User-Space DTX (Clipboard Detachment System) Interface

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 3)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 3)

Substitution definition contains illegal element <problematic>:

```
<problematic ids="id2" refid="id1">
    :c:type:`__u16 <__u16>`

.. |__u16| replace:: :c:type:`__u16 <__u16>`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 4)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 4)

Substitution definition contains illegal element <problematic>:

```
<problematic ids="id4" refid="id3">
    :c:type:`struct sdtx_event <sdtx_event>`

.. |sdtx_event| replace:: :c:type:`struct sdtx_event <sdtx_event>`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 5)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 5)

Substitution definition contains illegal element <problematic>:

```
<problematic ids="id6" refid="id5">
    :c:type:`enum sdtx_event_code <sdtx_event_code>`

.. |sdtx_event_code| replace:: :c:type:`enum sdtx_event_code <sdtx_event_code>`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 6)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 6)

Substitution definition contains illegal element <problematic>:

```
<problematic ids="id8" refid="id7">
    :c:type:`struct sdtx_base_info <sdtx_base_info>`

.. |sdtx_base_info| replace:: :c:type:`struct sdtx_base_info <sdtx_base_info>`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 7)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 7)

Substitution definition contains illegal element <problematic>:

```
<problematic ids="id10" refid="id9">
    :c:type:`struct sdtx_device_mode <sdtx_device_mode>`

.. |sdtx_device_mode| replace:: :c:type:`struct sdtx_device_mode <sdtx_device_mode>`
```

The `surface_dtx` driver is responsible for proper clipboard detachment and re-attachment handling. To this end, it provides the `/dev/surface/dtx` device file, through which it can interface with a user-space daemon. This daemon is then ultimately responsible for determining and taking necessary actions, such as unmounting devices attached to the base, unloading/reloading the graphics-driver, user-notifications, etc.

There are two basic communication principles used in this driver: Commands (in other parts of the documentation also referred to as requests) and events. Commands are sent to the EC and may have a different implications in different contexts. Events are sent by the EC upon some internal state change. Commands are always driver-initiated, whereas events are always initiated by the EC.

Contents

Nomenclature

- **Clipboard:** The detachable upper part of the Surface Book, housing the screen and CPU.
- **Base:** The lower part of the Surface Book from which the clipboard can be detached, optionally (model dependent) housing the discrete GPU (dGPU).

- **Latch:** The mechanism keeping the clipboard attached to the base in normal operation and allowing it to be detached when requested.
- **Silently ignored commands:** The command is accepted by the EC as a valid command and acknowledged (following the standard communication protocol), but the EC does not act upon it, i.e. ignores it.e upper part of the

Detachment Process

Warning: This part of the documentation is based on reverse engineering and testing and thus may contain errors or be incomplete.

Latch States

The latch mechanism has two major states: *open* and *closed*. In the *closed* state (default), the clipboard is secured to the base, whereas in the *open* state, the clipboard can be removed by a user.

The latch can additionally be locked and, correspondingly, unlocked, which can influence the detachment procedure. Specifically, this locking mechanism is intended to prevent the dGPU, positioned in the base of the device, from being hot-unplugged while in use. More details can be found in the documentation for the detachment procedure below. By default, the latch is unlocked.

Detachment Procedure

Note that the detachment process is governed fully by the EC. The `surface_dtx` driver only relays events from the EC to user-space and commands from user-space to the EC, i.e. it does not influence this process.

The detachment process is started with the user pressing the *detach* button on the base of the device or executing the `SDTX_IOCTL_LATCH_REQUEST` IOCTL. Following that:

1. The EC turns on the indicator led on the detach-button, sends a *detach-request* event (`SDTX_EVENT_REQUEST`), and awaits further instructions/commands. In case the latch is unlocked, the led will flash green. If the latch has been locked, the led will be solid red
2. The event is, via the `surface_dtx` driver, relayed to user-space, where an appropriate user-space daemon can handle it and send instructions back to the EC via IOCTLs provided by this driver.
3. The EC waits for instructions from user-space and acts according to them. If the EC does not receive any instructions in a given period, it will time out and continue as follows:
   - If the latch is unlocked, the EC will open the latch and the clipboard can be detached from the base. This is the exact behavior as without this driver or any user-space daemon. See the `SDTX_IOCTL_LATCH_CONFIRM` description below for more details on the follow-up behavior of the EC.
   - If the latch is locked, the EC will *not* open the latch, meaning the clipboard cannot be detached from the base. Furthermore, the EC sends an cancel event (`SDTX_EVENT_CANCEL`) detailing this with the cancel reason `SDTX_DETACH_TIMEDOUT` (see :ref:`events` for details).

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst`, **line 98**); *backlink*
>
> Unknown interpreted text role "ref".

Valid responses by a user-space daemon to a detachment request event are:

- Execute `SDTX_IOCTL_LATCH_REQUEST`. This will immediately abort the detachment process. Furthermore, the EC will send a detach-request event, similar to the user pressing the detach-button to cancel said process (see below).
- Execute `SDTX_IOCTL_LATCH_CONFIRM`. This will cause the EC to open the latch, after which the user can separate clipboard and base.

  As this changes the latch state, a *latch-status* event (`SDTX_EVENT_LATCH_STATUS`) will be sent once the latch has been opened successfully. If the EC fails to open the latch, e.g. due to hardware error or low battery, a latch-cancel event (`SDTX_EVENT_CANCEL`) will be sent with the cancel reason indicating the specific failure.

  If the latch is currently locked, the latch will automatically be unlocked before it is opened.
- Execute `SDTX_IOCTL_LATCH_HEARTBEAT`. This will reset the internal timeout. No other actions will be performed, i.e. the detachment process will neither be completed nor canceled, and the EC will still be waiting for further responses.
- Execute `SDTX_IOCTL_LATCH_CANCEL`. This will abort the detachment process, similar to `SDTX_IOCTL_LATCH_REQUEST`, described above, or the button press, described below. A *generic request* event (`SDTX_EVENT_REQUEST`) is send in response to this. In contrast to those, however, this command does not trigger a new detachment process if none is currently in progress.
- Do nothing. The detachment process eventually times out as described in point 3.

See :ref:`ioctls` for more details on these responses.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 137); *backlink*

Unknown interpreted text role "ref".

It is important to note that, if the user presses the detach button at any point when a detachment operation is in progress (i.e. after the EC has sent the initial *detach-request* event (`SDTX_EVENT_REQUEST`) and before it received the corresponding response concluding the process), the detachment process is canceled on the EC-level and an identical event is being sent. Thus a *detach-request* event, by itself, does not signal the start of the detachment process.

The detachment process may further be canceled by the EC due to hardware failures or a low clipboard battery. This is done via a cancel event (`SDTX_EVENT_CANCEL`) with the corresponding cancel reason.

User-Space Interface Documentation

Error Codes and Status Values

Error and status codes are divided into different categories, which can be used to determine if the status code is an error, and, if it is, the severity and type of that error. The current categories are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 162)

Unknown directive type "flat-table".

```
.. flat-table:: Overview of Status/Error Categories.
   :widths: 2 1 3
   :header-rows: 1

   * - Name
     - Value
     - Short Description

   * - ``STATUS``
     - ``0x0000``
     - Non-error status codes.

   * - ``RUNTIME_ERROR``
     - ``0x1000``
     - Non-critical runtime errors.

   * - ``HARDWARE_ERROR``
     - ``0x2000``
     - Critical hardware failures.

   * - ``UNKNOWN``
     - ``0xF000``
     - Unknown error codes.
```

Other categories are reserved for future use. The `SDTX_CATEGORY()` macro can be used to determine the category of any status value. The `SDTX_SUCCESS()` macro can be used to check if the status value is a success value (`SDTX_CATEGORY_STATUS`) or if it indicates a failure.

Unknown status or error codes sent by the EC are assigned to the `UNKNOWN` category by the driver and may be implemented via their own code in the future.

Currently used error codes are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 197)

Unknown directive type "flat-table".

```
.. flat-table:: Overview of Error Codes.
   :widths: 2 1 1 3
   :header-rows: 1

   * - Name
```

```
      - Category
      - Value
      - Short Description

    * - ``SDTX_DETACH_NOT_FEASIBLE``
      - ``RUNTIME``
      - ``0x1001``
      - Detachment not feasible due to low clipboard battery.

    * - ``SDTX_DETACH_TIMEDOUT``
      - ``RUNTIME``
      - ``0x1002``
      - Detachment process timed out while the latch was locked.

    * - ``SDTX_ERR_FAILED_TO_OPEN``
      - ``HARDWARE``
      - ``0x2001``
      - Failed to open latch.

    * - ``SDTX_ERR_FAILED_TO_REMAIN_OPEN``
      - ``HARDWARE``
      - ``0x2002``
      - Failed to keep latch open.

    * - ``SDTX_ERR_FAILED_TO_CLOSE``
      - ``HARDWARE``
      - ``0x2003``
      - Failed to close latch.
```

Other error codes are reserved for future use. Non-error status codes may overlap and are generally only unique within their use-case:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 234)

Unknown directive type "flat-table".

```
.. flat-table:: Latch Status Codes.
   :widths: 2 1 1 3
   :header-rows: 1

   * - Name
     - Category
     - Value
     - Short Description

   * - ``SDTX_LATCH_CLOSED``
     - ``STATUS``
     - ``0x0000``
     - Latch is closed/has been closed.

   * - ``SDTX_LATCH_OPENED``
     - ``STATUS``
     - ``0x0001``
     - Latch is open/has been opened.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 253)

Unknown directive type "flat-table".

```
.. flat-table:: Base State Codes.
   :widths: 2 1 1 3
   :header-rows: 1

   * - Name
     - Category
     - Value
     - Short Description

   * - ``SDTX_BASE_DETACHED``
     - ``STATUS``
     - ``0x0000``
     - Base has been detached/is not present.
```

```
   * - ``SDTX_BASE_ATTACHED``
     - ``STATUS``
     - ``0x0001``
     - Base has been attached/is present.
```

Again, other codes are reserved for future use.

Events

Events can be received by reading from the device file. They are disabled by default and have to be enabled by executing `SDTX_IOCTL_EVENTS_ENABLE` first. All events follow the layout prescribed by |sdtx_event|. Specific event types can be identified by their event code, described in |sdtx_event_code|. Note that other event codes are reserved for future use, thus an event parser must be able to handle any unknown/unsupported event types gracefully, by relying on the payload length given in the event header.

Currently provided event types are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 289)

Unknown directive type "flat-table".

```
.. flat-table:: Overview of DTX events.
   :widths: 2 1 1 3
   :header-rows: 1

   * - Name
     - Code
     - Payload
     - Short Description

   * - ``SDTX_EVENT_REQUEST``
     - ``1``
     - ``0`` bytes
     - Detachment process initiated/aborted.

   * - ``SDTX_EVENT_CANCEL``
     - ``2``
     - ``2`` bytes
     - EC canceled detachment process.

   * - ``SDTX_EVENT_BASE_CONNECTION``
     - ``3``
     - ``4`` bytes
     - Base connection state changed.

   * - ``SDTX_EVENT_LATCH_STATUS``
     - ``4``
     - ``2`` bytes
     - Latch status changed.

   * - ``SDTX_EVENT_DEVICE_MODE``
     - ``5``
     - ``2`` bytes
     - Device mode changed.
```

Individual events in more detail:

SDTX_EVENT_REQUEST

Sent when a detachment process is started or, if in progress, aborted by the user, either via a detach button press or a detach request (`SDTX_IOCTL_LATCH_REQUEST`) being sent from user-space.

Does not have any payload.

SDTX_EVENT_CANCEL

Sent when a detachment process is canceled by the EC due to unfulfilled preconditions (e.g. clipboard battery too low to detach) or hardware failure. The reason for cancellation is given in the event payload detailed below and can be one of

- `SDTX_DETACH_TIMEDOUT`: Detachment timed out while the latch was locked. The latch has neither been opened nor unlocked.
- `SDTX_DETACH_NOT_FEASIBLE`: Detachment not feasible due to low clipboard battery.
- `SDTX_ERR_FAILED_TO_OPEN`: Could not open the latch (hardware failure).

- SDTX_ERR_FAILED_TO_REMAIN_OPEN: Could not keep the latch open (hardware failure).
- SDTX_ERR_FAILED_TO_CLOSE: Could not close the latch (hardware failure).

Other error codes in this context are reserved for future use.

These codes can be classified via the SDTX_CATEGORY() macro to discern between critical hardware errors (SDTX_CATEGORY_HARDWARE_ERROR) or runtime errors (SDTX_CATEGORY_RUNTIME_ERROR), the latter of which may happen during normal operation if certain preconditions for detachment are not given.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 363)
>
> Unknown directive type "flat-table".
>
> ```
> .. flat-table:: Detachment Cancel Event Payload
>    :widths: 1 1 4
>    :header-rows: 1
>
>    * - Field
>      - Type
>      - Description
>
>    * - ``reason``
>      - |__u16|
>      - Reason for cancellation.
> ```

SDTX_EVENT_BASE_CONNECTION

Sent when the base connection state has changed, i.e. when the base has been attached, detached, or detachment has become infeasible due to low clipboard battery. The new state and, if a base is connected, ID of the base is provided as payload of type |sdtx_base_info| with its layout presented below:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 384)
>
> Unknown directive type "flat-table".
>
> ```
> .. flat-table:: Base-Connection-Change Event Payload
>    :widths: 1 1 4
>    :header-rows: 1
>
>    * - Field
>      - Type
>      - Description
>
>    * - ``state``
>      - |__u16|
>      - Base connection state.
>
>    * - ``base_id``
>      - |__u16|
>      - Type of base connected (zero if none).
> ```

Possible values for state are:

- SDTX_BASE_DETACHED,
- SDTX_BASE_ATTACHED, and
- SDTX_DETACH_NOT_FEASIBLE.

Other values are reserved for future use.

SDTX_EVENT_LATCH_STATUS

Sent when the latch status has changed, i.e. when the latch has been opened, closed, or an error occurred. The current status is provided as payload:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 414)
>
> Unknown directive type "flat-table".

```
.. flat-table:: Latch-Status-Change Event Payload
   :widths: 1 1 4
   :header-rows: 1

   * - Field
     - Type
     - Description

   * - ``status``
     - |__u16|
     - Latch status.
```

Possible values for `status` are:

- `SDTX_LATCH_CLOSED`,
- `SDTX_LATCH_OPENED`,
- `SDTX_ERR_FAILED_TO_OPEN`,
- `SDTX_ERR_FAILED_TO_REMAIN_OPEN`, and
- `SDTX_ERR_FAILED_TO_CLOSE`.

Other values are reserved for future use.

`SDTX_EVENT_DEVICE_MODE`

Sent when the device mode has changed. The new device mode is provided as payload:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 442)

Unknown directive type "flat-table".

```
.. flat-table:: Device-Mode-Change Event Payload
   :widths: 1 1 4
   :header-rows: 1

   * - Field
     - Type
     - Description

   * - ``mode``
     - |__u16|
     - Device operation mode.
```

Possible values for `mode` are:

- `SDTX_DEVICE_MODE_TABLET`,
- `SDTX_DEVICE_MODE_LAPTOP`, and
- `SDTX_DEVICE_MODE_STUDIO`.

Other values are reserved for future use.

IOCTLs

The following IOCTLs are provided:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 469)

Unknown directive type "flat-table".

```
.. flat-table:: Overview of DTX IOCTLs
   :widths: 1 1 1 1 4
   :header-rows: 1

   * - Type
     - Number
     - Direction
     - Name
     - Description

   * - ``0xA5``
     - ``0x21``
     - ``-``
```

```
    - ``EVENTS_ENABLE``
    - Enable events for the current file descriptor.

  * - ``0xA5``
    - ``0x22``
    - ``-``
    - ``EVENTS_DISABLE``
    - Disable events for the current file descriptor.

  * - ``0xA5``
    - ``0x23``
    - ``-``
    - ``LATCH_LOCK``
    - Lock the latch.

  * - ``0xA5``
    - ``0x24``
    - ``-``
    - ``LATCH_UNLOCK``
    - Unlock the latch.

  * - ``0xA5``
    - ``0x25``
    - ``-``
    - ``LATCH_REQUEST``
    - Request clipboard detachment.

  * - ``0xA5``
    - ``0x26``
    - ``-``
    - ``LATCH_CONFIRM``
    - Confirm clipboard detachment request.

  * - ``0xA5``
    - ``0x27``
    - ``-``
    - ``LATCH_HEARTBEAT``
    - Send heartbeat signal to EC.

  * - ``0xA5``
    - ``0x28``
    - ``-``
    - ``LATCH_CANCEL``
    - Cancel detachment process.

  * - ``0xA5``
    - ``0x29``
    - ``R``
    - ``GET_BASE_INFO``
    - Get current base/connection information.

  * - ``0xA5``
    - ``0x2A``
    - ``R``
    - ``GET_DEVICE_MODE``
    - Get current device operation mode.

  * - ``0xA5``
    - ``0x2B``
    - ``R``
    - ``GET_LATCH_STATUS``
    - Get current device latch status.
```

SDTX_IOCTL_EVENTS_ENABLE

Defined as `_IO(0xA5, 0x22)`.

Enable events for the current file descriptor. Events can be obtained by reading from the device, if enabled. Events are disabled by default.

SDTX_IOCTL_EVENTS_DISABLE

Defined as `_IO(0xA5, 0x22)`.

Disable events for the current file descriptor. Events can be obtained by reading from the device, if enabled. Events are disabled by default.

SDTX_IOCTL_LATCH_LOCK

Defined as `_IO(0xA5, 0x23)`.

Locks the latch, causing the detachment procedure to abort without opening the latch on timeout. The latch is unlocked by default. This command will be silently ignored if the latch is already locked.

SDTX_IOCTL_LATCH_UNLOCK

Defined as `_IO(0xA5, 0x24)`.

Unlocks the latch, causing the detachment procedure to open the latch on timeout. The latch is unlocked by default. This command will not open the latch when sent during an ongoing detachment process. It will be silently ignored if the latch is already unlocked.

SDTX_IOCTL_LATCH_REQUEST

Defined as `_IO(0xA5, 0x25)`.

Generic latch request. Behavior depends on the context: If no detachment-process is active, detachment is requested. Otherwise the currently active detachment-process will be aborted.

If a detachment process is canceled by this operation, a generic detachment request event (`SDTX_EVENT_REQUEST`) will be sent.

This essentially behaves the same as a detachment button press.

SDTX_IOCTL_LATCH_CONFIRM

Defined as `_IO(0xA5, 0x26)`.

Acknowledges and confirms a latch request. If sent during an ongoing detachment process, this command causes the latch to be opened immediately. The latch will also be opened if it has been locked. In this case, the latch lock is reset to the unlocked state.

This command will be silently ignored if there is currently no detachment procedure in progress.

SDTX_IOCTL_LATCH_HEARTBEAT

Defined as `_IO(0xA5, 0x27)`.

Sends a heartbeat, essentially resetting the detachment timeout. This command can be used to keep the detachment process alive while work required for the detachment to succeed is still in progress.

This command will be silently ignored if there is currently no detachment procedure in progress.

SDTX_IOCTL_LATCH_CANCEL

Defined as `_IO(0xA5, 0x28)`.

Cancels detachment in progress (if any). If a detachment process is canceled by this operation, a generic detachment request event (`SDTX_EVENT_REQUEST`) will be sent.

This command will be silently ignored if there is currently no detachment procedure in progress.

SDTX_IOCTL_GET_BASE_INFO

Defined as `_IOR(0xA5, 0x29, struct sdtx_base_info)`.

Get the current base connection state (i.e. attached/detached) and the type of the base connected to the clipboard. This is command essentially provides a way to query the information provided by the base connection change event (`SDTX_EVENT_BASE_CONNECTION`).

Possible values for `struct sdtx_base_info.state` are:

- `SDTX_BASE_DETACHED`,
- `SDTX_BASE_ATTACHED`, and
- `SDTX_DETACH_NOT_FEASIBLE`.

Other values are reserved for future use.

SDTX_IOCTL_GET_DEVICE_MODE

Defined as `_IOR(0xA5, 0x2A, __u16)`.

Returns the device operation mode, indicating if and how the base is attached to the clipboard. This is command essentially provides a way to query the information provided by the device mode change event (`SDTX_EVENT_DEVICE_MODE`).

Returned values are:

- `SDTX_DEVICE_MODE_LAPTOP`
- `SDTX_DEVICE_MODE_TABLET`
- `SDTX_DEVICE_MODE_STUDIO`

See |sdtx_device_mode| for details. Other values are reserved for future use.

SDTX_IOCTL_GET_LATCH_STATUS

Defined as `_IOR(0xA5, 0x2B, __u16)`.

Get the current latch status or (presumably) the last error encountered when trying to open/close the latch. This is command essentially provides a way to query the information provided by the latch status change event (`SDTX_EVENT_LATCH_STATUS`).

Returned values are:

- `SDTX_LATCH_CLOSED`,
- `SDTX_LATCH_OPENED`,
- `SDTX_ERR_FAILED_TO_OPEN`,
- `SDTX_ERR_FAILED_TO_REMAIN_OPEN`, and
- `SDTX_ERR_FAILED_TO_CLOSE`.

Other values are reserved for future use.

A Note on Base IDs

Base types/IDs provided via `SDTX_EVENT_BASE_CONNECTION` or `SDTX_IOCTL_GET_BASE_INFO` are directly forwarded from the EC in the lower byte of the combined |__u16| value, with the driver storing the EC type from which this ID comes in the high byte (without this, base IDs over different types of ECs may be overlapping).

The `SDTX_DEVICE_TYPE()` macro can be used to determine the EC device type. This can be one of

- `SDTX_DEVICE_TYPE_HID`, for Surface Aggregator Module over HID, and
- `SDTX_DEVICE_TYPE_SSH`, for Surface Aggregator Module over Surface Serial Hub.

Note that currently only the `SSH` type EC is supported, however `HID` type is reserved for future use.

Structures and Enums

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 712)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/surface_aggregator/dtx.h
```

API Users

A user-space daemon utilizing this API can be found at https://github.com/linux-surface/surface-dtx-daemon.

Docutils System Messages

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 279); *backlink*

Undefined substitution referenced: "sdtx_event".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 279); *backlink*

Undefined substitution referenced: "sdtx_event_code".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 378); *backlink*

Undefined substitution referenced: "sdtx_base_info".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 665); *backlink*

Undefined substitution referenced: "sdtx_device_mode".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\surface_aggregator\clients\(linux-master)(Documentation)(driver-api)(surface_aggregator)(clients)dtx.rst, line 692); *backlink*

Undefined substitution referenced: "__u16".