iai industrial systems
part of HID Global

Chip Encoding Interface Personalisation Systems

Software

Confidential information

Exported on
23/April/2025

Table of Contents

TCP/IP message structure 4
IAI server protocol 5
Passing the result back to the client 6
Supported commands 8
Error Codes 9
Configuration file 11
Location 11
Extended behavior 11
Hardware mapping 12

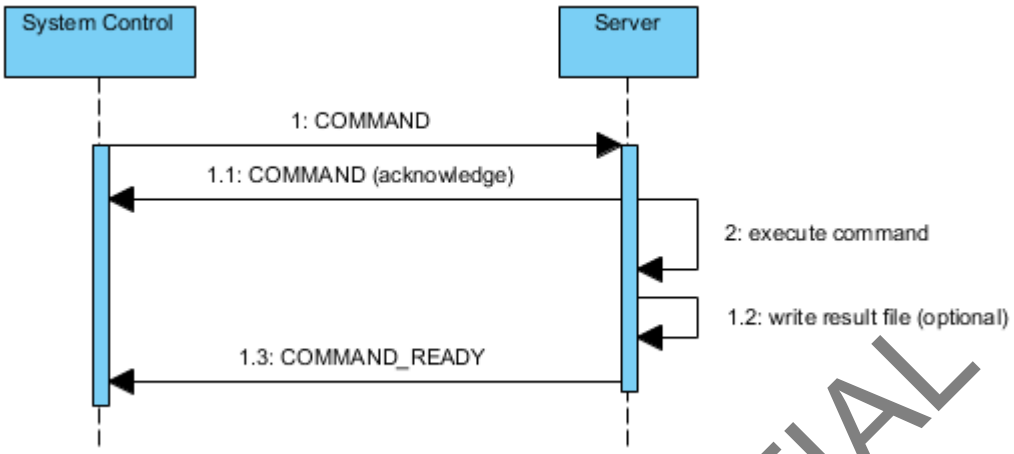
This document describes the Chip Encoding Interfacing for IAI's BookMaster and CardMaster personalisation systems.

TCP/IP message structure

The System Control Software and the Chip Programming Software interact via TCP/IP. The Chip Programming Software programs will act as server and the System Control Software as client.
For each encoding station the System Control software creates 1 general connection and 1 reader connection for each reader during start-up of the system. The first connection is used for general commands, the reader connections are used for reader specific commands. The order and total amount of connections is always: Connection 1 (general), Connection 2 (Reader 1), Connection 3 (Reader 2), etc. The System Control Software triggers the Programming Software by sending commands over one of the established connections.

IAI server protocol

The protocol used between client (SystemControl) and server (your Chip Server) is a rather simple TCP/IP message protocol. The protocol follows the basics of the "don't speak unless spoken to" philosophy. So the server will never communicate by itself, it will only respond on commands received from the client.



For example, a command that would fit in the above schematic could be WRITEANDVERIFY. Then we would get:

SC -> Server	WRITEANDVERIFY 1 A1234567	Parameter 1: indicates the reader board the message is send too Parameter A1234567: indicates the product number of the product currently in the reader
Server -> SC	WRITEANDVERIFY 1	The acknowledge message which in the case has to include the readerboard index (backwards compatibility reasons, it's strictly not necessary)
Server	Executes the WRITEANDVERIFY command	
Server	Optionally writes the result file	It up to the customer whether to use result files to communicate the result of the command to the client, or, to incorporate the result data in the TCP/IP message
Server -> SC	WRITEANDVERIFY 1_READY	The ready message, again, depending on the customers choice it can contain the result of the command. If result files are used, the ready message does not need more information. More on this topic later on in the document.

Every TCP message ends with a <LF> character (ASCII 0010). The client will always end each message with a <LF> character, and will expect a <LF> character at the end of every message from the server. If a <LF> is not present, the client will report a protocol error on the GUI.

Passing the result back to the client

The result of each command must be communicated back to the system control. This can be done either by writing a result file (an INI file) or by incorporating the result information in the ready response message. Both are explained in this document.

Result files

Result files are written by the server to a location which is accessible by the system control. The result file has the following structure:

```
[COMMAND]
ErrorCode=0
ErrorString=”Any string”
```

COMMAND: the command which is executed, e.g. “WRITEANDVERIFY 2”
ErrorCode: an integer which represents the error of the command
ErrorString: a human readable representation of the error

For example, a WRITEANDVERIFY result would look as follows:

```
[WRITEANDVERIFY 1]
ErrorCode=0
ErrorString=”no error”
```


Result over TCP

It's also possible to incorporate the result in the TCP ready message, which saves the trouble of dealing with files for inter process communication.
The command result information is added at the end of the ready message, separated by a colon. The result data itself is a list of key-value pairs, separated by <CR> characters (ASCII 0013).
An example of the ready message over TCP would look as follows:

```
WRITEANDVERIFY 1_READY:ErrorCode=0\rErrorMessage=no error\n
```

NOTE: The colon character is a reserved character, and cannot be used in the values of the key/value pairs. For example, the following message will fail to parse due to the colon in the ErrorMessage value:

```
WRITEANDVERIFY 4_READY:ErrorCode=1ErrorMessage=|CL:invalid response from server 500|
```

Supported commands

<command>	<argument(s)>	Description
INIT	<configuration file>	**General Connection** A trigger from the system control to initialize the server. If needed we can send a configuration file so that the server can receive configuration if needed. If there is no need for such configuration it can be ignored.
WRITEANDVERIFY	<#readerboard> <#product> <read number> <job name>	**Readerboard Connection** Start programming for specific reader board. **#readerboard:** The index of the reader **#product:** The product number which is used as reference to find product data files **read number:** [Optional]: the read number which is required by the chip server **job name:** [Optional]: the name of the job, which is required to create the relative path to the host data files Example: **WRITEANDVERIFY 2 XA00001 12345678 AB1234**

<command>	<argument(s)>	Description
READNUMBER	<#readerboard> <#product> <job name>	**Readerboard Connection** Read number in the chip. The read number has should be returned as part of the result of the command. **#readerboard:** The index of the reader **#product:** [Optional,BMPro only]: The product number which is used as reference to find product data files. This cannot be combined with the job name argument. **job name:** [Optional]: the name of the job, which is required to create the relative path to the host data files INI result file example: [READNUMBER 1] ErrorCode=0 ErrorString="no error" ReadNumber="XA000001" TCP result example READNUMBER 1_READY:ErrorCode=0\rReadNumber=XA00001 \r\n

Table 1 – Commands

Error Codes

The System Control enforces the following error code convention:

Value	Meaning
0	Chip successfully programmed.
< 0	Chip not programmed but chip is still programmable (product will receive rework status).

> 0	Chip programing failed. Chip can't be programmed again (product will receive reject status).

Configuration file

Location

The location of the configuration file can be defined in either the unit configuration or in the recipe. The value defined in the recipe takes precedence over the value defined in the unit configuration. The unit configuration value can be adjusted through the settings UI of the main application. To define the value in a recipe see the following examples:

CMO encode recipe

```
<Encode Contact="true" Contactless="true" ApplicationFile="S:
\IAI\Data\Recipes\Recipe\Encoding\EncodeApplication.ini" />
```

Note: For the CMO both the recipe value and the unit configuration value can be omitted. The INIT command is not sent to the server in such a case.

BMO encode recipe

```
<UnitRecipeConfigurations xsi:type="EncodeUnitRecipeConfiguration">
  <NoProcessing>false</NoProcessing>
  <ChipServerTimeOut>5000</ChipServerTimeOut>
  <ApplicationFile>S:\IAI\Data\Recipes\Recipe\Encoding\EncodeApplication.ini</
ApplicationFile>
</UnitRecipeConfigurations>
```


Extended behavior

The IAI machine can write which encoder positions are enabled in the configuration file send by the INIT command.

1. It will update the application file which is copied per server and configured before the INIT command is send.
2. It will add any reader known to the machine, starting with 01
3. 0 is not enabled, 1 is enabled
4. Any other content is not affected

```
[ChipApplication]
enabled01=1
enabled02=1
enabled03=0
enabled04=1
```

Hardware mapping