Custom Format Decoder

Plug-in User Guide

Contents

Overview

Read-a-Card plug-ins give you the ability to extend Read-a-Card's functionality to meet your own custom needs. By default, Read-a-Card will read the CSN/UID of a contactless card and format it in one of 6 fixed methods. But if instead you wish to format the CSN/UID differently, for example only read part of the UID, then Read-a-Card can return that data, formatted according to your requirements, using a software plug-in.

The Custom Format Decoder plug-in supports reading and decoding UIDs from all cards supported by Read-a-Card. It can also be used to decode the format of physical access credentials (PAC) read from HID Prox or iClass cards (assuming a suitable HID/Omnikey reader is used).

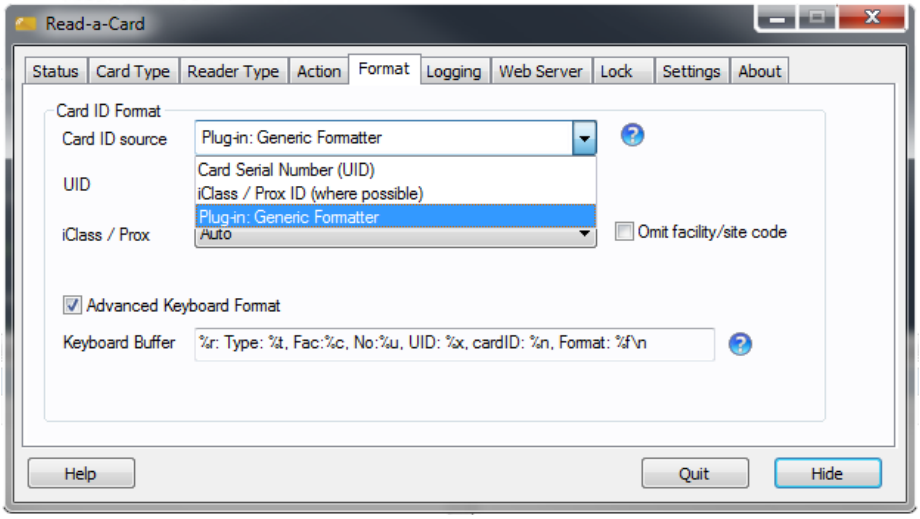
This document will guide you through the configuration and use of the Read-a-Card Custom Format Decoder plug-in.

Read-a-Card and plug-in DLLs

Read-a-Card supports plug-in extensions in the form of DLL files. These DLL files must reside in the “Plugins” folder within the Read-a-Card application folder. (e.g. C:\Program Files\Read-a-Card\Plugins).

The Custom Format Sector Decoder plug-in is included with recent versions of the Read-a-Card (v3.1.0 onwards). The plug-in DLL (CustomFormatDecoder.dll) is copied to the Plugins folder when Read-a-Card is installed.

Read-a-Card will display all available plug-ins in the Card ID source drop-down box on the Format tab. All plug-ins will have the “Plug-in:” prefix. Only one plug-in can be in use at any time.



Some plug-ins may require that Read-a-Card is licensed for the plug-ins to function; others may require specific configuration data. If a plug-in is unable to work for either of these reasons, it will appear in the drop-down list “greyed out”.

Clicking on the help icon (?) next to the format drop-down box will pop-up an information window showing the currently selected plug-in format name, version, format name and other features. This window also contains a link to edit the configuration file for the plug-in.

Plug-in Configuration

Read-a-Card format plug-ins are configured using a Windows configuration file.

In the case of the Custom Format Decoder plug-in, this file is named “CustomFormatDecoder.ini”. This file (which will be created automatically if necessary when the plug-in is first selected) should be located in the same folder as the Read-a-Card configuration file

On Windows XP/2000:

C:\Documents and Settings\All Users\Application Data\Read-a-Card\

On Windows Vista/Windows 7/Windows 8 :

C:\ProgramData\Read-a-Card\

In order for this plug-in to work, this file must at least contain the following:

```
[Config]
PluginVersion=1.0
PluginName=Custom Format Decoder Plug-in
```

PluginVersion specifies the minimum version of the plug-in that this configuration expects. If this value is greater than the actual plug-in version, an error message will be displayed when the plug-in is selected in Read-a-Card.

Plug-in name

The Read-a-Card format name displayed in the Card ID source drop-down box can optionally be configured in the ini file as follows:

```
[Config]
FormatUIName=My Name for this plug-in
```


ID Format Configuration

In addtion to the [Config] section, the following sections are recognised in the plug-in configuration file:

[Prox]

[iClass]

[UID]

The Prox and iClass sections allow specific configuration settings to be specified for each card type. The UID section specifies how the UID should be formated for all other card types. If either the Prox or iClass section is missing and a Prox or iClass card is presented then the settings in the [UID] section will be used to format the Prox or iClass PAC credential data.

Note that only HID Omnikey iClass readers are capable of reading the PAC credential from an iClass card application area. Other readers may be able to read the iClass Card Serial Number (CSN), effectively the UID of an iClass card.

Format Options

This plug-in offers two alternative methods, named DigitMask and BitMask, of extracting and formatting the raw data read from the card. These options are applicable to both UIDs (from any card type) and PAC (either Prox or iClass) credentials.

DigitMask is used in conjunction with the **OutputFormat** setting. It may be used to extract and arrange whole digits from of the card number after it has been interpreted according to the selected OutputFormat.

BitMask operates on individual bits within the raw data of the UID or PAC data. It can be used to discard start, stop and parity bits and to select which bits should be assigned to the facility code and which bits assigned to the card number. BitMask may be used in conjunction with the **BitOrder** and **ByteOrder** options to specify how the raw data should be interpreted. When using the BitMask option, the resulting card number and optional facility code are always returned as decimal values.

Note that the BitMask and DigitMask options cannot be used together for a given card type (if both are specified then only the DigitMask option will be used).

FACDigits and **CardNumDigits** are options which may be used with either the DigitMask or BitMask options to produce a consistent number of digits for the facility code or card number by adding as many leading zeros to each number as are required.

DigitMask

A DigitMask value can be used to indicate how the digits of the result of the OutputFormat process should be distributed between the facility code and card ID. For example:

DigitMask=**xxfff-cccccx**

The DigitMask is applied to the result of formatting the data according to the OutputFormat value. Only f (facility code) and c (card number) characters in the digit mask have any meaning. Any other characters indicate that the corresponding digit is to be ignored.

So, for example, with the above DigitMask setting specified, if the result of reading and formatting the data is the value “123456789012”, the resulting facility code will be 345 and the resulting card number will be 78901.

The following options allow the facility code and/or card number to be formatted with leading zeros:

FACDigits=4
CardNumDigits=7

Using the above example, these options would yield a facility code of 0345 and a card number of 0078901. The resulting ID (combined facility code and card number) would be 03450078901.

BitMask

More complex bitwise facility code and card number decoding can be performed with the BitMask option. Note that the BitMask and DigitMask options cannot be used together for a given card type (if both are specified then only the DigitMask option will be used).

BitMask=-----pfffffffffffffcccccccccccccccccccp--------
ByteOrder=0
BitOrder=0

The BitMask value is applied bitwise to the binary data read from the card, only characters c and f are used to indicate bits to be assigned to the card number or facility code respectively. All other characters indicate a bit to be ignored.

The **OutputFormat** option is not used in this case. Instead the bytes read from the card are first converted into a 64 bit integer value. The ByteOrder option determines whether the first or last byte should be considered the most significant (0=first byte is most significant, 1=first byte is least significant).

The **BitOrder** value may then used to reverse the order of the bits in the resulting facility code and card number.

BitOrder=0 implies that the leftmost character of the BitMask is matched to the most significant bit of the data.

BitOrder=1 implies that the rightmost character of the BitMask is applied to the least significant bit of the data and that the bit order is reversed (i.e. the least significant bit is treated as the most significant in the resulting facility code and card number)

So for example:

The following six bytes of data are read from the card’s memory:
C6 8B 06 92 48 14

ByteOrder=1 will reverse the byte order resulting in a 64 bit hex value of

144892068BC6

In binary this is:
0001 0100 0100 1000 1001 0010 0000 0110 1000 1011 1100 0110

Matching this to a BitMask and BitOrder as follows:

BitMask=------pfffffffffffffcccccccccccccccccccp
BitOrder=0

We get:

000101000100100010010010000001101000101111000110
-----pfffffffffffffccccccccccccccccccp

Facility code =00010010001001b = 489h = 1161 (decimal)
Card number = 0010000001101000101b = 10345h = 66373 (decimal)

The following options allow the facility code and/or card number to be formatted with leading zeros:

FACDigits=4
CardNumDigits=6

So using the above values the resulting facility code would be 0489, the card number would be 066373 and the overall combined ID would be 0489066373.

As another example, consider the following BitMask with BitOrder=1 used on the same data:

BitMask=pccccccccccccccccccfffffffffffffcp
BitOrder=1
ByteOrder=1

In this case we would get the following match:

```
000101000100100010010010000001101000101111000110
              pccccccccccccccccccfffffffffffffp
```

Note that the BitMask is right-aligned with the data due to BitOrder=1. This option also reverses the bit order giving:

Facility code =11000111101000b = 18e8h = 6376 (decimal)
Card number = 1011000000100100100b = 58124h = 360740 (decimal)

In this case the FACDigits=4 and CardNumDigits=6 options will not cause any additional leading zeros to be added because the values are already this number of digits.

The resulting combined ID will be 6376360740.

OutputFormat

This option determines how the resulting bytes will be formatted. It can have the following values:

1. **Decimal**
   This option takes the first 4 bytes of data (or all bytes if fewer than 4 have been read) and interprets them as a big-endian binary 32 bit number (i.e. the first byte in the sequence has the most significant value). The resulting 32 bit value is formatted as a 10 digit decimal number.
2. **Hex reversed**
   This option outputs all bytes of the data in hexadecimal (two ASCII numeric or uppercase hex characters per byte) in reverse order.
3. **Hex standard**
   This option outputs all bytes of the data in hexadecimal (two ASCII numeric or uppercase hex characters per byte) in the same order that they were read.
4. **Decimal 64 bit**
   This option takes the first 8 bytes of data (or all bytes if fewer than 8 have been read) and interprets them as a little-endian binary 64 bit number (i.e. the last byte in the sequence has the most significant value). The resulting 64 bit value is formatted as a decimal number with no leading zeros.
5. **Decimal 64 bit reversed**
   This option takes the last 8 bytes of data (or all bytes if fewer than 8 have been read) and interprets them as a big-endian binary 64 bit number (i.e. the first byte in the sequence has the most significant value). The resulting 64 bit value is formatted as a decimal number with no leading zeros.
6. **ASCII**
   This option outputs all bytes of data as ASCII printable characters. Any byte values that do not represent ASCII printable characters will be output as a '.' character.

ActionMask

You can control which Read-a-Card actions are taken (depending on the type of card) using the **ActionMask** setting. This allows the plug-in settings for a card type to suppress actions settings that would otherwise be enable by Read-a-Card's configuration settings.

ActionMask=00 (Hex pair e.g. "7E" for Action Mask 0111 1110)

The following is the bitwise representation of the Action Mask:

Action Mask:
X X X X X X X X
7 6 5 4 3 2 1 0
Bit 7 = Not Used
Bit 6 = Log Action 0 = Enable, 1 = Disable
Bit 5 = URL/HTTP Action 0 = Enable, 1 = Disable
Bit 4 = Command Action 0 = Enable, 1 = Disable
Bit 3 = Sound Action 0 = Enable, 1 = Disable
Bit 2 = Clipboard Action 0 = Enable, 1 = Disable
Bit 1 = Keyboard Action 0 = Enable, 1 = Disable
Bit 0 = Not used

