VTAP

Settings and Commands Reference

Core Firmware from v2.3.0.2
Connectivity Firmware from v1.0.4.3

VTAP25, VTAP50 and VTAP100

Revised June 2025 v4.21

DOT ORIGIN

If you need help to set up or use your VTAP reader, beyond what is contained in this Reference Guide, then please contact our support team.

Email: **vtap-support@dotorigin.com**

Download the latest documentation and firmware from **https://vtapnfc.com**

Telephone UK and Europe: +44 (0) 1428 685861

Telephone North America and Latin America: +1 (562) 262-9642

If you have any feedback on setting up or using your VTAP reader or this documentation, then please contact our support team. The product is constantly being reviewed and improved and we value feedback about your experience.

Dot Origin Ltd
Unit 7, Coopers Place Business Park, Combe Lane, Wormley
Godalming GU8 5SZ United Kingdom
+44 (0) 1428 685861

Contents

1 Using this guide

Settings and commands which have been added since the last publication of this guide are highlighted as NEW. Settings and commands with additional options or changed defaults in this version are highlighted as NEW OPTIONS.

You must be running the latest VTAP core firmware v2.3.0.2 or later (and on VTAP PRO devices the latest VTAP connectivity firmware v1.0.4.3) to access all of the highlighted commands or settings.

Core firmware and connectivity firmware

All VTAP readers use the same VTAP core firmware, although some settings and commands only have an effect on certain hardware, which is noted in this guide when it occurs.

IP-connected readers such as the VTAP100 PRO contain an additional processor module responsible for wireless or Ethernet comms, which runs separate VTAP connectivity firmware. This firmware is used in both Local and Cloud modes.

Visit **https://vtapnfc.com** if you need to download the latest firmware. (The API level quoted against each command is an alternative to core firmware version, which can be more easily queried and interpreted automatically by software. The full functionality of a command will only be available from VTAP readers running the stated API level or higher.) VTAP connectivity firmware can also be downloaded to update a VTAP reader locally, but VTAP readers in Cloud mode will be automatically updated via the VTAP Cloud.

If you are new to using a VTAP reader, you will find much more help in the VTAP Configuration Guide.

2 Essential files and processes

The VTAP Configuration Guide provides an introduction to using a VTAP reader, supplemented by VTAP Application Notes in more specialist areas.

There are a number of files and processes which are essential to operating your VTAP readers. Descriptions of those processes are repeated here for your convenience:

- Update firmware
- Add or delete keys
- Check status in BOOT.TXT

If you are new to using the VTAP reader, you will find much more help in the VTAP Configuration Guide.

2.1 Update firmware

To update the firmware on a VTAP reader:

1. Copy the core firmware image file `vtapware.dat` to the VTAP mass storage device. (It will be called `firmware.dat` for VTAP100 if the hardware version is v4 or earlier. You can identify the hardware revision if you check status in BOOT.TXT or use the `?b` command over a serial interface.)

   Note: The configuration process is the same for VTAP25, VTAP50 and VTAP100, but files on a VTAP reader can be specific to the type and hardware version of VTAP reader and are not always interchangeable.

   If you have an IP-connected reader, such as a VTAP PRO reader in Local mode, you can add a new VTAP connectivity firmware image file `vtapcfw.dat` in exactly the same way, but only update one firmware file at a time.

2. Reboot the device, either by using the `?REBOOT` command over a serial interface, by briefly disconnecting power.

   There will be a delay of a couple seconds when the VTAP reader boots up again and performs the update, then it will continue to operate as normal.

3. You can **Check status in BOOT.TXT** to see if the firmware version is as you expect.

If the new firmware does not work, reload the previous firmware.

2.2 Add or delete key files

There are two types of key that you may need to add to your VTAP reader:

- **Private keys** - the private key from each ECC key pair you have generated to secure your mobile wallet passes.

  Each is stored in a file with the name `private#.pem`, following the `.pem` format, where `#` is replaced with a number from `1` to `6`, matching the key slot you will save it in. (The demo passes are accessed using the key in KeySlot 6, so don't overwrite this one unless you are finished with Dot Origin demo passes.)

  The text of a `.pem` format file will look something like this:

```
-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIIEtIyvdGuRj+gRrTPn7+wpQ7XAhWfLAfmBzhtzjdrnQoAoGCCqGSM49
AwEHoUQDQgAEYzDKBwanQZs1TtuTsmrkpYjow8idfqMd0U/lwfpBdtqIjcCRowD1
lznasiT971AkZqvOZBfZTRnnjNBMu1uXzg==
-----END EC PRIVATE KEY-----
```

> **Note:** A VTAP reader cannot use more than 6 private key files.

  If you are not sure how to generate an ECC key pair please refer to the Application Note on ECC key pairs for more information.

- **Application keys** - these include AES or DESFire application keys to access NFC card data, or an OSDP secure channel base key.

  Each of your application keys must be saved in a text file, with the name `appkey#.txt`, where `#` is replaced with a number from `1` to `9`, matching the key slot you will save it in. Each text file should just contain one application key, in the appropriate format for its application, for example 32 hex digits in the case of a DESFire application key.

  The text of an application key text file will always start with `key=`and look something like:

```
key=bd6a15d1039e7527edfd01f37a220f3e
```

> **Note:** A VTAP reader cannot use more than 9 application key files.

2.2.1 Add a key

The same process is followed to add any type of key.

Step 1: Choose your key slot and name your key file appropriately

Check the `BOOT.TXT` file on your VTAP reader (or use a `?b` or `?info` command over a serial interface) to find out which key slots are already in use, before you add a new key. The number used in the filename of your key determines the key slot where that key will be saved.

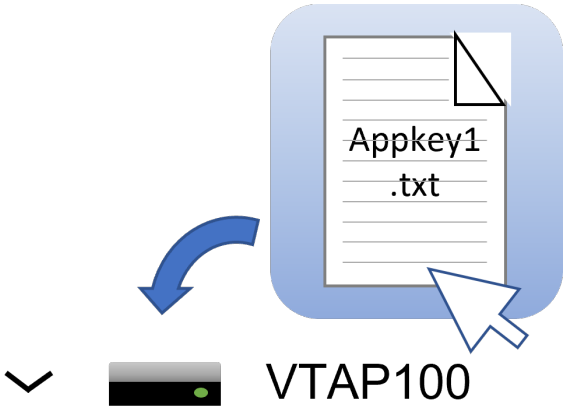
Note: If your key filename assigns a key to a key slot that is already in use, you will simply overwrite the current key in that slot with your new one.

Private keys must be in files named `private#.pem` and application keys in files named `appkey#.txt`, with `#` replaced by a number `1` to `6` for private keys or `1` to `9` for app keys, to identify your chosen key slot.

VTAP25 reader modules have a secure element, which is used by default to store all private keys, with EC encryption. (In all other VTAP readers these keys are stored in encrypted flash, as is the default for VTAP25 application keys.)

Step 2: Move or copy your key file to your VTAP reader

Load your keys by copying these files onto your VTAP reader. Just connect your VTAP reader to a PC via USB, so it appears as a mass storage device on the PC file system (unless disabled or locked), then you can drag and drop the files.



Alternatively, transfer the file to your VTAP reader using Zmodem to transfer over a serial interface.

Step 3: Load your key by rebooting or use a `?KEYLOAD` command - file disappears from filesystem

Power cycle your VTAP reader. (Disconnect or safely remove the drive from the PC then reconnect it.) When you reboot the VTAP reader your key will have been stored in hardware, and will no longer be listed as a file on the device.

Alternatively, send a `?KEYLOAD` command over a serial interface for the key to be loaded, to avoid the need to restart the VTAP reader.

Note: If you have not followed the `.pem` format exactly you will find that the VTAP reader will not consume the key when rebooted - it will remain visible in the file system. Check your key file text and try again.

2.2.2 Delete a key

To clear a key from a particular key slot, without replacing it with a new key, you prepare a special type of key file. The format for that file depends on the key type you need to clear.

Delete a private key

To delete a private key from slot `#` you create a key file called `private#.pem`. The content of this file is a single line:

```
-----DELETE EC PRIVATE KEY-----
```

Now add this special deletion key to the private key slot you need to clear, following the **Add a key** instructions above.

Delete an application key

To delete an application key from slot `#` you create a key file called `appkey#.txt`. The content of this file is a single line:

```
key=DELETE
```

Now add this special deletion key to the application key slot you need to clear, following the **Add a key** instructions above.

2.3 Check status in BOOT.TXT

If you navigate to the VTAP reader in the computer's file system. It will appear as an attached mass storage device and list the files contained, including the BOOT.TXT file.

Inspecting BOOT.TXT will give you essential information about your VTAP reader set up, at time of last reboot, which might be helpful when troubleshooting.

```
VTAP100
Boot time: 2001/01/01 00:00:00
Firmware: V2.2.5.0
Storage: Dataflash
Status: 0
Hardware: 5.01
Expansion: VTAP100C-V1-a2
VCP enabled
NCI: 0471125005-8C00
Serial number: 563230-798AEC17D053C05ADE6F6C36C79A6B12
VTAP label: CC123456
API level: 4
AppKeys used: 123------
```

Figure 2-1 Example VTAP100 v5 BOOT.TXT file

You are most likely to need:

- 'Serial number' ('ATCA' on VTAP100 v4a or earlier) - the hardware serial number for your VTAP reader.
- 'VTAP label' (if set) - the assigned serial number for your VTAP reader, which matches that on its label. This will not show if not set.
- 'Firmware' - the VTAP reader core firmware version in use. You will find the latest firmware versions at **https://www.vtapnfc.com/download/**
- 'Hardware' - the VTAP reader hardware version in use.
- 'API level' - indicates which serial or OSDP API commands are supported.
- 'KeySlots used:' - Indicates the ECC private keys loaded on the VTAP reader, to access VAS or Smart Tap passes. Helps you check whether you have uploaded the necessary ECC private keys, which can be unclear as the files are deleted when they are uploaded. These two examples show how to read this information:
  - 'KeySlots used:------' shows that no keys have been uploaded.
  - 'KeySlots used: 12--56' shows that key files 1 and 2 have been successfully uploaded, in addition to the defaults 5 and 6.
- 'AppKeys used:' Indicates the application keys (if any) uploaded to the VTAP reader for any other applications, such as keys loaded to use with DESFire applications.
- 'VCP enabled', if included - indicates that the virtual COM port has been enabled.

- 'Status' - should be 0 if operating normally, anything else indicates an error state.
- 'Expansion:' shows the name of the expansion board (if any) connected to the VTAP, for example: 'VTAP100W' for a Wiegand expansion board, 'VTAP100C' for a VTAP PRO BW expansion board, 'VTAP100E' for a VTAP PRO POE expansion board.
- 'Boot time' - The time at boot, which defaults to 1970/00/00 00:00:00 if power is removed to reboot.

If the configuration has been locked the `BOOT.TXT` file will end with the words LOCKED S/W or LOCKED H/W.

3 Settings

Settings are all of the parameters that can be set in a text file on your VTAP reader to control its operation. The settings are grouped according to which of the text files they can be used in. Most settings are used in the `config.txt` file.

3.1 config.txt

This section lists all valid settings for `config.txt`. `Config.txt` is a file which must contain the string `!VTAPconfig` at the start.

3.1.1 Apple VAS

Note: In these settings # is a number from 1 to 6, showing which settings form a group for reading each of 1 to 6 VAS passes linked to different Merchant IDs, which might be presented to this VTAP reader. If you have multiple IDs, the lowest numbered ID will be requested first, then continuing in ascending numeric order.

These settings are necessary to provide the VTAP reader with pairs of `VAS#MerchantID` and `VAS#KeySlot` that identify you as an Apple merchant entitled to read particular passes, and point to the appropriate private key to decode the data. You will separately need to upload the key files to those key slots. Help with this is included in **Add or delete keys**.

Settings below: **`VASKeySlot`** | **`VASMerchantID`** | **`VASMerchantURL`** | **`VASDefaultPassesEnabled`**

`VAS#KeySlot`

Definition:
An instruction to use the private key saved to a particular slot on the VTAP, when reading passes of type #.

Options:
`=1` to `=6`, identifying key file
`=0` or omitted (default), all available keys will be compared with the 4 byte hash of the public key for the data, to choose the right key.

If the data received by VTAP cannot be decrypted, the phone will register a pass read, but the data will not be output.

Default value: `=0`

Example value: `=2`

`VAS#MerchantID`

Definition:
An identifier supplied by a pass provider or customer to uniquely identify your passes in Apple Wallet applications

Options:

Default value: `N/A`

Example value: `=pass.com.pronto.fictionplc.demo`

VAS#MerchantURL

Definition:
A website URL to be visited when a pass of type # is presented. Not currently supported by iOS for VAS-only transactions.

Options:

Default value: `N/A`

Example value: `=https://dotorigin.com/`

VASDefaultPassesEnabled

Definition:
Used to restrict the number of VAS passes linked to different Merchant IDs, to be checked. Reduces the time spent testing each pass presented. Allows you to have alternative VAS pass details available in the configuration, ready to change over by serial command if or when they are needed.

Options:
In the example, only VAS2 and VAS3 passes will be enabled at startup

Default value: `=1,2,3,4,5,6`

Example value: `=2,3`

3.1.2 Google Smart Tap

A list of all valid configuration settings relating to Google Smart Tap controlling parameters in the `config.txt` file.

> **Note:** In these settings # is a number from 1 to 6, showing which settings form a group for reading each of 1 to 6 Smart Tap passes linked to different Collector IDs, which might be presented to this VTAP reader.

These settings are necessary to provide the VTAP reader with pairs of `ST#CollectorID` and `ST#KeySlot` that identify you as the Google merchant entitled to read particular passes, and point to the appropriate private key to decode the data. You will separately need to upload the key files to those key slots. Multiple collector IDs are not supported by Android, which means you cannot request more than one Collector ID from Google. Only one set should be live at any one time. Help with this is included in **Add or delete keys**.

Settings below: **STCollectorID** | **STKeySlot** | **STKeyVersion** | **STDefaultPassesEnabled**

ST#CollectorID

Definition:
An identifier supplied by Google to uniquely identify your passes in Google Wallet Smart Tap applications

Options:

Default value: `N/A`

Example value: `=97598013`

ST#KeySlot

Definition:
An instruction to use the private key saved to a particular slot on the VTAP, when reading passes of type #.

Options:
`=1` to `=6`, identifying key file.
`=0` or omitted (default) Google Wallet Smart Tap data will still be received and sent by the VTAP, only if the pass does not require authentication by the terminal.

Default value: `=0`

Example value: `=4`

ST#KeyVersion

Definition:
An instruction to use the private key with a particular version number saved on the VTAP, when reading passes of type #

Options:

Default value: `=0`

Example value: `=10`

STDefaultPassesEnabled

Definition:
Used to restrict the number of ST passes linked to different Collector IDs, to be checked. Reduces the time spent testing each pass presented. Allows you to have alternative ST pass details available in the configuration, ready to change over by serial command if or when they are needed.

Options:
In this example, only the ST5 pass will be enabled at startup

Default value: `=1,2,3,4,5,6`

Example value: `=5`

3.1.3 NFC card or tag

A list of all valid configuration settings which relate to extracting UID or block data from any type of card or tag, for the `config.txt` file. Be aware you may also need to use some type specific settings for **MIFARE Classic**, **MIFARE DESFire cards** and **NDEF records** which are grouped separately in the following sections.

The setting `NFCType2`, `NFCType4` or `NFCType5` is used to direct the VTAP reader to recognise and read particular types of data from the different NFC cards that may be presented. The settings prefixed `NDEFTag...` restrict the types of data that will be read.

`TagReadOffset` and `TagReadLength` are often used together, to define where to start extracting data from a data block and how much data to read.

If you choose `TagReadFormat=d` to require a decimal output of block data you can use `TagReadRightShift` to drop parity bits or `TagReadMinDigits` to add leading zeroes, when a fixed width output is required.

`TagByteOrder` can be used to reverse the byte order of data or UIDs, if that suits your application. Or you can use `TagByteOrderTypes` if you only want to reverse data for certain card type(s).

If `NFCType2=B` you can read up to 4 consecutive blocks of data from any NFC Type 2 cards/tags, for example MIFARE Ultralight and Ultralight C cards. It will also allow you to read block data from MIFARE Ultralight AES cards, authenticated using an AES key which is specified using `TagReadKeySlot`. Setting `TagReadKeyType=C` will enable CMAC verification, which may also be required (depending on your card configuration) for extra data security/integrity when using MIFARE Ultralight AES with authentication.

If `NFCType5=B` you can read block of data from NFC Type 5 cards/tags, for example ICODE cards. You will need to specify `TagReadBlockNum`, `TagReadLength`, `TagReadOffset`, `TagReadFormat` to define which blocks should be read.

Settings below: **`NFCType`** | **`NFCReportReadError`** | **`IgnoreRandomUID`** | **`TagByteOrder`** | **`TagByteOrderTypes`** | **`TagReadBlockNum`** | **`TagReadKey`** | **`TagReadKeySlot`** | **`TagReadKeyType`** | **`TagReadMinDigits`** | **`TagReadOffset`** | **`TagReadLength`** | **`TagReadFormat`** | **`TagReadRightShift`**

NFCType#

Definition:
Permits the cards or tags of an NFC Type # to be read, and defines which part of the card/tag data will be read.

Options:
For NFC Types:
=U or =1 to permit UID to be read (4 or 7 bytes usually in the first block).
=N or =2 to read NDEF records (NFCType 2 or 4).
=B or =3 to read or decode up to 16 bytes of memory block(s) (or 255 bytes with NFC Type 5) for example with MIFARE Ultralight, MIFARE Ultralight C or MIFARE Ultralight AES cards. (NFCType 2 or 5 only).
=D (NFC Type 4 only) to read DESFire secure data with **DESFire card or tag settings** settings.
=0 is disabled (default).

An order of precedence allows multiple selections from Pass, DESFire secure, Block data, NDEF, UID.
=NU or =UN would read UID if an NDEF record cannot be read.

Default value: =0

Example value: =U

NFCReportReadError

Definition:
Option to return an error payload when a pass or NFC Type 4 card read fails, making it possible to trigger alternative actions when an inappropriate card is presented to the VTAP reader.

Options:
=1 a failed pass or NFC Type 4 card read will return an error payload.
=0 (default) will not return any payload if an inappropriate card or pass is presented to the VTAP reader.

The returned error payload will begin ERROR:A, ERROR:G or ERROR:4 depending on whether an Apple (A), Google/Android (G) or NFC Type 4 (4) card/tag was detected in the VTAP NFC field. The error payload ends with a new line.

Note: When a VAS pass or payment card which requires user authentication is presented, on an iPhone in the VTAP NFC field, ERROR:A will not be triggered. In this case the iPhone returns a "not activated" error for the phone user, so that they can authenticate, then immediately present their pass or payment card again.

Default value: =0

Example value: =1

`IgnoreRandomUID`

Definition:
Filter out random tag reads. Some devices presented to the VTAP can appear as a Type 4 tag with random UID. May be needed if a VTAP is configured to read both passes and UID from NFC Type 4 tags.

Options:
`=1` to filter out random NFC Type 4 tag reads. Only useful if you have set `NFCType4=U`.

Default value: `=0`

Example value: `=1`

`TagByteOrder`

Definition:
Reverses the byte order of UID or block data read from all cards/tags. A UID of 1A2B3C4D becomes 4D3C2B1A. (Will be ignored if `TagByteOrderTypes` is set.)

Options:
`=1` in example reverses the byte order. It may be used in conjunction with `TagReadFormat` to produce hex-standard, hex-reversed decimal and decimal-reversed output of UIDs.

Default value: `=0`

Example value: `=1`

TagByteOrderTypes

Definition:
Reverses the byte order of UID or block data read from selected types of card/tag, where several types are being used in an application. A UID of 1A2B3C4D becomes 4D3C2B1A for your chosen card/tag type(s). (An alternative to `TagByteOrder`. If both are set `TagByteOrderTypes` take precedence and `TagByteOrder` will be ignored.)

Options:
Hex byte value, where each bit indicates reversal of data for a particular card/tag type, which can be added together:
Bit value 1: NFC Forum Type 1
Bit value 2: NFC Forum Type 2
Bit value 4: NFC Forum Type 3
Bit value 8: NFC Forum Type 4
Bit value 10: NFC Forum Type 5
Bit value 20: unused
Bit value 40: unused
Bit value 80: MIFARE Classic

`=80` in example reverses the byte order for MIFARE Classic data/UIDs only.
`=08` reverses the byte order for NFC Forum Type 4 (DESFire) data/UIDs only.
`=82` reverses the byte order for both MIFARE Classic and NFC Forum Type 2 (Ultralight) data/UIDs only.
`=00` (default) reverses nothing, any changes will be the result of `TagByteOrder`.
`=FF` reverses card/tag data on all card/tag types, equivalent to `TagByteOrder=1`.

Default value: `=0`

Example value: `=80`

TagReadBlockNum

Definition:
Select a block number to read.

Options:
MIFARE Classic 1K cards have 64 blocks (16 sectors of 4 blocks each) and 4K cards have 256 blocks (32 sectors of 4 blocks and 8 sectors of 16 blocks). These are numbered in decimal from 0 to 255. (Must also set `NFCType#=B` or `=3`, or `MIFAREClassic=B` or `=3`).

Default value: `=0`

Example value: `=56`

DEPRECATED

TagReadKey

Definition:
The key required to read block data from a MIFARE Classic card or tag. *Use of `TagReadKeySlot` now preferred.* [Does not support AES keys.]

Options:
6 bytes in hexadecimal

Default value: `N/A`

Example value: `=123ABC456DEF`

TagReadKeySlot

Definition:
Identifies which uploaded `appkey#.txt` file contains the key for accessing the block data from a MIFARE Classic card or tag or MIFARE Ultralight AES card. (An alternative to `TagReadKey` for MIFARE Classic card cards.)

Options:
`=1` to `=9`, to refer to the application key files uploaded as `appkey1.txt` through `appkey9.txt`

`=0` means no key specified (unless `TagReadKey` is set)

Default value: `=0`

Example value: `=1`

TagReadKeyType

Definition:
This is the key type, for the key set in TagReadKey.

Options:
`=A` or `=B` to describe the MIFARE Classic key type to be used with block data,

`=C` to enable AES CMAC secure message authentication / verification of block data. (Only applicable if `NFCType2=B` is also used with secured MIFARE Ultralight AES cards).

Default value: `=A`

Example value: `=B`

TagReadMinDigits

Definition:
Require fixed-width UID output by adding leading zeros as necessary.

Options:
1 to 20 digits, corresponding to the maximum number of decimal digits in a 64 bit value.

`=0` or omitted has no effect.
`=A` for automatic padding with leading zeros to length for UID: 10 digits for 32bit UID, 17 digits for 56bit UID or 20 digits for 64bit UID.
`=X` for extended automatic padding: 10 digits for 32bit UID, 18 digits for 56bit UID or 20 digits for 64bit UID.

Default value: `=0`

Example value: `=10`

TagReadOffset

Definition:
The byte offset within block or UID data. Describes the position to start reading data from, within a block.

Options:
`=0` to `=15`

If this offset is used together with TagReadLength, TagReadOffset + TagReadLength must be less than or equal to 16.

Default value: `=0`

Example value: `=5`

TagReadLength

Definition:
The number of bytes of block or UID data to read from a tag or card. (Except for DESFire tags or cards which have an overriding `DESFireReadLength` setting.)

Options:
`=1` byte to `=16` bytes

This number should not exceed `=4` if `TagReadFormat` is 'd'ecimal, as no more than 4 bytes are used in this case.

If this offset is used together with `TagReadOffset`, `TagReadOffset` + `TagReadLength` must be less than or equal to 16. (If it is not, the length will be reduced by truncating data.)

Default value: `=16`

Example value: `=4`

TagReadFormat

Definition:
Choose to format the extracted block data as ASCII, hex or decimal.

Options:
`=a` for ASCII characters (each byte is an ASCII character);
`=d` for decimal (interpret binary data as a 64 bit decimal value and output as ASCII decimal digits), in this case `TagReadLength` should not exceed 4 bytes;
`=h` for hexadecimal (convert binary data to ASCII hex digits with 2 digits per byte).

Default value: `=h`

Example value: `=d`

TagReadRightShift

Definition:
Number of bits to right shift data. It may be used to remove any parity bits included at the end of extracted binary data.

Options:
Number of bits to right shift decimal 64 bit block data. (Only relevant if `TagReadFormat=d`.)

Default value: `=0`

Example value: `=1`

3.1.4 MIFARE® Classic card or tag

A list of valid configuration settings for the `config.txt` file, which relate to extracting block data from MIFARE Classic cards or tags. (MIFARE is a registered trademark of NXP B.V.) You will also need to use some of the general **Card or tag settings** alongside these.

The setting `MIFAREClassic` is used to direct the VTAP reader to read various types of data from any MIFARE Classic cards that are presented. If you want to read blocks of data from MIFARE Classic or NFC cards or tags you will also need to use settings prefixed `TagRead...`

If `MIFAREClassic=B` to read block data, you need to specify `TagReadBlockNum`, `TagReadKey` or `TagReadKeySlot`, and `TagReadKeyType`. Together these define which block to read and provide the key information needed to decode that block.

Settings below: **`MIFAREClassic`**

`MIFAREClassic`

Definition:
Permits MIFARE cards or tags to be read, and defines which part of the card/tag data will be read. This can be the UID (4 or 7 bytes usually in the first block) or up to 16 bytes of data from another specified block. (Similar to NFCType for NFC cards).

Options:
`=U` or `1` to permit MIFARE UID to be read,
`=N` or `2` to read NDEF data (not currently supported on MIFARE Classic),
`=B` or `3` to read or decode a memory block.
`=0` is disabled (default).

An order of precedence allows multiple selections, from MIFARE just from Pass, Block data or UID. So `=BU` or `=UB` would read UID if an block data cannot be read.

Default value: `=0`

Example value: `=U`

3.1.5 MIFARE DESFire card or tag

A list of all valid configuration settings which relate to extracting secure data from DESFire cards or tags for the `config.txt` file. You may also need to use some of the general **Card or tag settings** alongside these.

DESFire cards may contain a number of applications, identified by an application ID. Each application may contain a number of data files, each identified by a file number, which may be individually protected. The VTAP reader supports a number of formats to read, decode or output the secure data. The format might be HID 10301 26-bit or HID 10301 37-bit. Reading data, from a DESFire card which contains secured data, therefore includes uploading the app key file, and providing information about the application ID and the key number to be used for authentication, along with the file number and the crypto algorithm for decoding each file and bit format.

To read DESFire cards will require setting `NFCType4=D`, uploading a suitable `appkey#.txt` files with the relevant application keys, and using all of these settings prefixed `DESFire....` Only DESFire cards which are unformatted or Key-ID 26-bit HID 10301 data are currently supported.

> **Note:** In these settings # is a number from 1 to 9, showing which settings form a group for reading each of 1 to 9 values from separate files and or applications on a DESFire card or tag. If you use multiple DESFire#... settings the values read will be output together, spaced by the `DESFireSeparator` string. The lowest numbered DESFire read will be first in the output string, then continuing in ascending numeric order. (For Wiegand data multiple reads are not supported, so only the lowest numbered DESFire#... settings will be used.) If no number is used the setting will be treated as set 1.

Some cards or passes can be set up so that each one carries a different key, although all are derived from the same master key. This is a feature of DESFire EV1 and EV2 cards, MIFARE2Go passes, Apple Wallet Access passes and others. One form of 'key diversification' scheme to support this is NXP AN10922. If your DESFire cards are using NXP AN10922 key diversification, you will need settings that are enabled by `DESFire#Diversification=1`. You will need to upload a Privacy key identified in `DESFire#PrivacyKeySlot`, and set a Privacy key number `DESFire#PrivacyKeyNum`, together with uploading System Identifier information (up to 16 bytes of data, saved as if it was another key) identified by `DESFire#SysIDKeySlot`. This is in addition to the usual settings needed to decode secured data in an encrypted application.

For examples refer to Application Notes.

Settings below: **DESFireAppID** | **DESFireCrypto** | **DESFireDiversification** | **DESFireFileID** | **DESFireFormat** | **DESFireKeyNum** | **DESFireKeySlot** | **DESFirePrivacyKeyNum** | **DESFirePrivacyKeySlot** | **DESFireReadLength** | **DESFireReadOffset** | **DESFireSysIDKeySlot** | **DESFireSysIDLength** | **DESFireSeparator**

DESFire#AppID

Definition:
Hex number identifying your DESFire application

Options:
24 bit number formatted as 6 hex digits with the most significant byte first

Note: The VTAP reader expects the `DESFireAppID` to be a 24 bit number formatted as 6 hex digits with the most significant byte first. However, some vendors and software treat the Application ID value as a byte sequence with the least significant byte first, which is the byte order used in communications with the card. If the VTAP reader fails to read your DESFire card application, try reversing the order of the `DESFireAppID` bytes. For example, if `DESFireAppID=F56400` try `DESFireAppID=0064F5`.

Default value: `N/A`

Example value: `=F56400`

DESFire#Crypto

Definition:
Identifies the cryptographic method used for DESFire cards or tags

Options:
`=3` identifies AES (default),
`=1` identifies 3DES cryptography,
`=0` for no cryptography

Default value: `=3`

Example value: `=1`

DESFire#Diversification

Definition:
Enables/disables DESFire key diversification settings.

Options:
`=1` enables key diversification in accordance with NXP AN10922 Symmetric key diversifications Application Note rev 2.2. (You will then also need to set `DESFire#PrivacyKeySlot`, `DESFire#PrivacyKeyNum` and `DESFire#SysIDKeySlot`).

`=0` disables the feature

Default value: `=0`

Example value: `=1`

DESFire#FileID

Definition:
Number identifying the file within your DESFire application to read

Options:
Use a value from 1 to 255

Default value: `N/A`

Example value: `=1`

DESFire#KeyNum

Definition:
Number identifying the application key needed to read your DESFire file

Options:

Default value: `N/A`

Example value: `=1`

DESFire#KeySlot

Definition:
Identifies which uploaded `appkey#.txt` file contains the key for accessing the DESFire file

Options:
`=1` to `=9`, to refer to the application key files uploaded as `appkey1.txt` through `appkey9.txt`

Default value: `N/A`

Example value: `=1`

DESFire#PrivacyKeyNum

Definition:
Number identifying the Privacy key within the DESFire application that is used to restrict access to the real UID, when a random UID is used to protect the card identity. Needed when key diversification is in use.

Options:
Valid numbers will depend on your cards/passes.

Default value: `N/A`

Example value: `=1`

DESFire#PrivacyKeySlot

Definition:
Identifies which key slot, filled by an uploaded `appkey#.txt` file, contains the Privacy key for accessing the UID. Needed when key diversification is in use.

Options:
`=1` to `=9`, to refer to the application key files uploaded as `appkey1.txt` through `appkey9.txt`

Default value: `N/A`

Example value: `=1`

NEW OPTIONS

DESFire#Format

Definition:
Identifies which bit format is used to store the data.

Options:
`=0` means no format (specify `DESFire#ReadLength` and `TagReadFormat` to determine how the data is output),
`=1` means KEY-ID v1 format (26 bit facility code and number format, H10301 compatible),
`=2` for KEY-ID v2 format (variable bit length including 26 bit)

Default value: `=0`

Example value: `=1`

DESFire#ReadLength

Definition:
The number of bytes of data to read from DESFire cards, distinct from TagReadLength which applies to other cards and tags.

Options:
`=1` byte to `=255` bytes. In practice limited by the data that a DESFire card can return in a single message, typically 240 bytes maximum.

(Default `=3` suits Key-ID encoded cards, however the setting is not required if `DESFire#Format` has selected a Key-ID format, as the length is then automatic.)

Default value: `=3`

Example value: `=4`

NEW

DESFire#ReadOffset

Definition:
The byte offset for the data. Describes the position to start reading data from, within a file.

Options:
`=0` to `=255`

Default value: `=0`

Example value: `=5`

DESFire#SysIDKeySlot

Definition:
Identifies which key slot, filled by an uploaded `appkey#.txt` file, contains the System Identifier information. Needed when key diversification is in use.

Options:
`=1` to `=9`, to refer to the application key files uploaded as `appkey1.txt` through `appkey9.txt`

Default value: `N/A`

Example value: `=1`

DESFire#SysIDLength

Definition:
Defines the length of the System Identifier key (number of bytes), when key diversification is in use. Optional when key diversification is in use.

Options:
`=0`, or omitting this setting, will automatically use the length of the stored System Identified app key

`=1` to `=16` will fix the length of app key in bytes

Default value: `=0`

Example value: `=1`

DESFireSeparator

Definition:
Defines a string to include in between the data obtained from separate DESFire card reads, when there is more than one.

Options:
Choose any separator that suits your application, up to 16 characters. (Note: Pre/postfix strings can still also be applied to the combined DESFire read output over a given interface.)

Default value: =,

Example value: =|

3.1.6 Access using Apple Wallet

A list of additional configuration settings for the `config.txt` file, which relate to extracting secure data from passes for Access using Apple Wallet.

Access using Apple Wallet is used by some customers in the US. It allows passes in Apple Wallet to act as keys to homes, cards and hotel rooms. This is distinct from Apple VAS passes for other purposes, enforcing a higher level of security, although it is similar in many ways.

The additional settings are `AccessTCI` which identifies you as a credential issuer in the Access using Apple Wallet programme, and `AccessAuthRequired` to require positive iPhone or Watch authentication with every tap to share an Access pass.

Settings below: **`AccessTCI`** | **`AccessAuthRequired`**

`AccessTCI`

Definition:
The TCI is an ID assigned by Apple to the Access using Apple Wallet credential issuer.

Options:
3 byte hex value which allows a matching pass to be brought up by an Apple iPhone or Apple Watch. A reboot is required to bring a change to this setting into effect.

Default value: `N/A`

Example value: `=203C20`

`AccessAuthRequired`

Definition:
Requires authentication of an Access pass in Apple Wallet, overriding any Express mode setting that may be in place.

Options:
`=1` to require authentication,
`=0` authentication not required.

A reboot is required to bring a change to this setting into effect.

Default value: `=0`

Example value: `=1`

3.1.7 NDEF card or tag

A list of all valid configuration settings which relate only to extracting NDEF records from NFC cards or tags for the `config.txt` file. You may also need to use some of the general **Card or tag settings** alongside these.

The `NDEFTagType4AID` and `NDEFTagType4AIDOnly` settings allow you to decide which NFC ISO application IDs (AIDs) should be tried when attempting to read NFCForum NDEF data from a Type4 card/tag (with `NFCType4=N`):

- The default behaviour is to try to select the NDEF-Tag application AID (D2760000850101h) as defined by the NFC Forum, but if `NDEFTagType4AID` is set then this AID will be tried first, in preference to the default.
- If the `NDEFTagType4AID` is not found, the VTAP will then try the default NDEF-Tag application AID.

This behaviour can be overridden by setting `NDEFTagType4AIDOnly=1`, so the VTAP will not attempt to use the default NDEFTag application AID, after failing to select the specified `NDEFTagType4AID`.

Settings below: **NDEFTagExtractType** | **NDEFTagExtractID** | **NDEFTagType4AID** | **NDEFTagType4AIDOnly**

NDEFTagExtractType

Definition:
Read only NDEF records of one type, such as type T (text). The output will be the payload of the first matching NDEF record found (by searching the NDEF records recursively). Where nothing is found that matches the constraints set, nothing will be output. If no constraints are set, all NFC data content will be output.

Options:
`=T` for text records (or omit setting). Only useful if you have set `NFCType2=N`, `NFCType3=N` or `NFCType4=N`.
`=U` to extract a URI from the data
`=<MyTagType>` using the code defined to select a proprietary NDEF record type definition (RTD)

Default value: `N/A`

Example value: `=T`
`=U`
`=MyTagType`

NDEFTagExtractID

Definition:
Read only NDEF records with a particular ID, such as 'name'. The output will be the payload of the first matching NDEF record found (by searching the NDEF records recursively). Where nothing is found that matches the constraints set, nothing will be output. If no constraints are set, all NFC data content will be output.

Options:
`=name` or omit setting.

Only useful if you have set `NFCType2=N`, `NFCType3=N` or `NFCType4=N`.

Default value: `N/A`

Example value: `=name`

NDEFTagType4AID

Definition:
Allows you to set a preferred AID to try when reading an NDEF Type 4 record.

Options:
The AID set here will be tried before trying the two standard NFC Forum application IDs (d2760000850101 followed by d2760000850100). The standard AIDs are always tried when this setting is not used.

Default value: `N/A`

Example value: `=f1234567890abcd`

NDEFTagType4AIDOnly

Definition:
Allows you to restrict the VTAP reader to only use the AID provided as the `NDEFTagType4AID` setting.

Options:
`=1` will prevent the VTAP reader from trying the standard NFC Forum application IDs (d2760000850101 followed by d2760000850100), so that only the value set for `NDEFTagType4AID` will be tried.
`=0` allows those standard AIDs to be tried if `NDEFTagType4AID` is not successful.

Default value: `=0`

Example value: `=1`

3.1.8 Keyboard/barcode reader emulation

A list of all valid configuration settings which relate to sending pass information over a keyboard/barcode reader emulation interface for the `config.txt` file.

If `KBLogMode=1` and `KBSource` is set, you can open an application on the computer connected to the VTAP reader, and pass or card data that is read will be typed to the screen. `KBSource` determines whether keyboard wedge/barcode mode is triggered only by mobile pass reads, or cards/tag reads too. You might need a `KBDelayMS` value to insert an additional delay between key presses, to suit your receiving application.

`KBPrefix` and `KBPostfix` can be optionally used to add extra characters before or after a pass read, to suit your application. These include adding pass type, as explained below:

Pass type character

Pass type is a single character to describe the source of a received pass payload, which may be an Apple VAS pass or Google ST pass, a particular type of NFC card, or other source such as a barcode scanner. The values that returned are:

A - Apple VAS pass
G - Google ST pass
0 - MIFARE card/tag
2 - NFC type 2 card/tag
4 - NFC type 4 card/tag
6 - NFC type 5 card/tag
E - Card emulation
Q - Scanner input
X - Key-ID v2 credential from Apple Wallet Access/ECP2 on iPhone
W - Key-ID v2 credential from Apple Wallet Access/ECP2 on Apple Watch

When `KBPassMode=1`, a character delimited section of the pass payload will be extracted. The delimiting character defaults to `|` but this can be changed to another character using `KBPassSeparator`. The payload is then considered as comprising a number of sections, any of which can be selected using `KBPassSection`. By identifying a `KBContentSeparator` this process can be repeated at a lower level, within a section. There is a simple example in the VTAP Configuration Guide.

`KBMap` optionally points to a separate file, where a list of key codes corresponding to particular characters, can be used to translate between the VTAP default US keyboard emulation and the keyboard language setting used by the host computer.

`KBEnable=0` allows you to disable the keyboard device function of a VTAP reader, which can be helpful if you are working with Android devices using the USB COMPort, to avoid a clash with the on-screen keyboard.

Settings below: KBDelayMS | KBEnable | KBMap | KBLogMode | KBPassContentMode | KBPassContentSection | KBPassContentSeparator | KBPassLength | KBPassMode | KBPassSection | KBPassSeparator | KBPassStart | KBPostfix | KBPrefix | KBSource

KBDelayMS

Definition:
Inserts a number of milliseconds delay between key down and key up and between consecutive key presses in the keyboard output.

Options:
Adjust to suit your receiving application between 5ms and 255ms.

Default value: `=5`

Example value: `=10`

KBEnable

Definition:
Option to disable the USB keyboard device function of a VTAP reader. This can be useful in Android USB serial integrations, where the presence of a USB keyboard device can disable the on-screen keyboard, which would then need to be overridden in phone settings.

Options:
`=1` (default) will enable the USB keyboard device of a VTAP reader.

`=0` will disable the USB keyboard device of a VTAP reader.

Note: The VTAP reader must be rebooted for a change to take effect, with the keyboard device added/removed from the composite device. When the VTAP reader reboots as a new device type it may be assigned different COM ports or drive letters.

Default value: `=1`

Example value: `=0`

KBMap

Definition:
Points to a keyboard map file and/or section of that file, which lists key codes corresponding to particular characters, in order to translate to a keyboard language setting used by the host computer. (Without this setting the VTAP keyboard emulation uses a US English keyboard mapping.)

Options:
`KBMap=<section>@<file_name>` is the syntax for this setting
`<file_name>` defaults to `kbmap.txt` if omitted.
`<section>` refers to a particular keyboard language such as `en-uk`, as the file may contain maps for multiple languages although the `KBMap` setting can only use one at a time.

If you need to use this feature please contact **vtap-support@dotorigin.com** for suitable `kbmap.txt` example files.

Default value: `N/A`

Example value: `=en-uk@kbmap.txt`

KBLogMode

Definition:
Turns the keyboard emulation feature on or off, sending tap payload data to the attached device as if it came from a keyboard. (Note: The VTAP reader will always appear to the attached device as an HID keyboard, see **KBEnable** if you need to suppress this function.)

Options:
`=1` starts keyboard emulation,
`=0` switches it off (default)

Default value: `=0`

Example value: `=1`

KBPassContentMode

Definition:
Turns on a subdivision of a data section into smaller content sections.

Options:
=0 off (default),
=1 on

To use this mode you must also supply `KBPassContentSeparator` if the default `%` is not the separator you want to use and `KBPassContentSection`.

Default value: `=0`

Example value: `=1`

KBPassContentSection

Definition:
Identifies the content section of interest, by finding KBPassContentSeparator characters. The first content section, before the first separator, is numbered 0.

Options:
Used only if `KBPassContentMode=1`.

Default value: `=0`

Example value: `=1`

KBPassContentSeparator

Definition:
Identifies the content separator character in the data which bounds content sections.

Options:
Used only if `KBPassContentMode=1` and a `KBPassContentSection` is set.

Default value: `=%`

Example value: `=%`

KBPassLength

Definition:
Number of characters of data to extract, starting from the position defined by `KBPassStart`.

Options:
Defaults to `0` meaning do not truncate data, send all available characters from the pass data, or extracted section of pass data.

Default value: `=0`

Example value: `=10`

KBPassMode

Definition:
Sets whether to extract parts of the data from the mobile NFC pass.

Options:
`=0` does not extract a character delimited section of the pass payload (`KBPassSection` is ignored).
`=1` uses the `KBPassSection` value to extract a character delimited section of the pass payload.

Default value: `=0`

Example value: `=1`

KBPassSection

Definition:
Identifies the section of interest, based on finding KBPassSeparator characters. The first section, before the first separator, is numbered 0.

Options:
Used only if `KBPassMode=1`.

Default value: `=0`

Example value: `=2`

KBPassSeparator

Definition:
Identifies the separator character in the data which bounds sections.

Options:
Used only if `KBPassMode=1` and a `KBPassSection` is set.

Default value: `=|`

Example value: `=|`

KBPassStart

Definition:
Number of characters into pass data to start reading, where first character is 0.

Options:
May be used with `KBPassLength`.

Default value: `=0`

Example value: `=5`

KBPostfix

Definition:
Adds special characters after data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0D` in the example adds a carriage return after the data,
`%0A` adds a new line after every pass or tag entry.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would append '$t' to the output.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `=%0A`

Example value: `=end%0D`

KBPrefix

Definition:
Adds special characters before the data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0A` in the example adds a new line linefeed before the data.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would prefix the output with '$t'.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `[Empty]`

Example value: `=%0Astart`

KBSource

Definition:
Controls which types of event data (pass reads, card/tag reads, serial commands) are sent as a keyboard entries.

Options:
The values are determined from a bitwise combination of the following hexadecimal values. Source options include these bit values for mobile passes and tag UIDs:
Bit 7 (0x80) = Mobile Pass (Apple VAS/Google Wallet Smart Tap)
Bit 6 (0x40) = STUID
Bit 5 (0x20) = Write to card emulation mode (`CardEmulationMode=1` see **VTAP NFC tag emulation settings**)
Bit 4 (0x10) = RFU
Bit 3 (0x08) = RFU
Bit 2 (0x04) = Scanners
Bit 1 (0x02) = Command interface messages (`>interface:type:message` see **Dynamic configuration commands**)
Bit 0 (0x01) = Card/Tag UID

Default value: `=A5`

Example value:

```
=80 (hex) will send only mobile NFC pass data
=A1 to send data from NFC passes and NFC cards/tags and card emulation
=83 to send passes, cards/tags and serial commands
```

3.1.9 Virtual COM port

A list of all valid configuration settings which relate to sending pass information over a virtual COM port interface for the `config.txt` file.

If `ComPortEnable=1`, `ComPortMode=1` and `ComPortSource` is set, pass or card data read by the VTAP reader will be sent over the virtual COM port. This is the virtual COM port in active mode. `ComPortSource` determines whether data is sent over the COM port interface only when there are mobile pass reads, or cards/tag reads too.

After setting `ComPortEnable=1` in `config.txt` the unit needs to be rebooted or power cycled. On a Windows PC, if you check the Device Manager, you will see under 'Ports (COM & LPT)' that there is now a VTAP reader that has been assigned a COM port. (If it says 'USB Serial Device' rather than 'VTAP100', you should right click on the `VTAP100.inf` or `VTAP.inf` file on the VTAP file system to install the correct driver.)

The COM port is described as being in either active or passive mode. Active mode means that pass or card data read by the VTAP reader will be sent over the COM port immediately, whenever it is read. Passive mode means that the VTAP reader will only send on that data in response to a command. These are listed in section **4**.

By default the VTAP reader is in active mode. Use `PassiveInterfaces=1` to enable passive mode and `CommandInterfaces=1` to allow commands to be received on the command interface. Set `InvalidDataCacheMS` for the time you want pass read data to be retained by the VTAP reader, while it is waiting for a Com Port request to send it onwards. There is a simple example in the VTAP Serial Integration Guide.

`ComPortPrefix` and `ComPortPostfix` can be optionally used to add extra characters before or after a pass read, to suit your application.

When `ComPortPassMode=1` you can use all of the other settings beginning `ComPortPass...` to extract only part of the mobile pass data. To do this you have to start by identifying a separator character in the data as `ComPortPassSeparator`. This allows the data to be regarded as a set of sections that can be identified by number. By identifying a `ComPortPassContentSeparator` this process can be repeated at a lower level, within a section. There is a simple example in the VTAP Configuration Guide.

If you want to receive serial QR/barcode scanner input on this interface , those commands are in the section **Serial barcode/QR scanner**.

Settings below: `CommandInterfaces` | `ComPortEnable` | `ComPortMode` | `ComPortPassContentMode` | `ComPortPassContentSection` | `ComPortPassContentSeparator` | `ComPortPassLength` | `ComPortPassMode` | `ComPortPassSection` | `ComPortPassSeparator` | `ComPortPassStart` | `ComPortPassPostfix` | `ComPortPrefix` | `ComPortSource`| `InvalidDataCacheMS` | `PassiveInterfaces`

CommandInterfaces

Definition:
Enables or disables your ability to send commands over particular interfaces.

Options:
`=1` enables the ComPort,
`=2` enables the Serial port,
`=4` enables the Serial2 port.

Add values together to enable multiple ports, so
`=3` enables both ComPort and Serial port.
`=7` (default) enables all ports.

Default value: `=7`

Example value: `=1`

ComPortEnable

Definition:
Enable a virtual COM port for the USB interface so the VTAP is treated by the connected PC as a COM port, as well as a mass storage device.

Options:
`=1` is enabled (default)
`=0` is disabled

Default value: `=1`

Example value: `=0`

ComPortMode

Definition:
Activate the COM port as an interface for receiving mobile pass data.

Options:
`=1` (default) if you want VTAP data sent over the COM port
`=0` is disabled

Default value: `=1`

Example value: `=0`

ComPortPassContentMode

Definition:
Turns on a subdivision of a data section into smaller content sections.

Options:
`=0` off (default),
`=1` on

To use this mode you must also supply `ComPortPassContentSeparator` if the default `%` is not the separator you want to use and `ComPortPassContentSection`.

Default value: `=0`

Example value: `=1`

ComPortPassContentSection

Definition:
Identifies the content section of interest, based on finding `ComPortPassContentSeparator` characters.

The first content section, before the first separator, is numbered 0.

Options:
Only used if `ComPortPassContentMode=1` is set.

Default value: `=0`

Example value: `=1`

ComPortPassContentSeparator

Definition:
Identifies the content separator character in the data which bounds content sections.

Options:
Only used if `ComPortPassContentMode=1` and `ComPortPassContentSection is set.`

Default value: `=%`

Example value: `=%`

ComPortPassLength

Definition:
Number of characters of data to extract, starting from the position defined by `ComPortPassStart`.

Options:
`=0` is default, meaning do not truncate data, send all available characters from the pass data, or extracted section of pass data.

Default value: `=0`

Example value: `=10`

ComPortPassMode

Definition:
Sets whether to extract parts of the data from the mobile NFC pass.

Options:
`=0` does not extract a character delimited section of the pass payload (`ComPortPassSection` is ignored).
`=1` uses the `ComPortPassSection` value to extract a character delimited section of the pass payload.

Default value: `=0`

Example value: `=1`

ComPortPassSection

Definition:
Identifies the section of interest, based on finding `ComPortPassSeparator` characters. The first section, before the first separator, is numbered 0.

Options:
Used only if `ComPortPassMode=1`.

Default value: `=0`

Example value: `=2`

ComPortPassSeparator

Definition:
Identifies the separator character in the data which bounds sections.

Options:
Only used if `ComPortPassMode=1` and a `ComPortPassSection` is set.

Default value: `=|`

Example value: `=|`

ComPortPassStart

Definition:
Number of characters into pass data to reading, where first character is 0.

Options:
May be used in conjunction with `ComPortPassLength`.

Default value: `=0`

Example value: `=5`

ComPortPostfix

Definition:
Adds special characters after data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0D` in the example adds a carriage return after the data,
`%0A` adds a new line after every pass or tag entry.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would append '$t' to the output.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `=%0D%0A`

Example value: `=end%0D`

ComPortPrefix

Definition:
Adds special characters before the data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0A` in the example adds a new line linefeed before the data.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would prefix the output with '$t'.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `[Empty]`

Example value: `=%0Astart`

ComPortSource

Definition:
Controls which types of event data (pass reads, card/tag reads, serial commands, scanner inputs) are sent to the COM port interface.

Options:
The values are determined from a bitwise combination of the following hexadecimal values. Source options include these bit values for mobile passes and tag UIDs:
Bit 7 (0x80) = Mobile Pass (Apple VAS/Google Wallet Smart Tap)
Bit 6 (0x40) = STUID
Bit 5 (0x20) = Write to card emulation mode (`CardEmulationMode=1` see **VTAP NFC tag emulation settings**)
Bit 4 (0x10) = RFU
Bit 3 (0x08) = RFU
Bit 2 (0x04) = Scanners
Bit 1 (0x02) = Command interface messages (`>interface:type:message` see **Dynamic configuration commands**)
Bit 0 (0x01) = Card/Tag UID

Default value: `=A5`

Example value:

```
=80 (hex) will send only mobile NFC pass data
=A1 to send data from NFC passes and NFC cards/tags and card emulation
=83 to send passes, cards/tags and serial commands
```

InvalidDataCacheMS

Definition:
A period to retain data after reading in milliseconds. Set to retain data in memory until requested, but without undue security risk to that data.

Options:
Example says retain data for 3000ms after a mobile pass, card/tag read. Used with interfaces in passive mode only.

Default value: `=3000`

Example value: `=3000`

`PassiveInterfaces`

Definition:
Enable passive mode at startup for any of the command interfaces.

Options:
The value is any combination of bit values (as per `CommandInterfaces` values) where
`=1` enables the ComPort,
`=2` enables the Serial port,
`=4` enables the Serial2 port.
`=0` (default) means passive mode is disabled on all interfaces.

`=3` in example enables passive mode on Serial and COMport.

Default value： `=0`

Example value： `=3`

3.1.10 Wiegand interface

A list of all valid configuration settings relating to the Wiegand interface for use in the `config.txt` file.

Note: The Wiegand interface is only available on VTAP100-PAC-W hardware.

If `WiegandMode=1` and `WiegandSource` is set, you can open an application on the computer connected to the VTAP reader, and pass or card data that is read will be sent over the Wiegand interface. `WiegandSource` determines whether data is sent over the Wiegand interface only when there are mobile pass reads, or cards/tag reads too.

`PassFormat` allows you to choose to interpret ASCII pass data as hex or decimal. By default 56 bits of data will be passed over the Wiegand interface, but you can change this by setting `PassWiegandBits`. The setting `PassWiegandParity` allows you to pad mobile pass data with zeroes, to imitate the inclusion of parity bits. This makes the pass data interchangeable that from with cards/tags which include parity bits, as long as parity is not being tested for validity.

When `WiegandPassMode=1` you can use all of the other settings beginning `WiegandPass...` to extract only part of the mobile pass data. To do this you have to start by identifying a separator character in the data as `WiegandPassSeparator`. This allows the data to be regarded as a set of sections that can be identified by number. By identifying a `WiegandPassContentSeparator` this process can be repeated at a lower level, within a section. There is a simple example in the VTAP Configuration Guide.

Settings below: **`WiegandMode`** | **`WiegandInputEnable`** | **`WiegandPassContentMode`** | **`WiegandPassContentSection`** | **`WiegandPassContentSeparator`** | **`WiegandPassLength`** | **`WiegandPassMode`** | **`WiegandPassSection`** | **`WiegandPassSeparator`** | **`WiegandPassStart`** | **`WiegandPassTypeIdent`** | **`WiegandPaddingMode`** | **`PassFormat`** | **`PassWiegandBits`** | **`PassWiegandParity`** | **`TagWiegandASCIIFormat`** | **`TagWiegandBits`** | **`TagWiegandParity`** | **`ScannerWiegandBits`**| **`WiegandSource`**

`WiegandMode`

Definition:
Activate the Wiegand interface for receiving mobile pass data.

Options:
`=1` sends VTAP data to the Wiegand interface,
`=0` switches it off (default)

Default value: `=0`

Example value: `=1`

WiegandInputEnable

Definition:
Control the VTAP response to Wiegand red, green and beep input signals, from a door controller.

Options:
`=81` (default) means that the VTAP will receive both beep and red and green LED inputs,
`=0` prevents any use of these inputs.

To drive VTAP responses (beep and LEDs) you must also have `WiegandMode=1`.

Default value: `=81`

Example value: `=0`

WiegandPassContentMode

Definition:
Turns on a subdivision of a data section into content sections.

Options:
`=0` off (default),
`=1` on.

To use this mode you must also supply `WiegandPassContentSeparator` if the default `%` is not the separator you want to use and `WiegandPassContentSection`.

Default value: `=0`

Example value: `=1`

WiegandPassContentSection

Definition:
Identifies the content section of interest, based on finding `WiegandPassContentSeparator` characters. The first content section, before the first separator, is numbered 0.

Options:
Only used if `WiegandPassContentMode=1` and `WiegandPassContentSection` is set

Default value: `=0`

Example value: `=1`

WiegandPassContentSeparator

Definition:
Identifies the content separator character in the data which bounds content sections.

Options:
Only used if `WiegandPassContentMode=1` and `WiegandPassContentSection` is set.

Default value: `=%`

Example value: `=%`

WiegandPassLength

Definition:
Number of characters of data to extract, starting from the position defined by `WiegandPassStart`.

Options:
Defaults to 0 meaning do not truncate data, send all available characters from the pass data, or extracted section of pass data.

Default value: `=0`

Example value: `=10`

WiegandPassMode

Definition:
Sets whether to extract parts of the data from the mobile NFC pass.

Options:
`=0` does not extract a character delimited section of the pass payload (`WiegandPassSection` is ignored).
`=1` uses the `WiegandPassSection` value to extract a character delimited section of the pass payload.

Default value: `=0`

Example value: `=1`

WiegandPassSection

Definition:
Identifies the section of interest, based on finding `WiegandPassSeparator` characters. The first section, before the first separator, is numbered 0.

Options:
Only used if `WiegandPassMode=1`.

Default value: `=0`

Example value: `=2`

WiegandPassSeparator

Definition:
Identifies the separator character in the data which bounds sections.

Options:
Only used if `WiegandPassMode=1` and `WiegandPassSection` is set.

Default value: `=|`

Example value: `=|`

WiegandPassStart

Definition:
Number of characters into string to start extracting data, where first character is 0.

Options:
May be used in conjunction with `WiegandPassLength`.

Default value: `=0`

Example value: `=5`

WiegandPassTypeIdent

Definition:
Inserts an additional leading byte of pass type identifier information in the Wiegand output. Either 01 for Apple VAS, or 02 for Google ST. This allows the controller or application receiving the Wiegand data (via a door controller) to distinguish between cards/tags and mobile wallet passes, as well as identifying the wallet type (Google or Apple) for reporting purposes.

Options:
`=0` disabled or
`=1` add the pass type identifier.

When this option is used, the Wiegand bit length is fixed to 56 bits + 8 bits type identifier = 64 bits total, so `PassWiegandBits` is ignored, if this option is enabled.

Default value: `=0`

Example value: `=1`

WiegandPaddingMode

Definition:
Controls where to apply padding zero bits when converting ASCII hexadecimal data which does not contain sufficient hex digits to produce the configured Wiegand bit length. Zeros can be added to LS (default) or MS end.

Options:
`=0` LS end padding (default),
`=1` MS end padding.

Default value: `=0`

Example value: `=1`

NEW OPTIONS

PassFormat

Definition:
Choose to interpret ASCII pass data characters as either hex or decimal, when converting the pass data to a Wiegand bit sequence.

Options:
`=d` for a decimal 64 bit number, from which the appropriate number of bits are output to Wiegand;
`=h` for hexadecimal, converted to a byte sequence from which the appropriate number of bits are output. (These lengths are both controlled by `WiegandPassBits`).
`=2` for KEY-ID v2 format data in VAS/ST wallet passes.

Default value: `=h`

Example value: `=d`

PassWiegandBits

Definition:
Specify the number of bits to output over the Wiegand interface from the start of the filtered pass data, where it otherwise defaults to 56.

Options:
`=1` to `=255` number of bits required. (Note: This number includes any parity bits.)

Default value: `=56`

Example value: `=64`

PassWiegandParity

Definition:
Adds calculated parity bits, or the equivalent of parity bits, to decimal pass data. This makes it possible to use mobile pass data formats that include parity bits. Parity bit equivalents can be used if the parity bit(s) are not being tested for validity. This is the same as bitwise left-shifting the data by the given number of bits within the 64 bit result.

Options:
`=1` Adds a single bit of 0 padding (for example in lieu of parity bits) to the least significant end of the data.
Used with `PassFormat=d` or `PassFormat=h`.

`=2` Adds calculated odd and even parity bits to Wiegand data. Used with either `PassFormat=d` or `PassFormat=h` data of any length, defined by `PassWiegandBits`. In this calculation the MS parity bit is calculated from the left half of the bits and the LS parity calculated from the right half of the bits, with the centre bit included in both calculations.

Default value: `=0`

Example value: `=2`

TagWiegandASCIIFormat

Definition:
Defines how to interpret ASCII tag or card data for output over a Wiegand interface.

Options:
=d is decimal
=h is hexadecimal
=a means ASCII (default)

Default value: =a

Example value: =d

TagWiegandBits

Definition:
Specify the number of bits to output over the Wiegand interface from the start of the extracted tag data, where it otherwise defaults to 56. Or allow the number of bits to be automatically determined from the data.

Options:
=0 to automatically determine the bit length from the extracted data, or
=1 to =255 to set to the number of bits required. (Note: This number includes any parity bits.)

If the number set exceeds the number of bits of data available, then that data will be padded with zero bits, following the WiegandPaddingMode setting for whether zeroes are added to the most or least significant bits.

Default value: =0

Example value: =64

TagWiegandParity

Definition:
Adds calculated parity bits, or the equivalent of parity bits, to tag data. This makes it possible to use tag or card data formats that include parity bits. Parity bit equivalents can be used if the parity bit(s) are not being tested for validity. This is the same as bitwise left-shifting the data by the given number of bits.

Options:
=1 Adds a single bit of 0 padding (for example in lieu of parity bits) to the least significant end of the data..

=2 Adds calculated odd and even parity bits to Wiegand data.

Default value: =0

Example value: =2

`ScannerWiegandBits`

Definition:
This allows you to specify the bit length of ASCII barcode scanner data which will be sent over the Wiegand interface, or allow that length to be automatically calculated (as 4 times the number of hex characters in the payload).

Options:
`=0` means the number of bits in the payload should be automatically calculated (as 4 times the number of hex characters), or
`=1` to `=255` to set to the number of bits required.

If the number set exceeds the number of bits of data available, then that data will be padded with zero bits, following the `WiegandPaddingMode` setting for whether zeroes are added to the most or least significant bits.

Default value: `=0`

Example value:
`=24`

`WiegandSource`

Definition:
WiegandSource controls which types of event data (pass reads, card/tag reads, serial commands) go to the Wiegand interface.

Options:
The values are determined from a bitwise combination of the following hexadecimal values. Source options include these bit values for mobile passes and tag UIDs:
Bit 7 (0x80) = Mobile Pass (Apple VAS/Google Wallet Smart Tap)
Bit 6 (0x40) = STUID
Bit 5 (0x20) = Write to card emulation mode (`CardEmulationMode=1` see **VTAP NFC tag emulation settings**)
Bit 4 (0x10) = RFU
Bit 3 (0x08) = RFU
Bit 2 (0x04) = Scanners
Bit 1 (0x02) = Command interface messages (`>interface:type:message` see **Dynamic configuration commands**)
Bit 0 (0x01) = Card/Tag UID

Note: If scanner data is sent to the VTAP Wiegand interface, the barcode data is assumed to be a sequence of bytes as ASCII hex digits. The first byte indicates the Wiegand bit length and subsequent bytes contain the Wiegand bit pattern.

Default value: `=A5`

Example value:

```
=80 (hex) will send only mobile NFC pass data
=A1 to send data from NFC passes and NFC cards/tags and card emulation
=83 to send passes, cards/tags and serial commands
```

3.1.11 Serial RS-232 interface

A list of all valid configuration settings for the `config.txt` file, which relate to sending pass information over the Serial, RS-232 interface on the VTAP50 or VTAP100 J1 connector.

Note: The Serial RS-232 port is available on VTAP50, or on VTAP100 v3a hardware onwards.

If `SerialMode=1` and `SerialSource` is set, you can open an application on the computer connected to the VTAP reader, and pass or card data that is read will be sent over the Serial RS-232 interface. `SerialSource` determines whether the Serial RS-232 interface is used to send mobile pass reads only, or cards/tag reads too.

You might need `SerialSettings` to change the standard parameters for data transfer from the 9600, n, 8, 1 default. `SerialStartup` can be used to set a message to transfer when the Serial interface is first enabled.

`SerialPrefix` and `SerialPostfix` can be optionally used to add extra characters before or after a pass read, to suit your application.

When `SerialPassMode=1` you can use all of the other settings beginning `SerialPass...` to extract only part of the mobile pass data. To do this you have to start by identifying a separator character in the data as `SerialPassSeparator`. This allows the data to be regarded as a set of sections that can be identified by number. By identifying a `SerialContentSeparator` this process can be repeated at a lower level, within a section. There is a simple example in the VTAP Configuration Guide.

If you want to receive serial QR/barcode scanner input on this interface, those commands are in the section **Serial barcode/QR scanner**.

Settings below: **SerialMode** | **SerialPassContentMode** | **SerialPassContentSection** | **SerialPassContentSeparator** | **SerialPassLength** | **SerialPassMode** | **SerialPassSection** | **SerialPassSeparator** | **SerialPassStart** | **SerialPostfix** | **SerialPrefix**| **SerialSettings** | **SerialSource** | **SerialStartup**

SerialMode

Definition:
Activate the Serial RS-232 interface for receiving mobile pass data.

Options:
`=1` (default) if you want VTAP data sent over the Serial RS-232 interface,
`=0` switches it off

Default value: `=1`

Example value: `=0`

SerialPassContentMode

Definition:
Turns on a subdivision of a data section into content sections.

Options:
`=0` off (default),
`=1` on.

To use this mode you must also supply `SerialPassContentSeparator` if the default `%` is not the separator you want to use and `SerialPassContentSection`.

Default value: `=0`

Example value: `=1`

SerialPassContentSection

Definition:
Identifies the content section of interest, based on finding `SerialPassContentSeparator` characters. The first content section, before the first separator, is numbered 0.

Options:
Only used if `SerialPassContentMode=1`.

Default value: `=0`

Example value: `=1`

SerialPassContentSeparator

Definition:
Identifies the content separator character in the data which bounds content sections.

Options:
Only used if `SerialPassContentMode=1` and `SerialPassContentSection` is set.

Default value: `=%`

Example value: `=%`

SerialPassLength

Definition:
Number of characters of data to extract, used in conjunction with `SerialPassStart`.

Options:
Defaults to 0 meaning do not truncate data, send all available characters from the pass data, or extracted section of pass data.

Default value: `=0`

Example value: `=10`

SerialPassMode

Definition:
Sets whether to extract parts of the data from the mobile NFC pass.

Options:
`=0` does not extract a character delimited section of the pass payload (`SerialPassSection` is ignored).
`=1` uses the `SerialPassSection` value to extract a character delimited section of the pass payload.

Default value: `=0`

Example value: `=1`

SerialPassSection

Definition:
Identifies the section of interest, based on finding `SerialPassSeparator` characters. The first section, before the first separator, is numbered 0.

Options:
Used only if `SerialPassMode=1`.

Default value: `=0`

Example value: `=2`

SerialPassSeparator

Definition:
Identifies the separator character in the data which bounds sections.

Options:
Only used if `SerialPassMode=1` and `SerialPassSection` is set.

Default value: `=|`

Example value: `=|`

SerialPassStart

Definition:
Number of characters into string to start extracting data, where first character is 0.

Options:
May be used with `SerialPassLength`.

Default value: `=0`

Example value: `=5`

SerialPostfix

Definition:
Adds special characters after data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0D` in the example adds a carriage return after the data,
`%0A` adds a new line after every pass or tag entry.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would append '$t' to the output.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `=%0D`

Example value: `=end%0D`

SerialPrefix

Definition:
Adds special characters before the data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0A` in the example adds a new line linefeed before the data.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would prefix the output with '$t'.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `[Empty]`

Example value: `=%0Astart`

SerialSettings

Definition:
Optional setting used to control the transfer of data over a serial connection.

Options:
Port settings: baud rate, parity, data bits, stop bits.

`=9600,n,8,1` is the default if not specified

Baud rate can be set to 115200 instead of 9600.
Parity could be o for odd, e for even or n for no parity.
8 data bits means that groups of 10 data bits will be sent over the serial link, a start bit, 8 data bits and a stop bit.
The stop bit can be transferred for 1, 1.5 or 2 times the period used to transfer a normal bit.

Default value: `=9600,n,8,1`

Example value: `=9600,n,8,1`

SerialSource

Definition:
SerialSource controls which types of event data (pass reads, card/tag reads, serial commands, scanner inputs) go to the serial interface.

Options:
The values are determined from a bitwise combination of the following hexadecimal values. Source options include these bit values for mobile passes and tag UIDs:
Bit 7 (0x80) = Mobile Pass (Apple VAS/Google Wallet Smart Tap)
Bit 6 (0x40) = STUID
Bit 5 (0x20) = Write to card emulation mode (`CardEmulationMode=1` see **VTAP NFC tag emulation settings**)
Bit 4 (0x10) = RFU
Bit 3 (0x08) = RFU
Bit 2 (0x04) = Scanners
Bit 1 (0x02) = Command interface messages (`>interface:type:message` see **Dynamic configuration commands**)
Bit 0 (0x01) = Card/Tag UID

Default value: `=A5`

Example value:

```
=80 (hex) will send only mobile NFC pass data
=A1 to send data from NFC passes and NFC cards/tags and card emulation
=83 to send passes, cards/tags and serial commands
```

SerialStartup

Definition:
Optional setting for transfer of data over a serial connection.

Options:
`="VTAP100/<version>\r"` is default. May be disabled by setting it to empty.

Default value: `="VTAP100/<version>\r"`

Example value: `=startup message`

3.1.12 Serial2 interface

A list of all valid configuration settings for the `config.txt` file, which relate to sending pass information over the Serial2 port. These settings are used for an RS-485 connection on VTAP100-PAC-485 readers.

Note: The Serial2 port is only available on VTAP100 v4 hardware onwards or VTAP50.

If `Serial2Mode=1` and `Serial2Source` is set, you can open an application on the computer connected to the VTAP reader, and pass or card data that is read will be sent over the Serial2 interface. `Serial2Source` determines whether the Serial2 interface is used to send mobile pass reads only, or cards/tag reads too.

You might need `Serial2Settings` to change the standard parameters for data transfer from the 9600, n, 8, 1 default. Serial2RS485 is used to set up RS-485 transmission over the Serial2 interface, rather than RS-232.

When `Serial2PassMode=1` you can use all of the other settings beginning `Serial2Pass...` to extract only part of the mobile pass data. To do this you have to start by identifying a separator character in the data as `Serial2PassSeparator`. This allows the data to be regarded as a set of sections that can be identified by number. By identifying a `Serial2ContentSeparator` this process can be repeated at a lower level, within a section. There is a simple example in the VTAP Configuration Guide.

If you want to receive serial QR/barcode scanner input on this interface, those commands are in the section **Serial barcode/QR scanner**.

Settings below: **`Serial2Mode`** | **`Serial2PassContentMode`** | **`Serial2PassContentSection`** | **`Serial2PassContentSeparator`** | **`Serial2PassLength`** | **`Serial2PassMode`** | **`Serial2PassSection`** | **`Serial2PassSeparator`** | **`Serial2PassStart`** | **`Serial2Postfix`** | **`Serial2Prefix`** | **`Serial2Settings`** | **`Serial2Source`** | **`Serial2RS485`**

`Serial2Mode`

Definition:
Activate the Serial2 interface for receiving mobile pass data.

Options:
`=1` (default) if you want VTAP data sent over the Serial2 interface,
`=0` switches it off

Default value: `=1`

Example value: `=0`

Serial2PassContentMode

Definition:
Turns on a subdivision of a data section into content sections.

Options:
`=0` off (default),
`=1` on.

To use this mode you must also supply `Serial2PassContentSeparator` if the default `%` is not the separator you want to use and `Serial2PassContentSection`.

Default value: `=0`

Example value: `=1`

Serial2PassContentSection

Definition:
Identifies the content section of interest, based on finding `Serial2PassContentSeparator` characters. The first content section, before the first separator, is numbered 0.

Options:
Only used if `Serial2PassContentMode=1`.

Default value: `=0`

Example value: `=1`

Serial2PassContentSeparator

Definition:
Identifies the content separator character in the data which bounds content sections.

Options:
Only used if `Serial2PassContentMode=1` and `Serial2PassContentSection` is set.

Default value: `=%`

Example value: `=%`

Serial2PassLength

Definition:
Number of characters of data to extract, used in conjunction with Serial2PassStart.

Options:
Defaults to 0 meaning do not truncate data, send all available characters from the pass data, or extracted section of pass data.

Default value: `=0`

Example value: `=10`

Serial2PassMode

Definition:
Sets whether to extract parts of the data from the mobile NFC pass.

Options:
`=0` does not extract a character delimited section of the pass payload (`Serial2PassSection` is ignored).
`=1` uses the `Serial2PassSection` value to extract a character delimited section of the pass payload.

Default value: `=0`

Example value: `=1`

Serial2PassSection

Definition:
Identifies the section of interest, based on finding `Serial2PassSeparator` characters. The first section, before the first separator, is numbered 0.

Options:
Only used if `Serial2PassMode=1`.

Default value: `=0`

Example value: `=2`

Serial2PassSeparator

Definition:
Identifies the separator character in the data which bounds sections.

Options:
Only used if `Serial2PassMode=1` and `Serial2PassSection` is set.

Default value: `=|`

Example value: `=|`

Serial2PassStart

Definition:
Number of characters into string to start reading data, where first character is 0.

Options:
May be used with `Serial2PassLength`.

Default value: `=0`

Example value: `=5`

Serial2Postfix

Definition:
Adds special characters after data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0D` in the example adds a carriage return after the data,
`%0A` adds a new line after every pass or tag entry.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would append '$t' to the output.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `=%0A`

Example value: `=end%0D`

Serial2Prefix

Definition:
Adds special characters before the data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0A` in the example adds a new line linefeed before the data.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would prefix the output with '$t'.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `[Empty]`

Example value: `=%0Astart`

Serial2Settings

Definition:
Optional setting used to control the transfer of data over the serial2 connection.

Options:
Port settings: baud rate, parity, data bits, stop bits.

`=9600,n,8,1` is the default if not specified

Baud rate can be set to 115200 instead of 9600.
Parity could be o for odd, e for even or n for no parity.
8 data bits means that groups of 10 data bits will be sent over the serial link, a start bit, 8 data bits and a stop bit.
The stop bit can be transferred for 1, 1.5 or 2 times the period used to transfer a normal bit.

Default value: `=9600,n,8,1`

Example value: `=9600,n,8,1`

Serial2Source

Definition:
Serial2Source controls which types of event data (pass reads, card/tag reads, serial commands, scanner inputs) go to the Serial2 interface.

Options:
The values are determined from a bitwise combination of the following hexadecimal values. Source options include these bit values for mobile passes and tag UIDs:
Bit 7 (0x80) = Mobile Pass (Apple VAS/Google Wallet Smart Tap)
Bit 6 (0x40) = STUID
Bit 5 (0x20) = Write to card emulation mode (`CardEmulationMode=1` see **VTAP NFC tag emulation settings**)
Bit 4 (0x10) = RFU
Bit 3 (0x08) = RFU
Bit 2 (0x04) = Scanners
Bit 1 (0x02) = Command interface messages (`>interface:type:message` see **Dynamic configuration commands**)
Bit 0 (0x01) = Card/Tag UID

Default value: `=A5`

Example value:

```
=80 (hex) will send only mobile NFC pass data
=A1 to send data from NFC passes and NFC cards/tags and card emulation
=83 to send passes, cards/tags and serial commands
```

Serial2RS485

Definition:
Automatically enables transmit driver for RS-485 transmission, rather than RS-232 over the serial2 interface.

Options:
=1 indicates the serial2 interface is used for RS-485 (on suitable hardware)

Default value: =1

Example value: =0

3.1.13 Serial barcode/QR scanner

These configuration settings for the `config.txt` file relate to enabling a serial scanner input (assumed to be an ASCII sequence) over a virtual COM port, RS-232, RS-485 or 3.3V serial interface. (There is a separate section for **Bluetooth barcode/QR scanner** inputs.)

`...ScannerMode` enables or disables the Serial scanner input feature, where `...` is `ComPort`, `Serial` or `Serial2` to identify the interface being used.

You will need to adjust `...ScannerDelimiter`, where `...` is `ComPort`, `Serial` or `Serial2` to identify the interface, if your scanner inputs are terminated by something other than a new line character.

Settings below: **ComPortScannerMode** | **ComPortScannerDelimiter** | **SerialScannerMode** | **SerialScannerDelimiter** | **Serial2ScannerMode** | **Serial2ScannerDelimiter**

ComPortScannerMode

Definition:
Controls whether a serial QR/barcode scanner input is expected on the COM port interface.
(Note: If this is enabled the COM port interface cannot be used for commands or OSDP.)

Options:
`=0` (default) serial QR code/barcode scanner input not permitted on the ComPort interface
`=1` enable serial QR code/barcode scanner input on the ComPort interface

Default value: `=0`

Example value: `=1`

ComPortScannerDelimiter

Definition:
Controls which character is expected to terminate each serial QR/barcode scanner input on the ComPort interface

Options:
Uses standard ASCII hex characters to describe the keystroke to expect.

`=%0d` (default) will treat a carriage return character as indicating the end of scanner input
`=%0a` would treat a new line as indicating the end of scanner input

Default value: `=%0d`

Example value: `=%0a`

SerialScannerMode

Definition:
Controls whether a serial QR/barcode scanner input is expected on the serial RS-232 interface.
(Note: If this is enabled the serial RS-232 interface cannot be used for commands or OSDP.)

Options:
=0 (default) serial QR code/barcode scanner input not permitted on the serial RS-232 interface
=1 enable serial QR code/barcode scanner input on the serial RS-232 interface

Default value: =0

Example value: =1

SerialScannerDelimiter

Definition:
Controls which character is expected to terminate each serial QR/barcode scanner input on the serial RS-232 interface

Options:
Uses standard ASCII hex characters to describe the keystroke to expect.

=%0d (default) will treat a carriage return character as indicating the end of scanner input
=%0a would treat a new line as indicating the end of scanner input

Default value: =%0d

Example value: =%0a

Serial2ScannerMode

Definition:
Controls whether a serial QR/barcode scanner input is expected on the serial2 interface.
(Note: If this is enabled the serial2 interface cannot be used for commands or OSDP.)

Options:
=0 (default) serial QR code/barcode scanner input not permitted on the serial2 interface
=1 enable serial QR code/barcode scanner input on the serial2 interface

Default value: =0

Example value: =1

`Serial2ScannerDelimiter`

Definition:
Controls which character is expected to terminate each serial QR/barcode scanner input on the serial2 interface

Options:
Uses standard ASCII hex characters to describe the keystroke to expect.

`=%0d` (default) will treat a carriage return character as indicating the end of scanner input
`=%0a` would treat a new line as indicating the end of scanner input

Default value: `=%0d`

Example value: `=%0a`

3.1.14 OSDP interface

A list of all valid configuration settings relating to the OSDP interface for use in the `config.txt` file.

Note: OSDP functionality is not supported on VTAP100 v4a hardware or earlier.

Note: You can use OSDP on any serial interface although using this protocol on the Serial2 RS-485 interface is the most common scenario, creating an RS-485 OSDP connection between a reader and door controller. To use OSDP on a different interface just substitute `Serial2...` in the settings for `Serial...` or `COMPort...`

In order to use these commands, you should first ensure that you have enabled Serial2 as an RS-485 interface using `Serial2RS485=1`, permit the sending of pass card/tag payloads over the Serial2 interface using `Serial2Source=A1` and chosen appropriate settings for `Serial2Settings` to match the required interface speed.

Settings below: **`Serial2OSDP`** | **`Serial2OSDPAddress`** | **`Serial2OSDPKeySlot`** | **`Serial2OSDPFormat`** | **`Serial2OSDPSecureOnly`**

`Serial2OSDP`

Definition:
Enable or disable use of OSDP on the Serial2 interface.

Options:
`=1` enables OSDP on the Serial2 interface,
`=0` disables OSDP on the Serial2 interface (default)

Default value: `=0`

Example value: `=1`

`Serial2OSDPAddress`

Definition:
Assigns the VTAP reader a numerical address, as a peripheral device under OSDP

Options:
Value 0 to 126

Default value: `=0`

Example value: `=1`

`Serial2OSDPKeySlot`

Definition:
Assigns any one of the 9 available encryption key slots, as the one to be used to hold the OSDP Secure Channel Base key

Options:
`=1` to `=9`, to refer to the application key files uploaded as `appkey1.txt` through `appkey9.txt`
`=0` no key slot specified, implies not secure channel

Default value: `=0`

Example value: `=1`

Serial2OSDPFormat

Definition:
Choose ASCII or Wiegand (raw) format for payload

Options:
`=Wiegand`, where the output will be a binary number comprised of a bit count and bit stream

Any value other than `=Wiegand` will output the tag data as ASCII text (not zero terminated). This can include the following values, used to define card event data to include, or require additional formatting of the output:

`$d` is the tag data,
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**),
`$$t` would append '$t' to the output,
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for keyslot used (1 to 6),
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.
`\n` is a new line character,
`\t` is a tab character,
`\\` would append '\' to the output.,
`\0` is binary 0,
`%XX` , where `XX` is a two character hex digit representing the binary value of a character to use (URL encoding style),
`%%` would append '%' to the output.

Default output, if this setting is omitted or no value is set, is equivalent to setting `=$t$n:$d` which will return data in the form `A16:123456` where the tag data is `123456`

Default value: `=$t$n:$d`

Example value:

```
=Wiegand
=start:$d:end ; returns 'start:123456:end' if 123456 is the tag data
=$t$n,$u:$d ; returns UID separated from tag type and merchant key slot
by a comma
```

Serial2OSDPSecureOnly

Definition:
Used to require that only the Secure profile is used, after initialisation.

Options:
`=1` is enabled to enforce secure mode.
`=0` is not enabled. It is possible that the implementation of OSDP on the controller would allow the VTAP reader to fall back to using the Basic profile, if Secure credentials failed.

Default value: `=1`

Example value: `=0`

3.1.15 Bluetooth BLE keyboard emulation

A list of all valid configuration settings for the `config.txt` file which relate to sending pass information over a BLE keyboard emulation interface, when the VTAP PRO is in Local mode.

Note: Bluetooth is only available on VTAP100-PRO-BW hardware.

`BTKeyboardMode` enables/disables the BLE keyboard emulation. When enabled, you can pair and connect the VTAP reader with a Bluetooth host device (such as a tablet or laptop), and start receiving the pass or card data on the text editor or other application that expects to receive a keyboard input. Like USB keyboard emulation, you need to ensure your cursor is at the right place (in an appropriate app) to receive the input.

When scanning for pairing or connecting, the VTAP PRO name will match its 'VTAP label' (check `boot.txt` or label on the bottom or the case for this serial number). By default no PIN is set, so the host device can automatically pair and connect to the VTAP PRO. Should you need to change the Bluetooth name for your VTAP PRO, use `BTKeyboardName`. If you want to add a PIN for pairing set `BTKeyboardPIN`.

`BTKeyboardSource` determines whether BLE keyboard mode responds to only mobile pass reads, or card/tag reads too. You might need a `BTKeyboardDelayMS` value to insert a delay between key presses, to suit your receiving application

`BTKeyboardPrefix` and `BTKeyboardPostfix` can be optionally used to add extra characters before or after a pass read, to suit your application.

When `BTKeyboardPassMode=1`, a character delimited section of the pass payload will be extracted. The delimiting character defaults to `|` but this can be changed to another character using `BTKeyboardPassSeparator`. The payload is then considered as comprising a number of sections, any of which can be selected using `BTKeyboardPassSection`. By identifying a `BTKeyboardContentSeparator` this process can be repeated at a lower level, within a section. This follows the same approach as over a USB keyboard emulation, for which there is a simple example in the VTAP Configuration Guide.

Settings below: **`BTKeyboardDelayMS`** | **`BTKeyboardMode`** | **`BTKeyboardName`** | **`BTKeyboardPassContentMode`** | **`BTKeyboardPassContentSection`** | **`BTKeyboardPassContentSeparator`** | **`BTKeyboardPassLength`** | **`BTKeyboardPassMode`** | **`BTKeyboardPassSection`** | **`BTKeyboardPassSeparator`** | **`BTKeyboardPassStart`** | **`BTKeyboardPIN`** | **`BTKeyboardPostfix`** | **`BTKeyboardPrefix`** | **`BTKeyboardSource`**

BTKeyboardDelayMS

Definition:
Inserts a number of milliseconds delay between key down and key up in the BLE keyboard output.

Options:
Adjust to suit your receiving application between 0ms and 255ms.

Default value: `=0`

Example value: `=5`

BTKeyboardMode

Definition:
Turns BLE keyboard emulation on or off.

Options:
`=1` enables BLE keyboard emulation,
`=0` switches it off (default)

Default value: `=0`

Example value: `=1`

BTKeyboardName

Definition:
This is the name that the VTAP BLE keyboard will advertise for discovery by a host device.

Options:
Up to 15 characters. If something longer is set it will be truncated.
Defaults to the VTAP assigned serial number. (VTAP Label value in `boot.txt`.)

Default value: `=<VTAP Label>`

Example value: `=CC123456`

BTKeyboardPassContentMode

Definition:
Turns on a subdivision of a data section into smaller content sections.

Options:
=0 off (default),
=1 on

To use this mode you must also supply `BTKeyboardPassContentSeparator` if the default `%` is not the separator you want to use and `BTKeyboardPassContentSection`.

Default value: `=0`

Example value: `=1`

BTKeyboardPassContentSection

Definition:
Identifies the content section of interest, by finding BTKeyboardPassContentSeparator characters. The first content section, before the first separator, is numbered 0.

Options:
Used only if `BTKeyboardPassContentMode=1`.

Default value: `=0`

Example value: `=1`

BTKeyboardPassContentSeparator

Definition:
Identifies the content separator character in the data which bounds content sections.

Options:
Used only if `BTKeyboardPassContentMode=1` and a `BTKeyboardPassContentSection` is set.

Default value: `=%`

Example value: `=%`

BTKeyboardPassLength

Definition:
Number of characters of data to extract, starting from the position defined by `BTKeyboardPassStart`.

Options:
Defaults to `0` meaning do not truncate data, send all available characters from the pass data, or extracted section of pass data.

Default value: `=0`

Example value: `=10`

BTKeyboardPassMode

Definition:
Sets whether to extract parts of the data from the mobile NFC pass.

Options:
`=0` does not extract a character delimited section of the pass payload (`BTKeyboardPassSection` is ignored).
`=1` uses the `BTKeyboardPassSection` value to extract a character delimited section of the pass payload.

Default value: `=0`

Example value: `=1`

BTKeyboardPassSection

Definition:
Identifies the section of interest, based on finding BTKeyboardPassSeparator characters. The first section, before the first separator, is numbered 0.

Options:
Used only if `BTKeyboardPassMode=1`.

Default value: `=0`

Example value: `=2`

BTKeyboardPassSeparator

Definition:
Identifies the separator character in the data which bounds sections.

Options:
Used only if `BTKeyboardPassMode=1` and a `BTKeyboardPassSection` is set.

Default value: `=|`

Example value: `=|`

BTKeyboardPassStart

Definition:
Number of characters into pass data to start reading, where first character is 0.

Options:
May be used with `BTKeyboardPassLength`.

Default value: `=0`

Example value: `=5`

BTKeyboardPIN

Definition:
Optional PIN for pairing the VTAP PRO reader with host device such as PC or tablet.

Options:
Numeric, 6 digits. By default a PIN is not set.

If a PIN of less than 6 digits is set, that PIN will be padded with leading zeros.
So, if you set BTKeyboardPIN=1234, your PIN will actually be 001234.

Default value: `N/A`

Example value: `=123456`

BTKeyboardPostfix

Definition:
Adds special characters after BLE keyboard data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0D` in the example adds a carriage return after the data,
`%0A` adds a new line after every pass or tag entry.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would append '$t' to the output.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `=%0A`

Example value: `=end%0D`

BTKeyboardPrefix

Definition:
Adds special characters before BLE keyboard data, if needed.

Options:
Use standard ASCII hex characters to describe the keystrokes to append.

`%0A` in the example adds a new line linefeed before the data.
`$t` would include the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).
`$$t` would prefix the output with '$t'.
`$t$n` will follow the pass type character with a digit for VAS merchant ID/ST collector ID index (1 to 6) and a digit for key slot used (1 to 6).
`$u` adds the UID as 8, 14 or 16 hex ASCII digits.

Default value: `[Empty]`

Example value: `=%0Astart`

`BTKeyboardSource`

Definition:
Controls which types of event data (pass reads, card/tag reads, serial commands) are sent as a BLE keyboard entries.

Options:
The values are determined from a bitwise combination of the following hexadecimal values. Source options include these bit values for mobile passes and tag UIDs:
Bit 7 (0x80) = Mobile Pass (Apple VAS/Google Wallet Smart Tap)
Bit 6 (0x40) = STUID
Bit 5 (0x20) = Write to card emulation mode (`CardEmulationMode=1` see **VTAP NFC tag emulation settings**)
Bit 4 (0x10) = RFU
Bit 3 (0x08) = RFU
Bit 2 (0x04) = Scanners
Bit 1 (0x02) = Command interface messages (`>interface:type:message` see **Dynamic configuration commands**)
Bit 0 (0x01) = Card/Tag UID

Default value: `=A5`

Example value:

```
=80 (hex) will send only mobile NFC pass data
=A1 to send data from NFC passes and NFC cards/tags and card emulation
=83 to send passes, cards/tags and serial commands
```

3.1.16 Bluetooth BLE beacon

A list of all valid configuration settings for the `config.txt` file which relate to enabling a Bluetooth advertisement beacon, when the VTAP PRO is in Local mode.

Note: Bluetooth is only available on VTAP100-PRO-BW hardware.

BLE Beacon functionality is enabled using the `BLEBeacon` setting.

Settings below: **BLEBeacon**

BLEBeacon

Definition:
Enables the BLE advertisement beacon on VTAP100-PRO-BW for use with any BLE beacon detecting application. BLE beacons can be used to provide location notifications, similar to GPS, but which will work where GPS will not, such as for indoor locations. One application is to trigger location notifications associated with Apple Wallet passes. When issuing a pass, a pass provider may set a location value that could comprise GPS coordinates or a BLE beacon UUID, with optional major and minor values.

Options:
The syntax used is

```
BLEBeacon=i,<name_for_logging>,<uuid>,<major>,<minor>
```

`<name_for_logging>` is an optional setting, up to 15 characters long, to allocate a name to this BLE beacon that may be reported. An example use would be a VTAP PRO reader in Cloud mode returning this name, as part of the tap information sent to a customer application, to indicate which beacon was being advertised at the time of the tap.
`<uuid>` is the UUID (unique identifier) of the beacon to advertise. You can use any value. If intended for use with Apple Wallet passes, the UUID must match that entered in the location field of the pass.
`<major>` and `<minor>` are decimal values (0 to 65535) that must be specified as part of the BLE beacon advertisement. A beacon detecting application can be configured to trigger when a specific UUID, and optionally specific major and/or minor numbers match those being advertised. For example, an Apple Wallet pass might use wildcard values to match any major or minor values with a specific UUID, or it might be set to only trigger when a specific UUID and major number is detected. This allows, for instance, detection of all branches of a franchise, or only those in specific regions (based on a matching major), or individual locations (based on matching major and minor).

Default value: `N/A`

Example value:

```
=i,loyaltybeacon,4CE2EF69-4414-469D-9D55-3EC7FCC31234,1,1
```

3.1.17 Bluetooth BLE GATT server

A list of all valid configuration settings for the `config.txt` file which relate to sending pass information over a BLE GATT server interface, when the VTAP PRO is in Local mode.

Note: Bluetooth is only available on VTAP100-PRO-BW hardware.

`BTServerMode` is used to switch between BLE GATT server and BLE keyboard operation. These two modes cannot be used at the same time.

When a BLE GATT server mode is enabled, use `BTServerName` if you want the Bluetooth advertised device name to be anything other than the VTAP reader's serial number. If you want to add a PIN for pairing set `BTServerPIN`.

Settings below: **`BTServerMode`** | **`BTServerName`** | **`BTServerPIN`**

`BTServerMode`

Definition:
Turns BLE GATT server on or off.

Options:
`=1` enables BLE GATT server (overrides any `BTKeyboardMode` setting),
`=0` switches it off (default)

Default value: `=0`

Example value: `=1`

`BTServerName`

Definition:
This is the name that the VTAP BLE GATT server will advertise for discovery by a host device.

Options:
Up to 15 characters. If something longer is set it will be truncated.
Defaults to the VTAP assigned serial number. (VTAP Label value in `boot.txt`.)

Default value: `=<VTAP Label>`

Example value: `=CC123456`

`BTServerPIN`

Definition:
Optional PIN for pairing the VTAP PRO reader with host device such as PC or tablet.

Options:
Numeric, 6 digits. By default a PIN is not set.

If a PIN of less than 6 digits is set, that PIN will be padded with leading zeros.
So, if you set BTKeyboardPIN=1234, your PIN will actually be 001234.

Default value: `N/A`

Example value: `=123456`

3.1.18 Bluetooth barcode/QR scanner

These configuration settings for the `config.txt` file relate to enabling a Bluetooth scanner input. (There is a separate section for **Serial barcode/QR scanner** inputs.)

Note: Bluetooth is only available on VTAP100-PRO-BW hardware.

By using these settings a Bluetooth classic HID barcode or QR scanner can be paired with a VTAP reader, so that the scanner output can be sent on by the VTAP reader just like pass, card or tag reads.

`BTScannerMode` enables or disables the Bluetooth scanner input feature.

`BTScannerOutput` sets the interface over which the scanned payload will be sent, and `ScannerWiegandBits` can be used if this data needs to be sent over Wiegand.

Note: These settings have been tested with a NETUM barcode scanner. Please contact **vtap-support@dotorigin.com** if you have any difficulty implementing these settings for a different type of scanner.

Settings below: **`BTScannerMode`** | **`BTScannerOutput`**

`BTScannerMode`

Definition:
Enable/disable Bluetooth scanner mode. If this is enabled and the VTAP PRO reader is not currently paired with a scanner, it will begin scanning to find a barcode/QR scanner it can pair with. When paired, scanning for new devices stops. When disabled any paired devices are removed and forgotten.

Options:
`=0` disabled
`=1` enabled

Default value： `=0`

Example value： `=1`

BTScannerOutput

Definition:
Controls which VTAP interface will be used to send the scanned payload from the paired scanner

Options:
The syntax required is `BTScannerOutput=<interface1>,<interface2>,<interface3>` with
`=c` for output to ComPort
`=s` for output over Serial RS-232
`=k` for output to USB keyboard emulation
`=w` for output over Wiegand (where the hardware allows)

Multiple selections can be separated with commas as in the examples.

Default value: `N/A`

Example value:
`=s`
`=k,c`
`=c,s,k`

3.1.19 LED control

A list of all valid configuration settings which relate to LED behaviour for use in the `config.txt` file.

> **Note:** VTAP100 and some VTAP50 readers have factory fitted LEDs, but these settings can also control an external RGB or serial LEDs fitted.

The VTAP100 has diagnostic LEDs for factory use and user feedback LEDs.

Use `LEDMode` to control the behaviour of any diagnostic LEDs. `LEDSelect` chooses to use user feedback LEDs in a position compatible with the case around the VTAP100. `LEDDefaultRGB` sets the default colour for those user feedback LEDs, when pass reads or errors are not being signalled.

`PassLED` chooses the colour to flash on user feedback LEDs, with duration and repeats, when a mobile pass is read. `TagLED` does the same for cards/tags. `PassErrorLED` chooses the colour to flash on user feedback LEDs, duration and repeats, when an error occurs during a mobile pass read.

`StartLED` chooses the LED colour, duration and repeats to trigger on startup.

There is a simple example in the VTAP Configuration Guide.

Settings below: **`LEDMode`** | **`LEDSelect`** | **`PassErrorLED`** | **`PassLED`** | **`TagLED`** | **`LEDDefaultRGB`** | **`LEDMaxSerial`** | **`LEDSerialGRB`** | **`LEDSerialRGB`** | **`StartLED`**

`LEDMode`

Definition:
Chooses a default mode of operation for diagnostic LEDs

Options:
`=0` Factory use LEDs 1-4 will flash in sequence but all on during NFC activity
`=1` Factory use LEDs all off but all on during NFC activity
`=2` Factory use LEDs will flash in sequence but no reaction to NFC activity
`=3` Factory use LEDs always off

Default value: `=0`

Example value: `=1`

LEDSelect

Definition:
Selects which RGB LED to use in response to pass reads, to suit the type of case around a VTAP100. For a VTAP50 using an external RGB LED board, use it to select the appropriate common cathode or common anode connection.

Options:
=0 External RGB LED (common cathode),
=1 on VTAP100: on-board LED pair shows through compact case window; On VTAP50: external RGB LED (common anode),
=2 on VTAP100: on-board LED pair shows through square case window; On VTAP50: external RGB LED (common anode),
=3 on VTAP50 only: External and on-board serial LEDs, where fitted.

Default value: =1

Example value: =2

PassErrorLED

Definition:
Chooses LED colour to flash on presentation of mobile NFC pass, if an error occurs

Options:
LED colour (use any hex RGB colour value) , LED on ms, LED off ms, number of repeats

Example sets a long 500ms flash in red on presentation of mobile NFC pass, if an error occurs

Default value: N/A

Example value: =FF0000,500

PassLED

Definition:
Chooses LED colour to flash on successful presentation of mobile NFC pass

Options:
LED colour (use any hex RGB colour value), LED on time in ms, LED off time in ms, number of repeats

Example sets two 100ms orange LED flashes, spaced by 50ms, on presentation of mobile NFC pass

Default value: N/A

Example value: =FF8000,100,50,2

TagLED

Definition:
Chooses LED colour to flash on presentation of a card/tag

Options:
LED colour (use any hex RGB colour value), LED on time in ms, LED off time in ms, number of repeats

Example sets 1 500ms green LED flash on presentation of card/tag

Default value: `N/A`

Example value: `=00FF00,500`

LEDDefaultRGB

Definition:
Chooses the default colour of the RGB LED. [On VTAP50 v2 only: A serial LED pattern can be chosen.]

Options:
`=FF8000` sets the LED to a warm orange colour as its default state, use any hexadecimal RGB value.
The second example sets the LED to white on most VTAP readers, but on a VTAP50 v2 with serial the LEDs are being instructed to follow a flash sequence defined in the `seq.comet` section of the `leds.ini` file.

Default value: `=FFFFFF`

Example value:
`=FF8000`

`=FFFFFF:seq.comet@leds.ini`

LEDMaxSerial

Definition:
Limits the maximum number of LEDs in a serial LED chain or matrix. [VTAP50 v2 only] Not required for external RGB LEDs.

Options:
The VTAP50 v2 has a limit of 1 to 255 serial LEDs in a chain. If future versions permit longer chains, there may be a need to limit the number used, for backward compatibility.

Default value: `=24`

Example value: `=255`

LEDSerialGRB

Definition:
Identify any serial external LEDs used which follow a GRB byte order when a mixture of GRB and RGB types are used. (When not specified a GRB byte order is the default assumed for all serial external LEDs.) [VTAP50 v2 only]

Options:
List LED numbers 1 to 255 that follow GRB byte order, separated by commas, or identify ranges such as 18-24, where the start and end of the range are separated by a hyphen.

If you accidentally include an LED number in both `LEDSerialRGB` and `LEDSerialGRB` settings it will be treated as RGB.

Default value: `=1-255`

Example value: `=3-9,16,18-24`

LEDSerialRGB

Definition:
Identify any serial external LEDs used which follow a RGB byte order when a mixture of RGB and GRB types are used. (When not specified a GRB byte order is the default assumed for all serial external LEDs.) [VTAP50 v2 only]

Options:
List LED numbers 1 to 255 that follow RGB byte order, separated by commas, or identify ranges such as 10-15, where the start and end of the range are separated by a hyphen, or =all to affect all LEDs.

If you accidentally include an LED number in both `LEDSerialRGB` and `LEDSerialGRB` settings it will be treated as RGB. Any LED number not set by `LEDSerialRGB` will be treated as GRB.

Default value: `N/A`

Example value: `=1,2,10-15,17`

StartLED

Definition:
Chooses LED colour or sequence on start up

Options:
LED colour (use any hex RGB colour value), LED on time in ms, LED off time in ms, number of repeats

Example sets two 100ms orange LED flashes, spaced by 50ms, on startup

Default value: `N/A`

Example value: `=FF8000,100,50,2`

3.1.20 Buzzer control

A list of all valid configuration settings which relate to buzzer behaviour for use in the `config.txt` file.

`PassBeep` chooses the buzzer duration and repeats to trigger when a mobile pass is read, following the same pattern as LED controls. `TagBeep` does the same for cards/tags. `PassErrorBeep` chooses the buzzer duration and repeats to trigger when an error occurs during a mobile pass read.

`StartBeep` chooses the buzzer duration and repeats to trigger on startup.

Note: Buzzer frequency cannot be changed on older VTAP100 readers, with v4a hardware or earlier.

A sequence of frequencies can be defined, in a section of the `config.txt` file or a separate text file, or by a single channel format 0 MIDI file, to produce a simple tune or chime rather than a monotone beep.

There are examples in the VTAP Configuration Guide and the syntax for describing a note sequence is explained there.

Settings below: **`PassBeep`** | **`PassErrorBeep`** | **`TagBeep`** | **`StartBeep`**

`PassBeep`

Definition:
Sets a beep on successful presentation of mobile NFC pass

Options:
Beep on ms, beep off ms, number of repeats[, optional beep frequency (defaults to 3136Hz if not set)][:<section_name>@<text_file> for beep sequence]
The first example sets two 100ms beeps, spaced by 50ms, at default frequency, on presentation of mobile NFC pass
Note: Setting frequency to 0, rather than omitting the setting, will switch the buzzer off.
The second example follows a beep sequence defined in the section `[passbeep]` of the `config.txt` file on the VTAP reader.
Note: If that section is not found the reader will default to the defined monotone beep.

Default value: `N/A`

Example value:

```
=100,50,2
=100,50,2:passbeep@config.txt
```

PassErrorBeep

Definition:
Sets a beep on presentation of mobile NFC pass, if an error occurs.

Options:
Beep on ms, beep off ms, number of repeats[, optional beep frequency (defaults to 3136Hz if not set)][:<section_name>@<text_file> for beep sequence]
Example sets a long 500ms beep, at default frequency, on presentation of mobile NFC pass if an error occurs
Note: Setting frequency to 0, rather than omitting the setting, will switch the buzzer off.
The second example follows a beep sequence defined in the section `[passerrorbeep]` of the `leds.ini` text file on the VTAP reader.
Note: If that section is not found the reader will default to the defined monotone beep.

Default value: `N/A`

Example value: `=500`

```
=500:passerrorbeep@leds.ini
```

TagBeep

Definition:
Sets a beep on presentation of card/tag

Options:
Beep on ms, beep off ms, number of repeats[, optional beep frequency (defaults to 3136Hz if not set)][:<section_name>@<text_file> for beep sequence]
Example sets 1 100ms beep on presentation of a card/tag at default frequency
Note: Setting frequency to 0, rather than omitting the setting, will switch the buzzer off.
The second example follows a beep sequence defined in the section `[tagbeep]` of the `beeps.ini` text file on the VTAP reader.
Note: If that section is not found the reader will default to the defined monotone beep.

Default value: `N/A`

Example value:

```
=100
=100:tagbeep@beeps.ini
```

StartBeep

Definition:
Sets a beep on startup

Options:
Beep on ms, beep off ms, number of repeats[, optional beep frequency (defaults to 3136Hz if not set)][:<section_name>@<text_file> for beep sequence]
Example sets three 200ms beeps, spaced by 200ms, at 1000Hz on startup
Note: Setting frequency to 0, rather than omitting the setting, will switch the buzzer off.
The second example follows a beep sequence defined in the midi file `chime.mid` on the VTAP reader.
Note: If that file is not found the reader will default to the defined monotone beep.

Default value: `N/A`

Example value: `=200,200,3,1000`

`=200,200,3,1000:midi@chime.mid`

3.1.21 VTAP NFC tag emulation

A list of all configuration settings which relate to VTAP NFC tag emulation for the `config.txt` file.

VTAP can emulate an NFC Forum Type 4 card or tag, containing NDEF encoded data. If you tap a phone on a VTAP in card emulation mode, a URL could be launched or Text data displayed on the tapping phone.

The VTAP emulated tag can also be written to, for example by an Android or iOS app. Any NDEF record data written by the app will not overwrite the emulated tag data, but instead is intercepted by the VTAP and could be output over any of the VTAP comms interfaces, treated as though it was a data read from an NFC tag.

`CardEmulationMode` lets you enable VTAP NFC tag emulation and `CardEmulationData` sets the data that will be passed when the VTAP is in a card emulation mode. These settings can be adjusted dynamically using the `?card` and `?cardmode` commands in the section **Dynamic configuration commands**.

There is more information in the VTAP Application Notes.

Settings below: **`CardEmulationMode`** | **`CardEmulationData`**

`CardEmulationMode`

Definition:
This enables or ends a mode, where the VTAP emulates an NFC Forum Type 4 card or tag, containing NDEF encoded data.

Options:
=0 to leave card emulation mode,
=1 to enter card emulation mode,
=2 to enter mixed mode, where the VTAP reads tags and passes but will also emulate a card.
For more detail refer to VTAP Application Notes.

Default value： `=0`

Example value： `=1`

CardEmulationData

Definition:
This defines the NDEF data to be sent by a VTAP reader that is in card emulation mode.

Options:
Format is =<type><:lang>,<NDEF>
<NDEF> data is assumed to be text if a <type> TEXT/URI/RAW/FILE is not included
TEXT (T) type is assumed to be English (en) unless another 2 character <lang>uage code is used
URI (U) requires a valid web address
RAW (R) is a raw binary message part for an NDEF record. Its length will be automatically set.
FILE (F) points to a text file on the VTAP containing one of the other command types.
For more detail refer to VTAP Application Notes.

Default value: `N/A`

Example value:

```
=TEXT,Hello World!
=URI,http://www.vtapnfc.com
=RAW,D101055402656e4869
=FILE,tagdata.txt
```

3.1.22 VTAP PRO in Local mode settings

A list of valid configuration settings relating to the VTAP PRO, which you may notice or need to use when in Local mode, in the `config.txt` file.

`CloudMode` and `CloudTapMode` are factory settings which control whether a VTAP PRO reader needs to make an IP-connection to the VTAP Cloud for management and/or tap processing. The alternative to Cloud Mode is Local Mode where a VTAP Cloud connection is not needed.

Note: The optional VTAP100 PRO I/O expansion board (VTAP100-PRO-EXP1) is required to use `PassRelay` and `TagRelay` settings.

Commands below: **`CloudMode`** | **`CloudTapMode`** | **`PassRelay`** | **`TagRelay`**

`CloudMode`

Definition:
A factory setting which determines whether a VTAP PRO reader needs to make an IP-connection to the VTAP Cloud for configuration and processing of tap data (Cloud mode) or whether configuration and processing of tap data is exclusively local to the VTAP reader (Local mode).

Options:
`=0` default value selects Local mode (only an option for VTAP100-PRO-BW readers)

`=1` will be the factory setting for a VTAP PRO reader sold in Cloud mode, for use with VTAP Cloud

Default value: `=0`

Example value: `=1`

`CloudTapMode`

Definition:
A factory setting which determines whether a VTAP PRO reader sends tap data to the VTAP Cloud for processing. (Contact **vtap-support@dotorigin.com** if you need help formatting tap data in the correct way for processing.)

Options:
`=0` selects tap processing locally on the VTAP reader. Tap data is not sent to VTAP Cloud.

`=1` is the factory setting for a VTAP PRO reader sold in Cloud mode, when tap processing is to be handled in the VTAP Cloud.

`=2` permits tap data to be sent to a custom URL for tap processing.

Default value: `=1`

Example value: `=0`

PassRelay

Definition:
Controls either of the two relays on the VTAP100-PRO in Local mode in response to a successful pass read

Options:
Syntax required is:

```
PassRelay=<relay_num> <action>[=param1[,param2][…]] [action[=params]]
```

<relay_num> is currently either 0 or 1 to refer to the two available relays

<action> can be one or more in a series of actions, as required, each separated by a space.
Actions can be one of:

```
on
off
timed=ontime[,offtime][,count]
delay=<time in ms>
```

If the off time is not specified, the on time is used. If the count is not specified, 1 is used. Delay is in milliseconds but can specify time in seconds or minutes with 's' or 'm'.

Default value: N/A

Example value:

```
PassRelay=0 timed=6000
;Switch ON relay 0 for 6000ms
```

TagRelay

Definition:
Controls either of the two relays on the VTAP100-PRO in Local mode in response to a successful card/tag read

Options:
Syntax required is:

```
TagRelay=<relay_num> <action>[=param1[,param2][…]] [action[=params]]
```

<relay_num> is currently either 0 or 1 to refer to the two available relays

<action> can be one or more in a series of actions, as required, each separated by a space.
Actions can be one of:

```
on
off
timed=ontime[,offtime][,count]
delay=<time in ms>
```

If the off time is not specified, the on time is used. If the count is not specified, 1 is used. Delay is in milliseconds but can specify time in seconds or minutes with 's' or 'm'.

Default value： N/A

Example value：

```
TagRelay=1 timed=1000,1000,2
;Switch ON relay 1 for 1000ms on, 1000ms off, repeated twice
```

3.1.23 Other settings

Other valid configuration settings for use in the `config.txt` file .

`MassStorageEnable` provides an alternative to a hardware lock on VTAP readers where jumpers may be difficult to access, but there is an enabled host connection via serial or USB.

`NFCDefaultEnable` allows the VTAP reader NFC field to be turned on or off to manage power consumption, where `SleepMode` and `SleepInactivityTimeMS` can be used to enter sleep mode, either immediately after startup or after a defined period of inactivity. `StartupDelayMS` provides a delay during start up to allow power to fully stabilise, for use primarily in situations with a Wiegand interface.

Settings below: **`LowPowerCardDetect`** | **`MassStorageEnable`** **`NFCDefaultEnable`** | **`SleepInactivityTimeMS`** | **`SleepMode`** | **`StartupDelayMS`**| **`ZmodemRxTimeout`**

NEW

`LowPowerCardDetect`

Definition:
Enable this setting to make use of low power card detection [VTAP 25 only] to reduce the average current used in an NFC polling loop. This is particularly useful for situations where a VTAP reader module is integrated in battery powered equipment. (`SleepMode` and `SleepInactivityMS` options can be used in conjunction with the `LowPowerCardDetect` option, to significantly reduce the power requirements of the VTAP reader in normal use.)

Options:
`=1` use low power card detection to reduce the current used in the NFC polling loop
`=0` do not use power reduction measures

Reboot after changing this setting for the change to take effect.

Default value: `=0`

Example value: `=1`

MassStorageEnable

Definition:
Allows a remote host to completely remove or restore mass storage access to VTAP readers with a config.txt setting. An alternative to a hardware lock, which avoids the need for changes to jumpers on the PCB, which may not be readily accessible.

Options:
Default value is enabled.

If the value is set `=0` when there are no command interfaces on any other serial ports, on reboot the setting will be ignored (treated as if it were `=1`), to avoid becoming locked out of your VTAP reader.

Default value: `=1`

Example value: `=0`

NFCDefaultEnable

Definition:
Determines whether NFC is enabled [default] or disabled at startup, in order to manage power consumption.

Options:
In the example NFC will be disabled at startup, to save power.

Default value: `=1`

Example value: `=0`

NEW

SleepInactivityTimeMS

Definition:
Allows you to specify a duration of NFC and serial comms inactivity, after which the VTAP reader will automatically enter low power sleep mode.

Sleep mode primarily runs the VTAP processor at very low power, shutting down all activity except monitoring for NFC and serial comms. Entering sleep mode also switches off all LED activity, which will resume when sleep mode ends.

(This is not available on older VTAP100 readers, with v4a hardware or earlier.)

Options:
Use any positive value in milliseconds, such as 5000.

Be aware that inactivity is checked approximately every 10ms, so the minimum effective value is 10. The timer will be reset by any communications, including a configuration change, and by anything detected in the NFC field.

`=0` is the default, where sleep mode is never triggered by inactivity

A VTAP reader will wake from sleep mode when there is an NFC event (tapping a card or phone) or receiver activity on a physical serial port. See the `?sleep` command for more details about waking from sleep mode.

CAUTION: As sleep mode disables USB completely, USB COM port activity cannot be used as a trigger to wake up the VTAP reader, although the reader should still wake when you tap a card or pass. Sleep mode is therefore not recommended for applications using only VTAP USB device interfaces.

Default value: `=0`

Example value: `=5000`

NEW

SleepMode

Definition:
Determines whether sleep mode should begin immediately after startup.

Sleep mode primarily runs the VTAP processor at very low power, shutting down all activity except monitoring for NFC and serial comms activity. Entering sleep mode also switches off all LED activity, which will resume when sleep mode ends.

(This is not available on older VTAP100 readers, with v4a hardware or earlier.)

Options:
`=1` enter sleep mode immediately after startup
`=0` do not enter sleep mode immediately after startup

A VTAP reader will wake from sleep mode when there is an NFC event (tapping a card or phone) or receiver activity on a physical serial port. See the `?sleep` command for more details about waking from sleep mode.

CAUTION: As sleep mode disables USB completely, USB COM port activity cannot be used as a trigger to wake up the VTAP reader, although the reader should still wake when you tap a card or pass. Sleep mode is therefore not recommended for applications using only VTAP USB device interfaces.

Default value: `=0`

Example value: `=1`

StartupDelayMS

Definition:
Delay in milliseconds before fully starting up to allow the power supply to stabilise.

Options:
Use a value such as 5000 when using an external power supply.
This could prevent possible file system corruption during installation, if VTAP could be wired up to a live external power supply (typically when using Wiegand or RS-485 expansions).

Default value: `=1000`

Example value: `=5000`

ZmodemRxTimeout

Definition:
Set the receive timeout to an appropriate duration.

Options:
Number of milliseconds

Default value: `=1000`

Example value: `=2000`

3.2 command.txt

This section lists all valid commands for `command.txt`. This is a file which must contain the string `!VTAPcommand` at the start.

There is a simple example in the VTAP Configuration Guide.

Commands below: **`reboot`** | **`refresh`** | **`remount`**

`reboot`

Definition:
Power cycle the VTAP, as if the USB cable was disconnected momentarily.

Options:

Default value: `N/A`

Example value: `reboot`

`refresh`

Definition:
Force the VTAP to re-read `config.txt`. This useful if you have renamed a file to be `config.txt` as file renaming is not automatically recognized by the VTAP.

Options:

Default value: `N/A`

Example value: `refresh`

`remount`

Definition:
Remove the drive from the operating system briefly and then attach it again. This will force the operating system to re-read any changes that were made to the file system by the VTAP.

Options:

Default value: `N/A`

Example value: `remount`

3.3 lock.txt

This section lists all valid commands for `lock.txt` which is needed to work with the software configuration lock. This is a file which must contain the string `!VTAPlock` at the start.

There is a simple example in the VTAP Configuration Guide.

Commands below: **`lock`** | **`unlock`**

`lock`

Definition:
Sets the password to the given value and locks the VTAP.

Options:

Default value: `N/A`

Example value: `=APa55word`

`unlock`

Definition:
Offers the given password to unlock the VTAP.

Options:

Default value: `N/A`

Example value: `=APa55word`

4 Commands

These are all of the commands that can be sent over a serial interface to control your VTAP reader.

Your command interface could be a virtual COM port or serial port. The commands can be sent using a program such as TeraTerm or PuTTY-nd for instance.

Situations where these might be used are described in the VTAP Serial Integration Guide. These include managing a VTAP reader in passive mode, changing `config.txt` or transferring files to the VTAP reader remotely.

For ease of use the commands are split into three types: **Data request commands**, **Remote management commands** and **Dynamic configuration commands**.

4.1 Data request commands

This section lists valid commands to send over a Command Interface, either a virtual COM port or serial port, using a program such as TeraTerm or PuTTY-nd for instance. You may also want to use commands in the **Remote management commands** and **Dynamic configuration commands** sections.

Note: When OSDP Secure mode is enabled on a particular Command Interface, `?b` is the only valid data request command.

These commands request data stored in the VTAP reader without changing it in any way.

Note: End any command sent with a <CR> or <LF>, but not both.

A `<setting>` refers to any of those listed in the section **config.txt settings**.

Commands below: **`%<setting>`** | **`?b`** | **`?events`** | **`?getserial`** | **`?h`** | **`?info`** | **`?r`** | **`?t`** | **`?temp`** | **`?type`** | **`?uid`**

%<setting>

Definition:
Request the current value of any setting that could be included in the `config.txt` file.

API level: 0

Options:
Multiple settings can be requested in a single line command, by separating each setting with the `|` character.
Using `%` on its own will return all current settings.

Example value: `%TagLED`

?b

Definition:
Reads VTAP boot information including core firmware version, API level and unique serial number.

API level: 0

Options:
Returns the BOOT.TXT data.
If OSDP is enabled on the port requesting this data (using **OSDP interface settings**) the data returned will also include '`OSDP:Enabled OSDP address: <OSDP_address_on_requesting_interface>`' at the end.
If an OSDP Secure Channel is being enforced on the port requesting this data it will return '`OSDP:Enabled,SecureOnly OSDP address: <OSDP_address_on_requesting_interface>`'

Example value: `?b`

NEW OPTIONS

?events

Definition:
Polls for any events (such as configuration change) that have happened in the VTAP reader since it was last polled for events.

API level: 7

Options:
A 32-bit hex number supplied with the command is a bit mask to select only certain events be reported. The default without a bit mask is to report all events (equivalent to using bit mask `ffffffff`).

Note: From API level 7, setting the top bit of the event mask parameter allows this command to temporarily configure which event notification bits will cause the WAKE pin (on VTAP25 only) to be asserted.

`00000001` is returned by the command if the configuration has been updated (bit 0 (0x01) set) since last poll.
`00000100` is returned by the command if a reboot has taken place (bit 2 (0x04) set) since last poll.
`00001000` is returned if a tap is available in the cache for reading with the `?r`, `?t` or `?type` commands (bit 3 (0x08) set).
`00010000` is returned if a VTAP100-PRO in Local mode notifies an input event (bit 4 (0x10) set) since last poll.
`00100000` is added when a VTAP reader wakes from sleep mode (bit 5 (0x20) set). (Sleep mode is not available on older VTAP100 readers, with v4a hardware or earlier.)

After reporting an event it is cleared.
If the bit mask used prevents the reporting of any event, it will remain pending for a subsequent poll.

Example value: `?events 00000001`

?getserial

Definition:
Retrieve VTAP reader serial number, which appears in `BOOT.TXT` as 'VTAP label'.

API level: 0

Options:
Will return the serial number (VTAP label), or `<NO SERIAL>` if a serial number has not been set.

Example value: `?getserial`

?h

Definition:
Retrieve public key hash information for the VTAP ECC keyslots, which can be used to work out which private keys have been loaded. The public key hash is the first 4 bytes of the 32 byte SHA-256 hash over the 32 byte ECC public key.

API level: 0

Options:
The block of public key hash data returned by the command is a 32 byte string.
The returned string starts with `A5`.
The public key data for the 6 key slots follows in order, 1 through 6.
Each key slot is described by a 1 byte flag (00=key slot empty, 01=key slot in use) then the 4 byte public key hash for that slot.
After all 6 keys the returned string ends with `5A`.

Example value: `?h`

?info

Definition:
Reads VTAP reader status information included in the BOOT.TXT file, including core firmware version, API level and unique serial number, equivalent to the `?b` command, but the data is returned as a JSON blob. (For VTAP PRO readers additional information is also included from the VTAP connectivity firmware, equivalent to the `?vcb` command.)

API level: 5

Options:
Returns the BOOT.TXT data and additional data from the VTAP connectivity firmware, if any.

The returned JSON string runs across multiple lines and terminates with OK. An example is shown below, from a VTAP100-PAC-485 reader:

```
{
  "Product": "VTAP100",
  "BootTime": "1970/01/01 00:00:00",
  "Firmware": "2.2.6.2",
  "Storage": "Dataflash",
  "Status": 0,
  "Hardware": "5.01",
  "Expansion": "VTAP100485LF-V1",
  "COMPortEnabled": true,
  "NCI": "0471125005-8C00",
  "HardwareID": "563230-0971BECA8676E452ADCDD8CA419635C8",
  "SerialNumber": "CC900000",
  "APILevel": 5,
  "AppKeysUsed": "12-4-6---",
  "KeySlotsUsed": "--3-56",
  "HWLocked": false,
  "SWLocked": false,
  "OSDP": {
           "Enabled": false,
           "SecureOnly": false,
           "Address": 0
    }
}
OK
```

Note: 'Serial number' from BOOT.TXT is labelled "HardwareID" in the JSON string, and 'VTAP label' from BOOT.TXT is labelled "SerialNumber". Other labels follow the BOOT.TXT names without spaces.

Example value: `?info`

?r

Definition:
Reads last NFC pass, card/tag data.
The tap data will be sent as long as it was read within the last `InvalidDataCacheMS` period and the request comes from an interface with a `...Source` setting that permits this type of data to be sent over that interface.
Up to three serial interfaces can request and receive this cached data for a single tap, at different times, up to the `InvalidDataCacheMS` period. The data will not be overwritten by a more recent tap until all potential interfaces have read the tap data or the `InvalidDataCacheMS period` is reached. [Note: On hardware VTAP100 v4 or earlier, there is only one cache, so there is a greater risk that the data could be overwritten by a more recent tap before it has been read over all interfaces].

API level: 0

Options:
This is only used in passive mode (selected with `?p`)

This command will only return pass/tag data where the `...Source` setting for the interface requesting the data permits it.

Example value: `?r`

?t

Definition:
Returns the type of the most recent cached NFC pass, card/tag read as a single character (as long as it was received inside the last `InvalidDataCacheMS` period) over the COM port.

API level: 0

Options:
Qualifiers instruct the VTAP to send an indicator of the type of NFC pass, card/tag data over a particular interface.

`?t COM` sends that NFC pass, card/tag data over COM port. (This is the default for `?t` when an interface is not specified.

Returned values are pass types (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**)

Example value: `?t COM`

?temp

Definition:
Returns the internal temperature of the VTAP.

API level: 0

Options:
Returns a Celsius value as reported by the processor.

Example value: `?temp`

?type

Definition:
Returns the type of the most recent cached NFC pass, card/tag read as a single character (as long as it was received inside the last `InvalidDataCacheMS` period) over the COM port or serial port, in passive mode.

API level: 0

Options:
Returned values are pass types (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**)

For Apple of Google passes the pass type is followed by VAS merchant ID/Smart Tap collector ID index and the key slot in use. In other cases -- follows the type.

Example value: `?type`

?uid

Definition:
Reads last card/tag UID. (Up to 16 characters in hex format).

API level: 4

Options:
This is only used in passive mode (selected with `?p`)

Example value: `?uid`

4.2 Dynamic configuration commands

This section lists valid dynamic commands to send over a Command Interface, either a virtual COM port or serial port, using a program such as TeraTerm or PuTTY-nd for instance. You may also want to use commands in the **Remote management commands** and **Data request commands** sections.

> **Note:** When OSDP Secure mode is enabled on a particular Command Interface, `?osdp` is the only valid dynamic configuration command.

Dynamic commands are used to manage the behaviour of a VTAP reader in passive mode, in real time. They support close integration of your VTAP reader with other systems. These commands are not saved in `config.txt` so they will only be valid until the next `?x` command, **reboot or refresh**, when the VTAP reader will return to following the settings in `config.txt`.

> **Note:** End any command sent with a <CR> or <LF>, but not both.

A `<setting>` refers to one of those listed in the section **config.txt settings**.

Commands below: **`>interface`** | **`>W`** | **`?a`** | **`?BEEP`** | **`?BEEPR`** | **`?card`** | **`?cardmode`** | **`?k`** | **`?KEYLOAD`** | **`?l`** | **`?LED`** | **`?LEDR`** | **`?NFC`** | **`?osdp`** | **`?p`** | **`?REBOOT`** | **`?STPasses`** |**`?tap`** | **`?u`** | **`?VASPasses`** | **`?x`**

`>interface`

Definition:
Send a message (for example as a virtual pass or tag payload) from any ComPort or serial interface to any of the other interfaces.
Note: Sending to a Wiegand interface `>W` is described separately.

API level: 3

Options:
The syntax used is `>interface:type:message`

`interface` is one of k, c, s, t, or b where k=Keyboard, c=ComPort, s=Serial, t=Serial2, b=Bluetooth keyboard. Valid choices depend on your hardware.
`type` is the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**).

The example here will send “hello”, as if it were an Apple pass payload, to the keyboard emulation interface.

Serial commands must be enabled over the corresponding interface. For example, if sending a message to ComPort, `ComPortSource=83` will enable pass read, card/tag reads and serial commands over the virtual COM port.

Example value: `>K:A:hello`

>W:T:YY:XXXXXXXX

Definition:
Send a message (for example as a virtual pass or tag payload) from any of the serial interfaces to a Wiegand interface [VTAP100-PAC-W only]. This is slightly different from the general case, as the message must take the form of a bit pattern and requires a bit length value.

API level: 0

Options:
`T` is type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**),
`YY` is the Wiegand bit length in hex and
`XXXXXXXX` is the Wiegand data in hex. (This is left justified, which means any padding bits to make a multiple of 8 will be in the LS bits).

The example here will send the 6 bit "B7A207", as an Apple pass payload, to the Wiegand interface.

Example value: `>W:A:06:B7A207`

?a

Definition:
Change COM port to active mode. Mobile pass, card/tag data will then be sent over the COM port automatically, whenever it is received.

API level: 0

Options:
-

Example value: `?a`

DEPRECATED

?BEEP

Definition:
The buzzer can be driven by this command over any serial command interface. *Use of `?BEEPR` now preferred.*

API level: 1

Options:
The syntax is:
`<on ms>,<off ms>,<number of repeats>[,<frequency>]`[:<section_name>@<text_file> for beep sequence]

In the first example, the buzzer will beep twice for 200ms on and 200ms off. (Frequency defaults to 3136Hz if not set).

The second example follows a beep sequence defined in the section `[tagbeep]` of the `config.txt` file on the VTAP reader.

Note: If the sequence definition section is not found, the reader will default to the defined monotone beep.

Example value:

```
?BEEP 200,200,2
?BEEP 200,200,2:tagbeep@config.txt
```

?BEEPR

Definition:
The buzzer can be driven by this command over any serial command interface.

API level: 1

Options:
The syntax is:
`<on ms>,<off ms>,<number of repeats>[,<frequency>]`[:<section_name>@<text_file> for beep sequence]

In the first example, the buzzer will beep twice for 200ms on and 200ms off. (Frequency defaults to 3136Hz if not set).

The second example follows a beep sequence defined in the section `[tagbeep]` of the `config.txt` file on the VTAP reader.
Note: If the sequence definition section is not found, the reader will default to the defined monotone beep.

Response provided OK<newline> or FAIL<newline>.

Example value:

```
?BEEPR 200,200,2
?BEEPR 200,200,2:tagbeep@config.txt
```

?card <type><:lang>,<NDEF>

API level: 0

Definition:
Set the `CardEmulationData` value dynamically over a serial interface. This defines the NDEF data to be sent by a VTAP reader that is in card emulation mode.

Options:
<NDEF> data is assumed to be text if a <type> is not included
TEXT (T) type is assumed to be english (en) unless another 2 character <lang>uage code is used
URI (U) requires a valid web address
RAW (R) is a raw binary message part for an NDEF record. Its length will be automatically set.
FILE (F) points to a text file on the VTAP containing one of the other command types.

For more detail refer to VTAP Application Notes.

Example value:

```
?card TEXT,Hello World!
?card URI,http://www.vtapnfc.com
?card RAW,D1010554026 56e4869
?card FILE,tagdata.txt
```

?cardmode <#>

API level: 0

Definition:
Set the `CardEmulationMode` value dynamically over a serial interface. This enables or ends a mode, where the VTAP emulates an NFC Forum Type 4 card or tag, containing NDEF encoded data.

Options:
=0 to leave card emulation mode,
=1 to enter card emulation mode,
=2 to enter mixed mode, where the VTAP reads tags and passes but will also emulate a card.
For more detail refer to VTAP Application Notes.

After this command the VTAP returns the number of its current `CardEmulationMode`.
Omitting a mode number <#> will also return the current `CardEmulationMode`.

Example value: `?cardmode 1`

DEPRECATED

?k

Definition:
Consumes any key files (such as `privateX.pem` or `appkey#.txt`) into secure storage, from the file system, without requiring a restart. Then remount the USB mass storage device after loading keys, forcing the USB host, such as Windows, to update its cache of the filesystem directory. *Use of `?KEYLOAD` now preferred.*

API level: 0

Options:
-

Example value: `?k`

?KEYLOAD

Definition:
Consumes any key files (such as `privateX.pem` or `appkey#.txt`) into secure storage, from the file system, without requiring a restart. Then remount the USB mass storage device after loading keys, forcing the USB host, such as Windows, to update its cache of the filesystem directory.

API level: 0

Options:
Response provided OK<newline> or FAIL<newline>.

Example value: `?KEYLOAD`

?l

Definition:
To lock the VTAP configuration using a password over any serial command interface.

API level: 0

Options:
The example will lock the VTAP configuration using the password APa55word.

Example value: `?l APa55word`

DEPRECATED

?LED

Definition:
The LED [Not VTAP50 v1] can be driven with this command over any serial command interface. Sequences of RGB LED flashes can be triggered, specifying the hexadecimal RGB colour. [VTAP50 v2 only: A serial LED pattern can be chosen.] *Use of `?LEDR` now preferred.*

API level: 0

Options:
LED colour (use any hex RGB colour value), LED on ms, LED off ms, number of repeats.
In the example, the LED will flash red 5 times for 100ms on and 100ms off.

[VTAP50 v2 only: follow the second example to instruct the LEDs to follow the `seq.test` portion of the `leds.ini` file in driving the chain of serial LEDs].

Example value:

```
?LED FF0000,100,100,5
?LED FFFFFF:seq.test@leds.ini
```

?LEDR

Definition:
The LED [Not VTAP50 v1] can be driven with this command over any serial command interface. Sequences of RGB LED flashes can be triggered, specifying the hexadecimal RGB colour. [VTAP50 v2 only: A serial LED pattern can be chosen.]

API level: 0

Options:
LED colour (use any hex RGB colour value), LED on ms, LED off ms, number of repeats.
In the first example, the LED will flash red 5 times for 100ms on and 100ms off.

[VTAP50 v2 only: follow the second example to instruct the LEDS to follow the `seq.test` portion of the `leds.ini` file in driving the chain of serial LEDs, or the third example to follow the contents of a `leds.ini` file by default.]

Response provided OK<newline> or FAIL<newline>.

Example value:

```
?LEDR FF0000,100,100,5
?LED FF0000:seq.test@leds.ini
```

?NFC

Definition:
To enable/disable the NFC field, to reduce power consumption. If no value is supplied with the command, it will read the current state of the NFC field, returning ON or OFF.

API level: 0

Options:
The example will enable the NFC field

Example value: `?NFC 1`

?osdp

Definition:
Enable/disable OSDP on a given port.

API level: 0

Options:
Using the syntax `?osdp [enable=<Serial2OSDP>] [keyslot=<Serial2OSDPKeySlot>] [address=<Serial2OSDPAddress>] [format=<Serial2OSDPFormat>] [secureonly=<Serial2OSDPSecureOnly>]` you can change any or all of these **OSDP interface settings** dynamically to change OSDP behaviour on the current port, until a reboot.

The command will always return the resulting OSDP state on the port, which could be `Enabled`, `Enabled`, `SecureOnly` or `Disabled`. So sending `?osdp` without any parameters just queries whether OSDP is enabled on this port.

Example value:
```
?osdp
?osdp enable=1 keyslot=1 address=0 format=wiegand secureonly=1
```

?p

Definition:
Change the COM port to passive mode. The COM port will then only send data in response to specific commands.

API level: 0

Options:
-

Example value: `?p`

?REBOOT

Definition:
To reboot the VTAP over any serial command interface.

API level: 0

Options:
Response provided OK<newline> or FAIL<newline>.

Example value: `?REBOOT`

?STPasses

Definition:
To dynamically change the Google Smart Tap pass enabled over any serial command interface. Avoids the need for frequent edits to `config.txt` when pass types often need to be changed.

API level: 0

Options:
The example will enable only Google SmartTap pass type ST4, if it is present in `config.txt`

Example value: `?STPasses 4`

?tap

Definition:
Allows you to emulate a pass or card tap by supplying the pass or card payload data. The `...Source` settings determine which interfaces should receive the data. Active interfaces receive the data directly. For passive interfaces the data is inserted into the tap cache to be read by any interfaces using the `?r` command. One use of this command is to facilitate testing.

API level: 2

Options:
The syntax required is:

```
?tap <T><M><K>,<passdata>
```

`<T>` is the pass type (A, G, 0, 2, 4, or 6, E, Q , X or W, see **Pass Type explainer**)
`<M>` is an optional merchant index character 0 to 9, defaults to 0 if omitted
`<K>` is the optional key slot character 1 to 9, defaults to 0 if omitted
`<passdata>` is the data to be recorded as if it were a pass read

The response will be either OK, or FAIL if no data is provided.

Any prefix/postfix strings configured are applied to keyboard, ComPort and serial outputs when in active mode. Pass data sent to Wiegand will have `PassFormat=h`, `d` or `a` and all other `PassWiegand...` options applied. Card/tag data sent to Wiegand will have `TagWiegandASCIIFormat` and `TagWiegandBits` applied.

Example value:

```
?tap A,1234546A
?tap G12,PASS12345678
?tap A1,NOKEYSPECIFIED
```

?u

Definition:
To unlock the VTAP configuration using a password over any serial command interface.

API level: 0

Options:
The example will unlock the VTAP configuration, using the password APa55word

Example value: `?u APa55word`

?VASPasses

Definition:
To dynamically change the Apple VAS passes enabled over any serial command interface. Avoids the need for frequent edits to `config.txt` when pass types often need to be changed.

API level: 0

Options:
The example will enable only Apple VAS passes VAS3, VAS4 and VAS5, if they are present in `config.txt`

Example value: `?VASPasses 3,4,5`

DEPRECATED

?x

Definition:
To reboot the VTAP over any serial command interface. *Use of ?REBOOT now preferred.*

API level: 0

Options:

Example value: `?x`

4.3 Remote management commands

This section lists the valid remote management commands to send over a Command Interface, either a virtual COM port or serial port, using a program such as TeraTerm or PuTTY-nd for instance. You may also want to use commands in the **Data request commands** and **Dynamic configuration commands** sections.

This section includes commands to directly change the `config.txt` file or to start a file transfer files from the VTAP file system via ZModem.

Note: End any command sent with a <CR> or <LF>, but not both.

A `<setting>` refers to one of those listed in the section **config.txt settings**.

The `@...` commands are similar to VTAP `?...` commands, but allow the VTAP interfaces to control functionality provided by the VTAP PRO expansion board [only a VTAP100-PRO-BW in Local mode], for example Bluetooth related functions.

Settings below: **`$setting`** | **`!filename`** | **`?reformat`**| **`?sleep`** | **`@BLEBeacon`** | **`@BTHostScan`** | **`@BTHostStopScan`** | **`@BTHostUnpair`**

$<setting>=<value>

Definition:
Set any setting that could be included in the `config.txt` file. (Note: The mass storage device will be temporarily dismounted after this action, to refresh operation of the VTAP.)

API level: 0

Options:
Multiple settings (up to 10) can be made in a single line command, by separating each setting with the <tab> character.
It is not recommended that you use this setting too frequently as it can impact flash memory life. If you require frequent changes in behaviour, use **dynamic commands**.

Example value: `$TagLED=FF00FF,300`

!<filename>

Definition:
This instructs the VTAP to send a copy of its file. Having sent this command to the VTAP you need to run ZModem receive in the client to collect this file. It will continue being sent until a ZModem acknowledgement is received.

API level: 0

Options:
In the example a copy of the file `boot.txt` will be sent over ZModem.

Example value: `!boot.txt`

`?reformat`

Definition:
To reformat the VTAP file system over any serial command interface, and then remount the USB and read in the configuration.

API level: 0

Intended for use in event of file system corruption. There is the option to backup files then restore them, after reformatting.

Always test or load your `config.txt` after reformatting, then reboot.

Options:
`?reformat preserve` will copy any `config.txt` and `serial.txt` files to RAM before reformatting the VTAP reader file system, then write those saved files back to the VTAP reader.

`?reformat empty` will simply reformat the VTAP reader file system and leave you to add the necessary files afterwards.

Example value: `?reformat preserve`

NEW

?sleep

Definition:
Instruct the VTAP reader to immediately enter a low power sleep mode.

API level: 7

Sleep mode primarily runs the VTAP processor at very low power, shutting down all activity except monitoring for NFC and serial comms activity. Entering sleep mode also switches off all LED activity, which will resume when sleep mode ends.

(This is not available on older VTAP100 readers, with v4a hardware or earlier.)

Options:
Response provided OK<newline> or FAIL<newline>.

A VTAP reader will wake from sleep mode when there is an NFC event (tapping a card or phone) or receiver activity on any serial port. The serial character which wakes up the VTAP reader will be not be received, it will only trigger wake up. An event bit will be set to record the wake up event. Then after a few milliseconds delay the VTAP reader will be ready for full operation.

CAUTION: As sleep mode disables USB completely, USB COM port activity cannot be used as a trigger to wake up the VTAP reader, although the reader should still wake when you tap a card or pass. Sleep mode is therefore not recommended for applications using only VTAP USB device interfaces.

For a VTAP25 reader module, we recommend you send `W\n` over serial1 or serial2 as a wake up instruction, then wait a few milliseconds for the reader to wake up, before sending any required commands. The WAKE pin (output) will be set high when the VTAP processor wakes up, or when there is an NFC payload to read. You can use this to wake your host processor, or as an indication that there is something to read, to avoid the need to continuously poll the VTAP25. The WAKE pin is reset when an `?r` or `?events` command is processed.

Example value: `?sleep`

@BLEBeacon

Definition:
API level: 0

Enables the BLE advertisement beacon on VTAP100-PRO-BW for use with any BLE beacon detecting application. BLE beacons can be used to provide location notifications, similar to GPS, but which will work where GPS will not, such as for indoor locations. One application is to trigger location notifications associated with Apple Wallet passes. When issuing a pass, a pass provider may set a location value that could comprise GPS coordinates or a BLE beacon UUID, with optional major and minor values.

Options:
The syntax used is

```
BLEBeacon i,<name_for_logging>,<uuid>,<major>,<minor>
```

`<name_for_logging>` is an optional setting, up to 15 characters long, to allocate a name to this BLE beacon that may be reported. An example use would be a VTAP PRO reader in Cloud mode returning this name, as part of the tap information sent to a customer application, to indicate which beacon was being advertised at the time of the tap.
`<uuid>` is the UUID (unique identifier) of the beacon to advertise. You can use any value. If intended for use with Apple Wallet passes, the UUID must match that entered in the location field of the pass.
`<major>` and `<minor>` are decimal values (0 to 65535) that must be specified as part of the BLE beacon advertisement. A beacon detecting application can be configured to trigger when a specific UUID, and optionally specific major and/or minor numbers match those being advertised. For example, an Apple Wallet pass might use wildcard values to match any major or minor values with a specific UUID, or it might be set to only trigger when a specific UUID and major number is detected. This allows, for instance, detection of all branches of a franchise, or only those in specific regions (based on a matching major), or individual locations (based on matching major and minor).

Using just @BLEBeacon on its own , or giving invalid beacon parameters will disable the beacon.

Example value:

```
@BLEBeacon i,loyaltybeacon,4CE2EF69-4414-469D-9D55-3EC7FCC31234,1,1
@BLEBeacon ; to disable
```

@BTHostScan

Definition:
Starts a Bluetooth scan. The VTAP PRO reader will begin scanning to find a barcode/QR scanner it can pair with. When paired scanning for new devices stops.

API level: 0

Options:
-

Example value: `@BTHostScan`

@BTHostStopScan

Definition:
Stops any ongoing Bluetooth scan.

API level: 0

Options:
-

Example value: `@BTHostStopScan`

@BTHostUnpair

Definition:
Unpairs all previously paired Bluetooth scanner devices, or a specific Bluetooth scanner device, associated with the VTAP PRO reader.

API level: 0

Options:
The syntax used is `@BTHostUnpair [<device_name_or_address>]`.

Without the optional `<device_name_or_address>` all Bluetooth scanner devices will be unpaired.

`<device_name_or_address>` can optionally refer to one of the currently paired scanner devices. (The number of paired scanner devices and the name of the first one can be found in `VCBOOT.TXT`, updated on boot. A device address is 6 bytes (12 hex characters).

Example value:

```
@BTHostUnpair
@BTHostUnpair C barcode scanner
```

4.4 Commands for VTAP PRO in Local mode

This section lists additional dynamic commands to send over a Command Interface to the VTAP100-PRO in Local mode, either using a virtual COM port or serial port, or using a program such as TeraTerm or PuTTY-nd for instance.

You may also want to use commands in the **Dynamic configuration commands**, **Remote management commands** and **Data request commands** sections.

The commands included here can be used when the VTAP100-PRO is in Local mode.

Note: The optional VTAP100 PRO I/O expansion board (VTAP100-PRO-EXP1) is required to use `?input` and `?relay` commands.

Commands below: **`?input`** | **`?relay`**| **`?vcb`**

`?input`

Definition:
Reads the current state of the input to the VTAP100-PRO in Local mode as either open or closed. (There is only one input.)

API level: 0

Options:
Syntax is `?input 0 [notify=open|1|close|0|any|none]`

`0` is the input number for future expansion (may be omitted)

The response to the command will be the current state of the input as either `1` (open) or `0` (closed) or `FAIL` (if the input number is not valid or the hardware does not have inputs).

The optional notify parameter controls whether input changes should be recorded, for return as bit 16 in response to an `?events` command.
`notify=open` or `=1` an event is recorded on changing the input from 0 to 1 (closed to open);
`notify=close` or `=0` an event is recorded on changing the input from 1 to 0 (open to closed);
`notify=any` an event is recorded for any change of input;
`notify=none` an event is never recorded as a result of an input change.

Default value: `N/A`

Example value:

```
?input
?input 0
?input 0 notify=open
```

?relay

Definition:
Controls either of the two relays on the VTAP100-PRO in Local mode

API level: 0

Options:
Syntax required is:

```
?relay <relay_num> <action>[=param1[,param2][…]] [action[=params]]
```

`<relay_num>` is currently either `0` or `1` to refer to the two available relays

`<action>` can be one or more in a series of actions, as required, each separated by a space.
Actions can be one of:

```
on
off
timed=ontime[,offtime][,count]
delay=<time in ms>
```

If the off time is not specified, the on time is used. If the count is not specified, 1 is used. Delay is in milliseconds but can specify time in seconds or minutes with 's' or 'm'.

Default value: `N/A`

Example value:

```
?relay 0 on
;Switch ON relay 0

?relay 1 delay=60000 on
;Switch ON relay 1 after 60s (60000ms)

?relay 0 timed=100,200,2
;Switch ON relay 0 twice for 100ms, with OFF interval 200ms

?relay 0 on delay=100 off delay=200 on delay=100 off
;Same result as previous example

?relay 0 on delay=5m timed=10s,20s,3
;Switch ON relay 0 after 5 mins, repeat 3 times 10s ON, 20s OFF
```

?vcb

Definition:
Reads boot information from the additional processor module responsible for wireless, Bluetooth or Ethernet comms, including paired device names and the hardware identifier for the VTAP PRO reader board assembly.

API level: 0

Options:
Returns VCBOOT.TXT data, for instance:

```
HW: VTAP100C V1 a4
FW: 1.0.3.3
RAM: 2097152
BT: 0
```

In this data FW identifies the VTAP connectivity firmware version.

Note: VTAP100E is connected over Ethernet, all other models use Wi-Fi.

Default value:

Example value: `?vcb`

A Index of Settings and Commands