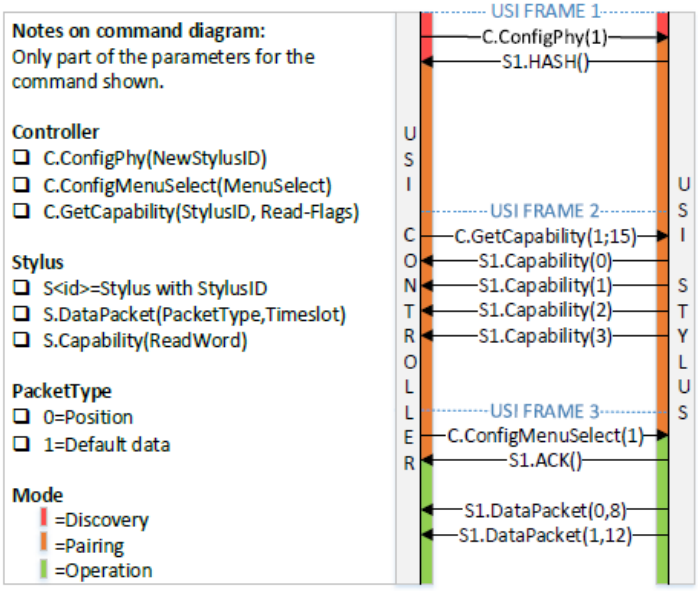


Figure 6-47 Controller Detecting a Stylus with Unknown Hash ID (Multiple Packet Read)

6.8.9.4 Timeout during Pairing

The timeout procedure handles protocol violations to avoid either the stylus or controller becoming stuck in a deadlock. However, significant time is lost if both go through the whole timeout process.

Figure 6-48 illustrates how a controller and stylus can time out during a Pairing state.

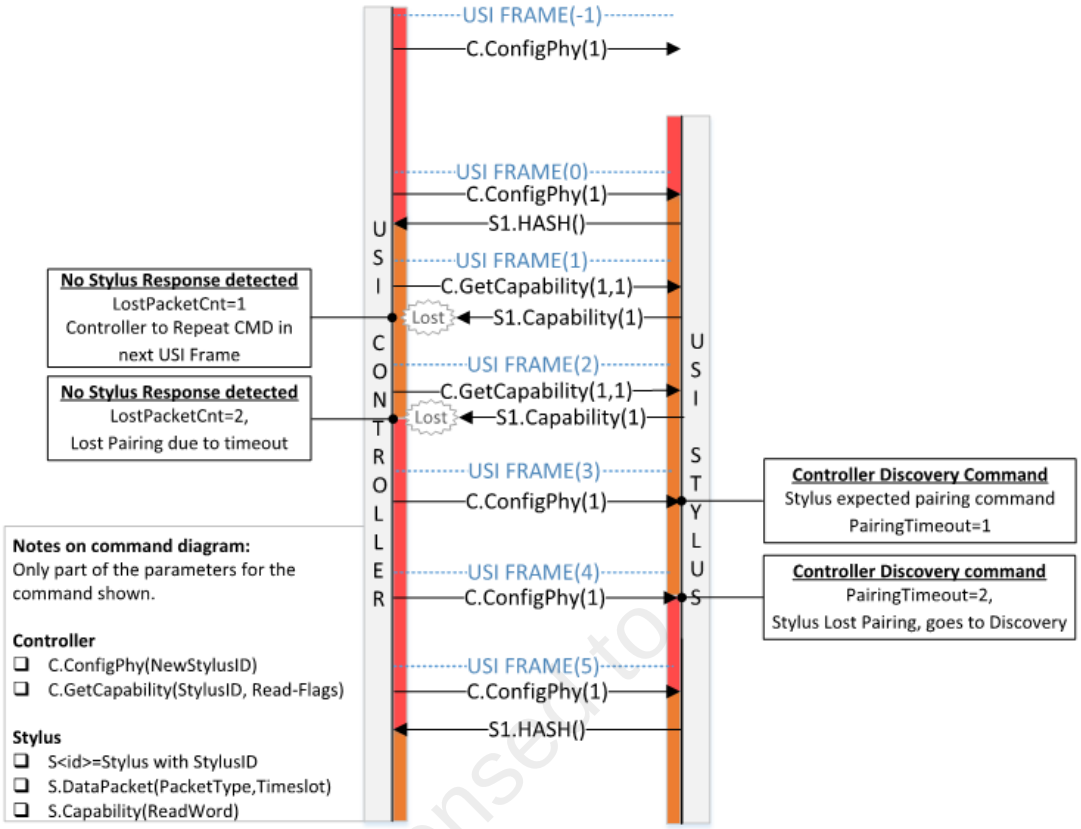


Figure 6-48 Both Controller and Stylus Timing Out (Not Recommended)

In this example the controller initially detects the stylus. However, the stylus' response to the Beacon is lost (typically due to either weak signal or interference). The controller then repeats the command for the first lost packet in an attempt to recover. If the second packet also is lost, as the illustration shows, the controller goes ahead sends out new invitation commands. The stylus is still trying to recover and looks in two USI frames for the controller addressing it. Because the controller has stopped addressing it, the stylus will, after two USI frames, time out and return to being unpaired.

Note that in this situation both the controller and the stylus time out. When the controller loses the stylus, the recommendation is that the controller unpair the stylus via an unpair command. The stylus will then be ready to accept the new invitation from the controller one cycle earlier than when it times out. Figure 6-49 illustrates this scenario.

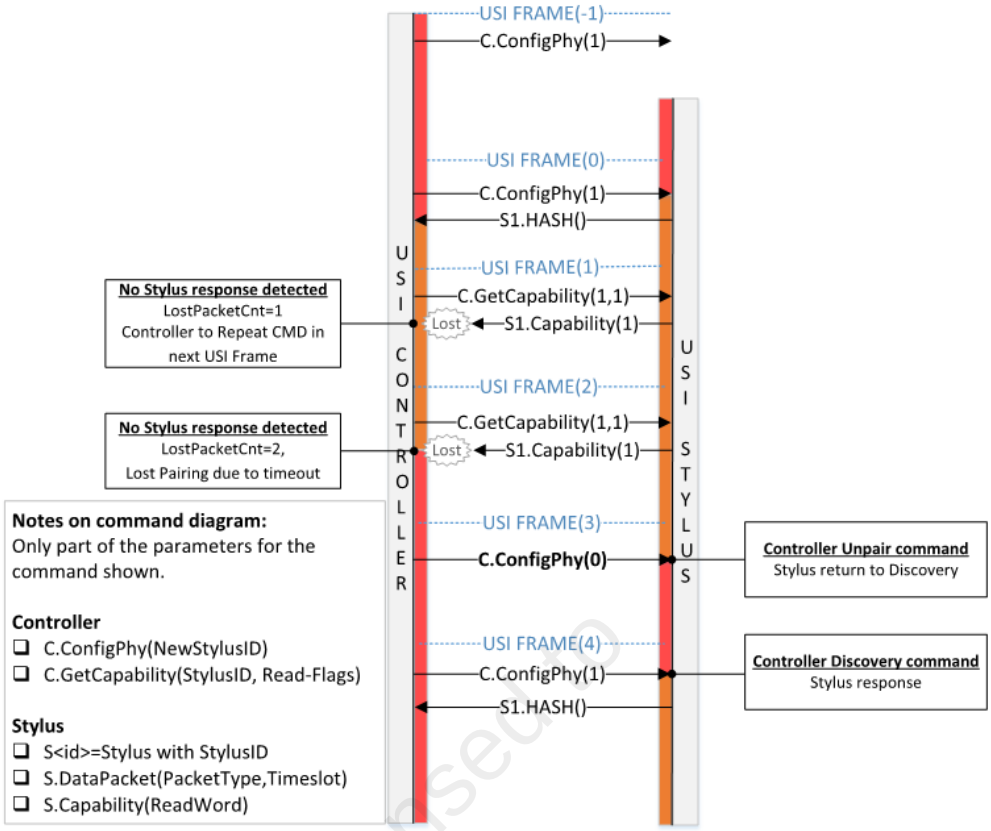


Figure 6-49 Controller Timing Out and the Stylus Being Unpaired (Recommended)

Another situation that could occur is that the stylus loses the controller after pairing has started. The stylus will therefore not respond to any of the controller commands and hence both the stylus and the controller will end up timing out. Figure 6-50 illustrates this.

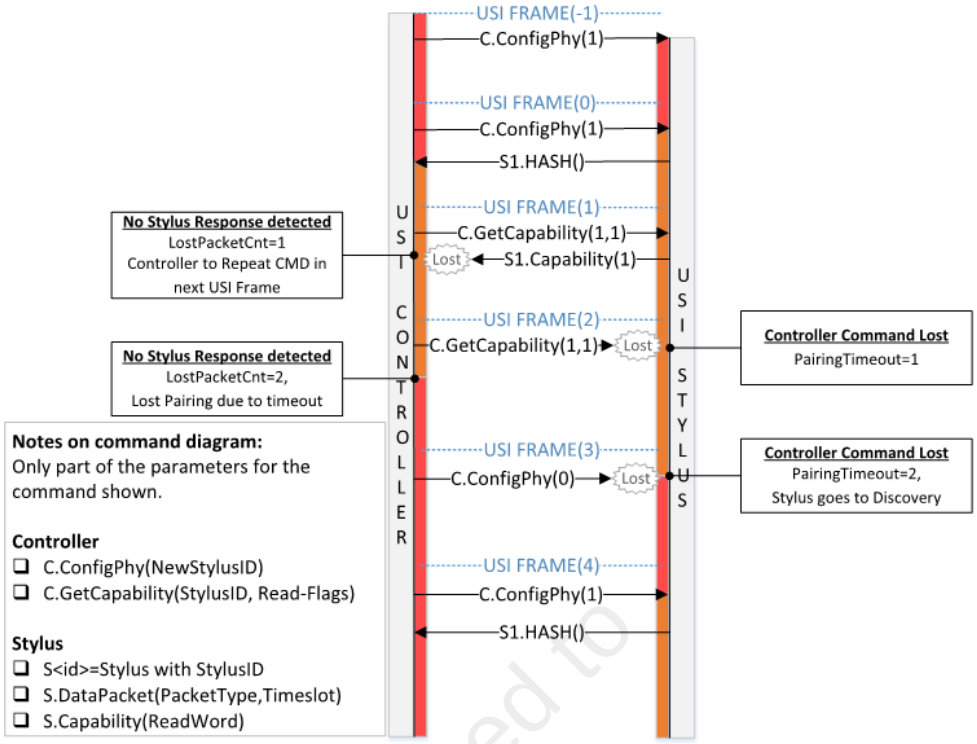


Figure 6-50 Both Controller and Stylus Timing Out When Stylus Packets are Lost

6.9 Operation Flow for Multiple Styluses

The following describes the possible combination of a USI Device and multiple styluses. Each stylus can either be a USI certified stylus or a non-USI Stylus (called a 'proprietary stylus').

6.9.1 Multiple USI Styluses

The USI specification provides for up to six simultaneous USI Styluses to be used with a device. The definition of 'simultaneous' is that the styluses are paired with the controller and either inking or hovering within range of the device at the same time.

It is the responsibility of the controller to manage the communication link between each of the individual styluses and the device and to ensure that there are no conflicts between stylus link parameters such as timeslots or downlink frequency.

Given the nature of the protocol only a single stylus at a time can be in the Pairing state with the controller, and subsequent styluses approaching may have to wait a number of USI frame periods to be discovered and subsequently paired with the controller. The controller shall handle each of the individual stylus links, even though each might be transitioning between the Discovery, Pairing and Operation states at random intervals.

The USI controller may have limited resources (too few timeslots or processing capability) and not be able to handle simultaneous styluses. If this happens, there is a status reporting mechanism (see section 7.3.2) in the host interface to indicate that the maximum number of styluses has been exceeded. This

mechanism is designed to provide information for application level end-user feedback on the capabilities of the USI enabled device. The mechanism relies on the ability of the controller to discover but not pair with a stylus beyond its capabilities.

Limited resources can also be caused by the capabilities of the USI Stylus. A fully featured stylus with vendor specific extensions and a high report rate could use a large proportion of the timeslots. There may not be insufficient slots available for additional styluses to be paired even though the total number of paired styluses does not exceed the controller's other capabilities. This condition can also be reported to the host in the status report.

The USI protocol provides a way for the controller to address either a single stylus or all styluses simultaneously. When multiple styluses are being used, they can all be addressed by using a Protocol command with StylusID field value = 7. This allows for commands such as unpairing all styluses from the device, or moving all styluses to a different downlink frequency to avoid interference.

6.9.2 Support for Proprietary Stylus and Proprietary Controller Implementations

As discussed in section 3.19, the USI specification allows device and stylus manufacturers to implement support for proprietary modes of operation, in addition to fully supporting the specified USI functions. The details of operation in proprietary modes are left to the vendor, except for discovery and switching between USI and proprietary modes.

To support such switching, either the controller or the stylus shall support a dual-mode operation (that is, it supports both the USI standard operation and the proprietary mode operation). The dual-mode implementation can switch to proprietary mode operation if it discovers that the other side is a proprietary implementation.

Of course, this assumes that the proprietary modes on both sides are compatible. For example, if the proprietary mode implementation on the controller is from vendor A and the proprietary mode implementation on the stylus is from vendor B, these implementations may not be compatible.

As described below there are two combinations for USI support of proprietary mode:

1. The USI dual-mode controller works with both the USI Stylus and a proprietary stylus.
2. The USI dual-mode stylus works with both the USI controller and a proprietary controller.

6.9.3 USI Dual-Mode Controller Operation with a Proprietary Stylus

Figure 6-51 describes the operation of a dual-mode controller. During its operation in the Discovery state, the controller looks for the USI Stylus. The controller repeatedly transmits Beacons and checks the ACK slot for a response from a USI Stylus. When the controller detects a USI Stylus response, it transitions to the Pairing state. A dual-mode controller also looks for a proprietary stylus in the USI frame period. It can intersperse this in the time gaps in the USI frame. It is required that the proprietary mode should also have a similar discovery mechanism to detect a proprietary stylus and then transition to its Operation state.

The USI discovery process includes transmitting a Beacon periodically with the time interval ranging from 4 ms to 25 ms. The beacon duration is 1 ms, followed by a turnaround gap of 250 µs and the possible ACK from 250 µs to 1 ms. Therefore a dual mode controller has to reserve between 1.5 ms to 3.25 ms for the USI discovery process. A portion of the remaining time can be used for the discovery of a proprietary stylus.

If the controller is not paired with any stylus, it is only sending the Beacons and waiting for ACKs, and there is no incoming data from any stylus. So if the beacon period is 16 ms, then between 14.5 ms

and12.75 ms is available to the controller to look for a proprietary stylus. The dual mode controller can take as much of this time as is necessary for the proprietary stylus discovery. This proprietary discovery process does increase the controller operation period and its power consumption. The dual mode controller vendor should choose carefully the length of time to detect the proprietary stylus response.

If the dual mode controller is already paired with a USI Stylus, then the whole of the USI frame period might be used for the downlink data slots, so there is no available time for a proprietary stylus to be discovered. Therefore a proprietary stylus cannot be connected to the controller when a single USI Stylus is already in Operation state. For the same reason, a proprietary stylus cannot operate in the USI controller's multiple stylus mode. But, of course, with a dual mode controller other USI Styluses can still be connected and supported simultaneously in a USI multiple stylus configuration.

If the controller is already paired with a proprietary stylus, then it potentially stops looking for a USI Stylus. During proprietary stylus operation the controller does not connect to any USI Styluses. Whether the controller accepts other proprietary styluses is dependent on the vendor implementation, and is out of scope for the USI Technical Specification.

When the user finishes the writing session (that is, when the stylus goes out of range of the device), the controller returns to its Discovery state and again starts looking for either an USI Stylus or a proprietary stylus.

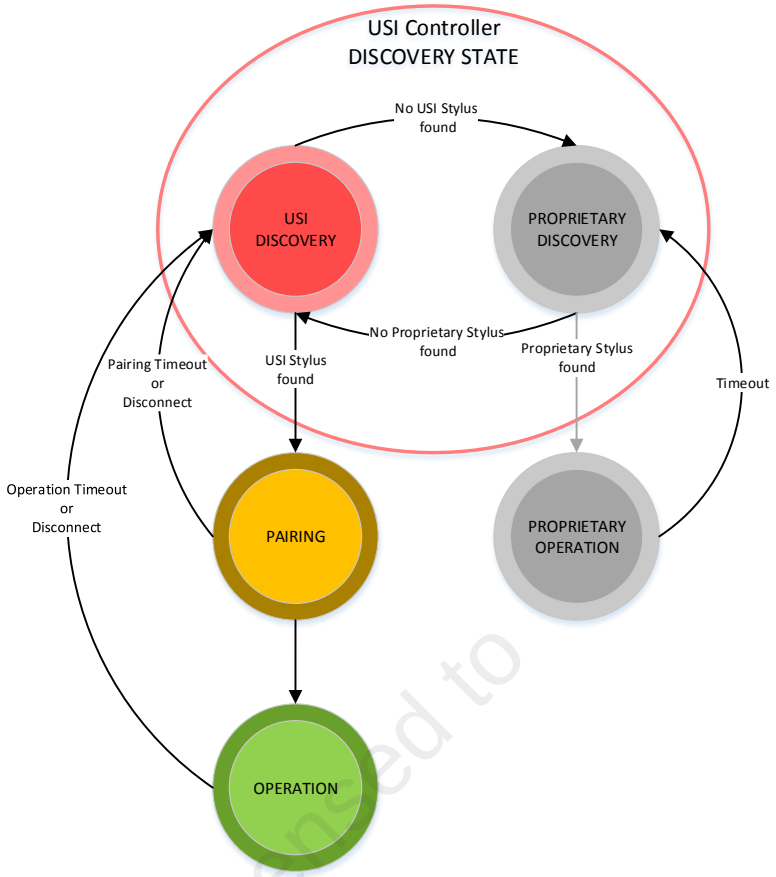


Figure 6-51 USI Dual Mode Controller Discovery State Processes

6.9.4 USI Dual-Mode Stylus Operation with a Proprietary Controller

Usually stylus operation and signaling is completely different for USI Styluses and proprietary styluses. The USI Stylus operates using a beacon signal from the controller, but a proprietary stylus typically operates independently from the controller.

To operate with a proprietary controller, the USI dual-mode stylus needs a mechanism to switch it to Operation state.

Since a USI frame period is 4 ms to 25 ms long, it is ensured that, if the stylus listens for a Beacon for 26 ms, it will receive the Beacon from a USI controller that is in range. Therefore, a dual-mode stylus needs a way to periodically find a maximum of 26 ms window of time to look for a USI controller. The rest of the time it can transmit the signal to operate in the proprietary mode. Since most proprietary mode operations are one-way communications, there might not be enough information available for the stylus to know if it is connected to a proprietary controller or not.

Once the dual-mode stylus receives a Beacon from a USI controller, it can start the pairing process and switch to USI mode. Once in this mode, it stops transmitting the proprietary signals.

§ §

7 Host Communication

7.1 Introduction

Because of its broad availability, the Human Interface Device (HID) specification defined by the Universal Serial Bus (USB) Implementers' Forum should be the software protocol employed between the controller and the host processor.

Only the HID descriptor, and not the device descriptor, is defined in the USI Technical Specification. The device descriptor depends on vendor values and hardware transport details that are determined by the implementation.

The API for applications to use the HID data is defined by the host operating system vendor. This definition is outside of the scope of the USI specification.

Some aspects of the data descriptor might not be supported by some operating systems and/or some applications.

7.2 Dynamic Discovery

HID discovery is static (it occurs only once, at controller device initialization), so HID usages cannot be updated to reflect the capabilities of a stylus when it is paired. To overcome this, the default descriptor shall include a report for all possible HID usages for a USI compliant stylus. This allows a newly discovered stylus to provide a minimum set or a full range of all possible data, depending on the stylus' capabilities. Any HID usages not provided with the currently in-use stylus are filled with default values (as specified below).

7.3 HID Reports

HID reports shall be used for host communications. There are three sets of HID reports - data, status and feature. These are described in detail in the next three sections.

The Report ID is only an example of several possible IDs; it can be changed to another ID, as appropriate. While there is one report ID for data and one for status, there are multiple feature report IDs.

7.3.1 Data Report

All controllers shall be able to report all possible stylus data items. The data report provides a maximal set of stylus data items. This data report shall be used for all stylus data transactions. Usages known by the controller, independent of the current stylus in use (such as coordinates), are fully specified. Usages that are only known after a stylus has been detected are given default ranges and values. It is the responsibility of the controller to scale values coming from the stylus to report a full range of values, and to fill in default values for any missing data in the data reports.

Implementation guidance: operating systems and applications often use data smoothing algorithms which assume that the data reported by a stylus occurs at equal time intervals. If the controller does not measure the stylus coordinates at equal intervals, the controller may have to sub-sample, super-sample, interpolate, extrapolate, or by some other means provide data through the interface at equal time

intervals. Also, due the nature of HID reports there is no simple way to indicate missing values (such as IMU data which may alternate reports). Again, due to smoothing algorithms assuming equally spaced data, repeating a previous value to fill in missing data will not produce the desired result. Therefore it is necessary to fill in missing data with extrapolated values from previous reports. A best effort should be made for initial data (if there are no previous values use zero, if there is only one previous value repeat that value).

7.3.1.1 Coordinates

(Usage(X) and Usage(Y))

Since the controller has complete knowledge of the attached sensor, the Physical Maximum, Unit, and Unit Exponent should be changed to reflect the actual physical size and resolution of the sensor. Also, the Logical Maximum should indicate the true maximum values output by the controller for the sensor. If the Logical Maximum is larger than 65535, the report size shall be increased and padding added (if necessary).

7.3.1.2 Transducer Index

(Usage(Transducer Index))

When multiple styluses are simultaneously tracked, the Transducer Index indicates which stylus produced the report. A unique Transducer Index shall be assigned to a stylus when it is detected and persists until it leaves range. The Transducer Index is arbitrary; its values may be reused.

Note: the stylus ID can be used for the Transducer Index, since it meets the requirements of uniqueness and persistence.

The Logical Maximum of the Transducer Index usage in the HID descriptor will be equal to the maximum number of simultaneous styluses. This still leaves one more index (zero) than the maximum number of styluses so that it is still possible to avoid immediately reusing the same index number.

Note that the Transducer Index usage appears several times throughout the descriptor

7.3.1.3 Tip Pressure

(Usage(Tip Pressure))

The Tip Pressure value output by the stylus shall be a 12 bit unsigned value.

7.3.1.4 Barrel Pressure

(Usage(Barrel Pressure))

The Barrel Pressure value output by the stylus shall be an 8 bit unsigned value. If the stylus does not support barrel pressure, the value reported is set to maximum (255).

Rationale: barrel pressure is used to change a brush dynamic, such as opacity. By setting Barrel Pressure to maximum, the full effect of the brush dynamic is applied.

7.3.1.5 Tip Switch

(Usage(Tip Switch))

The controller, not the stylus, generates the Tip Switch bit in the following manner (rationale: this eliminates the need for the stylus to send the Tip Switch bit):

1. If the tip pressure reported by the stylus is zero, then Tip Switch is set to zero. Otherwise the value of Tip Switch is set to 1.
2. If the stylus Eraser Ready bit is set, Tip Switch is not set (instead the Eraser bit is 1 when the reported tip pressure is non-zero).

7.3.1.6 Barrel Switches

(Usage(Barrel Switch) and Usage(Secondary Barrel Switch))

If one of the barrel switches is dedicated to an eraser affordance (that is, either a tail switch or a barrel button), it is not reported as a barrel switch by the stylus. Instead it is reported as an Eraser Ready bit by the stylus. The secondary barrel switch is a relatively new HID function that might not be supported in the data parser provided by the operating system.

7.3.1.7 Invert

(Usage(Invert))

The stylus provides an Eraser Ready bit. This bit shall be set when the eraser affordance is in use, meaning either that the tail end of the stylus is closest to the surface (if there is an eraser switch and sensor in the tail) or that the barrel button dedicated to erasing has been pressed. The touch controller uses this bit as the Invert bit for HID purposes.

7.3.1.8 Eraser

(Usage(Eraser)):

The controller, not the stylus, handles the following functionality (rationale: It eliminates the need for the stylus to send an Erasing indication):

1. Unless the Eraser Ready bit from the stylus is set, the Erase bit is zero.

2. If the Eraser Ready bit from the stylus is set and the Tip Pressure value reported by the stylus is zero then Eraser is set to zero. Otherwise Eraser is set to 1.

7.3.1.9 In Range

(Usage(In Range)):

The controller generates the In Range bit. A report with the In Range bit set is sent by the controller when a stylus is detected and its capabilities are known. All subsequent reports while receiving data from the stylus shall have In Range set. At least one report with In Range clear shall be sent when the stylus is no longer detected.

7.3.1.10 Orientation

(Usage(X Tilt), Usage(Y Tilt), and Usage(Twist)):

Any orientation values not supported by the stylus shall be reported as zero. Tilt values shall have a physical range from -90 to 90 degrees (measured between the stylus and screen surface), although they may be reported in radians. Twist values shall have a physical range from 0 to 360 degrees, although they may be reported in radians.

The physical and logical ranges shall be reported as two decimal places of resolution for degrees, or four decimal places for radians. If the stylus reports lower resolution values, these ranges shall be scaled to meet this requirement.

7.3.1.11 IMU

Any IMU values not supported by the stylus are reported as zero.

IMU values from the stylus are passed through unmodified except for the required extrapolation (no scaling or axis translations applied).

The IMU uses the HID Sensor Usage Page.

7.3.1.11.1 Accelerometer

(Usage (Acceleration Axis X), Usage (Acceleration Axis Y), and Usage (Acceleration Axis Z))

Accelerometer values from the stylus are placed in the Acceleration usages.

7.3.1.11.2 Gyroscope

(Usage (Angular Velocity X Axis), Usage (Angular Velocity Y Axis), and Usage (Angular Velocity Z Axis))

Gyroscope values from the stylus are placed in the Angular Velocity usages.

7.3.1.11.3 Magnetometer

(Usage (Heading X Axis), Usage (Heading Y Axis), and Usage (Heading Z Axis))

Magnetometer values from the stylus are placed in the Heading usages.

7.3.1.12 Charge Level

(Usage(Battery Strength)):

Charge level shall be reported in percentage of full capacity. The report shall be in the range of 0 to 100 percent. If the Stylus does not support reporting the charge level, and only supports reporting low-battery indications, then this value shall be reported as 1 (1%) when the battery is low and 100 (100%) when the battery is not low.

7.3.1.13 Serial Number

(Usage(Transducer Serial Number)):

The serial number is the 52 bit serial number from the stylus with the 12 bit Vendor ID from the stylus prepended to make a 64 bit value. The concept of serial number is a relatively new HID addition that might not yet be supported in the data parser provided in the operating system.

7.3.1.14 Preferred Color

(Usage(Preferred Color)): (New HID usage)

The preferred color from the stylus capabilities is placed in the Preferred Color. If there is no preferred color from the stylus, the field shall be filled in with 0xFF (no preference). Preferred color is a new HID function.

Refer to Appendix H for information about the PreferredColor index values a stylus can use.

7.3.1.15 Preferred Line Width

Usage(Preferred Line Width)): (New HID usage 0x5E)

The preferred width from the stylus capabilities is placed in the Preferred Line Width. If there is no preferred line width from the stylus, the field shall be filled in with 0xFF (no preference). Preferred Line Width is a new HID usage.

7.3.1.16 Preferred Line Style

(Usage(Preferred Line Style)): (New HID usage 0x70)

The preferred type from the stylus capabilities is placed in the Preferred Line Style. If there is no preferred type from the stylus, the field shall be filled in with (Usage (No preference)) (New HID usage 0x77). Preferred Line Style is a new HID usage.

Styles may be one of the following:

- 1: Ink (Usage (Ink)) (New HID usage 0x72)
- 2: Pencil (Usage (Pencil)) (New HID usage 0x73)
- 3: Highlighter (Usage (Highlighter)) (New HID usage 0x74)
- 4: Chisel Marker (Usage (Chisel Marker)) (New HID usage 0x75)
- 5: Brush (Usage (Brush)) (New HID usage 0x76)
- 6: No Preference (Usage (No Preference)) (New HID usage 0x77

7.3.1.17 Vendor Defined Data and Usage

(Usage Page(Vendor Defined Usage Page 1) Usage(Vendor Usage 2)):

One or two custom 16 bit words received from the stylus are placed into the vendor defined usage at the end of the report. Any unused words shall be set to zero.

7.3.2 Status Report

Usage (Digitizer Error)): (New HID Usage 0x81 on the Digitizer usage page)

All controllers shall include the status report descriptor specified below to indicate error conditions.

It is the responsibility of the controller to send status indications.

The Status Report uses the HID Digitizer Error usage array (New usage 0x81 on the Digitizer usage page). One of the following error usages array index is sent:

- 1: Normal Operation
  (Usage (Err Normal Status)): (New HID Usage 0x82) - operating within specification (there is no requirement to indicate this condition). This might be sent to indicate an error has cleared.

- 2: Too many transducers
  (Usage (Err Transducers Exceeded)): (New HID Usage 0x83) – more styluses are in range than can be supported.
- 3: Unable to support all stylus features
  (**Usage (Err Full Trans Features Unavail)): (New HID Usage 0x84)** – usually this indicates the touch controller was unable to provide enough timeslots to transfer all stylus data and could only support higher priority data.
- 4: The energy source is running low
  (Usage (Err Charge Low)): (New HID Usage 0x85) –this indicates the stylus' battery or super-cap needs to be recharged.
  There is also a Transducer Index usage. This usage is filled in with the Transducer Index of a stylus reporting a low charge. Otherwise this usage is undefined.

7.3.3 Feature Reports

Diagnostic and stylus information is available through a series of HID feature reports.

Report IDs in the sample descriptor are for illustration only and may be changed. Feature reports are discovered thru their HID usages.

HID Get Feature Report and HID Set Feature Report commands are issued to obtain the information.

7.3.3.1.1 Diagnostic Command

(Usage (Digitizer Diagnostic)): (New HID Usage 0x80 on the Digitizer usage page)

The diagnostic command allows a host based test application to issue commands to a stylus and receive the results. The touch controller forwards the command to the stylus and returns the result.

The lower 33 bits of data for a HID SetFeature request are treated as the "CRC, Protocol Command Data, Command ID, StylusID, and Preamble" as shown in Figure 5-6". The remaining bits are ignored. It is the requester's responsibility to compute the CRC data and insert the preamble bits. This allows testing of the CRC and Preamble. The touch controller then sends the resulting command to the styluses.

The response from the stylus and its error status is stored in the touch controller for later retrieval by a HID Get Feature Report request. The lower 16 bits of the response to a HID GetFeature report contains the response from the stylus. The response contains the following:

GetFeature results				
Bits:	**63:23**	**22**	**21:16**	**15:0**
Contents:	Unused/Undefined	EIA	Error	Command Response

- EIA – Error Information Available (An Error code Is Available)
  - 0: There was no error and the error field is undefined
  - 1: The command resulted in an S.REJECT(…) and the reason is in the error field
- Error – the error code if an S.REJECT() was received
- Command Response – the value returned from the stylus in response to the command, if any.

Therefore, executing a command consists of sending a HID Set Feature Report to send the command to the stylus followed by a HID Get Feature Report to retrieve the results.

7.3.3.1.2 Set Transducer Index Selector

Usage (Transducer Index):

The Get features below get a feature for a given transducer. Since HID does not allow any parameters for a get feature, the single byte value of the Set Transducer Index Selector informs the touch controller which transducer index the following get feature applies to. The requester can do the Set Transducer Index Selector to a specific value and can do multiple subsequent Get feature requests. The transducer index in the get feature should be tested to insure it matches the requested index in Set Transducer Index Selector, as another process could potentially issue a Set Transducer Index Selector in the middle of a Set index Get feature sequence.

7.3.3.1.3 Get Stylus Firmware Info

Usage (Transducer Software Info.)): (New HID Usage 0x90)

The Get Stylus Firmware Information command returns the vendor, product, and firmware version information. This can be used to determine when a firmware update is needed.

The command contains a one byte Transducer Index, 16 bits of Vendor ID (Usage (Transducer Vendor ID)) (new HID usage 0x91), 64 bits of Product ID (Usage (Transducer Product ID)) (new HID usage 0x92), and a Software Version report (Usage (Software Version)) (new HID usage 0x2A on the Generic Device usage page) consisting of one byte each of Major (Usage (Major)) (new HID usage 0x2D) and Minor (Usage Minor)) (new HID usage 0x2E) HID usages. Information is filled in for the transducer number given by the Set Transducer Index Selector feature. The Vendor ID is filled in with the USI Vendor-ID from a C.GetGID command. The Product ID is filled in with the remainder of response from the C.GetGID command with the upper 12 bits (USI Vendor-ID) set to zero. The version information is filled in with the results of the C.GetFirmwareVersion command. All but Transducer Index are new HID usages.

7.3.3.1.4 Get Stylus USI Version

Usage (Protocol Version)): (New HID Usage 0 x2B Generic Controls Page)

The Get Stylus USI Version command returns the USI version information.

The command contains a one byte Transducer Index and a Protocol Version report consisting of one byte each of Major (Usage (Major)) (usage 0x2A on the Generic Device usage page) and Minor (Usage Minor)) (usage 0x2B) HID usages which are filled in with the results of the C.GetUSIVersion command for the transducer number indicated by the Set Transducer Index Selector feature. These are new HID usages.

7.3.3.1.5 Get/Set Preferred Color

(Usage (Protocol Version)): (New HID Usage 0x5C Digitizer Page)

The command contains a one byte Transducer Index HID usage, one byte Preferred Color usage (Usage (Preferred Color)) (usage 0x5C), and one bit Preferred Color is Locked usage (Usage Preferred Color is Locked)) (usage 0x5D) with seven upper bits of padding.

On a HID GetFeature the Preferred Color and Preferred Color is Locked values are filled in with the results of the C.GetCapability command for the transducer number indicated by the Set Transducer Index Selector feature.

On a HID SetFeature a C.SetColor command is issued for the transducer number indicated in the Transducer Index usage with the Preferred Color indicated in the Preferred Color usage. The Preferred Color is Locked usage is ignored.

These are new HID usages.

7.3.3.1.6 Get/Set Preferred Line Width

(Usage (Preferred Line Width)): (New HID Usage 0x5E Digitizer Page)

The command contains a one byte Transducer Index HID usage, one byte Preferred Line Width usage (Usage (Preferred Line Width)) (usage 0x5E), and one bit Preferred Line Width is Locked usage (Usage Preferred Line Width is Locked)) (usage 0x5F) with seven upper bits of padding.

On a HID GetFeature the Preferred Line Width and Preferred Line Width is Locked values are filled in with the results of the C.GetCapability command for the transducer number indicated by the Set Transducer Index Selector feature.

On a HID SetFeature a C.SetWidth command is issued for the transducer number indicated in the Transducer Index usage with the Preferred Width indicated in the Preferred Width usage. The Preferred Width is Locked usage is ignored.

These are new HID usages.

7.3.3.1.7 Get/Set Preferred Line Style

(Usage (Preferred Line Style)): (New HID Usage 0x70 Digitizer Page)

The command contains a one byte Transducer Index HID usage, one byte Preferred Line Style usage (Usage (Preferred Line Style)) (usage 0x70), and one bit Preferred Line Style is Locked usage (Usage Preferred Line Style is Locked)) (usage 0x71) with seven upper bits of padding.

On a HID GetFeature the Preferred Line Style and Preferred Line Style is Locked values are filled in with the results of the C.GetCapability command for the transducer number indicated by the Set Transducer Index Selector feature.

On a HID SetFeature a C.SetType command is issued for the transducer number indicated in the Transducer Index usage with the Preferred Style indicated in the Preferred Style usage. The Preferred Style is Locked usage is ignored.

Styles may be one of the following:

- 1: Ink (Usage (Ink)) (New HID usage 0x72)
- 2: Pencil (Usage (Pencil)) (New HID usage 0x73)
- 3: Highlighter (Usage (Highlighter)) (New HID usage 0x74)
- 4: Chisel Marker (Usage (Chisel Marker)) (New HID usage 0x75)
- 5: Brush (Usage (Brush)) (New HID usage 0x76)
- 6: No Preference (Usage (No Preference)) (New HID usage 0x77)

7.3.3.1.8 Get/Set Buttons

(Usage (Barrel Switch))

The command contains a one byte Transducer Index HID usage, and a three-byte array of button definitions. Each array element is tagged with one of the button usages as follows:

- Barrel Switch (Usage 0x44): The barrel switch closest to the stylus tip
- Secondary Barrel Switch (Usage 0x5A): The barrel switch second closest to the stylus tip.
- Eraser (Usage 0x45): The switch on the tail, or a barrel switch dedicated to an erase function.

Each of the three switches can take on one of the following switch function usages:

- Switch Unimplemented (new HID Usage 0xA4): This reports a switch as “missing”, such as the secondary barrel switch for a stylus with only one barrel switch. A switch cannot be Set to Unimplemented.
- Barrel Switch (Usage 0x44): The switch will act as if it were the barrel switch.
- Secondary Barrel Switch (Usage 0x5A): The switch will act as if it is the secondary barrel switch.
- Eraser (Usage 0x45): The switch will provide an Eraser Ready indicator.
- Switch Disabled (new HID Usage 0xA3): The switch exists but will not send any status to the host.

On a HID GetFeature the button values are filled in with the results of the C.GetCapability command for the transducer number indicated by the Set Transducer Index Selector feature.

On a HID SetFeature a C.SetButtons command is issued for the transducer number indicated in the Transducer Index usage and all three buttons are set.

7.3.3.1.9 Get/Set Vendor Extension

(Usage (Vendor Usage 1)): (0x01 Vendor Define Page)

The command contains a one byte Transducer Index HID usage and a one word vendor specific usage.

On a HID GetFeature the word of vendor specific usage is filled in with the results of the C.GetVendorExtension(…) command for the transducer number indicated in the Set Transducer Index Selector feature.

On a HID SetFeature, a C.SetVendorExtension(…) command is issued for the transducer number indicated in the Transducer Index usage with the word vendor specific usage placed in the VendorExtension field

7.4 HID Descriptor for a Data Report

The following is the HID descriptor for the data report that includes all the usages from section 7.3.1. A USI controller shall support this descriptor and send the HID input reports that are conformant with this descriptor.

```
0x05, 0x0d,             // USAGE_PAGE (Digitizers)
0x09, 0x02,             // USAGE (Pen)
0xa1, 0x01,             // COLLECTION (Application)
0x09, 0x20,             //  USAGE (Stylus)
0xa1, 0x00,             //  COLLECTION (Physical)
0x85, HID_REPORTID_TABLET,   //     REPORT_ID ()
0x05, 0x01,             //     USAGE_PAGE (Generic Desktop)
0xa4,                 //     PUSH
0x09, 0x30,             //     USAGE (X)
0x35, 0x00,             //     PHYSICAL_MINIMUM (0)
0x46, 0x3a, 0x20,         //     PHYSICAL_MAXIMUM (8250)
0x15, 0x00,             //     LOGICAL_MINIMUM (0)
0x26, 0xf8, 0x52,         //     LOGICAL_MAXIMUM (21240)
0x55, 0x0d,             //     UNIT_EXPONENT (-3)
0x65, 0x13,             //     UNIT (Eng Lin:Distance)
```

```
0x75, 0x10,             //    REPORT_SIZE (16)
0x95, 0x01,             //    REPORT_COUNT (1)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x09, 0x31,             //    USAGE (Y)
0x46, 0x2c, 0x18,       //    PHYSICAL_MAXIMUM (6188)
0x26, 0x6c, 0x3e,       //    LOGICAL_MAXIMUM (15980)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0xb4,               //    POP
0x05, 0x0d,             //    USAGE_PAGE (Digitizers)
0x09, 0x38,             //    USAGE (Transducer Index)
0x95, 0x01,             //    REPORT_COUNT (1)
0x75, 0x08,             //    REPORT_SIZE (8)
0x15, 0x00,             //    LOGICAL_MINIMUM (0)
0x25, MAX_SUPPORTED_STYLI,  //    LOGICAL_MAXIMUM (2)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x09, 0x30,             //    USAGE (Tip Pressure)
0x75, 0x10,             //    REPORT_SIZE (16)
0x26, 0xff, 0x0f,       //    LOGICAL_MAXIMUM (4095)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x09, 0x31,             //    USAGE (Barrel Pressure)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x09, 0x42,             //    USAGE (Tip Switch)
0x09, 0x44,             //    USAGE (Barrel Switch)
0x09, 0x5a,             //    USAGE (Secondary Barrel Switch)
0x09, 0x3c,             //    USAGE (Invert)
0x09, 0x45,             //    USAGE (Eraser)
0x09, 0x32,             //    USAGE (In Range)
0x75, 0x01,             //    REPORT_SIZE (1)
0x95, 0x06,             //    REPORT_COUNT (6)
0x25, 0x01,             //    LOGICAL_MAXIMUM (1)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x95, 0x02,             //    REPORT_COUNT (2)
0x81, 0x03,             //    INPUT (Cnst,Var,Abs)
0x09, 0x3d,             //    USAGE (X Tilt)
0x55, 0x0e,             //    UNIT_EXPONENT (-2)
0x65, 0x14,             //    UNIT (Eng Rot:Angular Pos)
0x36, 0xd8, 0xdc,       //    PHYSICAL_MINIMUM (-9000)
0x46, 0x28, 0x23,       //    PHYSICAL_MAXIMUM (9000)
0x16, 0xd8, 0xdc,       //    LOGICAL_MINIMUM (-9000)
0x26, 0x28, 0x23,       //    LOGICAL_MAXIMUM (9000)
0x95, 0x01,             //    REPORT_COUNT (1)
0x75, 0x10,             //    REPORT_SIZE (16)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x09, 0x3e,             //    USAGE (Y Tilt)
0x81, 0x02,             //    INPUT (Data,Var,Abs)
0x09, 0x41,             //    USAGE (Twist)
0x15, 0x00,             //    LOGICAL_MINIMUM (0)
0x27, 0xa0, 0x8c, 0x00, 0x00,  //    LOGICAL_MAXIMUM (36000)
0x35, 0x00,             //    PHYSICAL_MINIMUM (0)
0x47, 0xa0, 0x8c, 0x00, 0x00,  //    PHYSICAL_MAXIMUM (36000)
```

```
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x05, 0x20,          //    USAGE_PAGE (Sensors)
0x0a, 0x53, 0x04,       //    USAGE (Data Field: Acceleration Axis X)
0x65, 0x00,          //    UNIT (None)
0x16, 0x01, 0xf8,       //    LOGICAL_MINIMUM (-2047)
0x26, 0xff, 0x07,       //    LOGICAL_MAXIMUM (2047)
0x75, 0x10,          //    REPORT_SIZE (16)
0x95, 0x01,          //    REPORT_COUNT (1)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x54, 0x04,       //    USAGE (Data Field: Acceleration Axis Y)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x55, 0x04,       //    USAGE (Data Field: Acceleration Axis Z)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x57, 0x04,       //    USAGE (Data Field: Angular Velocity Axis X)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x58, 0x04,       //    USAGE (Data Field: Angular Velocity Axis Y)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x59, 0x04,       //    USAGE (Data Field: Angular Velocity Axis Z)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x72, 0x04,       //    USAGE (Data Field: Heading X Axis)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x73, 0x04,       //    USAGE (Data Field: Heading Y Axis)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x0a, 0x74, 0x04,       //    USAGE (Data Field: Heading Z Axis)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x05, 0x0d,          //    USAGE_PAGE (Digitizers)
0x09, 0x3b,          //    USAGE (Battery Strength)
0x15, 0x00,          //    LOGICAL_MINIMUM (0)
0x25, 0x64,          //    LOGICAL_MAXIMUM (100)
0x75, 0x08,          //    REPORT_SIZE (8)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x09, 0x5b,          //    USAGE (Transducer Serial Number)
0x25, 0xff,          //    LOGICAL_MAXIMUM (-1)
0x75, 0x40,          //    REPORT_SIZE (64)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x06, 0x00, 0xff,       //    USAGE_PAGE (Vendor 1)
0x09, 0x5b,          //    USAGE (Transducer Serial Number)
0x75, 0x20,          //    REPORT_SIZE (32)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x05, 0x0d,          //    USAGE_PAGE (Digitizers)
0x09, 0x5c,          //    USAGE (Preferred Color)
0x26, 0xff, 0x00,       //    LOGICAL_MAXIMUM (255)
0x75, 0x08,          //    REPORT_SIZE (8)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x09, 0x5e,          //    USAGE (Preferred Line Width)
0x81, 0x02,          //    INPUT (Data,Var,Abs)
0x09, 0x70,          //    USAGE (Preferred Line Style)
0xa1, 0x02,          //    COLLECTION (Logical)
0x15, 0x01,          //      LOGICAL_MINIMUM (0)
0x25, 0x06,          //      LOGICAL_MAXIMUM (5)
```

```
0x09, 0x72,             //      USAGE (Ink)
0x09, 0x73,             //      USAGE (Pencil)
0x09, 0x74,             //      USAGE (Highlighter)
0x09, 0x75,             //      USAGE (Chisel Marker)
0x09, 0x76,             //      USAGE (Brush)
0x09, 0x77,             //      USAGE (No preference)
0x81, 0x20,             //      INPUT (Data,Ary,Abs,NPrf)
0xc0,               //    END_COLLECTION
0x06, 0x00, 0xff,       //     USAGE_PAGE (Generic Desktop)
0x09, 0x01,             //     USAGE (Vendor Usage 1)
0x15, 0x00,             //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
0x75, 0x10,             //     REPORT_SIZE (16)
0x95, 0x01,             //     REPORT_COUNT (1)
0x81, 0x02,             //     INPUT (Data,Var,Abs)
0xc0,               //    END_COLLECTION
```

7.5 HID Descriptor for Status Reports

The following is the HID descriptor for the status report that A USI controller shall support for reporting status and error conditions.

```
0x05, 0x0d,             //    USAGE_PAGE (Digitizers)
0x85, HID_REPORTID_ERROR,    //    REPORT_ID ()
0x09, 0x81,             //    USAGE (Digitizer Error)
0xa1, 0x02,             //    COLLECTION (Logical)
0x09, 0x38,             //     USAGE (Transducer Index)
0x75, 0x08,             //     REPORT_SIZE (8)
0x95, 0x01,             //     REPORT_COUNT (1)
0x15, 0x00,             //     LOGICAL_MINIMUM (0)
0x25, MAX_SUPPORTED_STYLI,    //     LOGICAL_MAXIMUM (2)
0x81, 0x02,             //     INPUT (Data,Var,Abs)
0x09, 0x81,             //     USAGE (Digitizer Error)
0x15, 0x01,             //     LOGICAL_MINIMUM (0)
0x25, 0x04,             //     LOGICAL_MAXIMUM (3)
0x09, 0x82,             //     USAGE (Err Normal Status)
0x09, 0x83,             //     USAGE (Err Transducers Exceeded)
0x09, 0x84,             //     USAGE (Err Full Trans Features Unavail)
0x09, 0x85,             //     USAGE (Err Charge Low)
0x81, 0x20,             //     INPUT (Data,Ary,Abs,NPrf)
0xc0,               //    END_COLLECTION
```

7.6 HID Descriptor for Feature Reports

Following is the HID descriptor for the Get/Set Feature Reports.

```
// Get/Set Line Color
0x85, HID_REPORTID_GETSET_COLOR,//    REPORT_ID ()
0x09, 0x5c,             //    USAGE (Preferred Color)
0xa1, 0x02,             //    COLLECTION (Logical)
```

```
0x15, 0x00,             //    LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,       //    LOGICAL_MAXIMUM (2)
0x75, 0x08,             //    REPORT_SIZE (8)
0x95, 0x01,             //    REPORT_COUNT (1)
0x09, 0x38,             //    USAGE (Transducer Index)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x5c,             //    USAGE (Preferred Color)
0x26, 0xff, 0x00,       //    LOGICAL_MAXIMUM (255)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x5d,             //    USAGE (Preferred Color is Locked)
0x75, 0x01,             //    REPORT_SIZE (1)
0x95, 0x01,             //    REPORT_COUNT (1)
0x25, 0x01,             //    LOGICAL_MAXIMUM (1)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x95, 0x07,             //    REPORT_COUNT (7)
0xb1, 0x03,             //    FEATURE (Cnst,Var,Abs)
0xc0,               //  END_COLLECTION

// Get/Set Line Width
0x85, HID_REPORTID_GETSET_WIDTH,//  REPORT_ID ()
0x09, 0x5e,             //  USAGE (Preferred Line Width)
0xa1, 0x02,             //  COLLECTION (Logical)
0x09, 0x38,             //    USAGE (Transducer Index)
0x15, 0x00,             //    LOGICAL_MINIMUM (0)
0x25, MAX_SUPPORTED_STYLI,  //    LOGICAL_MAXIMUM (2)
0x75, 0x08,             //    REPORT_SIZE (8)
0x95, 0x01,             //    REPORT_COUNT (1)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x5e,             //    USAGE (Preferred Line Width)
0x26, 0xff, 0x00,       //    LOGICAL_MAXIMUM (255)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x5f,             //    USAGE (Preferred Line Width is Locked)
0x75, 0x01,             //    REPORT_SIZE (1)
0x25, 0x01,             //    LOGICAL_MAXIMUM (1)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x75, 0x07,             //    REPORT_SIZE (7)
0xb1, 0x03,             //    FEATURE (Cnst,Var,Abs)
0xc0,               //  END_COLLECTION

// Get/Set Line Style
0x85, HID_REPORTID_GETSET_STYLE,//  REPORT_ID ()
0x09, 0x70,             //  USAGE (Preferred Line Style)
0xa1, 0x02,             //  COLLECTION (Logical)
0x75, 0x08,             //    REPORT_SIZE (8)
0x95, 0x01,             //    REPORT_COUNT (1)
0x15, 0x00,             //    LOGICAL_MINIMUM (0)
0x25, MAX_SUPPORTED_STYLI,  //    LOGICAL_MAXIMUM (2)
0x09, 0x38,             //    USAGE (Transducer Index)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x70,             //    USAGE (Preferred Line Style)
```

```
0xa1, 0x02,             //    COLLECTION (Logical)
0x25, 0x06,             //      LOGICAL_MAXIMUM (5)
0x09, 0x72,             //      USAGE (Ink)
0x09, 0x73,             //      USAGE (Pencil)
0x09, 0x74,             //      USAGE (Highlighter)
0x09, 0x75,             //      USAGE (Chisel Marker)
0x09, 0x76,             //      USAGE (Brush)
0x09, 0x77,             //      USAGE (No preference)
0xb1, 0x20,             //      FEATURE (Data,Ary,Abs,NPrf)
0xc0,               //    END_COLLECTION
0x09, 0x71,             //    USAGE (Preferred Line Style is Locked)
0x75, 0x01,             //    REPORT_SIZE (1)
0x25, 0x01,             //    LOGICAL_MAXIMUM (1)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x75, 0x07,             //    REPORT_SIZE (7)
0xb1, 0x03,             //    FEATURE (Cnst,Var,Abs)
0xc0,               //  END_COLLECTION

// Get/Set Diagnostic
0x85, HID_REPORTID_DIAGNOSE,      //  REPORT_ID ()
0x09, 0x80,             //  USAGE (Digitizer Diagnostic)
0x15, 0x00,             //  LOGICAL_MINIMUM (0)
0x25, 0xff,             //  LOGICAL_MAXIMUM (-1)
0x75, 0x40,             //  REPORT_SIZE (64)
0x95, 0x01,             //  REPORT_COUNT (1)
0xb1, 0x02,             //  FEATURE (Data,Var,Abs)

// Get/Set Buttons
0x85, HID_REPORTID_GETSET_BUTTONS,//  REPORT_ID ()
0x09, 0x44,             //  USAGE (Barrel Switch)
0xa1, 0x02,             //  COLLECTION (Logical)
0x09, 0x38,             //    USAGE (Transducer Index)
0x75, 0x08,             //    REPORT_SIZE (8)
0x95, 0x01,             //    REPORT_COUNT (1)
0x25, MAX_SUPPORTED_STYLI,    //    LOGICAL_MAXIMUM (2)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x15, 0x01,             //    LOGICAL_MINIMUM (0)
0x25, 0x03,             //    LOGICAL_MAXIMUM (5)
0x09, 0x44,             //    USAGE (Barrel Switch)
0xa1, 0x02,             //    COLLECTION (Logical)
0x09, 0xa4,             //      USAGE (Switch Unimplemented)
0x09, 0x44,             //      USAGE (Barrel Switch)
0x09, 0x5a,             //      USAGE (Secondary Barrel Switch)
0x09, 0x45,             //      USAGE (Eraser)
0x09, 0xa3,             //      USAGE (Switch Disabled)
0xb1, 0x20,             //      FEATURE (Data,Ary,Abs,NPrf)
0xc0,               //    END_COLLECTION
0x09, 0x5a,             //    USAGE (Secondary Barrel Switch)
0xa1, 0x02,             //    COLLECTION (Logical)
0x09, 0xa4,             //      USAGE (Switch Unimplemented)
```

```
0x09, 0x44,             //      USAGE (Barrel Switch)
0x09, 0x5a,             //      USAGE (Secondary Barrel Switch)
0x09, 0x45,             //      USAGE (Eraser)
0x09, 0xa3,             //      USAGE (Switch Disabled)
0xb1, 0x20,             //      FEATURE (Data,Ary,Abs,NPrf)
0xc0,               //    END_COLLECTION
0x09, 0x45,             //    USAGE (Eraser)
0xa1, 0x02,             //    COLLECTION (Logical)
0x09, 0xa4,             //      USAGE (Switch Unimplemented)
0x09, 0x44,             //      USAGE (Barrel Switch)
0x09, 0x5a,             //      USAGE (Secondary Barrel Switch)
0x09, 0x45,             //      USAGE (Eraser)
0x09, 0xa3,             //      USAGE (Switch Disabled)
0xb1, 0x20,             //      FEATURE (Data,Ary,Abs,NPrf)
0xc0,               //    END_COLLECTION
0xc0,               //  END_COLLECTION

// Get Firmware Version
0x85, HID_REPORTID_GET_FIRMWARE,//  REPORT_ID ()
0x75, 0x08,             //  REPORT_SIZE (8)
0x95, 0x01,             //  REPORT_COUNT (1)
0x05, 0x0d,             //  USAGE_PAGE (Digitizers)
0x09, 0x90,             //  USAGE (Transducer Software Info.)
0xa1, 0x02,             //  COLLECTION (Logical)
0x09, 0x38,             //    USAGE (Transducer Index)
0x25, MAX_SUPPORTED_STYLI,  //    LOGICAL_MAXIMUM (2)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x91,             //    USAGE (Transducer Vendor ID)
0x75, 0x10,             //    REPORT_SIZE (16)
0x26, 0xff, 0x0f,       //    LOGICAL_MAXIMUM (4095)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x09, 0x92,             //    USAGE (Transducer Product ID)
0x75, 0x40,             //    REPORT_SIZE (64)
0x25, 0xff,             //    LOGICAL_MAXIMUM (-1)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)
0x05, 0x06,             //    USAGE_PAGE (Generic Device)
0x09, 0x2a,             //    USAGE (Software Version)
0x75, 0x08,             //    REPORT_SIZE (8)
0x26, 0xff, 0x00,       //    LOGICAL_MAXIMUM (255)
0xa1, 0x02,             //    COLLECTION (Logical)
0x09, 0x2d,             //      USAGE (Major)
0xb1, 0x02,             //      FEATURE (Data,Var,Abs)
0x09, 0x2e,             //      USAGE (Minor)
0xb1, 0x02,             //      FEATURE (Data,Var,Abs)
0xc0,               //    END_COLLECTION
0xc0,               //  END_COLLECTION

// Get USI Version
0x85, HID_REPORTID_GET_PROTOCOL,//  REPORT_ID ()
0x05, 0x06,             //  USAGE_PAGE (Generic Device)
```

```
0x09, 0x2b,             //    USAGE (Protocol Version)
0xa1, 0x02,             //    COLLECTION (Logical)
0x05, 0x0d,             //     USAGE_PAGE (Digitizers)
0x25, MAX_SUPPORTED_STYLI,    //     LOGICAL_MAXIMUM (2)
0x09, 0x38,             //     USAGE (Transducer Index)
0xb1, 0x02,             //     FEATURE (Data,Var,Abs)
0x05, 0x06,             //     USAGE_PAGE (Generic Device)
0x09, 0x2b,             //     USAGE (Protocol Version)
0xa1, 0x02,             //     COLLECTION (Logical)
0x09, 0x2d,             //      USAGE (Major)
0x26, 0xff, 0x00,         //      LOGICAL_MAXIMUM (255)
0xb1, 0x02,             //      FEATURE (Data,Var,Abs)
0x09, 0x2e,             //      USAGE (Minor)
0xb1, 0x02,             //      FEATURE (Data,Var,Abs)
0xc0,                 //     END_COLLECTION
0xc0,                 //    END_COLLECTION

// Get/Set Vendor Specific
0x85, HID_REPORTID_GETSET_VENDOR,//  REPORT_ID ()
0x06, 0x00, 0xff,         //    USAGE_PAGE (Generic Desktop)
0x09, 0x01,             //    USAGE (Vendor Usage 1)
0xa1, 0x02,             //    COLLECTION (Logical)
0x05, 0x0d,             //     USAGE_PAGE (Digitizers)
0x09, 0x38,             //     USAGE (Transducer Index)
0x75, 0x08,             //     REPORT_SIZE (8)
0x95, 0x01,             //     REPORT_COUNT (1)
0x25, MAX_SUPPORTED_STYLI,    //     LOGICAL_MAXIMUM (2)
0xb1, 0x02,             //     FEATURE (Data,Var,Abs)
0x06, 0x00, 0xff,         //      USAGE_PAGE (Generic Desktop)
0x09, 0x01,             //     USAGE (Vendor Usage 1)
0x75, 0x10,             //     REPORT_SIZE (16)
0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
0xb1, 0x02,             //     FEATURE (Data,Var,Abs)
0xc0,                 //    END_COLLECTION

// Set Select Transducer Index
0x85, HID_REPORTID_SET_TRANSDUCER,//  REPORT_ID ()
0x05, 0x0d,             //    USAGE_PAGE (Digitizers)
0x09, 0x38,             //    USAGE (Transducer Index)
0x75, 0x08,             //    REPORT_SIZE (8)
0x95, 0x01,             //    REPORT_COUNT (1)
0x15, 0x00,             //    LOGICAL_MINIMUM (0)
0x25, MAX_SUPPORTED_STYLI,    //    LOGICAL_MAXIMUM (2)
0xb1, 0x02,             //    FEATURE (Data,Var,Abs)

0xc0                  // END_COLLECTION
```

§ §

8 Power Management (Informative)

8.1 USI Stylus Power Management

Because the stylus has limited power storage, power consumption is a critical issue in its implementation. This chapter provides a power management reference model as an informative guideline for USI Stylus implementers.

8.2 USI Stylus States and Power Modes

Power management in the USI Stylus is dependent on what it needs to do in different operating states (Discovery, Pairing and Operation). When it is in a different operating state, the stylus' power consumption requirements and limitations are different. Figure 8-1 contains a flow diagram of the stylus' operating states. Power usage guidance is added beside each state description to suggest a power usage model for each of the three critical modules/components of the USI Stylus – transmitter (Tx), receiver (Rx), and micro control unit (MCU), as shown in Figure 8-2.

The USI Stylus Tx is designed specifically for transmitting signals from the stylus to the controller. These signals provide information that has to be received and processed by the controller, such as pressure information and button information. The stylus Rx is designed for receiving uplink Beacon signals from the controller. The MCU is designed to manage the interactions between the stylus' Tx and Rx, the calculation and detection of pressure information from the force sensor, and the stylus' operations during its power saving modes.

Each individual module/component of the stylus can be in any of the three basic power consumption modes – active, standby, and off.

For the Tx:

- Active: the Tx is currently transmitting a signal.
- Standby: the Tx is ready to transmit a signal but no signal is currently being transmitted.
- Off: the Tx has no power.

For the Rx:

- Active: the Rx is currently receiving a signal.
- Standby: the Rx is ready to receive a signal, but no signal is currently being received.
- Off: the Rx has no power.

For the MCU:

- Active: MCU is managing the interactions between the stylus' Tx and Rx, and/or is detecting and calculating pressure information from the force sensor, and/or is conducting limited operations during the stylus' power saving modes.
- Standby: the MCU is only responsible for managing the Rx so that it is ready to receive signals from the controller.
- Off: the MCU is not doing anything and is in its lowest power state. It may be completely powered off or have just enough power to be able to wake when needed. This choice is implementation dependent.

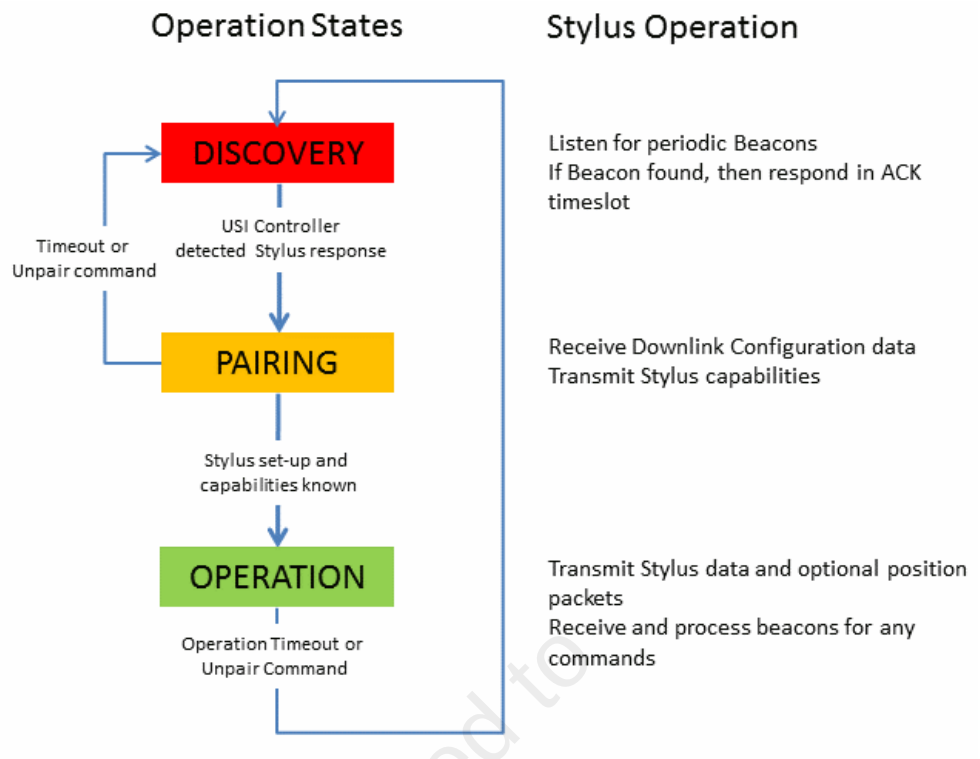


Figure 8-1 Flow Diagram of Stylus Operating States

8.2.1 Discovery State

When the USI Stylus is in the Discovery state, the MCU starts in standby ('Stby') mode and the MCU commands the Rx to be in standby mode as well. An implementation may keep the USI Stylus Rx in off mode most of the time, and move into standby mode periodically to check for Beacons.

When it is checking for a Beacon, the Rx shall be in standby mode for 26 ms to make sure that it has waited for a possible maximum USI frame period. It may then go into off mode for some time. The exact length of time the Rx stays in off mode depends on the implementation. For example, some implementations may keep the Rx in off mode until the user picks up the stylus and a motion is detected. Other implementations may simply adhere to a 50% duty cycle by alternating the Rx between standby mode for 25 ms and off mode for 25 ms. The power and latency trade-offs are critical factors for the designers of each stylus implementation.

When the Rx detects the first edge of a Beacon, the Rx changes into active mode and starts receiving the beacon data. Likewise, the MCU moves into active mode to command the Tx to change to standby mode and be ready for transmitting an ACK. Finally, the Tx transitions into active mode to transmit the ACK to the controller. At this moment, the MCU, Rx, and Tx are all in active mode and ready to receive, transmit, and manage the interactions between the controller and the USI Stylus. An implementation can do further optimization by turning off Rx while sending data on the downlink and turning off Tx while receiving the Beacon. Thus only Rx or Tx is active at any point of time.

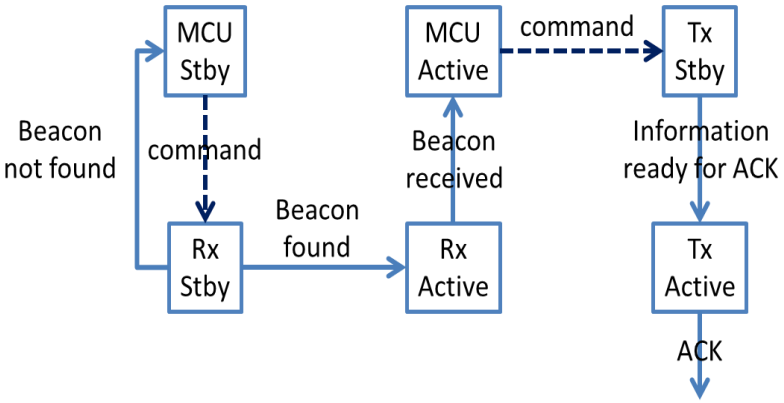


Figure 8-2 Power Modes in the USI Stylus MCU, Rx and Tx (Discovery)

8.2.2 Pairing State

When the stylus is in the Pairing state, the MCU, Rx, and Tx are all in active mode and ready to receive, transmit and manage the interactions between the controller and the USI Stylus. Based on the time slot assigned for mutual communication, the modes of the Rx, Tx and MCU can be changed back and forth between standby mode and active mode to optimize power saving, as Figure 8-3 shows. During the assigned time slot the MCU is in active mode and commands the Tx and Rx to be in active mode. The length of time that the Tx or Rx is in active mode depends on the commands they receive from the MCU. Other than during the assigned time slot, the MCU, Rx and Tx shall be in standby mode.

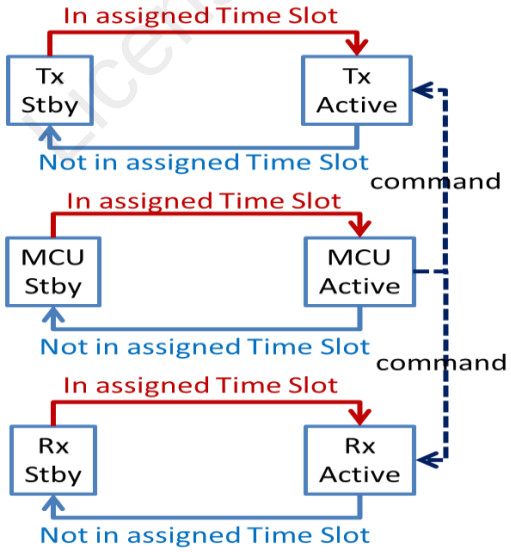


Figure 8-3 USI Stylus Modes Alternate in the MCU, Rx and Tx

8.2.3 Operation State

Similar to the actions while the stylus is in the Pairing state, when the USI Stylus is in the Operation state the MCU, Rx, and Tx may alternate between standby mode and active mode to optimize power savings.

The states may change, of course, according to the presence of the time slot that was assigned for communication with the controller. See Figure 8-3.

8.2.4 Power Off State (Optional)

If there is no further operation within a predefined period during the unpaired state, the USI Stylus may be completely powered off. When the end user wants to use the USI Stylus, a user-trigger event is needed to power the stylus on. Typical triggers include the user pushing a button on the stylus or putting a slight pressure on the stylus tip.

Appendix A – Definitions, Acronyms, Abbreviations

A.1 Definitions

active stylus	A stylus that contains electronics and a power source (such as a battery or parasitic energy conduit). The active stylus transmits signals to the sensors on the device; these signals result in precisely locating the stylus' position on the device's screen. In addition, the stylus transmits data such as pressure and button information.
all-in-one	A large screen (typically larger than 25 inch) computer that includes all of the functionality of a personal computer (processor, memory, communications, display, etc.) in a single unit. When the all-in-one is placed flat on a table or mounted on a wall (similar to a white board), multi-user touch and multiple stylus input become critical to its design functionality. A portable version of the all-in-one is battery powered for transportation to multiple locations.
barrel pressure	Some styluses can sense pressure placed on the body of a stylus (the barrel) and report that information to the controller. The collection of this pressure information may be triggered by the user pressing a button or holding his fingers on the stylus' body.
Beacon	A periodic signal sent by the USI controller to allow all USI Styluses to discover the existence of the controller. A USI Stylus listens for Beacons and, after it receives a Beacon, it can set its timing reference, start the pairing process and begin communications with the controller.
button	A stylus typically has one or more buttons. These can be used to convey information to the application that the user wants to do something special, such as launching a menu to pick a new color for the stylus touch on the controller.
controller	A microcontroller that is a component of a device and receives sensor signals from the device's touch screen. The sensor signals contain both touch and stylus information. The controller is the master in the USI control protocol.
device	A computing system that has a touch screen. This is a product (such as a phone, tablet, laptop, PC, all-in-one, etc.) that can be used with an active stylus.
digitizer	A sensor grid that acquires stylus data and sends it to the controller. The digitizer might also be referred to as 'sensor' or 'sensor panel'.
disconnect	A state in which the communications between a stylus and a touch screen are in the process of being disabled.
discovery	The detection and identification of a USI Stylus performed by a USI controller. (See the definition of 'Beacon'.) Also: the state in a USI Stylus in which the discovery process takes place.
downlink	A means of communication (link) used by the USI Stylus to transfer information 'down' to the USI controller.

dwell time	An average minimum time that a stylus remains in contact with the screen after it first touches the screen for tapping.
eraser affordance	Mechanism on a stylus that is used to trigger an eraser function. Either a tail switch or a stylus barrel button.
frame	A frame is the set of fields and packets that transmitted between the start of one Beacon the start of the next Beacon. The frame includes the uplink Beacon, downlink ACK and the downlink data.
host processor	The main processing component of the device. Typically this runs the device's operating system and the touch and stylus information are sent to this device by the controller.
inking	The visual show of digital 'ink' (visual marks) on a display. For example, a writing application shows the characters input by a writer, and a drawing application shows shapes and colors drawn by an artist.
link	A link is the communication channel between the USI Stylus and the USI controller. The link is a two way communication that consists of both the uplink and the downlink.
mode	Or 'operation mode'. A form of operation. There are two types of modes: (a) Compliant with the USI Compliance Test Plan: USI compliant, proprietary or dual mode (which includes both USI compliant and proprietary operations) (b) Number of styluses connected to the controller: single stylus mode, multiple stylus mode.
multiple devices	The term 'multiple devices' refers to many devices that the user may want to use with the USI Stylus. These might include smartphones, tablets, laptops, PCs, all-in-ones, etc.
multiple styluses	A protocol controller can talk to multiple styluses concurrently. This is referred to as a 'multiple stylus' configuration.
operating system	The firmware/software that is running on the host processor and that uses the processed stylus and touch data to enable applications to interact with the user.
packet	The response from a stylus (acting as a slave to the controller) that is formed by combining one or more timeslots.
pairing	The process of connecting a USI Stylus to the USI controller. This includes the process of discovering the capabilities, and exchange configuration parameters to make the stylus ready to start inking.
passive stylus	A stylus that does not contain electronics, a power source or reactive components. It may be possible to support pressure information and a thin tip with a passive stylus, but the passive stylus cannot provide the quality information provided by an active stylus.
pressure	Force. See 'tip pressure' and 'barrel pressure'
proprietary stylus	A stylus that does not comply with the USI Compliance Test Plan, but which might still operate with a USI compliant device.
session	A session between a USI controller and a USI Stylus is in progress when both are in either Discovery or Operation state. If either the controller or stylus disconnects and returns to the Discovery state, the session is considered to have ended.
signature	An electronic signature in the writer's handwriting using the stylus as the writing device.
slot	A 250 µs period of time in the USI communications protocol. This is the smallest unit of time allocated for the USI Stylus to send its information to the controller. Also see 'timeslot'.

stylus	An instrument (frequently pen-like) designed to work with a touch capable device, much as a pen works with paper. A stylus facilitates writing, drawing or pointing. In the USI control protocol the stylus is the slave to the controller acting as master.
tilt	Information about how the stylus is oriented with respect to the touch surface of the device.
timeslot	One of a number of evenly divided intervals of the time period between uplink Beacons. Also see 'slot'.
time to contact	The length of the interval from the time that a stylus is in its resting position (e.g., a pen in a user's hand while the user is thinking) to the time the stylus tip touches the device screen.
tip	The stylus' physical point or surface that contacts the touch screen.
tip pressure	Force on the tip of a stylus that can be sensed by the stylus and reported to the controller. This force indicates how the user is pressing the stylus against the touch screen.
touch	A process that is caused by either finger or stylus contact with a 'touch-sensitive' screen.
touch capable device	A product that incorporates a touch interface. Examples include smartphones, tablets, notebook computers, etc.
touch input	Finger touch input for sensors.
touch screen	The part of a device that is sensitive to touch. Note: the touch screen might or might not be a display; it might be a surface that provides no graphical information to the user, but still is used for the device's direct communication with a stylus.
transducer index	Index of the stylus reported to the operating system. This index is important for distinguishing styluses when multiple styluses are connected to a device
unpaired	A USI Stylus that is not yet connected to a device and currently is in its Discovery state.
uplink	A means of communication (link) on which the USI controller sends information and control commands 'up' to a USI Stylus.
USI controller	A controller in a device that is compliant with the USI Compliance Test Plan. A USI Stylus always works on a device that has a USI controller.
USI Stylus	A stylus that is compliant with the USI Compliance Test Plan. A USI Stylus always works on a device that has a USI controller.
writing	The act of a stylus operating on a touch capable device that is running a text application. Also see 'inking'.

A.2 Acronyms and Abbreviations

ACK	acknowledgement
ANOVA	analysis of variance
CRC	cyclic redundancy check
D-BPSK	differential binary phase shift keying
dpi	(screen) dots per inch
DSSS	direct sequence spread spectrum
F, pF, fF	farads, pico farads, femto farads
fC, pC	femto Coulomb, pico Coulomb
FFT	fast Fourier transform
HID	human interface device
I2C	inter integrated circuit (communication or bus)
ID	identifier
IMU	inertial measurement unit
LED	light emitting diode
LSB	least significant bit
MCU	micro control unit
MLS	maximum length sequence
MSB	most significant bit
OS	operating system
PC	personal computer
PN	pseudo noise
Rx	receive
SPI	serial peripheral interface (bus)
Stby	standby
TS0, TS1, … TSN	time of the start of a specific timeslot (0 through N)
T_{ACK}	time of the start of an ACK
TAG	turnaround gap
T_{BEACON}	time of start of a Beacon
T_{EOB}	time of end of a Beacon
Tx	transmit
UI	user interface
USB	Universal Serial Bus ™
USI™	Universal Stylus Initiative ®
V, Vpp, Vrms	volts, volts (peak to peak), volts (root mean square)

Appendix B – References

General information on HID usages:
http://www.usb.org/devleopers/hidpage/

HID Multitouch Report Descriptor from USB Implementers Forum
http://www.usb.orgg/developers/hidpage/HUTRR34.zip

Supporting Usages in Digitizer Report Descriptors, Microsoft,
https://msdn.microsoft.com/en-us/library/windows/hardware/jj151564(v=vs.85).aspx

Digitizer Drivers for Windows Touch and Pen-Based Computers, Microsoft, October 21, 2010
download.microsoft.com/download/a/d/f/.../DigitizerDrvs_touch.docx

Polhemus Device
http://polhemus.com/motion-tracking/all-trackers/g4

http://polhemus.com/_assets/img/G4_Brochure.pdf

USI Compliance Test Plan, Universal Stylus Initiative, 2016

Color Index Values
http://www.w3.org/TR/SVG/types.html#ColorKeywords

Hash Algorithm
https://code.google.com/p/smhasher/wiki/MurmurHash3

European Common External Power Supply (EPS) Memorandum of Understanding (MoU)
http://ec.europa.eu/DocsRoom/documents/2418/attachments/1/translations/en/renditions/native

Additional information on the MoU available at:
http://ec.europa.eu/growth/sectors/electrical-engineering/rtte-directive/common-charger/index_en.htm

Appendix C – Trademarks and Patents

USI™

UNIVERSAL STYLUS INITIATIVE®

Pen Design™

USI CERTIFIED™

USI UNIVERSAL STYLUS INITIATIVE & Pen Design™

Appendix D– Time to Ink Analysis

D.1 Objectives

Conduct an experiment to determine the minimum available time to ink after the stylus is detected within range. Measure the following:

- The approach time (from stylus detection to surface contact)
- The dwell time (time until stylus is lifted or moved after the initial contact).

D.2 Description of Research

Participants

- 8 right handed participants with normal or corrected vision

Apparatus

- Polhemus G4 wireless magnetic tracking device. Data acquired at 120 Hz, latency < 10 ms, accuracy of 0.001 inch linear and 0.001 degree angular. Participants use a passive stylus on a Dell 10 inch tablet.

Experimental Design

- The experimental design is a 3x5x4 repeated measures design with each participant exposed to each experimental condition. Participants will repeat each trial 10 times.

Independent Variables

- Three independent variables were tested to determine their impact on approach time and dwell time:
  1. Use case
  2. Distance to target
  3. Target size
- The objective was to find the scenario that resulted in the minimum approach and dwell times.

Dependent variables to be measured:

- Time to contact – time from a distance X above target, to target contact
- Dwell time – time from when the stylus touches the target to when the stylus is lifted from the target surface.

D.2.1 Use Cases

High Precision

- User starts with palm down, holding the stylus similar to writing posture with a pen. The user raises the stylus, moves it to a target to the left. The movement should occur largely at the wrist.
- Trial runs were made with the stylus at 5 different starting distances from the surface (6 cm, 5 cm, 4 cm, 3 cm, 2 cm) with horizontal movement controlled at 2 cm.

- For each starting distance, the users were presented with different target sizes in order to test for interactions of speed and precision

See Figure D-1 for a graphical representation of this test case.

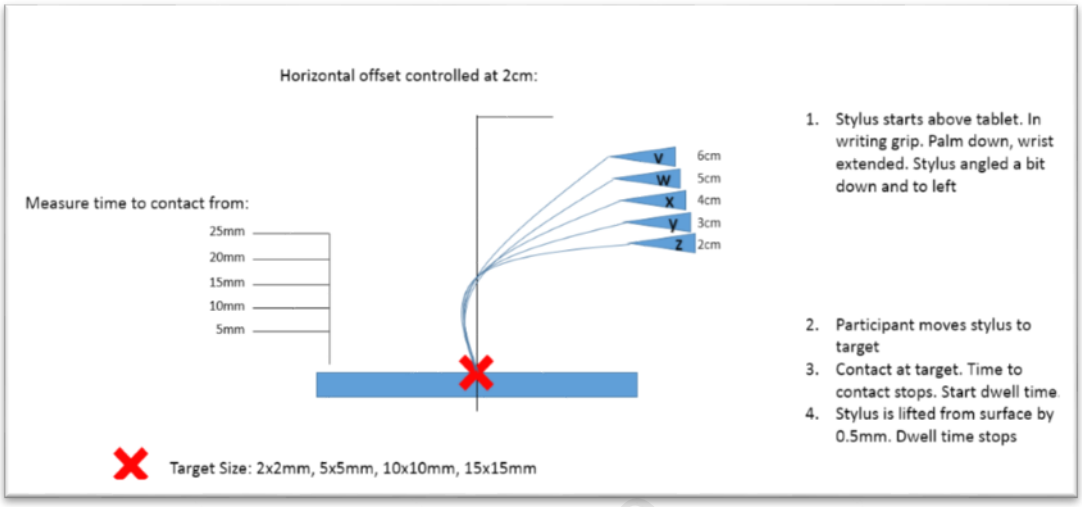


Figure D-1 High Precision Use Case

Vertical Approach

User stands above the workstation and starts holding the stylus in a pinch grip from above the tablet. The user lowers the stylus to a target on the tablet. The analogy of this use case is a teacher standing above a student's work, and descending with a red pen to mark the student's work.

Trial runs were made with the stylus at 5 different starting distances from the surface (40 cm, 30 cm. 20 cm, 10 cm, 5 cm) with horizontal movement controlled at 2 cm.

For each starting distance, the users were presented with different target sizes in order to test for interactions of speed and precision. See Figure D-2 for a graphical representation of this test case.

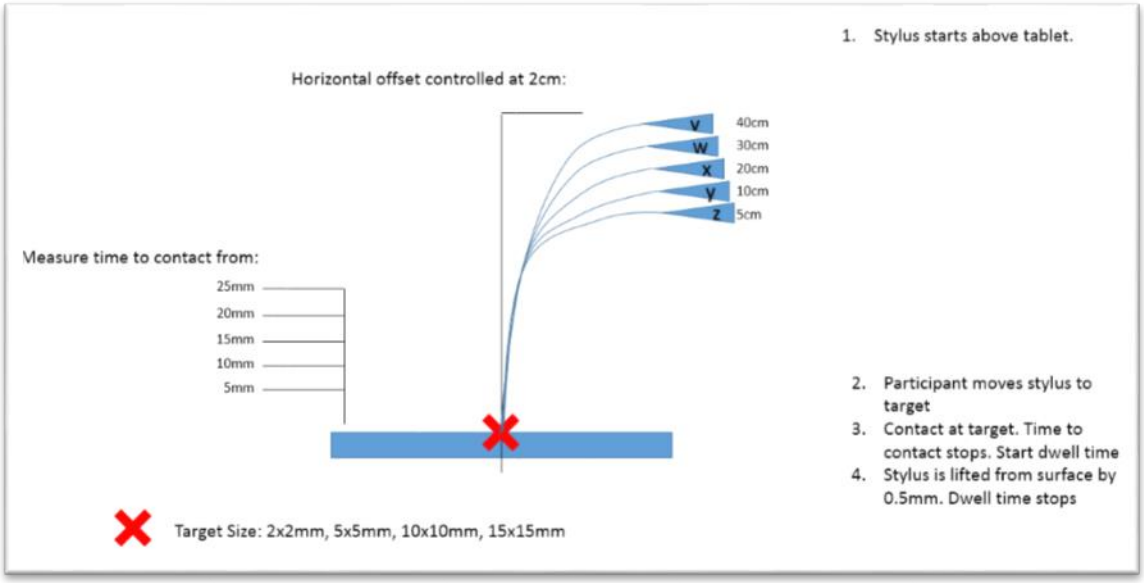


Figure D-2 Vertical Approach Use Case

Move From Edge

User starts by holding the stylus similar to writing posture as in use case 1. The wrist is rotated such that the palm is facing left. The user moves the stylus to a target to the user's left. The movement largely occurs at the elbow. The analogy for this use case is a graphical artist pausing to view work on the screen, relaxing the artist's hand to the right of the tablet, identifying a correction to make and re-engaging the tablet.

Trial runs were made with the stylus at 5 different starting distances from the edge of the screen (6 cm, 5 cm, 4 cm, 3 cm, 2 cm) with vertical height controlled at 2 cm.

For each starting distance, the users were presented with different target sizes in order to test for interactions of speed and precision

See Figure D-3 for a graphical representation of this test case:

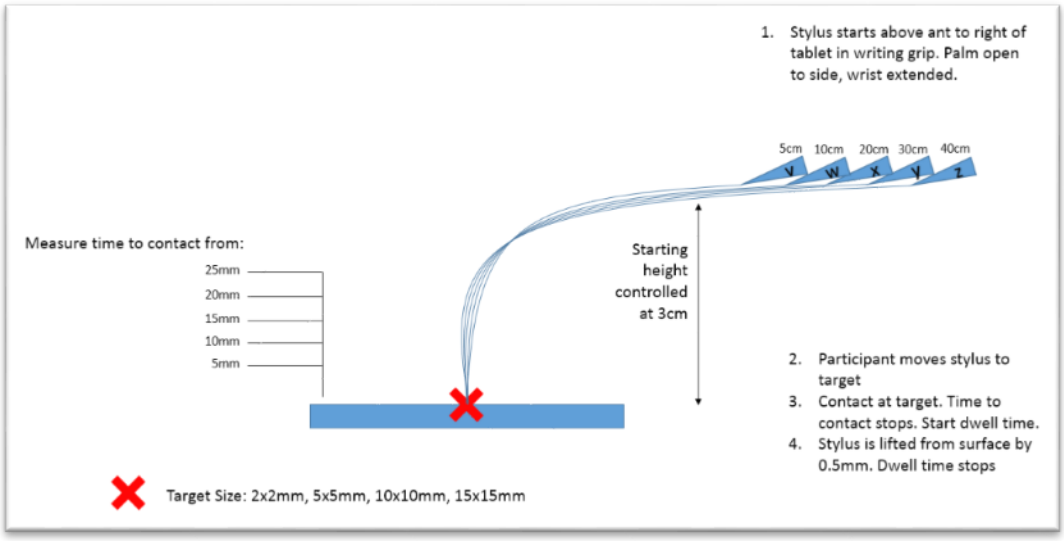


Figure D-3 Move From Edge Use Case

D.3 Data Analysis

Data analysis is broken into two sections, approach speed and dwell time as measured by the above experiments.

D.3.1 Approach Speed

D.3.1.1 Data Quality

Although somewhat bi-modal and skewed, the data was sufficiently Gaussian ($P < 0.005$) for statistical comparisons.

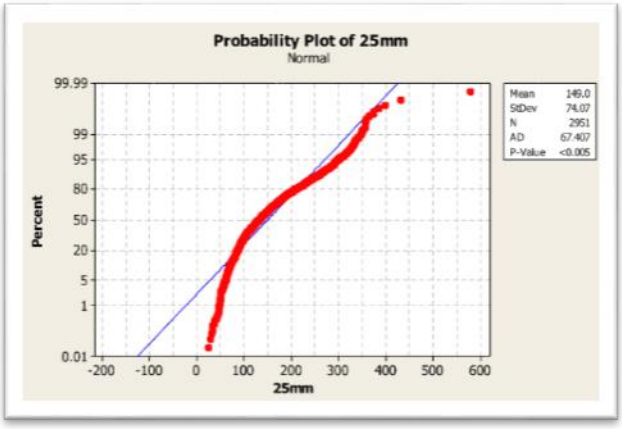


Figure D-4 Data Quality

D.3.1.2 Test Conditions

An ANalysis Of Variance (ANOVA) was done for the test condition case (move from edge, vertical, and precision) at the various heights. The Move from Edge (horizontal) test condition has the least delay to reach the surface and is significantly faster ($p < 0.001$) than Vertical or Precision for all heights. The analysis for 5 mm and 25 mm heights are shown below:

One-way ANOVA: 5 mm versus trial

```
Source  DF  SS    MS    F     P
trial    2  39393 19697 84.23 0.000

                    Individual 95% CIs For Mean Based on
                    Pooled StDev
Level       N  Mean StDev -----+---------+---------+---------+----
horizontal 1151 26.29 12.68 (--*--)
precision   640 36.07 20.78                    (---*---)
vertical   1160 29.98 14.04          (--*--)
                    -----+---------+---------+---------+----
                     27.0    30.0    33.0    36.0
```

One-way ANOVA: 25 mm versus trial

Source DF SS MS F P
trial 2 737094 368547 70.33 0.000

Individual 95% CIs For Mean Based on Pooled StDev

Level	N	Mean	StDev	----+---------+---------+---------+----
horizontal	1151	132.88	63.75	(--*-)
precision	640	175.07	88.76	(---*--)
vertical	1160	150.57	70.37	(-*--)

----+---------+---------+---------+----
135 150 165 180

D.3.1.3 Starting Distance

ANOVA on the starting distance shows that starting distance was not significant. This analysis was only done for the Move from Edge condition at 10 mm height only since this represents the worst-case test condition and probable minimum height requirement. The results are shown below:

One-way ANOVA: 10 mm versus distance

Source DF SS MS F P
distance 4 2548 637 0.98 0.418

Individual 95% CIs For Mean Based on Pooled StDev

Level	N	Mean	StDev	----+---------+---------+---------+----
5	221	53.64	25.31	(---------*---------)
10	239	54.40	28.76	(---------*---------)
20	229	52.33	24.74	(---------*---------)
30	226	50.17	24.82	(---------*---------)
40	236	53.64	23.51	(---------*---------)

----+---------+---------+---------+----
48.0 51.0 54.0 57.0

D.3.1.4 Target Size

ANOVA shows that larger targets were slightly faster ($P < 0.035$). This analysis was only done for the Move from Edge condition at 10 mm. The results are shown below:

One-way ANOVA: 10 mm versus size

Source DF SS MS F P
size 3 5591 1864 2.88 0.035
error 1147 742662 647
Total 1150 748253

Individual 95% CIs For Mean Based on Pooled StDev

Level	N	Mean	StDev	-+---------+---------+---------+-------
2	276	55.08	25.75	(---------*---------)
5	285	55.14	27.06	(---------*---------)
10	298	50.62	24.50	(---------*---------)

```
15   292 50.79 24.46   (-------*---------)
                  ----+---------+---------+---------+-------
                    48.0    51.0    54.0    57.0
```

D.3.1.5 Height

The relationship between time and height is exceptionally linear. This means we can interpolate intermediate values of height is needed.

Variable	Total Count	Mean	StDev	Minimum	Q1	Median	Q3
5 mm	2953	29.863	15.718	4.000	18.000	26.000	38.000
10 mm	2953	59.518	29.799	10.000	36.000	52.000	75.000
15 mm	2953	89.261	44.406	17.000	55.000	78.000	113.000
20 mm	2953	119.18	59.22	23.00	73.00	104.00	151.00
25 mm	2953	148.98	74.07	24.00	91.00	130.00	189.00

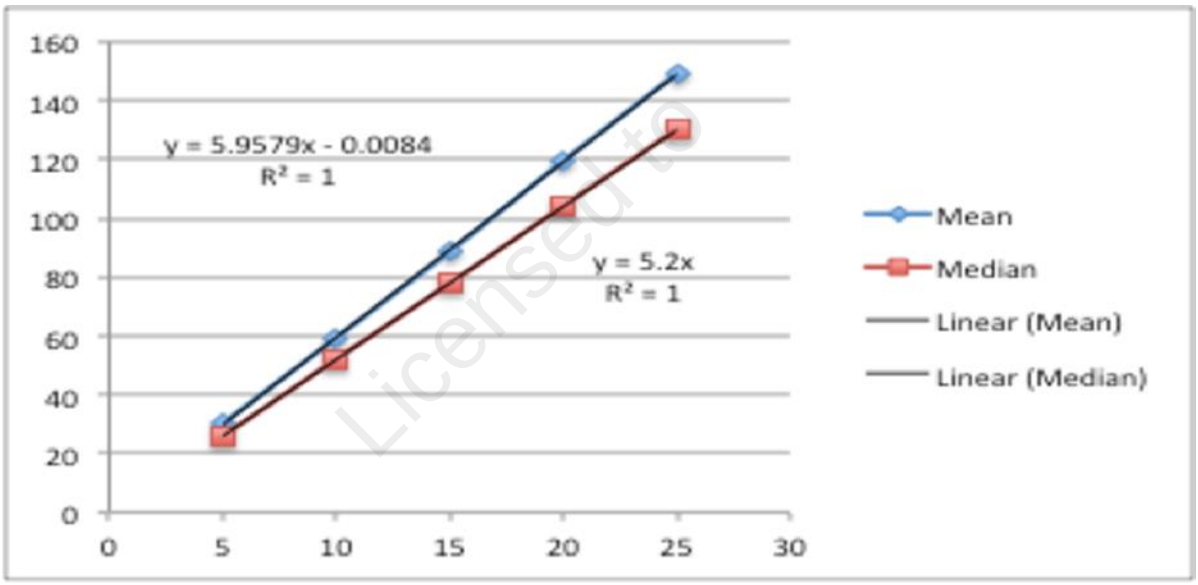


Figure D-5 Height vs Approach Time

D.3.1.6 Approach Speed Conclusions

Conclusions about approach speed:

- The Move from Edge condition is significantly faster.
- Starting distance is not significant.
- 10 mm and 20 mm target sizes are faster than smaller targets.
- Time and height are linear.

D.3.1.7 Worst Case Approach Time

The worst case (fastest) approach time is the Move from Edge condition with a 10 or 20 mm target size. The statistics for this at a 10 mm height is as follows:

Descriptive Statistics: 10 mm

Variable	N	Mean	SE Mean	StDev	Minimum	Q1	Median	Q3	Maximum
10 mm	1151	52.855	0.752	25.508	13.000	34.000	46.000	67.000	135.000

The different median and mean indicates a skew to the curve, which is shown in Figure D-6:

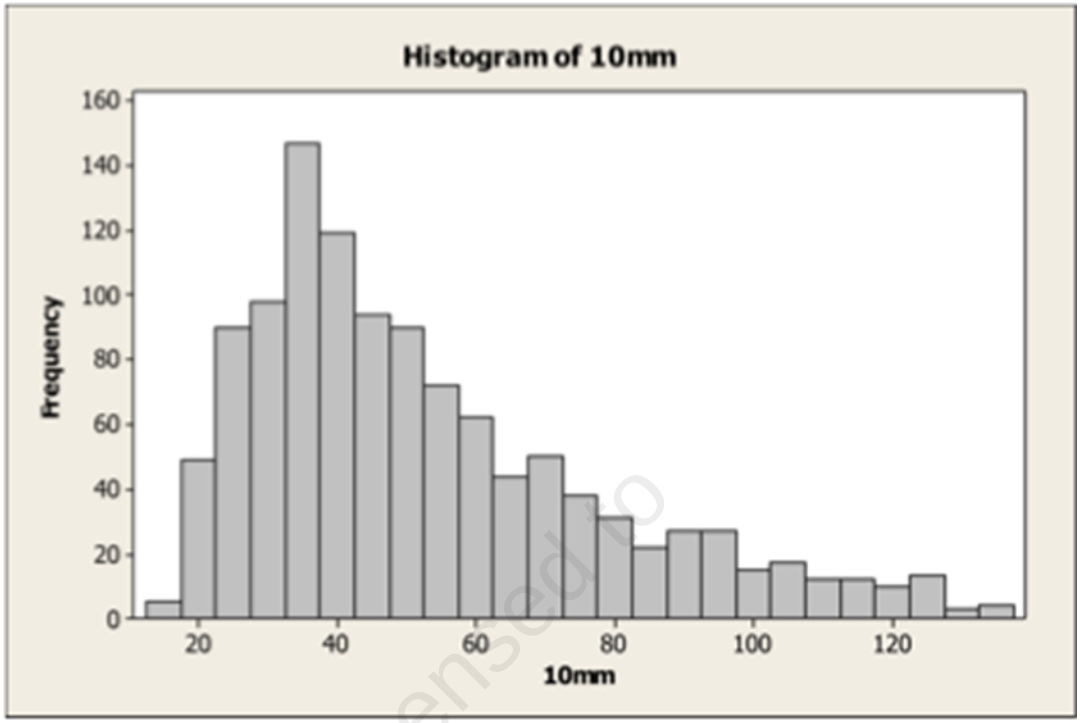


Figure D-6 Histogram of Move from Edge Condition with a 10 or 20 mm Target Size and 10 mm Height

Because of the skew, mean and standard deviation is not a good measure of minimum times. Looking at quartiles, 75% of the samples are 34 ms or longer.

D.3.2 Dwell Time

D.3.2.1 Data Quality

Although somewhat skewed, the data were sufficiently Gaussian ($P < 0.005$) for statistical comparisons.

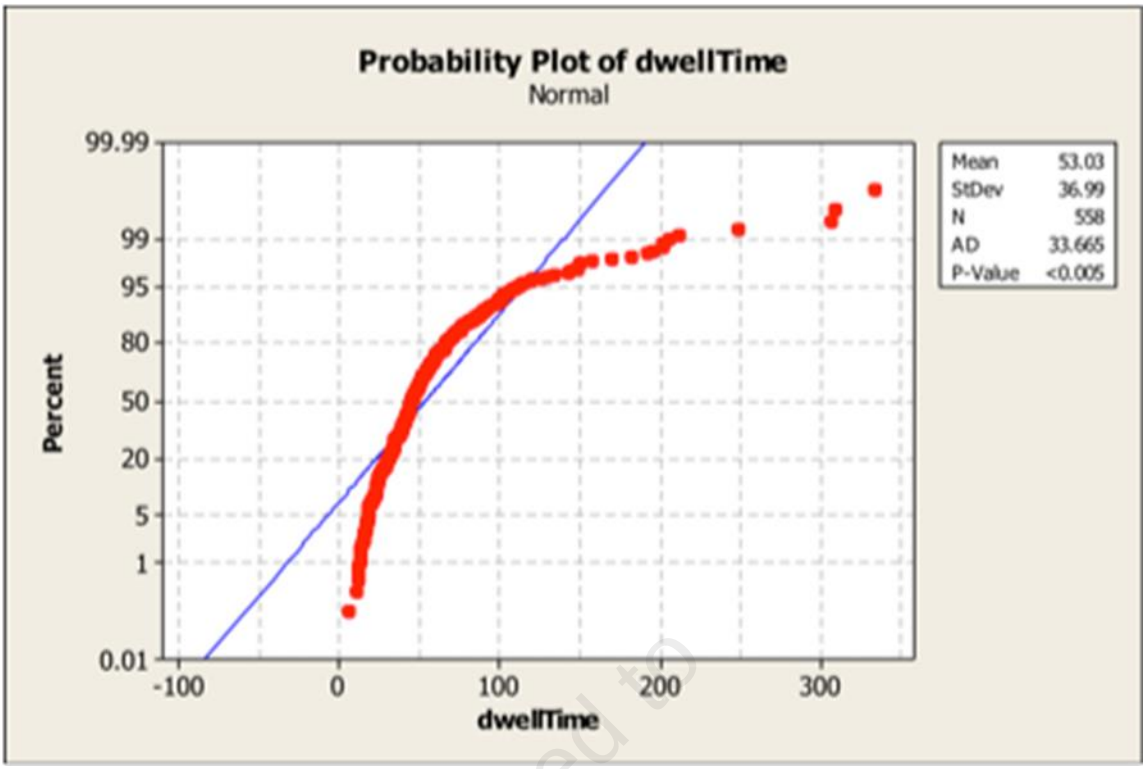


Figure D-7 Dwell Time Data Quality

D.3.2.2 Test Condition

ANOVA for the test condition shows that the Move from Edge test condition has the shortest dwell time and is significantly faster ($p < 0.001$) than Vertical or Precision for all heights

One-way ANOVA: dwellTime versus trial

Source	DF	SS	MS	F	P
trial	2	85566	42783	14.40	0.000

Individual 95% CIs For Mean Based on Pooled StDev

```
Level       -+---------+---------+---------+-------
horizontal   (------*-----)
precision                 (------*-----)
vertical          (------*-----)
          -+---------+---------+---------+-------
         49.0      56.0      63.0      70.0
```

D.3.2.3 Starting Distance

ANOVA on the starting distance shows that a higher starting distance was slightly faster ($P < 0.019$). The results are shown below:

One-way ANOVA: dwellTime versus distance

Source	DF	SS	MS	F	P
distance	4	16080	4020	2.98	0.019

```
                    Individual 95% CIs For Mean Based on
                    Pooled StDev
Level   N  Mean StDev  --+---------+---------+---------+------
 5    113 51.73 28.88        (-------*-------)
10    111 49.35 34.25      (-------*-------)
20    109 59.89 46.74               (-------*-------)
30    117 58.29 43.79               (-------*-------)
40    108 45.56 24.71  (-------*-------)
                  --+---------+---------+---------+------
                  40.0      48.0      56.0
```

D.3.2.4 Target Size

ANOVA shows that target size is not significant. The results are shown below:

One-way ANOVA: dwellTime versus size

```
Source  DF    SS   MS    F     P
size     3  5409 1803 1.32 0.267
```

```
                    Individual 95% CIs For Mean Based on
                    Pooled StDev
Level   N  Mean StDev ----+---------+---------+---------+----
 2    129 56.36 39.98           (----------*----------)
 5    146 55.93 37.53           (----------*----------)
10    138 49.18 29.44 (----------*----------)
15    145 50.82 39.91    (----------*----------)
                  ----+---------+---------+---------+----
                  45.0      50.0      55.0      60.0
```


D.3.2.5 Dwell Time Conclusions

The dwell time testing led to the conclusions:

- Move from Edge condition is fastest.
- Starting distance of 40 cm is fastest.
- Target size is not significant.

D.3.2.6 Worst Case Dwell Time

The worst case (fastest) dwell time is the Move from Edge condition starting at 40 mm. The statistics for this are as follows:

Descriptive Statistics: dwellTime

Variable	Distance	N	Mean	StDev	Minimum	Q1	Median	Q3	Maximum
dwellTime	40	108	45.56	24.71	10.00	30.00	39.00	57.25	142.00

The different median and mean indicates a skew to the curve, which is shown in Figure D-8:

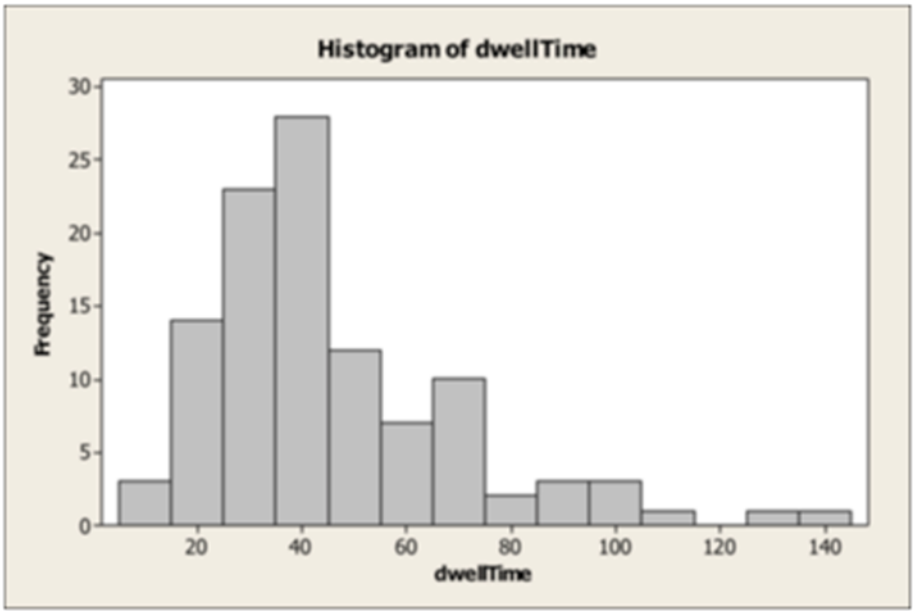


Figure D-8 Histogram Of Dwell Time for Move from Edge Condition at 40 mm Starting Distance

Because of the skew, mean and standard deviation is not a good measure of minimum times. Looking at quartiles, 75% of the samples are 30 ms or longer.

D.3.3 Approach Time vs Dwell Time

There is no correlation between approach time and dwell time, as is shown in Figure D-9:

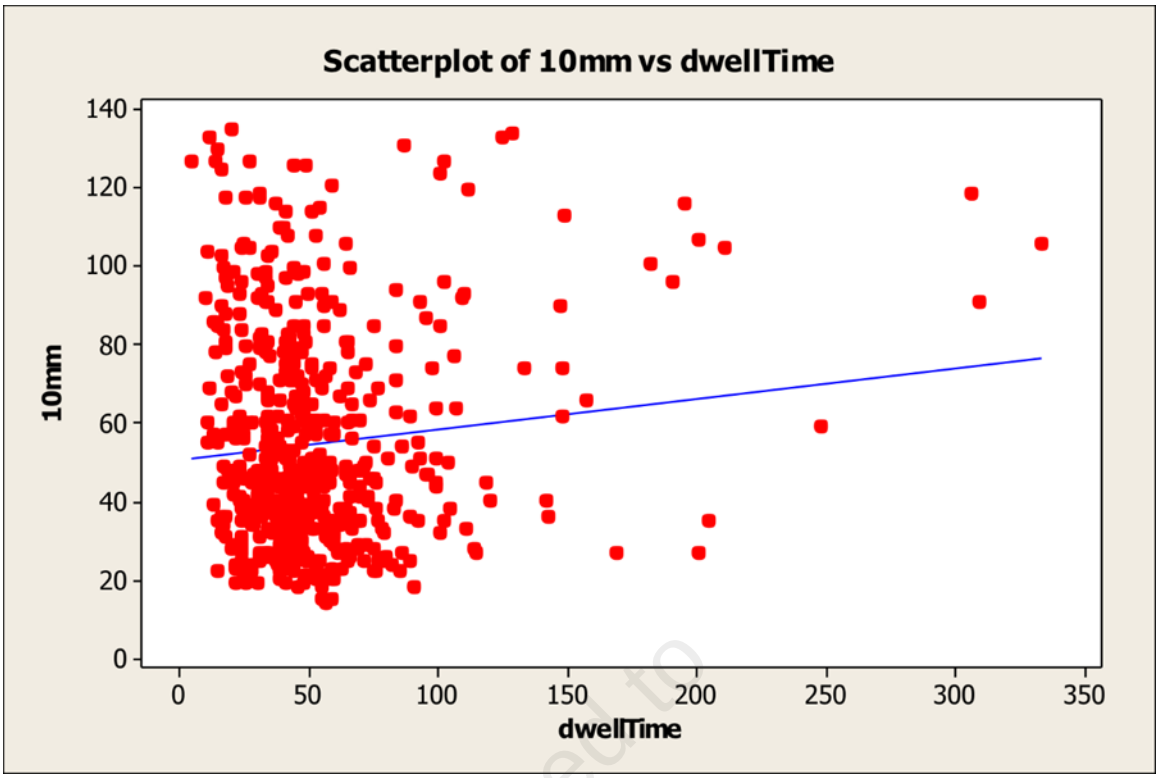


Figure D-9 Approach Time vs Dwell Time

Appendix E – Frequency Selection Analysis

The chart in Figure E-1 shows the frequency bands occupied by various devices that would be in close proximity to a USI controller and stylus. The USI uplink DSSS spreading code was tuned to maximize energy in the 200 - 500 KHz range. USI downlink D-BPSK frequencies were selected to operate in slightly broader range of 100 - 500 KHz range to allow more flexibility for the USI controller to adapt to system noise sources. While the quiet ranges as per the chart are in 200-275 khz and 350-500khz, an implementation can use the other frequencies in the 100-500khz range, if it determines that there is no interference at that frequency at the time the controller-stylus link is operating.

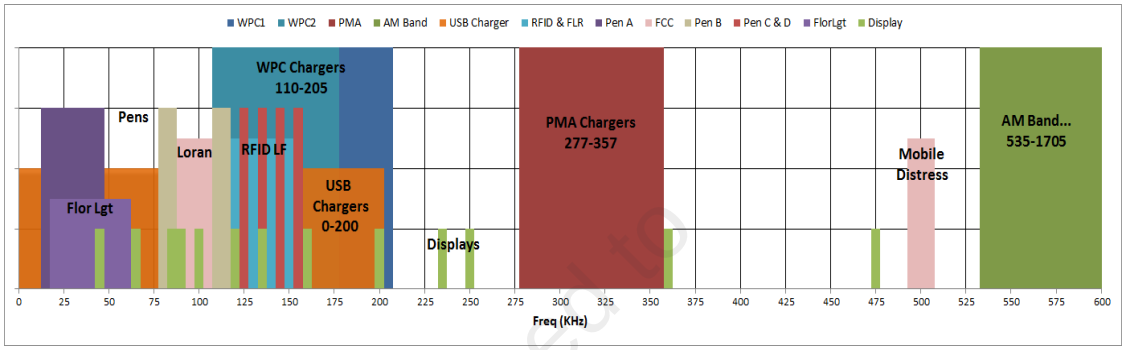


Figure E-1 Frequency Selection

§ §

Appendix F – Vendor ID Codes

Table 6-19 defines the stylus' global ID, which contains USI Vendor-ID[11:0], that is unique to each vendor. The following table defines the Vendor-ID codes assigned by USI. Vendors shall use their unique VendorID in all their USI specification based products.

Vendor ID	Vendor Name
0-3	Reserved (Legacy use)
4-9	Reserved for Universal Stylus Initiative, Inc.
10-19	For USI Vendor Experimental Use; Not to be used in production release
20-4095	Assigned by Universal Stylus Initiative, Inc. Refer to http://www.UniversalStylus.org

Appendix G– Hash Algorithm

This appendix describes the hash algorithm of the USI Stylus hash function. See section 6.7 for details on what this algorithm is used for.

G.1 Input

The hash function input is 96 bits of data that can be organized into six 16 bit words. Word0 is the least significant word (rightmost), and Word5 is the most significant word (leftmost). The input information is:

- Word0: stylus global ID word 0 (GID0)
- Word1: stylus global ID word 1 (GID1)
- Word2: stylus global ID word 2 (GID2)
- Word3: stylus global ID word 3 (GID3)
- Word4: 8 bits of PreferredType in LSByte, 8 bits of PreferredWidth in MSByte and
- Word5: 8 bits of PreferredColor in LSByte and 2 bits of SEP. all other MSbits set to zero.

Following table further clarifies this.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WORD0 (GID0)	Device-ID[15:0]															
WORD1 (GID1)	Device-ID[31:16]															
WORD2 (GID2)	Device-ID[47:32]															
WORD3 (GID3)	Vendor-ID[11:0]												Device-ID[51:48]			
WORD4	PreferredWidth[7:0]								PreferredType[7:0]							
WORD5	0	0	0	0	0	0	SEP[1:0]		PreferredColor[7:0]							

A 32 bit seed value is also input to the algorithm. This value shall be 0x55534931.

G.2 Output

The hash function outputs a 32 bit integer. This is then split into two halves:

- The 16 LSBs are used in the stylus response S.HASH(…) to the C.ConfigPhy(…) command and also in C.VerifyHashLow(…).
- The 16 MSBs are used by the controller when it sends the C.VerifyHashHigh(…) command.

G.3 Algorithm

Algorithm is Murmer3 and can be downloaded from:

https://code.google.com/p/smhasher/wiki/MurmurHash3 .

USI will only use the hash function murmurHash3_x86_32().

§ §

Appendix H– Preferred Color Index Values

The following table defines the 8-bit index value to be used for the PreferredColor. The indices correspond to the keywords defined in http://www.w3.org/TR/SVG/types.html#ColorKeywords and cover major colors.

Table H-1 8-bit Index values for PreferredColor

Index	Name	RGBValues
0	AliceBlue	#F0F8FF
1	AntiqueWhite	#FAEBD7
2	Aqua	#00FFFF
3	Aquamarine	#7FFFD4
4	Azure	#F0FFFF
5	Beige	#F5F5DC
6	Bisque	#FFE4C4
7	Black	#000000
8	BlanchedAlmond	#FFEBCD
9	Blue	#0000FF
10	BlueViolet	#8A2BE2
11	Brown	#A52A2A
12	BurlyWood	#DEB887
13	CadetBlue	#5F9EA0
14	Chartreuse	#7FFF00
15	Chocolate	#D2691E
16	Coral	#FF7F50
17	CornflowerBlue	#6495ED
18	Cornsilk	#FFF8DC
19	Crimson	#DC143C
20	Cyan	#00FFFF
21	DarkBlue	#00008B
22	DarkCyan	#008B8B

Index	Name	RGBValues
23	DarkGoldenRod	#B8860B
24	DarkGray	#A9A9A9
25	DarkGreen	#006400
26	DarkKhaki	#BDB76B
27	DarkMagenta	#8B008B
28	DarkOliveGreen	#556B2F
29	DarkOrange	#FF8C00
30	DarkOrchid	#9932CC
31	DarkRed	#8B0000
32	DarkSalmon	#E9967A
33	DarkSeaGreen	#8FBC8F
34	DarkSlateBlue	#483D8B
35	DarkSlateGray	#2F4F4F
36	DarkTurquoise	#00CED1
37	DarkViolet	#9400D3
38	DeepPink	#FF1493
39	DeepSkyBlue	#00BFFF
40	DimGray	#696969
41	DodgerBlue	#1E90FF
42	FireBrick	#B22222
43	FloralWhite	#FFFAF0
44	ForestGreen	#228B22
45	Fuchsia	#FF00FF
46	Gainsboro	#DCDCDC
47	GhostWhite	#F8F8FF
48	Gold	#FFD700
49	GoldenRod	#DAA520
50	Gray	#808080
51	Green	#008000
52	GreenYellow	#ADFF2F
53	HoneyDew	#F0FFF0
54	HotPink	#FF69B4

Index	Name	RGBValues
55	IndianRed	#CD5C5C
56	Indigo	#4B0082
57	Ivory	#FFFFF0
58	Khaki	#F0E68C
59	Lavender	#E6E6FA
60	LavenderBlush	#FFF0F5
61	LawnGreen	#7CFC00
62	LemonChiffon	#FFFACD
63	LightBlue	#ADD8E6
64	LightCoral	#F08080
65	LightCyan	#E0FFFF
66	LightGoldenRodYellow	#FAFAD2
67	LightGray	#D3D3D3
68	LightGreen	#90EE90
69	LightPink	#FFB6C1
70	LightSalmon	#FFA07A
71	LightSeaGreen	#20B2AA
72	LightSkyBlue	#87CEFA
73	LightSlateGray	#778899
74	LightSteelBlue	#B0C4DE
75	LightYellow	#FFFFE0
76	Lime	#00FF00
77	LimeGreen	#32CD32
78	Linen	#FAF0E6
79	Magenta	#FF00FF
80	Maroon	#800000
81	MediumAquaMarine	#66CDAA
82	MediumBlue	#0000CD
83	MediumOrchid	#BA55D3
84	MediumPurple	#9370DB
85	MediumSeaGreen	#3CB371
86	MediumSlateBlue	#7B68EE

Index	Name	RGBValues
87	MediumSpringGreen	#00FA9A
88	MediumTurquoise	#48D1CC
89	MediumVioletRed	#C71585
90	MidnightBlue	#191970
91	MintCream	#F5FFFA
92	MistyRose	#FFE4E1
93	Moccasin	#FFE4B5
94	NavajoWhite	#FFDEAD
95	Navy	#000080
96	OldLace	#FDF5E6
97	Olive	#808000
98	OliveDrab	#6B8E23
99	Orange	#FFA500
100	OrangeRed	#FF4500
101	Orchid	#DA70D6
102	PaleGoldenRod	#EEE8AA
103	PaleGreen	#98FB98
104	PaleTurquoise	#AFEEEE
105	PaleVioletRed	#DB7093
106	PapayaWhip	#FFEFD5
107	PeachPuff	#FFDAB9
108	Peru	#CD853F
109	Pink	#FFC0CB
110	Plum	#DDA0DD
111	PowderBlue	#B0E0E6
112	Purple	#800080
113	RebeccaPurple	#663399
114	Red	#FF0000
115	RosyBrown	#BC8F8F
116	RoyalBlue	#4169E1
117	SaddleBrown	#8B4513
118	Salmon	#FA8072

Index	Name	RGBValues
119	SandyBrown	#F4A460
120	SeaGreen	#2E8B57
121	SeaShell	#FFF5EE
122	Sienna	#A0522D
123	Silver	#C0C0C0
124	SkyBlue	#87CEEB
125	SlateBlue	#6A5ACD
126	SlateGray	#708090
127	Snow	#FFFAFA
128	SpringGreen	#00FF7F
129	SteelBlue	#4682B4
130	Tan	#D2B48C
131	Teal	#008080
132	Thistle	#D8BFD8
133	Tomato	#FF6347
134	Turquoise	#40E0D0
135	Violet	#EE82EE
136	Wheat	#F5DEB3
137	White	#FFFFFF
138	WhiteSmoke	#F5F5F5
139	Yellow	#FFFF00
140	YellowGreen	#9ACD32
141-254	Reserved	
255	No Preferred Color	

§ §

Appendix I – USI™ Test Commands

I.1 Write Command definitions ID=50 to ID=57

Write commands allow for test modes and test data to be setup as well as statistics about the Stylus RX and TX channels to be gathered.

Assumptions:-

1. All commands assume that the Stylus has been paired correctly, so that it has timeslots allocated and frequencies of operation defined, as well as beacon period.
2. Test modes that change the normal operation mode can be exited via any button press.

The following table gives details of the new commands

Table I-1 Test Commands

Command	WRITE-DATA[15:0]	Command-ID[5:0]
Test_Exit	Clears any test settings/parameters and returns to Normal operation. This command can also be achieved manually by pressing any of the buttons, while in Test Modes Test Modes are defined as any command between 54 – 57 are active Data = 0xFEDC – Exit Test Mode Data != 0xFEDC – No Exit	50
Soft_Reset	Data = 0xBA98 – Apply Reset Data != 0xBA98 – No Reset Performs a equivalant Power-on-reset of the Stylus, Conditons after reset should reflect the state of the Stylus when power-on reset has occurred.	51
Echo_Packet	Returns the value recieved by the Stylus 0x0000 ... – Value to be returned in Data Packet 0xFFFF Note: Default value after reset = 0x0000 Stylus should calculate the CRC of Data Value used	52
Statistics	Data = 0x3210 – **Statistics_Clear** , clears all values to 0x0000. The values remain at 0x0000 until enabled via the Enable data value. Data = 0x7654 – **Statistics_Enable** enables updating of Statistics (refer to Read Commands) Data = 0x89AB – **Statistics_Freeze** , stop incrementing values.	53

Command	WRITE-DATA[15:0]	Command-ID[5:0]
Test_Mode	Assumption on entry to this mode is that Normal Pairing has occured between controller and stylus and data and/or position packets have been defined. See Table I- below for Individual bit allocation Only an individual bits can be enabled and there is no all zero command	54
Vendor_Test_Mode	Value specified by Vendor Examples of functionality that could be provided are 1. Serial Data stream recieved by stylus output on a test point 2. Enable a port on the Stylus to monitor status messages 3. Enable control port to configure and control data transmitted 4. ..	55
Reserved	Reserved for USI Test	56-57

The following table describes the Test-Mode command definitions.

Table I-2 Test Mode Command Definitions

WRITE-DATA[15:0]	Name	Description
0x0001	Increment_Data	Starting at Data value set by the Echo_Packet command increment by 1 count on every enabled Data Packet setup during pairing Wrap around on 0xFFFF to 0X0000
0x0002	Enable_All_Data	Stylus transmits a Data Packet on all available timeslots in USI Frame. The number of timeslots is based on the USI Frame cadence set during pairing and the timeslot size. (e.g., 16.6ms enables 29 Data Packets at 0.5ms) The Data value returned is the value set by the Echo_Packet command
0x0004	Increment_All_Data	Combination of above two commands Increment_Data and Enable_All_Data
0x0008	Enable_All_Position	Stylus transmits a Position Tone on all available time slots in USI Frame. The number of timeslots is based on the USI Frame cadence set during pairing and the timeslot size. (e.g., 16.6ms enables 14 Position Packets at 1.0ms, assuming ACK Packet = 0.5ms) The Tone used is the one dedfined during pairing.
0x0010	Tone	Send a continuous tone as defined during the pairing operation. The stylus is not required to recieved or respond to any beacon commands. Note: This operation mode can not be exited via the Test_exit or Soft_reset command. Only a button press or removal of power can exit this mode.

WRITE-DATA[15:0]	Name	Description
0x0020	Reserved	
0x0040	Reserved	
0x0080	Reserved	
0x0100	Reserved	
0x0200	Reserved	
0x0400	Reserved	
0x0800	Reserved	
0x1000	Reserved	
0x2000	Reserved	
0x4000	Reserved	
0x8000	Reserved	
All other values	Not allowed	Return S.REJECT()

I.2 Read Command Definitions, Subcommand = 50 & 51

The following commands allow for Debug parameters to be read from the Stylus.

Table I-3 – Read Debug Commands

Command	READ-FLAGS[8:0]	SubCommand-ID[5:0]	Command-ID[5:0]
Get_RX_Statistics	Individual per command	50	63
Get_TX_Statistics		51	63
Reserved for USI Test		52-57	63

The following table shows the RX_Statistics, these are 16bit fields that increment on the given condition when enabled via the `Statistics_Enable` command. By default they values are undefined and need to be cleared before use using the `Statistics_Clear` command.

The values can be read out at any time. However to enable a consistent set of values to be read the values can be prevented from being incremented by using the `Statistics_Freeze` command.

Table I-4 Get_Rx_Statistics

Command Name:	**Get_RX_Statistics(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	Beacons Received – Independent of Stylus ID															
n=1	WRITE Commands Received – Specific to Stylus ID of Stylus															
n=2	READ Commands Received - Specific to Stylus ID of Stylus															
n=3	DSS – Code error in Data Bit															
n=4	CRC error in Command															
n=5	Stylus received Signal Strength															
n=6-8	reserved															

The following table gives the TX_Statistics which shows the Stylus responses and events that have occurred.

Table I-5 Get_Tx_Statistics

Command Name:	**Get_TX_Statistics(...)**															
	Stylus data fields															
READ-FLAGS[n]=1	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
n=0	S.HASH() responses made															
n=1	S.ACK() WRITE responses made															
n=2	S.REJECT() WRITE responses made															
n=3	S.REJECT() READ responses made															
n=4	Number of successful pairing events															
n=5	Number of Discovery TimeOuts															
n=6	Number of Pairing TimeOuts															
n=7	Number of Unpairing commands															
n=8	Reserved															

Appendix J – CRC Generation Pseudo-Code

The following lists the high level flow (sample Ccode) that is applicable for both the controller and the stylus.

```
int CRC_Calc(unsigned _int32 data,
        const int data_word_length,
        int crc_polynomial,
        const int crc_polynomial_length,
        const int seed,
        const int finalXORval)
{

   // We are calculating from MSB, so get the mask
   const _int32   msb_mask = 1 << (data_word_length-1);

   // Shift the CRC polynomial to match
   crc_polynomial = crc_polynomial << (data_word_length - crc_polynomial_length);

   for(int bidx = 0;bidx < data_word_length; bidx++) {
      if(data & msb_mask)
         data = data ^ crc_polynomial;

      // Shift the data left by one bit
      data = data << 1;
   }

   // The high 'crc_polynomial_length'-1 bits are the CRC (before 'finalXOR')
   // Extract them
   int crc = data >> (data_word_length - (crc_polynomial_length - 1));

   // Now XOR with the 'finalXORval'
   crc = crc ^ finalXORval;

   return (crc);
}

int CRC4_DownLink(unsigned _int32 data) {
   // For Downlink, the parameters are
   // data - as passed
   // datalength = 16 bits
   // Polynomial = 11001B = 25
   // polynomial length = 5 bits
   // seedvalue = 0
   // finalXORval = 0xF = 15
   return CRC_Calc (data, 16, 25, 5, 0, 15);
}

int CRC5_UpLink(unsigned _int32 data) {
   // For Uplink, the parameters are
   // data - as passed
```

```
  // datalength = 25 bits
  // Polynomial = 111001B = 57
  // polynomial length = 6 bits
  // seedvalue = 0
  // finalXORval = 0
  return CRC_Calc (data, 25, 57, 6, 0, 0);
}
```

Appendix K – Charger Interference in USI™ Model

K.1 Purpose

The purpose of the charger interference framework is to provide a description of the worst-case interference from noisy AC/DC chargers that a Universal Stylus will need to work with. This framework is intended to describe this specification and a quantitative method for characterizing chargers which is:

- Useful for the purpose of Universal Stylus definition,
- Appropriate for the variety of chargers that may be encountered,
- Easily understandable to all, and
- Straightforward to measure.

K.2 Framework

The framework is based upon the common-mode noise specification in the European Common External Power Supply (EPS) Memorandum of Understanding (MoU) (see Appendix B for reference link). This Appendix quantifies the charger's common-mode (i.e., shared between the output 0V and +V lines) noise properties, with respect to earth ground. The quantities specified in the MoU and used in the USI spec are:

- Maximum AC peak-to-peak voltage,
- Maximum EPS switching peak-to-peak voltage in four frequency bands, and
- Maximum slew rate for the EPS switching signal.

The MoU also specifies a minimum load to line capacitance, although that quantity is not included in this spec. In addition, the MoU specifies a maximum occupied bandwidth; this is excluded from the present specification in lieu of a quiet sensing band specification (described below).

The specifications from the MoU are adopted, with the following modifications:

- The actual numbers in this document are modified to apply to Universal Stylus performance.
- The frequency bands for EPS switching peak-to-peak voltage are modified as appropriate for the Universal Stylus model.
- An additional spec is added, which is the requirement for a quiet sensing band to allow for Universal Stylus operation.

The model adopted is illustrated in Figure K-1. The charger is modeled as a DC voltage supply affected by some interference waveform V_I between its 0V line and earth ground. It may have some internal impedance Z_{EARTH} which limits the charge pushing capability of the interference source. The Controller is modeled as a resistive load with some capacitance to earth ground C_D. The capacitance of the user touching the screen is labeled as C_F. It is the interference voltage waveform across this capacitance V_F that is of interest to the touch system.

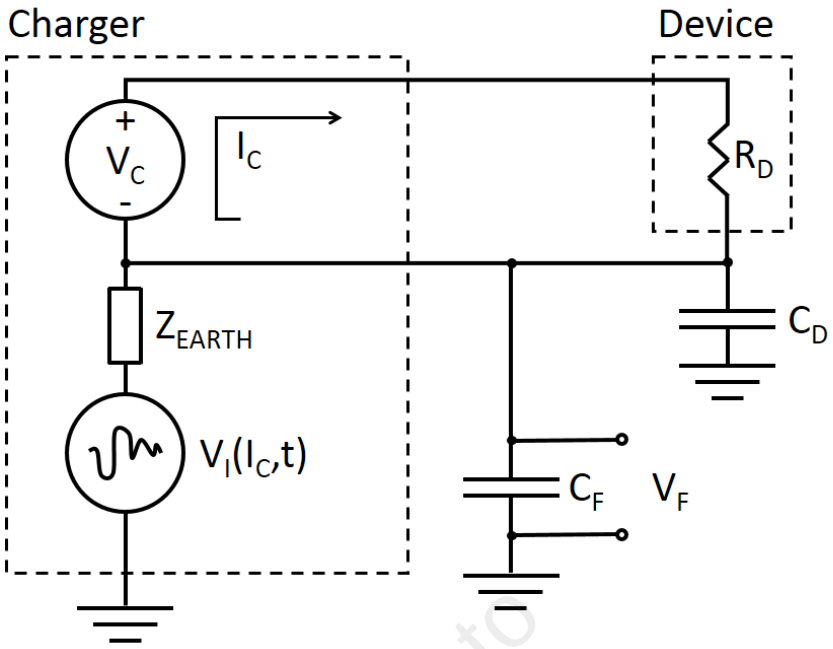


Figure K-1 Charger model used in this specification.

The detailed testing procedure is specified at the end of this document.

The requirements on this model are specified in the following table:

Spec Name	Unit	Spec Value
Max AC V_{pp}	Volts	200
Max EPS V_{pp} (1kHz - 50kHz)	Volts	3.3
Max EPS V_{pp} (50-200kHz)	Volts	3
Max EPS V_{pp} (200-600kHz)	Volts	1
Max EPS V_{pp} (600kHz-2MHz)	Volts	1
One Quiet Band V_{pp} (100kHz band, between 100-500kHz)	Volts	0.002
Max EPS Slew in any 1µs window	Volts	3

Note that only one quiet band is required by the spec. It must be located between 100-500kHz and be 50kHz wide.

These specifications must be met at all currents that might be drawn from the power supply.

K.3 Assumptions

This specification makes the following assumptions:

- A charger is adequately described as a voltage source across its outputs up to some maximum power rating, and
- The mechanism of interference coupling from the charger into the capacitive touch system is purely due to the floating ground of the device, as compared to the well-grounded user.

K.4 Measurement

The European standard includes a description of the testing method. This method is used, with some modifications. A schematic of the test set-up is shown in Figure K-2.

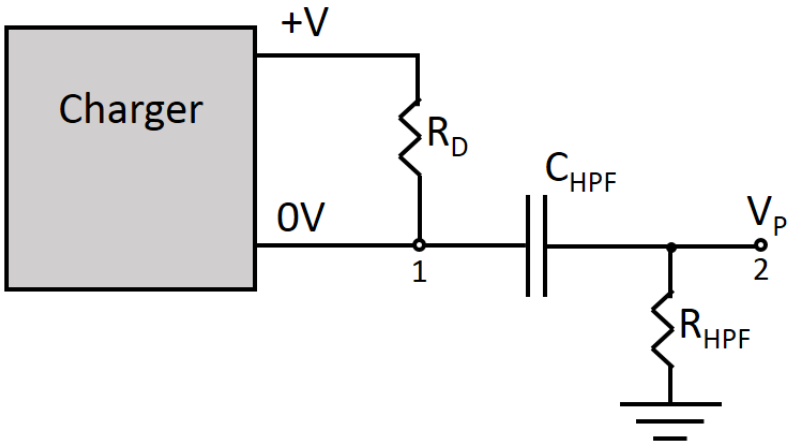


Figure K-2 Schematic drawing of the test set-up

The following materials are required for this test:

- The charger to be tested.
- An AC power supply for the chargers capable of supplying $121V_{rms}$ and $242V_{rms}$ at 60Hz with sufficient current to run the chargers.
- A set of power resistors to use as R_D.
- A 100pF capacitor for C_{HPF} and a 250kΩ resistor for R_{HPF}.
- A digital oscilloscope for capturing waveforms of the voltage of interest V_P with a sampling rate of at least 5MS/s of time length of at least 20ms. The probes should have 10MΩ resistance and 15pF capacitance.
- Software for digitally filtering the data captured by the digital oscilloscope.

This is the testing procedure:

1. Connect the charger to the AC power supply, with voltage set to the maximum rated AC voltage of the charger.
2. Measure the AC Vpp. This is done by using the test set-up shown in Figure K-2, with C_{HPF}=100pF, and R_{HPF}=250kΩ. The oscilloscope probe is used to probe the point marked "1" in the figure (ie, the non-high pass filtered signal). A 20ms traced is captured, and the maximum peak-to-peak voltage swing is measured.
3. Measure the remaining parameters. This is done by using the same test set-up, but probing point "2" instead of point "1". The signal is now high-pass filtered. A 20ms trace is captured with the digital oscilloscope. The following calculations are done:
    a. The Max EPS Vpp values are measured by applying a digital low-pass filter (for the <50kHz band) or a digital band-pass filter (for the other bands). The filters will have no more than 1dB attenuation in the pass-band and no less than 60dB attenuation in the stop-bands. The stop-bands start 5kHz away from the pass-bands.
    b. The quiet band is measured by filtering the data with a set of digital band-pass filters and measuring the Vpp of the results. The filters will have a pass-band of width 50kHz, no more

than 1dB attenuation in the pass-band, and no less than 60dB attenuation in the stop-bands. The stop-bands start 5kHz away from the pass-bands. The central frequencies of the filters will range from 125kHz to 475kHz in 5kHz increments.

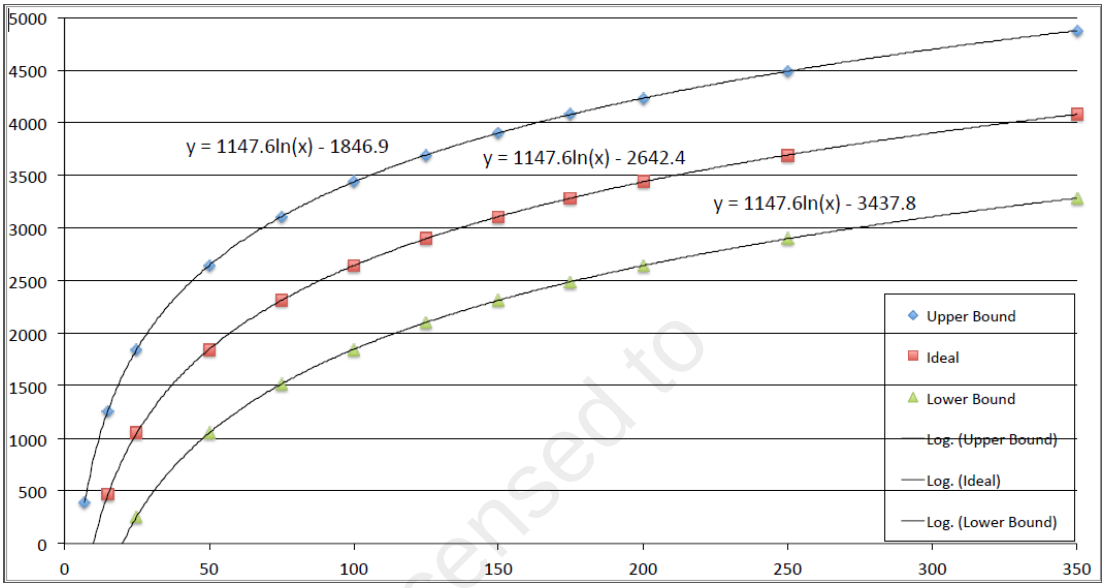
c. The slew rate is measured by finding the maximum voltage change over a 1µs time interval in the time series (without digital filtering).

4. Repeat steps 2 and 3 with a variety of R_D values, so to test the range of allowable I_D.

Appendix L – Pressure Log Curve Profile

The following graphs represent the log curve profiles that shall be used by the stylus to convert the raw gram force value (X-axis) to the 12-bit unsigned value (Y-axis) reported by the Stylus to the Controller.

The graph shows an ideal curve (in middle) and the upper bounds and the lower bounds of the curves. The actual curve used by the Stylus must be within these bounds.



The constants used in creating the above profile are stated below.

Upper Bound : $y = 1147.6\ln(x) - 1846.9$

Ideal : $y = 1147.6\ln(x) - 2642.4$

Lower Bound : $y = 1147.6\ln(x) - 3437.8$

The reported levels of pressure shall fall within the tolerance described above.

A logarithmic fit to the data shall have an r^2 goodness of fit of 90% or greater (it has to be a good fit to a log curve).

A linear fit to the data must have an r^2 goodness of fit of less than the logarithmic goodness of fit (the curve must be more logarithmic than linear).