Voyantic

Ensurance™ Recipe Builder

User manual

Ensurance™ Case Builder

Version 11/2019

1 Important Information

READ THE COMPLETE USER GUIDE CAREFULLY BEFORE USING THE ENSURANCE™ SYSTEM.

Voyantic Ltd. operates a policy of ongoing development. Voyantic Ltd. reserves the right to make changes and improvements to any of the products described in this user guide without prior notice.

THE CONTENTS OF THIS USER GUIDE ARE PROVIDED "AS IS". EXCEPT AS REQUIRED BY APPLICABLE LAW, NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, ARE MADE IN RELATION TO THE ACCURACY, RELIABILITY OR CONTENTS OF THIS USER GUIDE. VOYANTIC LTD. RESERVES THE RIGHT TO REVISE THIS USER GUIDE OR WITHDRAW IT AT ANY TIME WITHOUT PRIOR NOTICE.

General Terms and Conditions of Voyantic Ltd. shall apply.
http://www.voyantic.com/termsandconditions.pdf

The Ensurance system components can use radio frequencies, use of which may require local or otherwise applicable governmental or official approvals or permissions.
VOYANTIC LTD. DOES NOT WARRANT ANY TYPE OF APPROVAL FOR ENSURANCE, REELSURANCE PRO, OR THE INSTALLED COMPONENTS. VOYANTIC LTD. SHALL UNDER NO CIRCUMSTANCES BE LIABLE OF ANY USE OF ANY OF THESE ITEMS.

Use of the Ensurance software requires a valid license. Any copyrights, patents and other intellectual property rights (including the right to change and further develop) in and to the Ensurance (including any related documentation and other materials delivered by Voyantic Ltd.) shall belong to Voyantic Ltd.

Voyantic©, Ensurance™ , Reelsurance™, , Reelsurance lite™, Reelsurance Pro™ Tagsurance™, Tagsurance HF™ Snoop Pro™, Readformance™, Tagformance™, Tagformance lite™ and Tagformance Pro™ are trademarks of Voyantic Ltd. For improving clarity of the text these trademarks are not marked in the manual text.

2 Table of Contents

About

This document gives an overview for creating personalization recipes for Ensurance system by using either a graphical tool or manually. The text covers definition of the utilization of the case builder software, configuration files and data formats.

Chapter 3 contains instructions on how to install Ensurance Case Builder software and list the generic system requirements. It also lists what other software is required to operate the system.

Chapters 4 and 5 explain how to create a recipe for Voyantic Ensurance personalization system. Chapter 4 describes the process of creating the recipe and introduces graphical Ensurance case builder. Chapter 5 describes the contents of the recipe files in detail and introduces a method of programming the recipes manually.

3 Installation

3.1 Before You Begin

3.1.1 Install Software

The basic Ensurance package contains the Ensurance Case Builder software which will be descripted within this manual. Please refer to Ensurance GUI User manual for the necessary installation procedures.

3.1.2 System Requirements

Minimum system requirements for Ensurance Case Builder:

- Windows 7 / Windows 8 / Windows 10
- 2.4 GHz Pentium 4 or equivalent (e.g. 1.6 GHz Pentium M or AMD Athlon 2800+)
- 2 GB RAM
- 1 GB free hard disk space
- Minimum screen resolution 1024 x 768
- Mouse, keyboard

4 Ensurance Case Builder

4.1 Overview

The Case Builder is a graphical tool for creating compatible test recipes for Ensurance personalization system. It consists of 4 main parts:

- **Control Panel**: It allows to insert the Master Case file name, Save the current configuration, Load a previous test recipe and Clear the current worksheet.
- **Stage Builder**: It allows to configure the necessary sub-processes for a specific test recipe by defining their correct order, Connection & Hardware Settings, Task definition (if any) and the I/O mappings.
- **Variable Manager:** It allows to define all the necessary variables (System, Data Conversion, File origin and Incremental) for a specific test recipe.
- **Mapping list:** Summary table for all the I/O mappings and Data Converters within a specific test recipe.

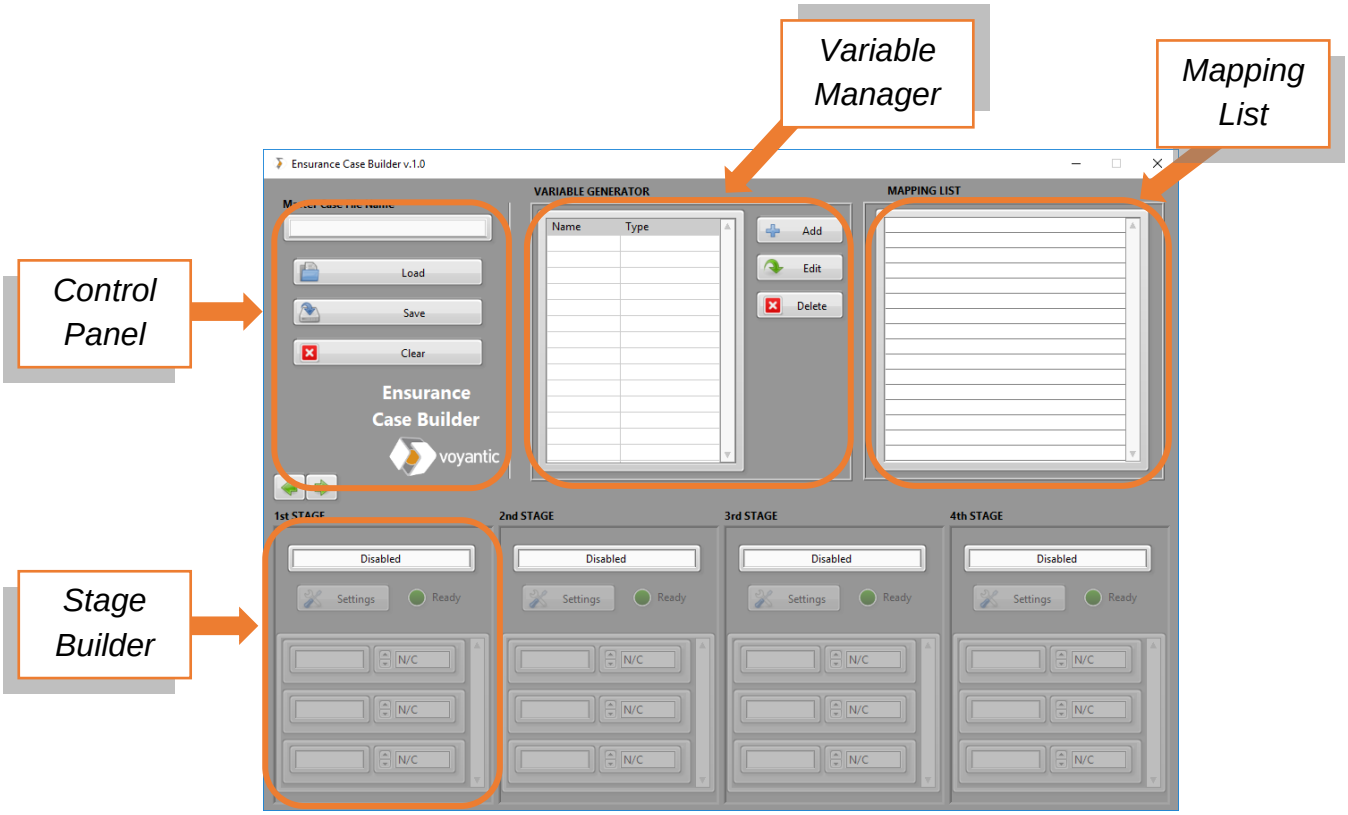

Figure 1: Ensurance Case Builder Overview

4.2 Definition of the process steps

The first step to be defined is the number of sub-processes (devices) involved in a specific test recipe and their sequential order in the system. The user can define up to 12 stages through the matching case builders by following a left-right order. Within a single “Stage Builder” you may define:

- **Device Type**: UHF Encoder, UHF Performance Tester (Tagsurance UHF), HF Encoder, HF Performance Tester (Tagsurance HF), Printer, Barcode Reader.
- **Device Settings**: Connection and HW Configurations, tasks list (if available)
- **I/O Mapping Array**: It defines the external I/O connections

In the following paragraphs a detailed description of the device settings will be shown.

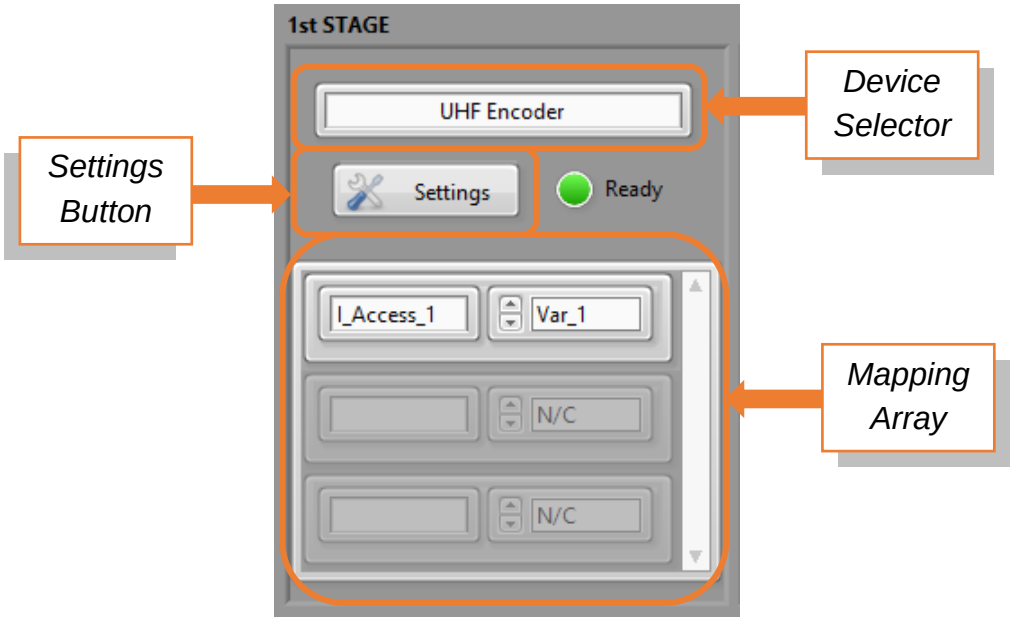

Figure 2: Stage Builder Front Panel

4.2.1 Definition of UHF encoding tasks

Once the UHF Encoder has been selected through the Device Selector drop-down menu, by pressing the Settings button, it is possible to setup the connection and HW information and the define a proper tasks list.

4.2.1.1 Connection settings

The connection and HW settings for the UHF Encoder are listed here:

- IP Address
- TCP Port (keep the default value “1234”)
- Trigger Source -> SW or HW
- Trigger Mode -> Falling Edge or Rising Edge
- TX Power (dBm) -> Transmission power of the UHF Encoder

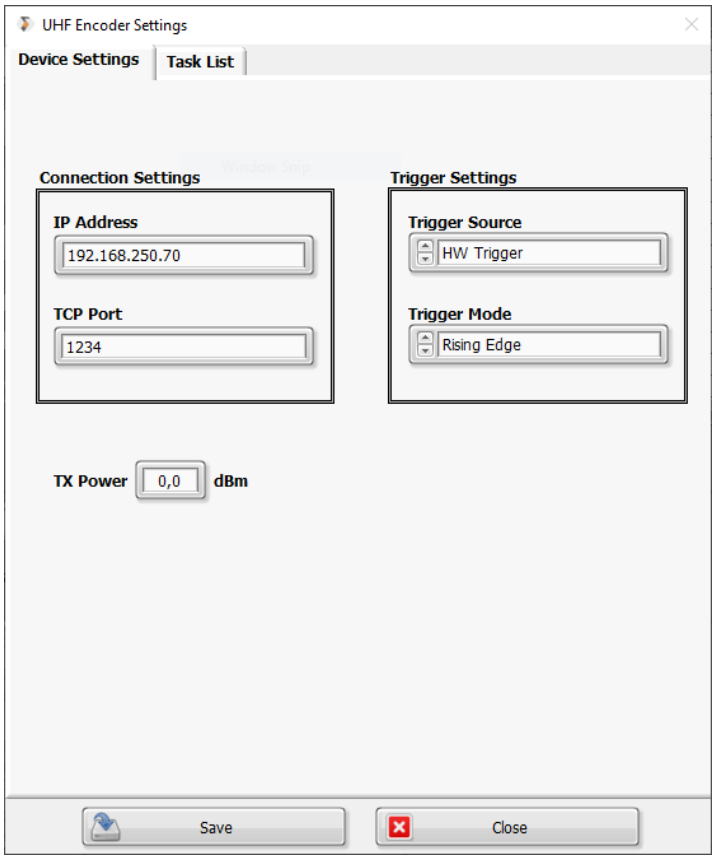

Figure 3: UHF Encoder Device Settings page

4.2.1.2 Task options

The Case Builder includes an automatic tool for creating and defining a UHF Encoder task list. The reader supports the following commands (ISO-18000-63): Inventory, Select, Access, Read, Write, BlockWrite, Lock, and Kill.

VerifyRead command is for the memory content verification. UHF Encoder will read the memory content by using a Read command, and compares the read data with the data given as parameter.

Note: The Inventory command is mandatory for each case file. The Select command is defined here after the Inventory, but UHF Encoder will process the Selects before the Inventory.

Once a specific task is selected from the Task Type drop-down menu, the user can modify the task parameters and add it within the task list by pressing the Add button. It is possible to edit or remove a task from the list by pressing the matching buttons.

Some commands can also share some data (you may recognize them whenever a “Use Result as Variable” checkbox is present) with other sub-processes as well as gathering some input data (recognizable by the “Use Variable Data” checkbox). Here you can find a list of all the possibilities:

- **Access**: The Password parameter (Variable Input)
- **Write**: The Data Word parameter (Variable Input)
- **BlockWrite**: The Data Words array parameter (Variable Input)
- **Kill**: The Password parameter (Variable Input)
- **Read**: The data read from the memory (Shared Output)
- **VerifyRead**: The comparison data (Variable Input)

Once the tasks list is correctly defined, press the Save button to go back to the main interface.

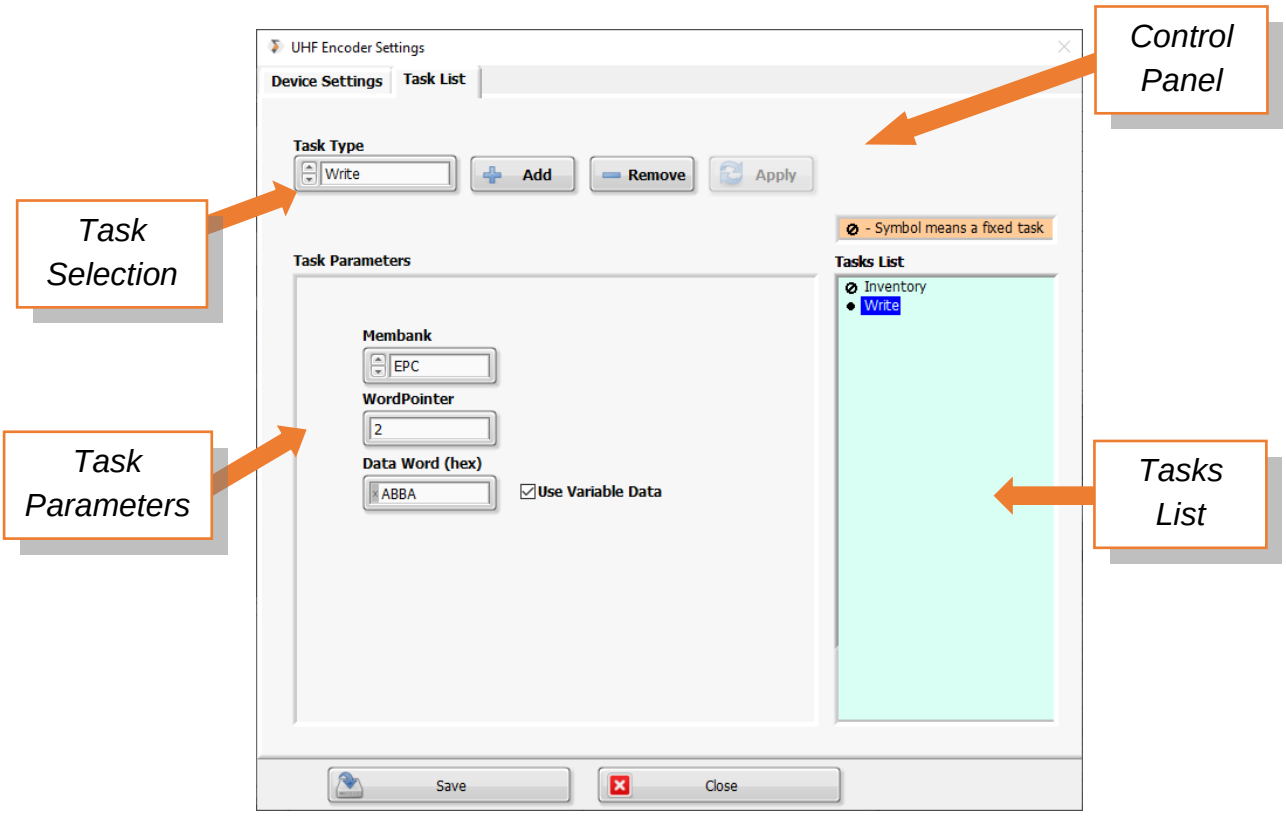

Figure 4: UHF Encoder Tasks list creator

4.2.2 Definition of UHF Performance tester tasks

Once the Tagsurance UHF has been selected through the Device Selector drop-down menu, by pressing the Settings button, it is possible to setup the connection and HW information and the define a proper test list (with a dedicated Case Builder).

4.2.2.1 Connection settings

The connection settings for Tagsurance UHF are listed here:

- Name: Device nickname
- COM Port: Dedicated serial port (It is possible to gather this information automatically if the device is actually connected with the same working laptop which is running the Ensurance Case Builder)
- Device Name: Device name stored in the device memory (It is possible to gather this information automatically if the device is actually connected with the same working laptop which is running the Ensurance Case Builder).

4.2.2.2 Hardware and test options

The Case Builder includes a link to a Tagsurance Test Manager in which it is possible to define both Hardware settings and the list of the tests they have to performed (please refer to Tagsurance 2 manual for detailed instruction about the usage of the Test Manager).

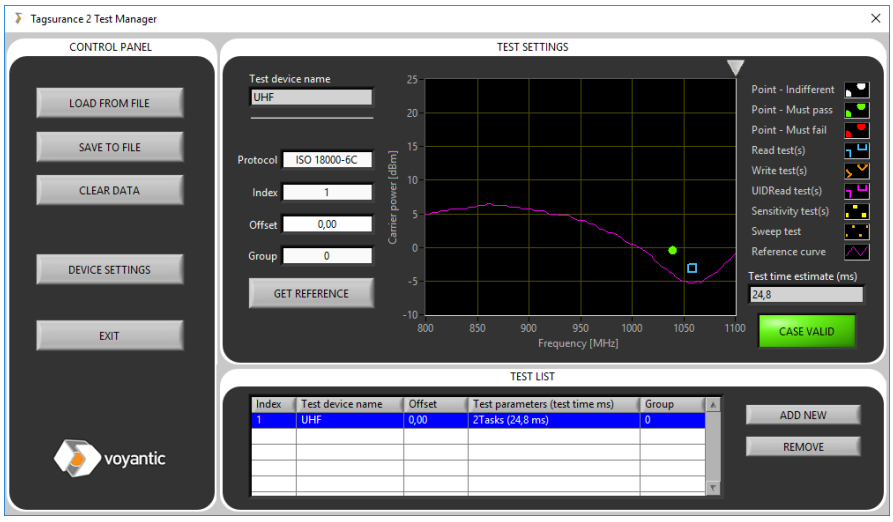

Figure 5: Tagsurance UHF Test Manager view

After a proper case file has been defined and correctly saved, it is possible to share the data read towards a Read test (EPC, TID, User and Reserved memory) by clicking the matching “Use Result as Variable?” checkbox within the *Shared Reads* list.

Once the Tagsurance UHF settings are correctly defined, press the Save button to go back to the main interface.

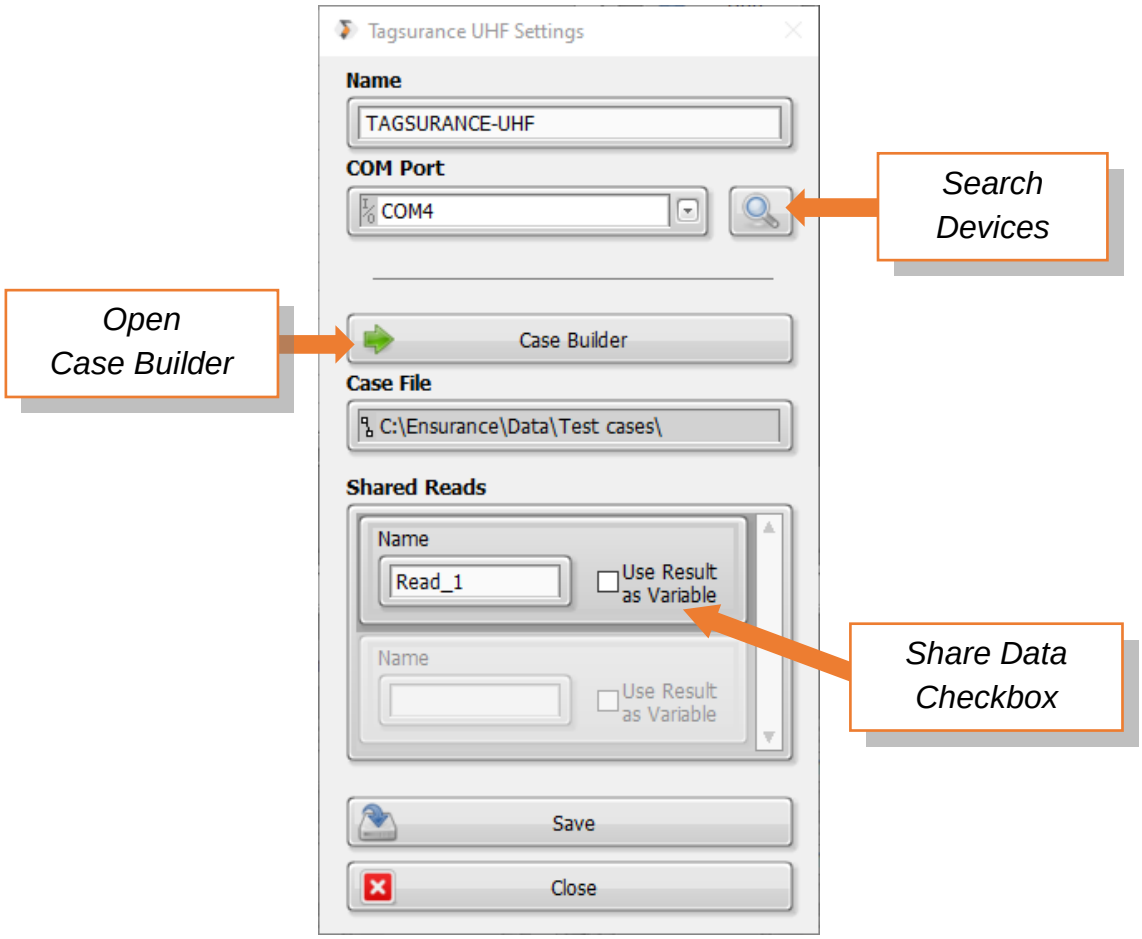

Figure 6: Tagsurance UHF settings view

4.2.3 Definition of HF encoding tasks

Once the HF Encoder has been selected through the Device Selector drop-down menu, by pressing the Settings button, it is possible to setup the connection and HW information and the define a proper task list.

4.2.3.1 Connection settings

The connection and HW settings for the HF Encoder are listed here:

- COM Port: Serial Port name
- Timeout [ms]
- Trigger Mode: Low State, High State, Falling Edge, Rising Edge
- Busy/Ready Inverted? checkbox
- Pass/Fail Inverted? Checkbox

Without inversion, signals are at high state when *Busy* and *Pass*

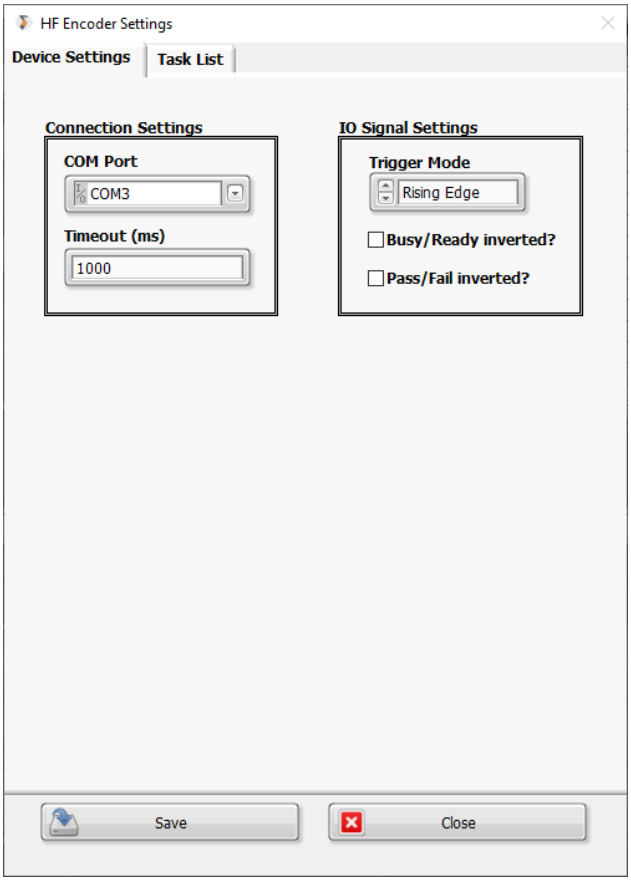

Figure 7: HF Encoder Settings View

4.2.3.2 Task options

The Case Builder includes an automatic tool for creating and defining a HF Encoder task list. The reader supports the following commands and matching tag families:

- **NTAG203**: Carrier, Activate, Read, Write, Lock
- **NTAG210**: Carrier, Activate, Read, Write, Authenticate
- **NTAG213**: Carrier, Activate, Read, Write, Lock, Authenticate, NFC URL, NFC Text
- **SLIX2**: Carrier, Activate, Read, Write Single Block, NFC URL, NFC Text, Verify Read, Verify NFC URL, Verify NFC Text

Once a specific task is selected from the Task Type drop-down menu (the task availability will depend on the selected Tag Type), the user can modify the task parameters and add it within the task list by pressing the Add button. It is possible to edit or remove a task from the list by pressing the matching buttons.

Note: The *Carrier ON* and *Activate* tasks at the beginning of the task list, and the Carrier OFF at the end of the task list are mandatory. Those are always ready on the list, and they cannot be removed.

Some commands can also share some data (you may recognize them whenever a “Use Result as Variable” checkbox is present) with other sub-processes as well as gathering some input data (recognizable by the “Use Variable Data” checkbox). Here you can find a list of all the possibilities:

- **Activate**: It will share the UID data (Variable Output)
- **Read**: Read Page (Variable Output)
- **Write**: Data Array parameter (Variable Input)
- **Authenticate**: Password array parameter (Variable Input)
- **NFC URL**: URL parameter (Variable Input)
- **NFC Text**: Text parameter (Variable Input)
- **Read (SLIX2)**: Read Block (Variable Output)
- **Write Single Block**: Data Array parameter (Variable Input)
- **Verify Read**: Data Array parameter (Variable Input)
- **Verify NFC URL**: URL parameter (Variable Input)
- **Verify NFC Text**: Text parameter (Variable Input)

Once the tasks list is correctly defined, press the Save button to go back to the main interface.

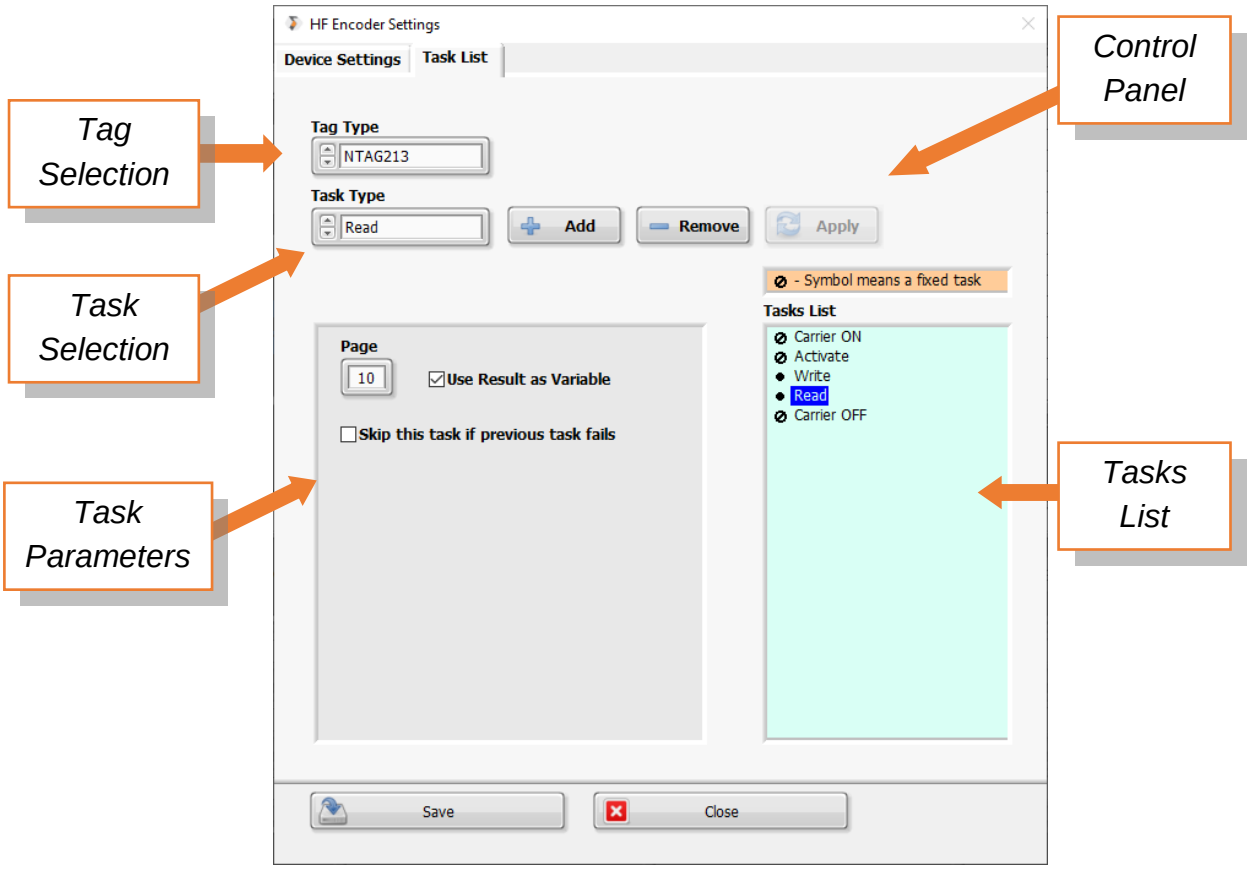

Figure 8: HF Encoder Tasks list creator

4.2.4 Definition of HF Performance tester tasks

Once the Tagsurance HF has been selected through the Device Selector drop-down menu, by pressing the Settings button, it is possible to setup the connection and HW information and the define a proper test list (with a dedicated Case Builder).

4.2.4.1 Connection settings

The connection settings for Tagsurance HF are listed here:

- Device Name: Device nickname
- Network Address: Dedicated connection address (It is possible to gather this information automatically if the device is actually connected with the same working laptop which is running the Ensurance Case Builder)
- TCP Timeout (ms)

4.2.4.2 Hardware and test options

The Case Builder includes a link to a Tagsurance Test Manager in which it is possible to define both Hardware settings and the list of the tests they need to be performed (please refer to Tagsurance 2 manual for detailed instruction about the usage of the Test Manager).

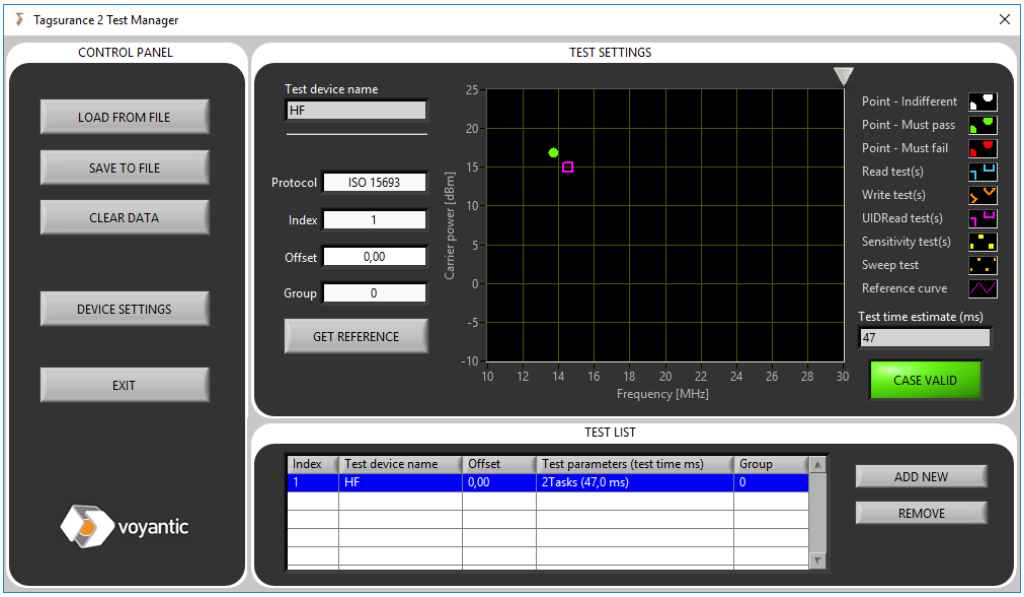

Figure 9: Tagsurance HF Test Manager view

Once the Tagsurance HF settings are correctly defined, press the Save button to go back to the main interface.

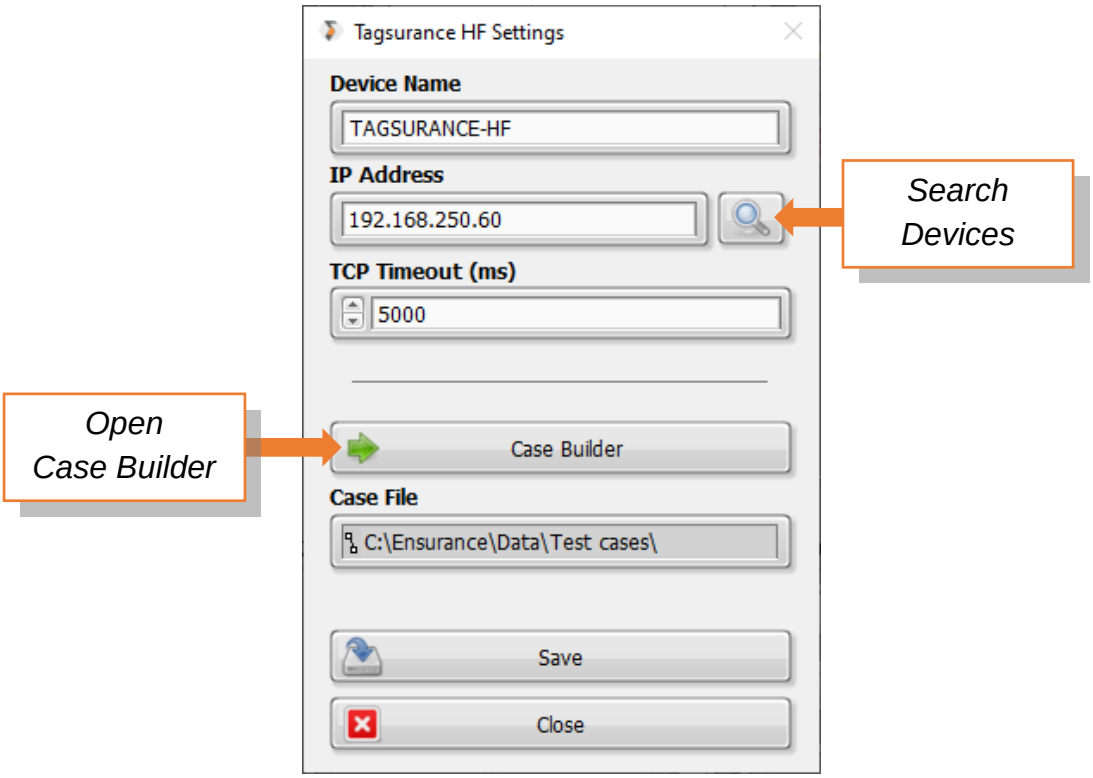

Figure 10: Tagsurance HF Settings view

4.2.5 Definition of variable data printing tasks

Once the Printer has been selected through the Device Selector drop-down menu, by pressing the Settings button, it is possible to setup the connection, HW information and Job/Field list definition.

4.2.5.1 Connection settings

The connection settings for the Printer are listed here:

- COM Port: dedicated serial port
- Baud Rate
- Timeout (ms)
- Trigger Source: HW or SW

4.2.5.2 Jobs and Field Data

The user can define:

- A Jobs List through the matching array and eventually enable the possibility to switch between them on-the-fly by checking the “Variable Jobs” checkbox; therefore a Variable Input will be generate (Job selector).
- A “Bad Tag Marker” job by enabling the “Bad Tag Marking?” checkbox and filling the matching field (it cannot be left empty once it has been enabled)
- A Field Data list: for each Field a Name, Line Wrap Width and Type need to be declared and a new Variable Input will be generated automatically.

Once the Printer settings are correctly defined, press the Save button to go back to the main interface.

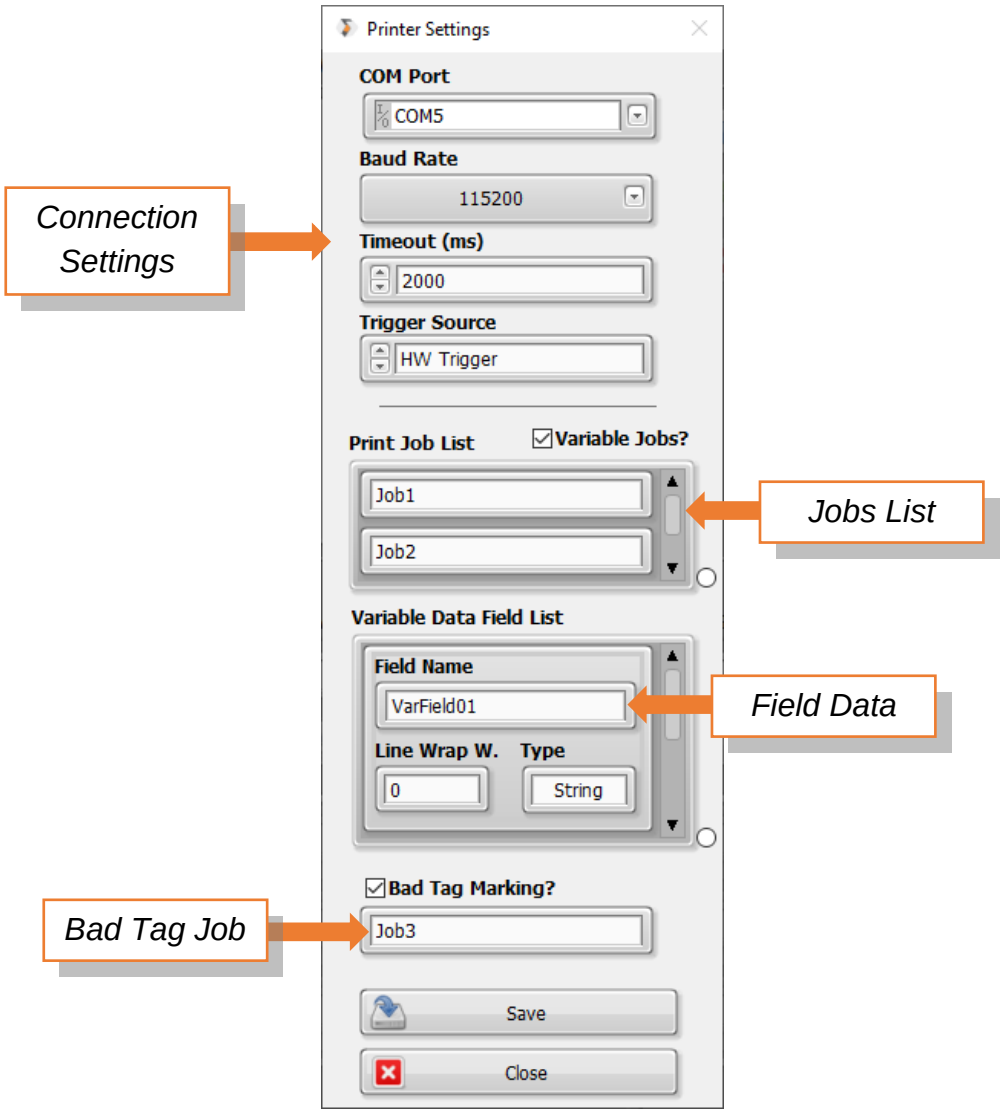

Figure 11: Printer Settings View

4.2.6 Definition of 1D/2D code reading tasks

Once the Barcode Reader has been selected through the Device Selector drop-down menu, by pressing the Settings button, it is possible to setup the connection and the camera properties.

4.2.6.1 Connection settings

The connection settings for the Printer are listed here:

- IP Address
- TCP Port: default value is 2111
- Timeout [ms]: default value is 500ms

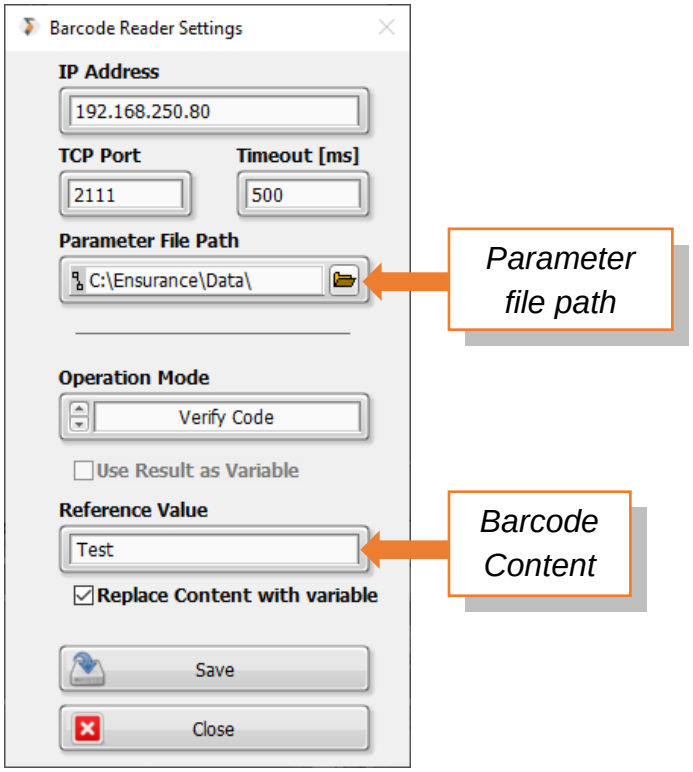

Figure 12: Barcode Reader Settings view

4.2.6.2 Camera properties

The Barcode Reader sub-process can be used to either read or verify a code.

- Code Reading: the read code can be shared as a variable.
- Verification: the user needs to insert the Barcode content constant. In case of variable content from label to label, the constant content will be overwritten with variable from the other subprocesses.

4.2.6.3 Parameter file

The parameter file will be generated through a specific barcode reader configurator (Please refer to the correct manual). From the Case Builder you just need to point out the correct file path from the matching field.

4.3 Variables and mapping definition

In the previous chapters you have learned how to define the sub-processes settings and tasks lists as well as their sequence within the test recipe. The next step is defining the variables involved through the variable manager, create and finalize the correct mapping between sub-processes.

4.3.1 Variable Manager

Variable Manager is an integrated tool for creating, editing and deleting variables by using the matching buttons. The variable types that can be generated are:

- **System**: They are generically used for creating a direct connection between multiple subprocesses or between a sub-process and a data converter variable
- **Data Converter**: They are used to usually connect 2 sub-processes which they require a conversion algorithm in order to be correctly mapped.
- **File Read**: These variables take their values from within an external CSV file
- **Autogenerated**: These variables are generated independently from any other processes

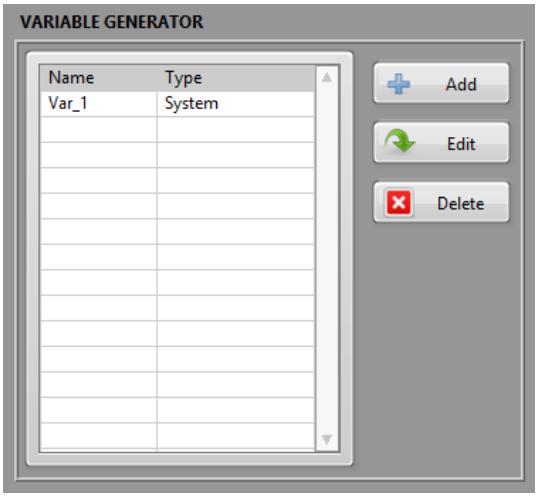

Figure 13: Variable Manager view

4.3.2 Variable Types

4.3.2.1 File Read Variable

The “File Read” variable allows the user to create a source link with a CSV file. The specific variable settings are:

- Variable File Path: It is the file of the connected CSV file.
- Column in the file: This is the index of the CSV file table column to which the variable is connect with. The index of the first column is 0.
- Variable Type: Hex, Dec (U32) or Ascii string

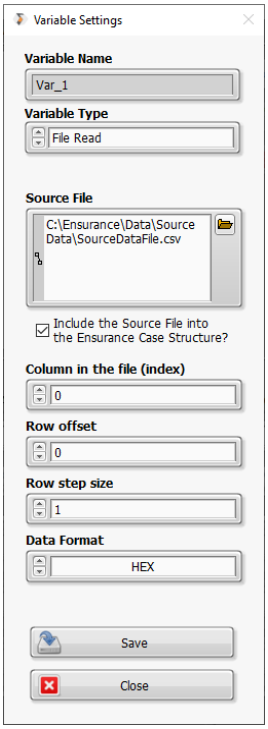

Figure 14: File Origin Variable Settings

4.3.2.2 Autogenerated Variables

The “**Incremental Value**” variables allow the user to define an internal counter with different characteristics. The specific variable settings are:

- Data format: Hex (0) or Dec (1) number and String (2)
- Initial value: Initial value of the counter
- Step size: Increment width
- Minimum length (bytes): detailed description in the chapter 5.
- String prefix (String optional): detailed description in the chapter 5.

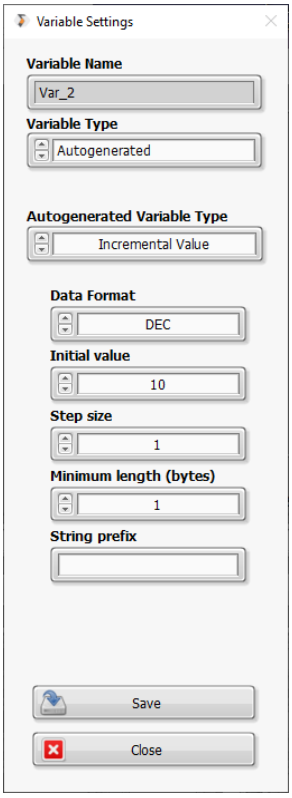

Figure 15: Incremental Value Settings (Numeric Counter from 0 with Step size = 1)

The “**Fixed Value**” variables allow the user easily define the data to be used in personalization process. Available data formats are:

- Byte array
- U8 Hex (single byte)
- U32 decimal number (always four bytes)
- Ascii string.

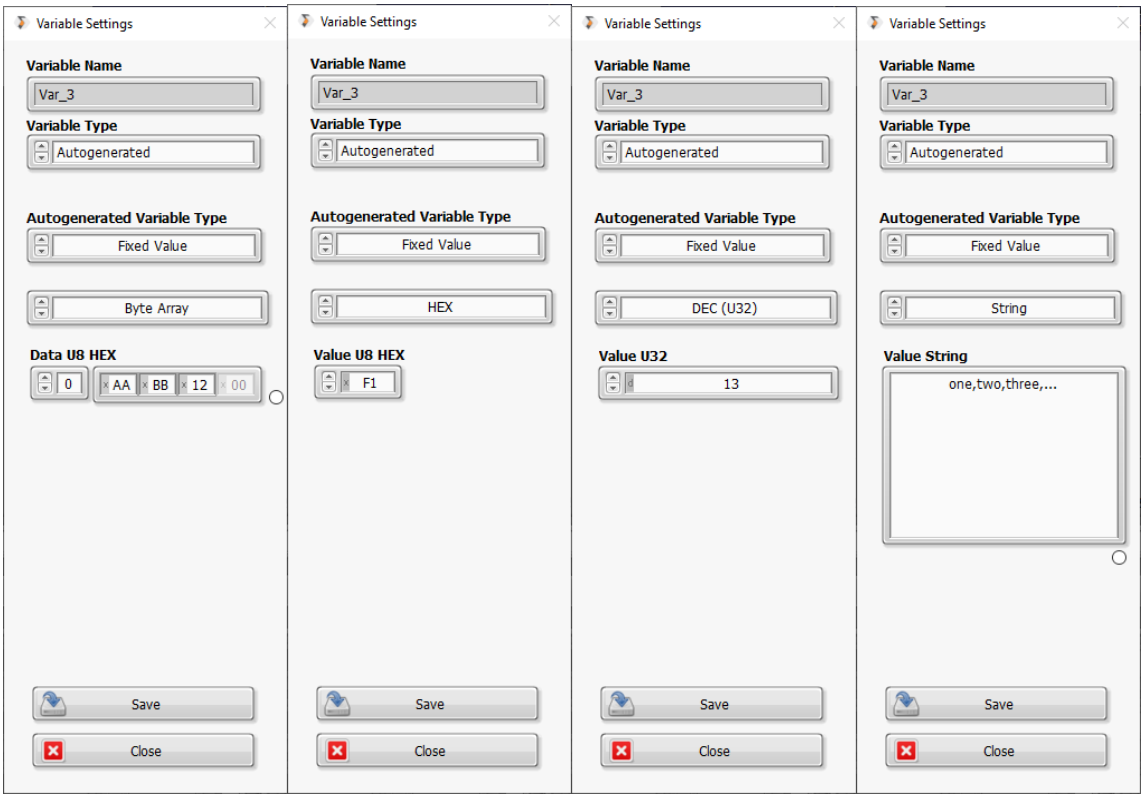

Figure 16: Fixed Value Settings

4.3.3 Data Converter Variables

The Data Converter variables are used to connect two sub-processes which they require a conversion algorithm in order to be correctly mapped or it is required to convert a data read from a “sourcing” variable (File origin or Incremental). The available data converters are: Memory to String, String to Memory, Zero Padding, Join Variables, Crop Variable, GS1 EPC GIAI-96 (you will find a detailed description in the chapter 285).

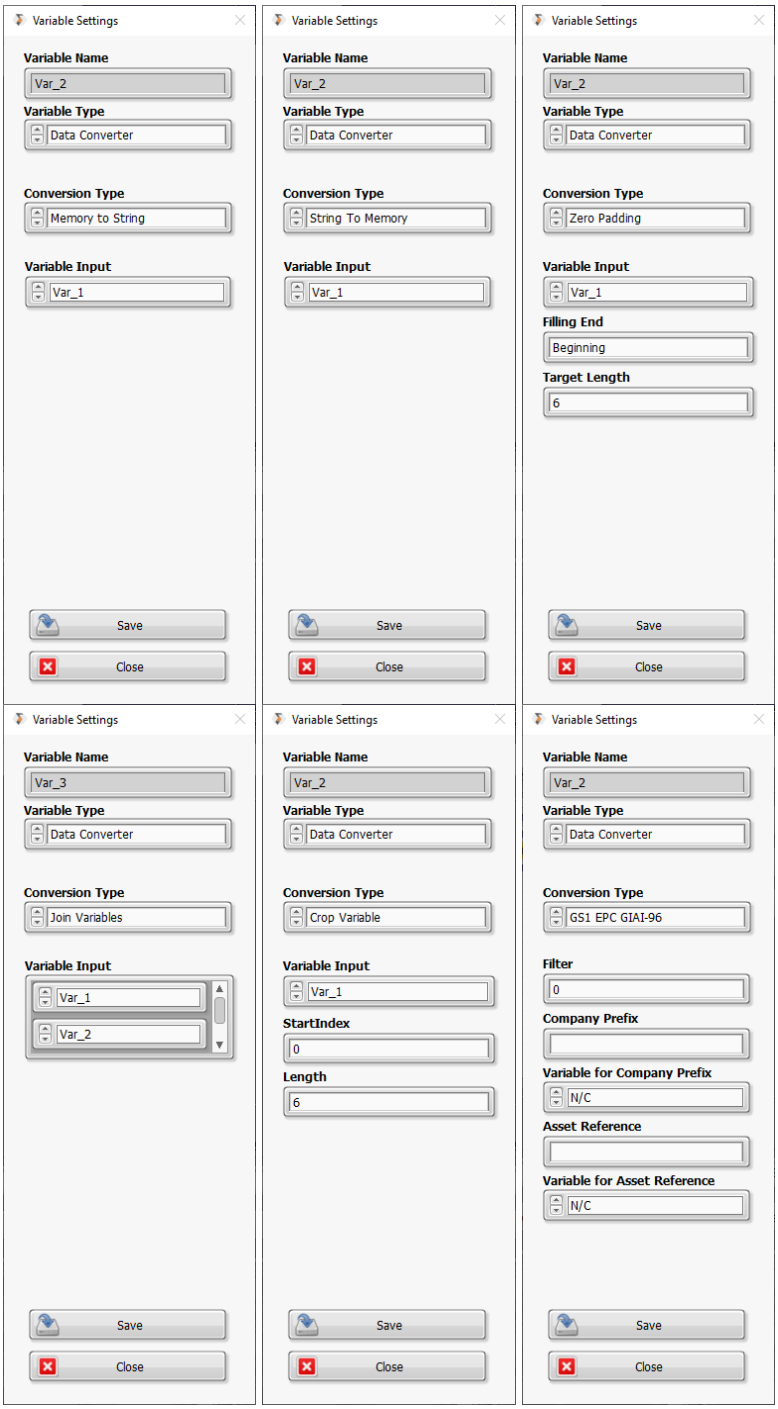

Figure 17: Data Converters Settings

4.3.4 Linking variables and sub-processes

Once all the sub-processes and the variables involved in the test recipe are correctly defined, the last step before saving the configuration is mapping all the sub-processes I/O. In order to make the process as clear as possible, let's take the following example:

Sub-processes sequence:

1. **Tagsurance HF**
    - Quality control
    - Point test at 13,56 MHz, 20 dBm
2. **HF Encoder**
    - Device and Trigger Settings
    - Tasks list:
        - Carrier On?
        - Activate (Read UID)
        - Verify NFC URL
        - Carrier Off
3. **Code Reader**
    - Device Settings
    - QR Code Verification

Variables List:

- Var 1: File Origin
- Var 2: System
- Var 3: Data Converter (Var 2 -> Memory to String -> Var 3)
- Var 4: Data Converter (Var1, Var 3 -> Join Variables -> Var 4)

The following screenshot shows an overview of the GUI before the mapping operation.

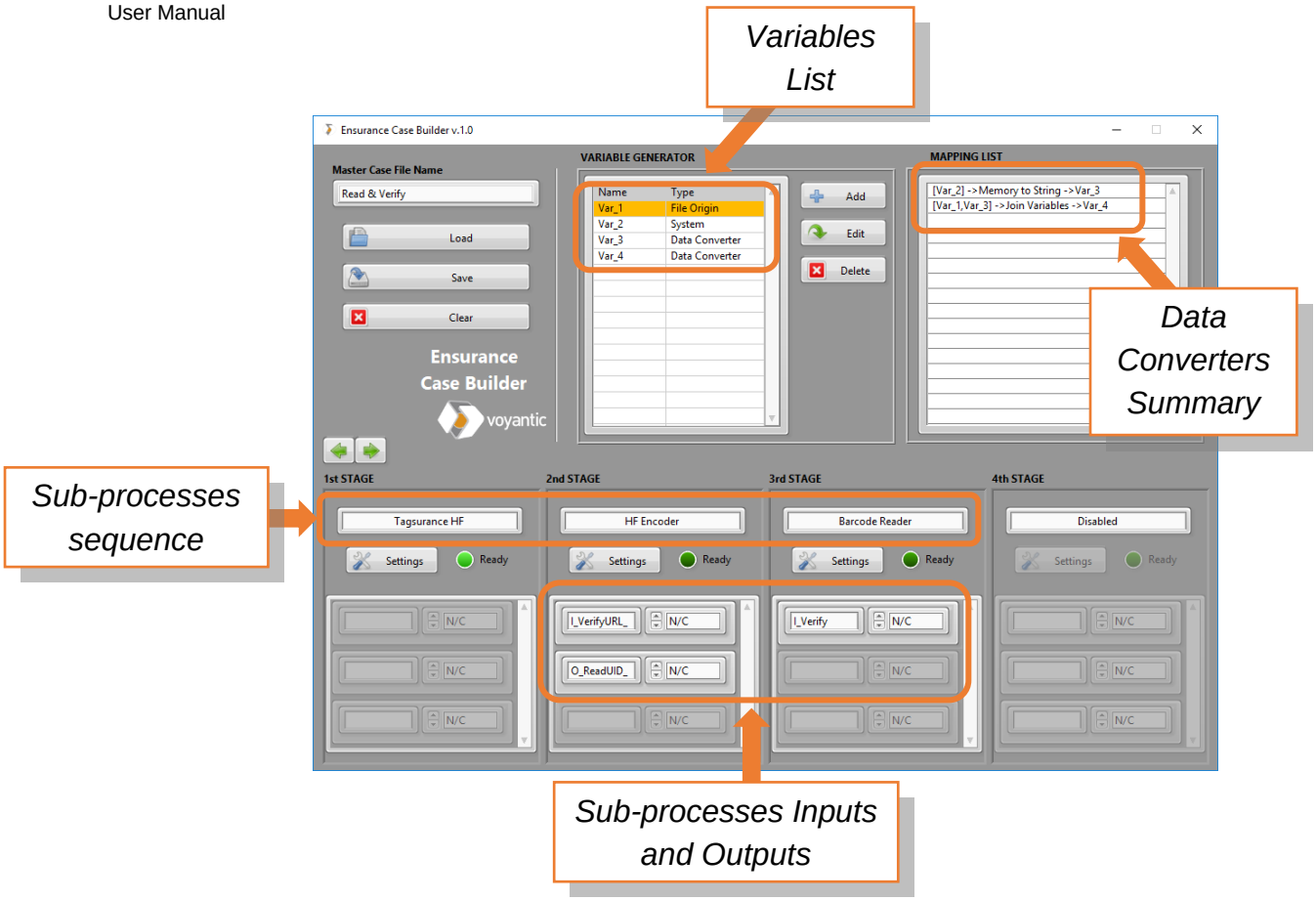

Figure 18: Example -> Variables List & configured sub-processes (not mapped)

The sub-processes Inputs and Outputs are not mapped yet (N/C), therefore the “Ready” led is not steady green.

The requested mapping for this example is the following:

1. **File Origin (Var 1)** -> URL Data
2. **System (Var 2)** -> UID Data
3. Encoder Read Output -> System (Var 2) -> **Memory to String (Var 3)**
4. File origin (Var 1), Memory to string (Var 3) -> **Join Variables (Var 4)** -> Code Reader Input

You connect the sub-processes Inputs and Outputs by choosing the correct variable through the correspondent drop-down menu.

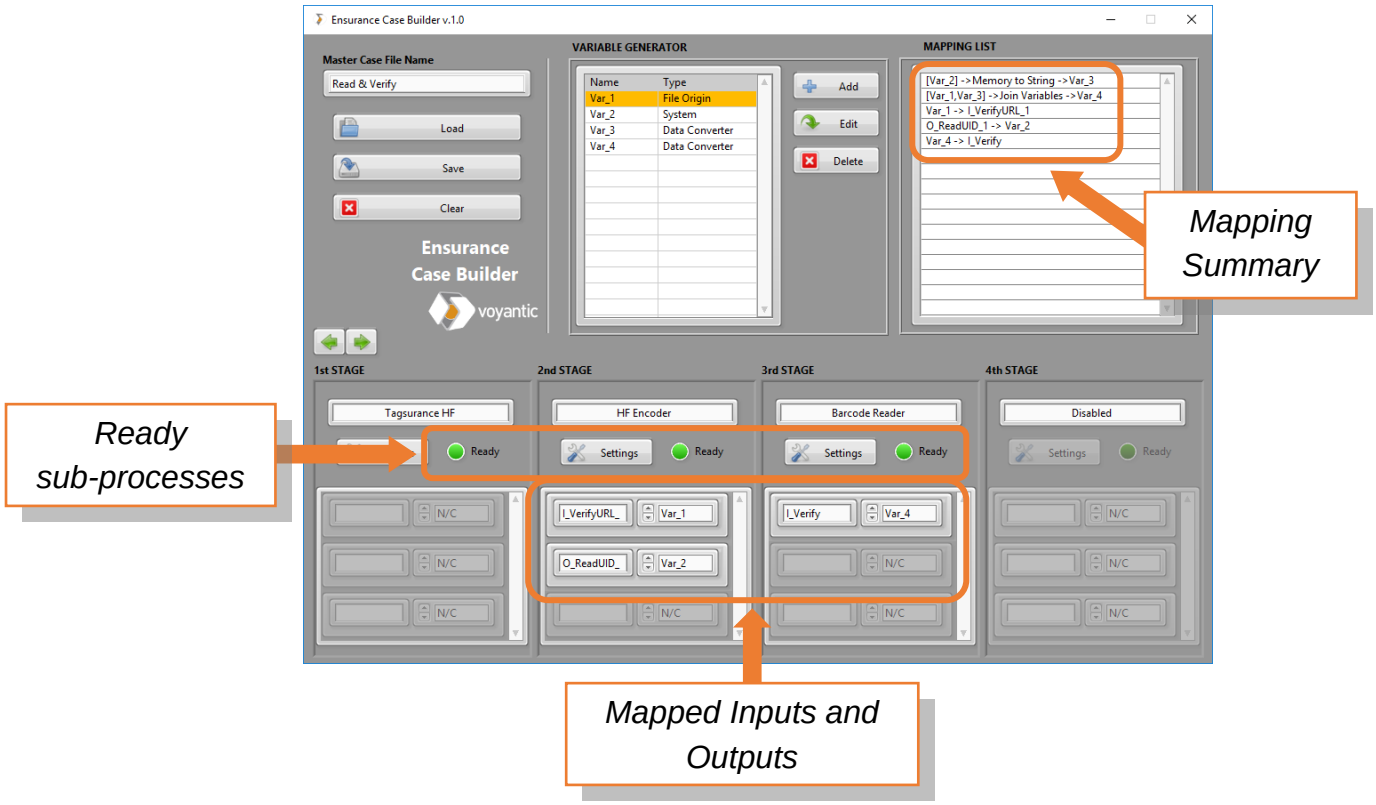

Figure 19: Example -> Mapped Inputs and Outputs

All the Ready LEDs are steady green as the sub-processes Inputs and Outputs are mapped. We are finally ready to generate the recipe files.

4.3.5 Customization possibilities

In case the available data converters don't cover all the test requirements, please contact Voyantic support for further investigation as it might be possible adding a custom converted.

4.4 Creating the recipe file(s)

We are finally approaching the final step which consists of creating the needed recipe files. If we keep in consideration the same example, we can insert a Master case file name on the GUI left-corner and press the Save button. If everything goes smoothly, you should see a confirmation pop-up appearing.

The Case Builder will automatically create a new folder (same name as the Master Case file name) underneath the /Data/Case files folder. Within the folder, all the necessary files are included (you can find a detailed description in the Chapter 5):

- Sub-Processes subfolders: a sub-folder (which includes the device configuration file and task file if exists) will be generated for each defined sub-process.
- "Variables" sub-folder: it includes the Data converters (Variable Data Conversion Definitions.ini), File Origin and Incremental (Variable Data Source Configuration.ini) variables definitions. It also includes the necessary CSV files used by the File Origin variables.
- Master Case file

In order to run the test recipe correctly, you need to maintain the same file structure underneath the created folder (Read & Verify in the example)

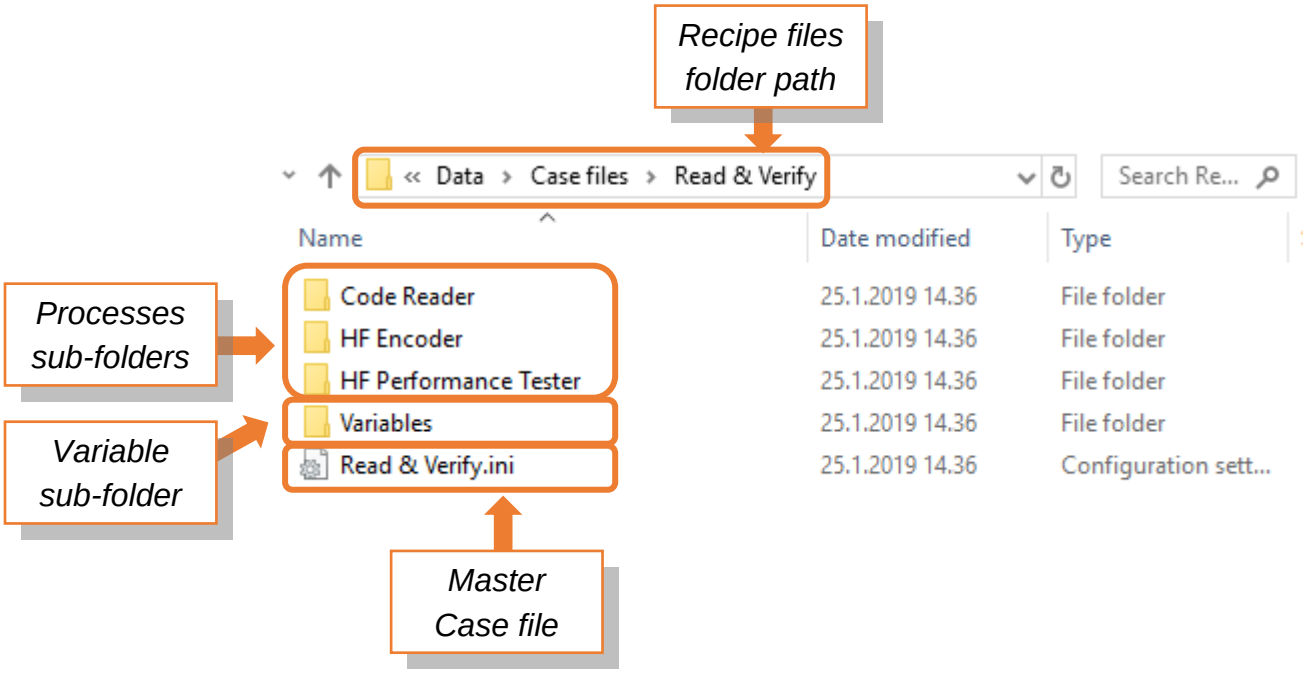

Figure 20: Test Recipe file hierarchy

5 The recipe files

5.1 Overview

5.2 The Master Case file

The master case file is used as a top-level definition of the whole system.

It is used to:

- define and configure all the stages that are to be used in the system
- define how source variables are to be generated/received to the system
- define how internally variables are to be converted/modified when moving between stages, sources and others

[Variables]

Definitions = "..\Variables\Variable Data Source Configuration.ini"

Variables definitions are used to define input variables to the system. This is a separate configuration file.

[Data Conversions]

Definitions = "..\Variables\Variable Data Conversion Definitions.ini"

Variables converters definitions are used to define conversion algorithms to the system. This is a separate configuration file.

[Stages]

Stages is a list used to define a user-friendly name for each stage **and** to define the order in which they are located in the system. For example, a stage at position 1 is physically located before a stage at position 2,3,4 and so on.
Note: *For each stage defined, there must be a corresponding section with that same name in the master case file.*

The following example is just to be used for reference, more or fewer stages can be defined.

1 = "Code Reader"

2 = "HF Performance Tester"

3 = "HF Encoder"

4 = "UHF Performance Tester"

5 = "UHF Encoder"

6 = "Printer"

Each stage defined in the [Stages]-section will follow a certain common pattern:

[stage name as defined in the stages-section]

OFFSET

Reserved for future use, use a value of 0

TYPE

This defines what the actual device the stage consists of. Valid values are:

Type	***Description***
TS_HF	*Tagsurance HF*
HF_Encoder	*HF Encoder*
TS_UHF	*Tagsurance UHF*
Rain_Encoder	*UHF Encoder*
Sick_Lector_631	*Sick Lector 631 image based code reader, 2D*
VideoJet_DataFlex_6420	*VideoJet DataFlex 6420 & 6530 thermal transfer printer*

[HF Performance Tester]

OFFSET = 0

TYPE = TS_HF

DeviceConfig = "..\HF Performance Tester\Devices.txt"
File path of Tagsurance HF "Devices"-file, generated by Tagsurance HF case-builder

TaskList = "..\HF Performance Tester\TS-HF test case.txt"
File path of Tagsurance HF "Test case"-file, generated by Tagsurance HF case-builder

[HF Encoder]

OFFSET = 0

TYPE = HF_Encoder

DeviceConfig = "..\HF Encoder\HF Encoder Device Config.ini"
File path to configuration file

TaskList = "..\HF Encoder\HF Encoding Tasks.ini"
File path to the encoding task list

[UHF Performance Tester]

OFFSET = 0

TYPE = TS_UHF

DeviceConfig = "..\UHF Performance Tester\Tagsurance UHF Device config.ini"
File path to Tagsurance UHF “device config”-file, generated by the Tagsurance UHF case-builder

TaskList = "..\UHF Performance Tester\Tagsurance UHF Case File.txt"
File path to Tagsurance UHF “task list”-file, generated by the Tagsurance UHF case-builder

[UHF Encoder]

OFFSET = 0

TYPE = Rain_Encoder

DeviceConfig = "..\UHF Encoder\UHF Encoder Device Config.ini"
File path of UHF Encoder configuration file

TaskList = "..\UHF Encoder\UHF Encoding Tasks.ini"
File path of UHF Encoder task list

[Code Reader]

OFFSET = 0

TYPE = Sick_Lector_631

DeviceConfig = "..\Code Reader\Settings.ini"
File path of Sick Lector 631 configuration file

[Printer]

OFFSET = 0

TYPE = VideoJet_DataFlex_6420

DeviceConfig = "..\Printer\VideoJet config.ini"

File path to the VideoJet DataFlex 6420 configuration file

5.3 Processes settings

5.3.1 UHF Encoder

5.3.1.1 Configuration file

[TCP]

IP [STRING]

Can also be the reader host name if LAN setup supports it e.g FX500802A10

PORT [U32]

TCP port number, unless otherwise specified use port number 1234

[RF]

TX PWR [FLOAT]

Transmitter power, in dBm. Parameter precision of 0.1 dBm.

[TRIGGER]

Source [U32]

0 → Software trigger

1 → Hardware trigger

GPIO [U32]

1 → Keep this value with Voyantic adapter

2 → for future purposes

Level [U32]

0 → Low level at pin is seen as a trigger

1 → High level at pin is seen as a trigger

Note: The trigger signal levels are defined at the UHF Encoder's GPIO connector. Once the IO-adapter is connected, it inverts the trigger signal that should be taken into account when creating the device configuration for the Ensurance RAIN Encoder.

[OUTPUT]

Invert BusyReady [TRUE/FALSE]

(Optional) Invert the polarity of the Busy/Ready pin output from the Zebra reader GPIO connector – Default FALSE, which means the high state is Busy

Invert PassFail [TRUE/FALSE]

(Optional) Invert the polarity of the Pass/Fail pin output from the Zebra reader GPIO connector – Default FALSE, which means the high state is Pass

5.3.1.2 Task file

[Configuration]

BlockWrite Words [U8]

(Optional parameter) Sets the number of maximum words to be included into each BlockWrite command sent to the tag. If more than that are defined in a BlockWrite task, the task will result in multiple BlockWrite commands being sent to the tag (Default value = 1)

RF Link [U8]

(Optional parameter) Allows for selection of different RF-link settings. See table below for details (Default value: 3)

RF Link	Tari [µs]	PIE	BLF [kHz]	M
0	**25**	**2**	**240**	**2**
1	**12,5**	**1,5**	**240**	**2**
2	**18,8**	**1,5**	**320**	**2**
3	**12,5**	**1,5**	**320**	**2**

[Inventory]

Select [U8]

Session [U8]

Target [U8]

Q [U8]

[Task N]

Every task is defined by a new section named Task N, where N is a unique number (incrementing, starting from 1)

The type of task is defined by the key “TaskType”, see below.

TaskType [String] = Any string of the following: Access, BlockWrite, Kill, Read, Write, Select

Common for all tasks except the Select task are the input and output variable keys.

***Note**: Even though all tasks can have variables assigned to them, only a few will actually make use of them. See each task to find out if a variable can be of used with it.*

Variable keys are:

Var In [U16]

Var Out [U16]

Here are the different tasks:

[Task N]

TaskType [String] = **Select**

Target [U8]

Action [U8]

Membank [U8]

Pointer [U16_HEX]

Length [U8]

Truncate [U8]

Bitmask [BITS]

The *Select* command will be processed before the inventory sequence even though it is defined “under” the inventory.

[Task N]

TaskType [String] = **Access**

Password [U32_HEX]

Variable In modifies Password

[Task N]

TaskType [String] = **Read**

Membank [U8]

Wordpointer [U16_HEX]

Wordcount [U8]

Variable Out generated from bytes read from the tag

[Task N]

TaskType [String] = **Write**

Membank [U8]

Wordpointer [U16_HEX]

DataWord [U16_HEX]

Variable In modifies DataWord

[Task N]

TaskType [String] = **BlockWrite**

Membank [U8]

Wordpointer [U16_HEX]

DataWords [U16_HEX_ARRAY] *Variable In modifies DataWords*

[Task N]

TaskType [String] = **Lock**

MaskAction [U32]

[Task N]

TaskType [String] = **Kill**

Password [U32_HEX]

[Task N]

TaskType [String] = **VerifyRead**

Membank [U8]

Wordpointer [U16_HEX]

DataWords [U16_HEX_ARRAY] *Variable In modifies DataWords*

5.3.2 UHF Performance Tester

5.3.2.1 Configuration file

[Device config]

COM PORT [STRING]

Serial com port number (e.g. COM12)

Serial protection [STRING]

Setup if any serial communication protection is to be used

- *None*
- *Checksum*
- *Framesync*
- *Chekcsum & framesync*

Carrier before ms [FLOAT]

Set the carrier before time (milliseconds)

Trigger source [STRING]

- *Internal*
- *External, rising edge*
- *External, falling edge*

Pass fail mode [STRING]

- *Static*
- *Static, inverted*
- *Pulse*
- *Pulse, inverted*

[Read N]

Var Out [U16]

We define output variables here. Note that output variables can only be generated from read commands at the moment. The "N" stands for the N:th read command in the TS-UHF case file. Indexing starts at 1. For example, to produce output data variable 5 on the second read command we would create section "[Read 2]" containing "Var out [U16] = 5".

5.3.2.2 Task file

Please refer to Tagsurance UHF manual in order to know the task file details

5.3.3 HF Encoder

5.3.3.1 Configuration file

[Reader Plugin]

ReaderPlugin — *Reader Plugin file name OR file path*

[COMS]

Port [STRING]

Timeout [I32]

[IO]

Busy/Ready Invert [TRUE/FALSE]

Pass/Fail Invert [TRUE/FALSE]

Trigger Timeout ms [I32] — *Setting this to -1 will allow for infinite trigger waiting*

Trigger Mode [U8]

- 0 → A high pin state is seen as a trigger
- 1 → A low pin state is seen as a trigger
- 2 → A rising edge on pin is seen as a trigger
- 3 → A falling edge on pin is seen as a trigger

5.3.3.2 Task file

Chip Specific Plugin Configuration

[IC Type]

Plugin [PATH] — *Chip Plugin file name OR file path*

Common configurations for all tasks

Var In [U16]

Var Out [U16]

Skip if previous fail? [TRUE/FALSE]

NTAG203 Task list

[Task N]

Task Type [String] = **Activate**

[Task N]

Task Type [String] = **Carrier**

Carrier on? [TRUE/FALSE]

[Task N]

Task Type [String] = **Read**

Page [U8_HEX]

[Task N]

Task Type [String] = **Write**

Page [U8_HEX]

Data [U8_HEX_ARRAY] *Size = 4*

Variable In modifies Data bytes

[Task N]

Task Type [String] = **Lock**

Lock pages [U8_HEX_ARRAY]

Blocking pages [U8_ HEX _ARRAY]

NTAG210 Task list

NTAG210 supports *Activate, Carrier, Read* and *Write* tasks listed previously, plus *Authenticate.*

[Task N]

Task Type [String] = **Authenticate**

Password [U8_HEX_ARRAY] *Size = 4*

Variable In modifies Password

NTAG213 Task list

NTAG210 supports *Activate, Carrier, Read, Write, Lock* and *Authenticate* tasks listed previously, plus *NFC URL and NFC Text.*

[Task N]

Task Type [String] = **NFC URL**

URL [STRING]

String representing the actual web address. “http://www.voyantic.com” for example.
Will use UTF-8 encoding scheme.

UID Mirror [TRUE/FALSE]

This activates a function in the NTAG213 chip which will append the UID of the chip to the end of the web address and after a prefix if that is specified.

Counter Mirror [TRUE/FALSE]

Similar to UID Mirror but this will append a “read counter” value to the end of the URL after a prefix if that is specified.

Mirror Prefix [STRING]

A mirror prefix that will be added at the end of the URL and before the UID/Counter Mirror. This can be any free string.

Identifier Code [I16]

The identifier code as defined by NFC Forum URI Record Type Definition. It can be used to shorten the length of a URL by replacing the protocol field of the URL. A reader will use this code to prepend the protocol field to the URL it reads from the tag.
Setting Identifier Code = 0 will disable the usage of this functionality (this is the default value if it’s left undefined in the configuration file)
A few codes and their associated protocol fields are:
Identifier Code 0 = Not applicable, no protocol prepending used
Identifier Code 1 = “http://www.”
Identifier Code 2 = “https://www.”
Identifier Code 3 = “http://”
Identifier Code 4 = “https://”

Note: In a case where both UID Mirror and Counter Mirror are activated at the same time, the UID Mirror will be put in front of the UID Counter and they will be separated by a “x”.

[Task N]

Task Type [String] = **NFC Text**

Text [STRING]

String containing the text to be written. “This is a Voyantic Encoder text” for example. Will use UTF-8 encoding scheme.

Language Code [String]

Language code according to RFC 5646. For example, “fi” for Finnish, “de” for German, “fr” for French, etc.

Mirror Prefix [STRING]

A mirror prefix that will be added at the end of the text and before the UID Mirror. Not used if UID Mirror is not active.
This can be any free string.

UID Mirror [TRUE/FALSE]

This activates a function in the NTAG213 chip which will append the UID of the chip to the end of the text and after a prefix if that is specified.

Counter Mirror [TRUE/FALSE]

Similar to UID Mirror but this will append a “read counter” value to the end of the URL after a prefix if that is specified.

Variable In modifies Text

Example 1: URL with NTAG213 mirror functionality enabled

Setting URL to “http://www.voyantic.com/” and activating both UID and Counter Mirrors. We set the prefix to something useful, let’s say “Mirrors Here:”.

The resulting URL will be (when read by an NFC reader):
http://www.voyantic.com/MirrorsHere:123456789101213x0000001

Example 2: URL using the Identifier Code to reduce the memory footprint of the URL string in tag memory

Setting URL to “voyantic.com/”
Identifier Code shall be set to 1, meaning that we have removed the “http://www.”-protocol field from our URL and that we want the NFC reader to automatically prepend “http://www.” to our URL.

The resulting URL will be (when read by an NFC reader):
“http://www.voyantic.com/”

SLIX2 Task list

[Task N]

Task Type [String] = **Activate**

[Task N]

Task Type [String] = **Carrier**

Carrier on? [TRUE/FALSE]

[Task N]

Task Type [String] = **Read**

Block [U8_HEX]

[Task N]

Task Type [String] = **Write Single Block**

Block [U8_HEX]

Data [U8_HEX_ARRAY] *Size = 4*

Variable In modifies Data

[Task N]

Task Type [String] = **Verify Read**

Block [U8_HEX]

Data [U8_HEX_ARRAY] *Size = 4, data to check against tag memory block*

Variable In modifies Data

[Task N]

Task Type [String] = **NFC URL**

URL [STRING]

String representing the actual web address. “http://www.voyantic.com” for example.
Will use UTF-8 encoding scheme.

UID Mirror [TRUE/FALSE]

This emulates the UID Mirror functionality of NTAG213 chip which will append the UID of the chip to the end of the web address and after a prefix if that is specified.

Mirror Prefix [STRING]

A mirror prefix that will be added at the end of the URL and before the UID Mirror.
This can be any free string.

Identifier Code [I16]

The identifier code as defined by NFC Forum URI Record Type Definition. It can be used to shorten the length of a URL by replacing the protocol field of the URL. A reader will use this code to prepend the protocol field to the URL it reads from the tag.
Setting Identifier Code = 0 will disable the usage of this functionality (this is the default value if it’s left undefined in the configuration file)
A few codes and their associated protocol fields are:
Identifier Code 0 = Not applicable, no protocol prepending used
Identifier Code 1 = “http://www.”
Identifier Code 2 = “https://www.”
Identifier Code 3 = “http://”
Identifier Code 4 = “https://”

Perma Lock [TRUE/FALSE]

Activate/deactivate permanent locking of the specific memory area that contain the NFC data. Tag must support the 15693 optional command “Lock Block”.

Variable In modifies URL

[Task N]

Task Type [String] = **NFC Text**

Text [STRING]

String containing the text to be written. “This is a Voyantic Encoder text” for example. Will use UTF-8 encoding scheme.

Language Code [String]

Language code according to RFC 5646. For example, “fi” for Finnish, “de” for German, “fr” for French, etc.

UID Mirror [TRUE/FALSE]

This emulates the UID Mirror functionality of an NTAG213 chip which will append the UID of the chip to the end of the text and after a prefix if that is specified.

Mirror Prefix [STRING]

A mirror prefix that will be added at the end of the text and before the UID Mirror. Not used if UID Mirror is not active.
This can be any free string.

Postfix [STRING]

A postfix that will be added at the end of the text and UID Mirror.
This can be any free string. If UID Mirror is not active the postfix will be added to the end of the text instead.

Perma Lock [TRUE/FALSE]

Activate/deactivate permanent locking of the specific memory area that contain the NFC data. Tag must support the 15693 optional command “Lock Block”.

Variable In modifies Text

[Task N]

Task Type [String] = **Verify NFC URL**

URL [STRING]

String representing the URL/web address to verify against. “http://www.voyantic.com” for example.

UID Mirror [TRUE/FALSE]

Activates/deactivates if UID mirror functionality is present in the URL.

Mirror Prefix [STRING]

The mirror prefix to be used if UID Mirror is activated when verifying the URL contents.

Identifier Code [I16]

The identifier code, as defined by NFC Forum URI Record Type Definition, to be checked in the URL.

Variable In modifies URL to be verified

[Task N]

Task Type [String] = **Verify NFC Text**

Text [STRING]

String containing the text to be verified. “This is a Voyantic Encoder text” for example.

Language Code [String]

Language code to verify according to RFC 5646. For example, “fi” for Finnish, “de” for German, “fr” for French, etc.

UID Mirror [TRUE/FALSE]

This allows for verification of an NFC Text where UID Mirror was used in encoding. See SLIX2 NFC Text task description for more info.

Mirror Prefix [STRING]

This allows for verification of an NFC Text where a Mirror Prefix was used. See SLIX2 NFC Text task description for more info.

Postfix [STRING]

This allows for verification of an NFC Text where a Postfix was used. See SLIX2 NFC Text task description for more info.

Variable In modifies Text to be verified

5.3.4 HF Performance Tester

5.3.4.1 Configuration file

Please refer to Tagsurance 2 user manual

5.3.4.2 Task file

Please refer to Tagsurance 2 user manual

5.3.5 Printer

5.3.5.1 Configuration file

[COM]

Port [String]

Serial port number (e.g. COM12)

Baud [U32]

Baud rate used for serial communication, set to 115200 unless otherwise specified

Timeout [I32]

Serial communication timeout value in milliseconds, set to 1000 unless specified

[TRIGGER]

Source [U8]

- 0 → SW trigger
- 1 → HW trigger

[JOBS]

Var In [U16]

Bad Tag Marker [STRING] (optional)

Define bad tag marker job (if the key exists -> conditionality)

JobN Name [STRING]

N in JobN is read as Job ID

Var In changes the active Job ID according to the first byte in the variable data.

[DataN]

Var In [U16]

Field Name [STRING]

Name of the field in the print-job that is to be set by this data. Note: Make sure when creating a print job (through the VideoJet software suite) that the field names used match the field names in this file.

Line Wrap Width [U32]

(Optional) Can be used to insert newline-characters into the variable before it's sent to the printer. Leaving the undefined disables the newline-insertion-functionality.

Type [U8]

0 → U32 Hex (Variable data is converted MSB first)
1 → U32 Decimal (Variable data is converted MSB first)
2 → Text (ASCII according to
http://zone.ni.com/reference/en-XX/help/371361L-01/glang/ascii_codes/)

Var In sets the Field data belonging to Field Name to a HEX string representation of the variable data bytes.
E.g. if Field Name [STRING] = "var00" and Type [U8] = 1 receiving variable data [1;2;5;10] would send "var00=A050201" to the printer before next print is issued.

For example, if Type is set to 3, receiving variable data [86;111;121;97;110;116;105;99] would send "var00=Voyantic" tot the printer.

5.3.6 Code Reader

5.3.6.1 Configuration file

[TCP]

IP [STRING]

Port [U16]

TCP Port, unless otherwise specified use 2111

Packet Size [I16]

TCP packet sizes, unless otherwise specified use 1024

Packet Timeout [I16]

Timeout in ms, unless otherwise specified use 500

[Camera Settings]

Parameters [PATH]

File path to the Sick Lector configuration tool-generated parameters-file

[OUTPUT PARSING]

No Read Token [STRING]

Unless camera is specifically configured to use another output string when code-reading fails use "NoRead" as default.

Variable Separator [STRING]

This is a string which separates the output data from a barcode read. The camera can (if configured to) output separate data from one single barcode, so in that case we define a string that is used as a separator between the data. This string needs to match the camera setting. Unless configured to something else in the camera use "+" as default.

[DATAN]

Set N for each data section to a unique number. This is used to separate the sections from each other (two identically named sections are not allowed in the same file).

Var Out [U16]

Variable Out number associated with the data

Camera Var Name [STRING]

This string needs to match one of the output data in the camera parameter file. More precisely it needs to exist in the "OFtplstr1" parameter in the file.
Example: *If we are outputting the barcode content, named "BC" in the parameter file, and we want to send this data to another process in Ensurance, we need to assign an output variable to it by setting for example:*

[DATA1]
Var Out [U16] = 5
Camera Var Name [STRING] = "BC"

The above will output the barcode as variable number 5.

[VERIFICATION]

Var In [U16]

Barcode Content [STRING]

Set to the expected barcode content that the camera will attempt to verify.

Wildcard single [STRING]

Define a string that will be used to identify a single wildcard character in Barcode Content-string (default value: ?).

Wildcard multiple [STRING]

*Define a string that will be used to identify multiple wildcard characters in Barcode Content-string. For now, the number of wildcard characters cannot be changed, so any number of characters will be accepted in its place (default value: *)*

Variable In modifies the Barcode Content.

Example usage:

Barcode Content [STRING] = "?www.voyantic.com"
Wildcard single [STRING] = ""*
Wildcard multiple [STRING] = "?"

*The above setup will verify and **accept** a barcode content that reads as:* www.voyantic.com

and also accept: http://www.voyantic.com

Since we are using the "?" as a wildcard for multiple characters.

*But will **not** accept a barcode if it reads as:* www.voyantic.com/products

5.4 Variable Data Conversions

Each section in a variable converter setup file is for a separate variable converter. Hence, one file can define multiple converters at once.

Common for all converters is the type definition and variable in/out definitions.

Note that there can be multiple input variables, so that's why they're defined as an array.

- Var In [U16_ARRAY]
- Var Out [U16]
- Type [STRING]

The actual type of converter is determined by the Type string

5.4.1 String to memory

This converter will use the first Var In in the array and ignores the rest.

This converter will treat the input data array as a character array, convert it to a string representation, read the string as if it is a hexadecimal string and output it as a U8 decimal array.

Type = "String to memory"

Example of data conversion:

***Input data** = [49;50;51;52;53;54]*

Converted to string = "123456"

Interpreted as hex array of bytes [0x12,0x34,0x56]

***Output data**= [18;52;86] (decimal representation)*

5.4.2 Memory to string

This converter will use the first Var In in the array and ignores the rest.

This converter will treat the input data array as a byte array, convert bytes to characters and combine them to one string. Then convert the string to byte array where one byte represents one character.

Type = "Memory to string"

Example:

***Input data** = [18;52;86] (decimal representation)*

As hexadecimal representation=[0x12;0x34;0x56]

Combined characters to one string = "123456"

Converted to byte array = [49;50;51;52;53;54]

***Output data**= [49;50;51;52;53;54]*

5.4.3 Zero Padding

This converter uses the first Var In in the array and ignores the rest.

Type = "Fill_Zeros"

Filling end [STRING]

- "End" *Append zeros to the end of the Var In data*
- "Beginning" *Append zeros to the beginning of the Var In data*

Target length [U32]

Sets the length of the data after padding, sent out as Var Out.

5.4.4 Join Variables

This converter will combine the all input variables defined in Var In array. The variables are combined in the same order they are listed in the array.

Type = "Join_variables"

Example:

Type = "Join_variables"
Var Out [U16] = 4
Var In [U16_Array] = [3;1;2] *Where 1 = [0x01;0xA1], 2 =[0x 2B;0x01], 3 =[0x FF]*

As a result, the output variable "4" = [0xFF;0x01;0xA1;0x2B;0x01]

5.4.5 GS1 Electronic Product Code

These converters will create an Electronic Product Code in binary format that can be written into the *EPCglobal UHF Class 1 Gen 2* tag memory. The conversions are made according to the *GS1 EPC Tag Data Standard*.

5.4.5.1 GS1 EPC GIAI-96

Global Individual Asset Identifier (GIAI) consists of two elements: *GS1 Company Prefix* and *Individual Asset Reference*. In addition to these, the *Filter* value is required for the binary coding.

Keys for the GS1 EPC GIAI-96 converter:

Type = “GS1-EPC-GIAI-96”

Company Prefix [STRING]

Company Prefix variable [U16]
(Optional) Define the input variable with which the Company Prefix default value will be overwritten.

Asset Reference [STRING]

Asset Reference variable [U16]
(Optional) Define the input variable with which the Asset Reference default value will be overwritten.

Filter [U8]

Note: Although the Company Prefix and Asset Reference consist of numbers only, the format of the parameters is STRING.

Example:

[Conversion D]
Type [String] = "GS1-EPC-GIAI-96"
Var In [U16_ARRAY] = [7;8]
Var Out [U16] = 9
Filter [U8] = 0
Company Prefix [String] = "6400001"
Company Prefix variable [U16] = 7
Asset Reference [String] = "40000000000000000"
Asset Reference variable [U16]= 8

Based on the conversion default values, the result is (HEX) 341586A0048E1BC9BF040000, which can be written to the RFID Tag EPC Memory Bank (starting at bit 20h)

5.5 Variable Data Sources

The variable generators produce the variables which are originating from somewhere else than the sub-processes' results. The Variable Generators are defined in a variable generator setup file, where is one section for each Variable Generator.

5.5.1 CSV File Reader

The File Reader reads variable data from file CSV file where the separator is ";". The file can contain data for several variables. In the file, one row contains data for one tag, and each column contains data for one variable. Data is read in order, so that the data in the first row (by default) will be addressed to the first personalized tag.

Keys for the File Reader Variable Generator:

- Type [String] = "CSV_file_reader"
- File_Path [PATH] = "C:\Ensurance\Data\Case Files\Example\variable_data_source_file.csv"
- Row_Offset [U16] = 0 (Optional, default = 0)
- Row_Step [U16] = 1 (Optional, default = 1)
- Var Out [I16_ARRAY] = [-1;1;2;3]
- Variable_data_formats [U8_ARRAY] = [0;0;1;2]

Variable_ID_order defines to which variable the columns will be connected. With "-1", the column can be ignored, i.e. the data in that column won't be connected to any variable.

Variable_data_format defines the data format in the file: 0 = HEX, 1 = Number(U32), 2 = String.

Example:

[Variable Data File]
Type [STRING] = "CSV_file_reader"
File_Path [PATH] = "..\variable_data_source_file.csv"
Var Out [I16_ARRAY] = [1;2]
Variable_data_formats [U8_ARRAY] = [0;2]

Data lengths of different formats:

HEX: minimum data length is 8 bits (1 byte)	*(Hex) B = 0x0Bh, (Hex) 13B = 0x013Bh*
Number (U32): length is always 32 bits	*(Dec) 7 = 0x00000007h*
String: one Char represents 8 bits	*(String) "7" = 0x37h*

5.5.2 Incremental Value

The Incremental Value variable generator produces unique variable data for each tag starting from given initial value, which will be incremented with defined step from tag to tag.

The variable data format can be chosen to be either a number or a string.
Number “5” = 0x05, and string “5” = 0x35

In case the format is string, the minimum length for the value can be set, e.g. value 5 will be presented as “0005”. Also the prefix can be defined for the string format value. The prefix is not calculated as part of minimum length: value 5 with minimum length of 4 and prefix of “SKU6645” cause result “SKU66450005”.

Keys for Incremental Value Variable Generator:

- Type [STRING] = “Incremental_value”
- Var Out [U16] = 4
- Data_format [U8] = 2
- Initial_value [U32] = 0
- Step_size [U8] = 1 *(optional, default = 1)*
- Minimum_length [U8] = 0 *(optional, default = 0)*
- String_prefix [STRING] = “Prefix” *(optional, default = empty)*

Formats: 1 = number (U32), 2 = string

Example:

[Incremental value]
Type [STRING] = “Incremental_value”
Var Out [U16] = 4
Data_format [U8] = 2
Initial_value [U32] = 0
Step_size [U8] = 1
Minimum_length [U8] = 0
String_prefix [STRING] = "Prefix"

The Minimum_length defines the number of bytes for the data. In case of a number,the leading zero bytes are added, and in case of a string, the leading zeros characters are added (1 char = 8 bits = 1 byte)
As an example, when the Minimum_length = 4

- *number “7” = 0x00000007h*
- *number #377” = 0x00000179h*
- *string “7” -> string “0007” = 0x30303037h*

5.5.3 Fixed Value

The Fixed Value variable generator produces the same static variable data for each tag. Data can be defined either as Byte Array, or by using HEX, DEC (U32), or String formats.

Keys and example values for Fixed Value Variable Generator:

- Type [STRING] = “Fixed_value”
- Var Out [U16] = 5
- one of following:
  - Data [U8_HEX_ARRAY] = [0A;B1;2C;D3]
  - Value [U8_HEX] = F5
  - Value [U32] = 27
  - Value [STRING] = “FreeText”

