CRANE® PAYMENT SOLUTIONS

Remote Coin Programming

Implementation for NRI Coin Validators

Authors:	Georgiy Kalchev
Department:	Crane Payment Solutions / NRI R&D
Version:	0.1
Date:	2014.02.19

Contents

Document History

Date	*Version*	*Changes*
19.02.2014	0.1	First official version

1 Introduction

Remote Coin Programming, from now on referred to as RCP, is a standard feature of Money Controls devices, that allows coin data to be programmed or updated in field. The notion “remote” has two meanings here:

Firstly, coins can be programmed via the ccTalk communication interface without any manual coin insertion. The host machine just transfers the selected coin data file into a validator, which does the whole calculation job. One of the main advantages of this approach is the relative simplicity of the host machine software, as it does not need to concern about technical peculiarities of each and every device connected to it. All this is handled inside of RCP-capable devices.

Secondly, the coin data to be programmed can be downloaded from some remote central server (HTTP, FTP etc) and then be locally distributed among host machines. This path can be very flexible. In any case, the last member of the coin data distribution link is the ccTalk interface to each individual coin validator.

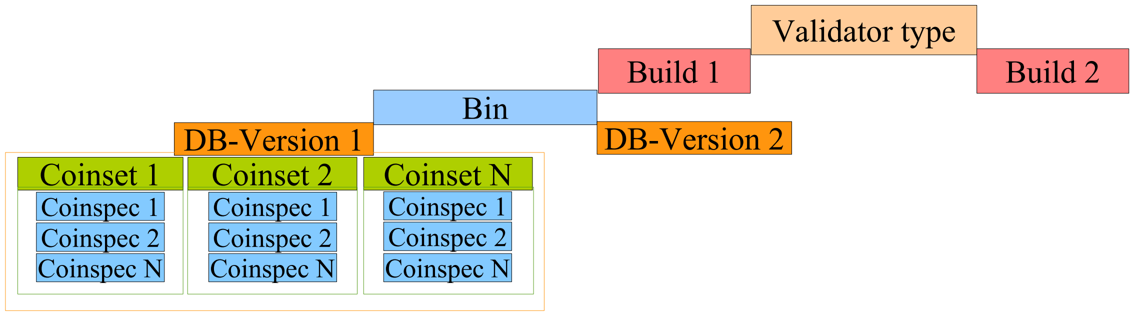
NRI has adopted this feature in some of its coin validators, **v2eagle** and **v2colibri**. The aim was to achieve maximum compatibility with the existing Money Controls products in order to make the transition for the customers as seamless as possible. This means, that the NRI RCP validators should be fully compatible with the existing hardware/software infrastructure provided by Money Controls for their products and running on customer sites.

Money Controls SR5i validator was chosen as a reference for the NRI implementation. The original SR5i RCP implementation is described in *[TSP121 SR5i ccTalk Protocol Manual].* This document describes the concepts and technical details of the NRI RCP functionality.

Please read this document thoroughly, as its every chapter presents vital information for application programmers who wish to employ RCP.

2 Data structure and definitions

The RCP data structure is tree-like and is backed by ordinary file system objects, like folders and files. In general, *coinspecs* (***coin spec****ifications*) are files, the rest is folders. The whole tree can be, for example, archived for later distribution.



The elements of the tree are described below, in the bottom-to-top direction.

2.1 Coinspecs

RCP allows single coins to be uploaded and programmed into a validator. A single coin with a certain security level (i.e. acceptance window bandwidth) has one so-called *coinspec* (**coin spec**ification) file, recognized by the file extension **.bin**.

Example: For a 5-Eurocent coin with a standard, narrow and super narrow bandwidth there are three coinspecs for Money Controls customers: “EU005A**-0**.bin”, “EU005A**-2**.bin” and “EU005A**-7**.bin”. The suffix **-N** indicates the security level – more on coinspec file naming conventions in chapter 4.1.

A coinspec file contains all the data needed for coin identification and channel window calculation like algorithmic parameters, addends and formulas. The channel windows calculation is done by a validator's firmware.

2.2 Coinsets

Although single coins (coinspecs) can be programmed independently and randomly (purely from a technical standpoint), this is seldom done in this way. Instead, coinspecs are grouped into *coinsets*, i.e. a collection of somehow related coinspecs. Usually, one coinset corresponds to some country or currency and is named appropriately: “EU” means Euro, “GB” – Great Britain, “US” – USA etc. This grouping guarantees, that there is no “cross-credit” effect (i.e. overlapping channel windows) among coinspecs within a coinset. Programming random coinspec combinations cannot rule out cross-credit situations.

Normally, the validator manufacturer manages coinsets and guarantees intra-coinset coinspec compatibility. Managing coinsets also includes versioning and delivery to customers. But, of course, customers can create and manage their own coinsets, suited for their needs. In such cases though, customers should be responsible for coinspec compatibility.

Conceptionally, coinsets should be viewed as containers for some logically related and mutually compatible coinspecs. Physically, coinsets are directories filled with their coinspec files.

To provide some accounting, consistency and coinspec management control, there must be some kind of versioning. The so-called *coinspec issue numbers* serve this purpose. All coinspecs within their coinset should have the same issue number. If some coinspec gets modified and, as a result, its issue number increases, that same issue number must be then unconditionally assigned to the rest of the coinset's coinspecs. Issue numbers are managed locally per coinset and are coinset-specific, i.e. are not globally unique.

The issue number is also part of a coinspec data, and can be directly seen in its DOS stub (see *chapter 4.3)*.

2.3 Database (DB) version

Since there can be several variants of a validator type, each differently equipped (sensor types etc), coinspecs should be separately generated and tuned for each variant. To identify validator variants there are *database versions,* one per variant. For instance, a v2eagle with a hardness detection sensor has a DB version 2, with a rim sensor – 3, without the both – 1. A v2colibri with a hardness detection has a DB version 2.

Physically, DB versions are also folders, named according to the scheme “Db-NNN”, where NNN is the version number, padded with zeroes from the left (Db-001). These folders are populated with coinset folders.

Similar to an issue number, a DB version is a part of a coinspec data payload. If a validator determines that its own DB version does not match the one of a coinspec, it cancels the processing, ignores the upload and returns the error code 236 (see below).

2.4 Other elements

The higher-level structure elements (see picture above) are defined as follows:

Bin

This folder contains the actual coinset sub-folders with coinspec files. Coinspec files have the extension “**.bin**”.

The presence of such folder implies, that there can potentially be other types or classes of data files.

Build

Specific build ID of a given validator type. There may be default, or standard, builds or some customer-specific builds. Currently, the default build 0 “DE0” is supported.
The build string is a part of the DOS stub in a coinspec file and is directly visible.

Validator type

This highest-level folder indicates the type of validator, “EAGLE” for v2eagle and “COLIBRI” for v2colibri. Validator type string is also available within the DOS stub of every belonging coinspec file.

3 Implementation

The following sections describe the implemented ccTalk commands for RCP and all relevant error codes.

RCP features still missing or not supported:

1. ".nnn" files for quick updates are not supported (not needed for NRI validators).
2. Extra files and macro files are not (yet) supported.

3.1 Implemented commands

ccTalk Header 96, sub-headers are listed in the table below:

Table 3.1: Implemented commands

Sub-header	***Command***
255	Begin packet upload
254	Upload packet data
253	End packet upload & program
249	Remove coin signature
248	Request extended coin id
245	Remove all coin signatures
244	Request currency code
243	Request coin issue

NOTE: All RCP commands return NO REPLY on 2 fundamental failures: RCP feature is not available (not built in at all or currently deactivated) in a validator, or a command is either not supported (like **Sub-header 247 "Calculate ROM secure signature"**) or invalid. In all other cases either ACK or some payload data will be returned.

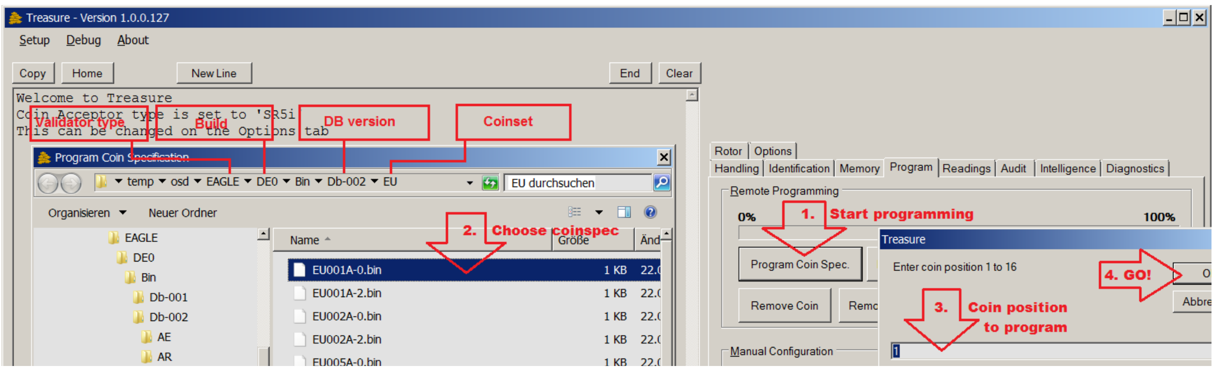
Sub-header	**255, Begin packet upload**
Parameters	none
Reply	ACK on success or none on failure
Action	The device is prepared for RCP. If the preparation fails, the internal error code is saved, which will prevent any further RCP processing. This should be the very first command in the programming sequence.

Sub-header	**254, Upload packet data**
Parameters	Portion of a coinspec file contents, limited by the ccTalk protocol physical constraints (255 bytes max minus 1-byte sub-header).
Reply	ACK on success or none on failure
Action	Sequentially transfer the selected coinspec file, portion by portion, into the device by executing this command multiple times. The coinspec contents may not be modified in any way. If the internal device error has been set after the “*Begin packed upload*”, the data will be ignored. If any error occurs during the upload, the internal error will be set, and the further data will be ignored.

Sub-header	**253, End packet upload & program**
Parameters	One or more 1-byte coin positions, ranging 1...16, to be programmed with the transferred coinspec, typically one.
Reply	ACK on RCP success; 1-byte code on RCP error; none on failure
Action	This is the final command in the RCP sequence. It specifies what coin position(s) is/are to be programmed with the completely transferred coinspec. If the internal device error has been set during the “*Begin packed upload*” or “*Upload packet data*”, nothing will be done and this error will be returned. If any error occurs during the programming, it will be immediately returned. Error codes are detailed in chapter 3.2.

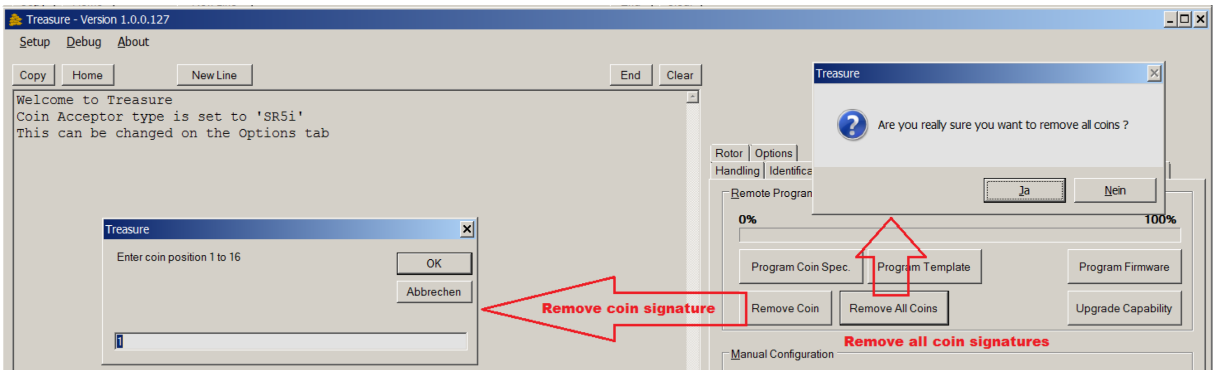
To repeat the RCP process, start with the “*Begin packet upload*”.

Below is a screen-shot of a Money Controls development tool ***Treasure***, showing how to trigger a coinspec programming sequence. Click on the “Program Coin Spec.” button to get to the dialog window that selects a desired coinspec. After selecting a coinspec, another dialog box will ask for a coin position to program with a coinspec. Pressing the OK button will execute the sequence described above.



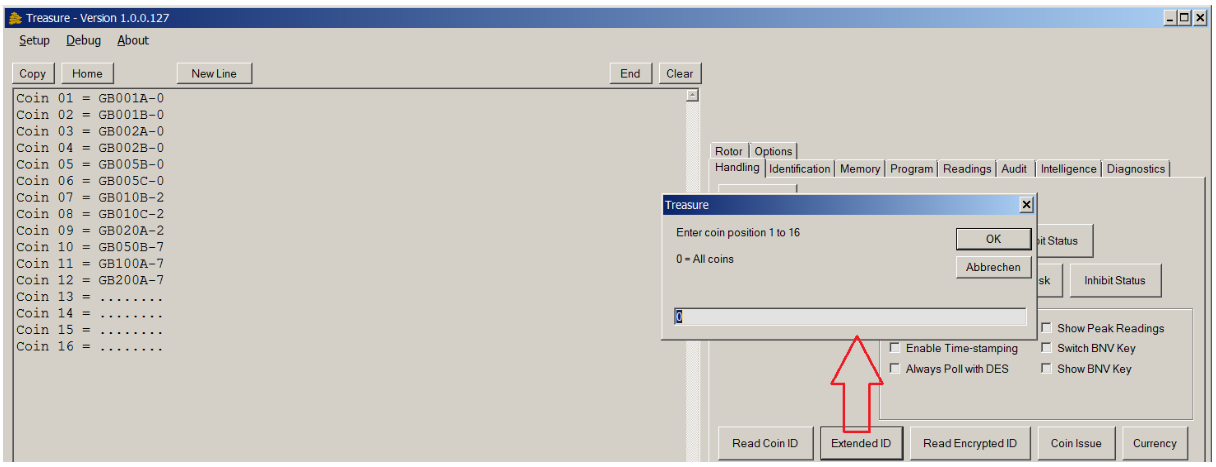
```
Welcome to Treasure
Coin Acceptor type is set to 'SR5i
This can be changed on the Options tab
```

Sub-header	**249, Remove coin signature**
Parameters	1-byte coin position ranging 1...16
Sub-header	**245, Remove all coin signatures**
Parameters	none
Reply	ACK on removal success; 1-byte code removal error; none on failure
Action	Depending on the command, either all 16 or a certain coin position signature will be removed. Removing a signature means clearing a coin identification, deactivating its channel and closing its acceptance window. If any error occurs during the programming, it will be immediately returned. Error codes are detailed in chapter 3.2.



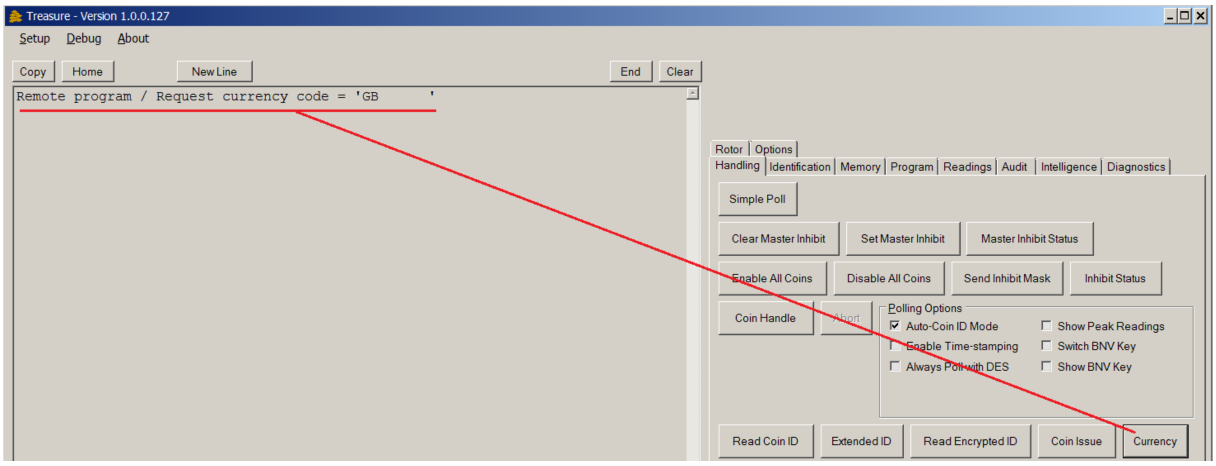
```
Welcome to Treasure
Coin Acceptor type is set to 'SR5i'
This can be changed on the Options tab
```

Sub-header	**248, Request extended coin ID**
Parameters	1-byte coin position ranging 1...16
Reply	8-char ASCII string or none on failure
Action	Returns an 8-char ASCII string, consisting of a standard ccTalk 6-char coin ID plus a **'-N'** suffix. **N** specifies the coin security level. See chapter 4.2 for details. If the specified coin position is unprogrammed/erased, the returned string will contain 8 ASCII dots.



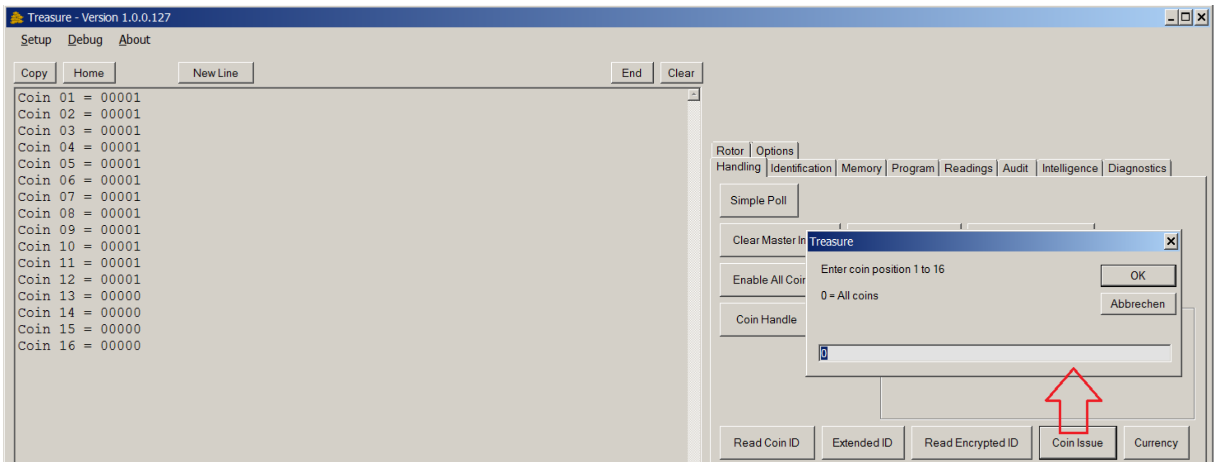
```
Coin 01 = GB001A-0
Coin 02 = GB001B-0
Coin 03 = GB002A-0
Coin 04 = GB002B-0
Coin 05 = GB005B-0
Coin 06 = GB005C-0
Coin 07 = GB010B-2
Coin 08 = GB010C-2
Coin 09 = GB020A-2
Coin 10 = GB050B-7
Coin 11 = GB100A-7
Coin 12 = GB200A-7
Coin 13 = ........
Coin 14 = ........
Coin 15 = ........
Coin 16 = ........
```

Sub-header	**244, Request currency code**
Parameters	none
Reply	8-char ASCII string or none on failure
Action	Returns an 8-char ASCII string, representing all coin currencies programmed in a validator. Each currency is coded with 2 ASCII chars, extracted from each standard coin ID. Therefore at most 4 currencies can be represented. Unused currency codes are filled with ASCII spaces.



```
Remote program / Request currency code = 'GB      '
```

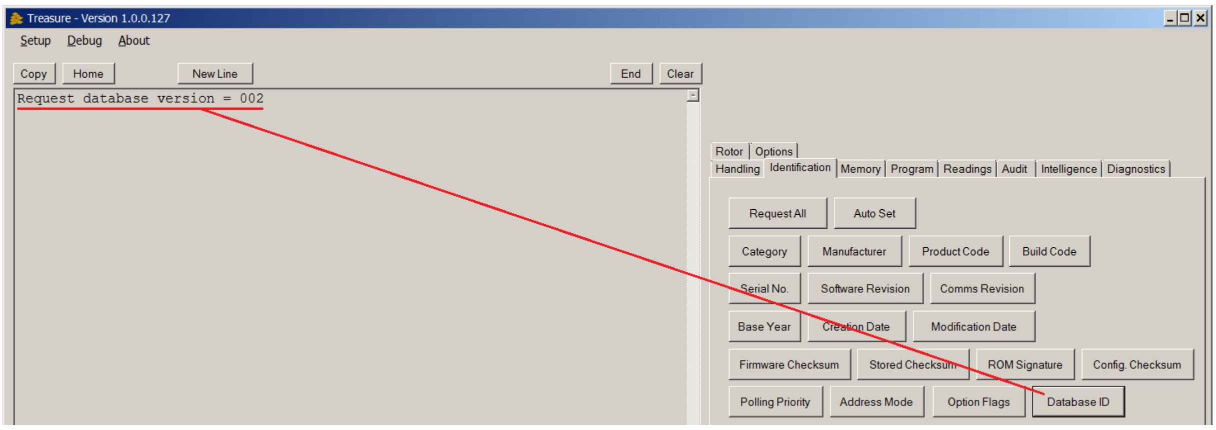
Sub-header	**243, Request coin issue**
Parameters	1-byte coin position ranging 1...16
Reply	5-char ASCII string or none on failure
Action	Returns a 5-char ASCII string, representing the coinspec issue number of the coin position. The string is padded with '0'-chars from the left. Maximal value is 99999, minimal valid is 1. Zero means the issue number is not available.



```
Coin 01 = 00001
Coin 02 = 00001
Coin 03 = 00001
Coin 04 = 00001
Coin 05 = 00001
Coin 06 = 00001
Coin 07 = 00001
Coin 08 = 00001
Coin 09 = 00001
Coin 10 = 00001
Coin 11 = 00001
Coin 12 = 00001
Coin 13 = 00000
Coin 14 = 00000
Coin 15 = 00000
Coin 16 = 00000
```

Not technically part of the RCP command set, but still worth mentioning, is the "*Request database version*" command (Header 243).

Header	**243, Request database version**
Parameters	none
Reply	1-byte database version, binary value.
Action	The returned database version can be used to select the correct database version folder in the RCP tree, described in chapter 2. The value of zero means, that the RCP functionality is deactivated in the validator.



```
Request database version = 002
```

3.2 Error codes

The error codes returned by certain RCP commands have a (slightly) different meaning in the NRI context, due to the very different internal technical implementation.

Table 3.2: Implemented error codes

Error Code	***Money Controls Description***	***NRI Meaning***
255	Packet too short	Wrong packet length within a coinspec → invalid coinspec file
254	Packet too long	The received data structure is too large to be saved in the validator, or too much data in a packet → invalid coinspec file
253	Too much data	The size of the reception buffer is too small to hold a portion of a coinspec during upload. Or programming was initiated, but the reception buffer still contains unprocessed coinspec data. → Invalid coinspec file or wrong command sequencing
252	File verification error 1	Coinspec CRC error → invalid coinspec file
251	File verification error 2	Invalid payload data in a coinspec → invalid coinspec file
240	Unsupported packet header	Invalid DOS stub or wrong packet sequence within a coinspec → invalid/incompatible coinspec file (destined for other validator type)
239	Unsupported file format	Unrecognized packet within a coinspec → invalid coinspec file
236	Non-matching database version	Non-matching database version
231	Sensor mismatch	Wrong/invalid/not found MSID → invalid coinspec file
210	Coinspec not uploaded or incomplete	Coinspec not yet fully uploaded, but coin programming or deletion was attempted → invalid coinspec file or wrong command sequencing
200	Internal neural network error	Internal calculation or any other error
100	Illegal coin position	Illegal coin position
1		Invalid reference tables in a validator → defective validator

4 Coinspec file format

As stated in previous chapters, a coinspec is an ordinary file. The contents of the file are device-specific, i.e. coinspecs designed for *v2eagle* cannot be successfully programmed into *v2colibri*. As the file extension **.bin** implies, most of the file contents is binary, except for the *DOS stub* explained in 4.3. The contents of a coinspec file may not be modified in any way, it should be transferred to a validator as is.

4.1 File naming convention

The naming scheme for all RCP coinspec files is **CCvvvR-N.bin**, where:

- **CC**: 2-char currency code (see ISO 4217)
- **vvv**: Coin value
- **R**: Coin revision
- **N**: Mapped coin security level (see next section)

4.2 Security Level mapping

Due to differences in level designators, the NRI channel security levels are be mapped to the Money Controls compatible levels. The table below lists all currently implemented mappings.

Table 4.1: Security Level Mapping

NRI Level	***MC Level***	***Acceptance window bandwidth***
1	1	wide
2	0	standard (normal)
3	2	narrow
4	7	super-narrow

4.3 DOS Stub

The first 23 bytes of any coinspec file are directly human-readable in any text editor, being an ASCII string. This string is called a *DOS stub*. The first 2 chars are always a 'C' followed by a space. This is a sort of marker, signifying a **C**oinspec data.

Following the marker is an uppercase string, that matches the “Validator Type” folder name described in chapter 2 above. Validators check this portion to make sure the coinspec is compatible with them. “EAGLE” stands for v2eagle, “COLIBRI” stands for v2colibri.

Separated by several spaces follows a concatenation of a 3-char Build-ID with a 2-digit database version. In conclusion, also separated by several spaces, is the 5-digit zero-padded coinspec issue number.

Example: Coinspec for v2eagle, Build DE0, DB version 1, issue number 5
```
C EAGLE   DE001   00005
```
Example: Coinspec for v2colibri, Build DE0, DB version 2, issue number 3
```
C COLIBRI DE002   00003
```