STX Driver Description

Table of Content

1 Legacy Command Set

The Legacy Command Set is a one-to-one portation the Liconic ActiveX Library. The intention of the Legacy Command Set is to ensure backward compatibility to users who are familiar with these commands. However it is strongly recommended to not use this command set for new developments. Use the New Command Set instead. All this command you can find in STXCommands.Java file. These commands are commented.

2 New Command Set

The New Command Set was developed to ease integration of Liconic products and to provide maximum functionality to the end-user. The New Command Set has been reduced to 26 very powerful commands. These commands are straight forward and will satisfy all needs for common integration. In order to continuously improve our products we are interested in receiving your feed-back about this driver and/or inputs for improvements.

2.1 Communication Using New Command Set

In an integration of Liconic STX instruments each instrument is considered as and a unit within a system. Such system consists of one or more units. All communication is handled by two TCP/IP Ports. In addition information between the User Software and the STX Driver is shared by STX Files. SXT files are strictly text based. The new STX Driver therefore provides a most universal interface allowing simple integration under all modern platforms.
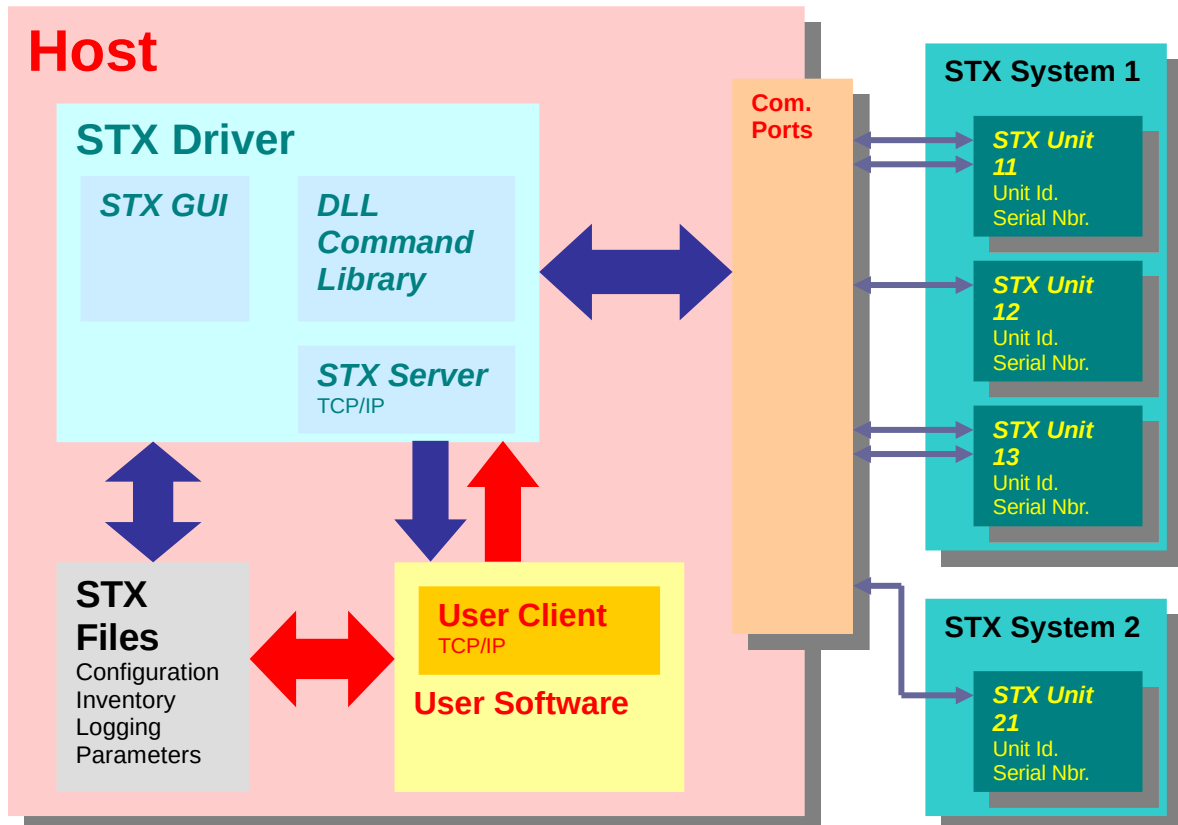
The above picture gives an overview the elements used for an integration of STX instruments. The user software is the integrators software that may control additional equipment such as a robot or may include a data base interface. In this application the user software is located on the same computer as the STX Drivers and the comports to the STX units.

All communication between the STX elements and the user software is channelled through

- TCP/IP command port
- Common STX Files

Hardware related functions, service or maintenance command are performed through the STX GUI which is also part of the STX driver. The STX GUI may give access to the user.
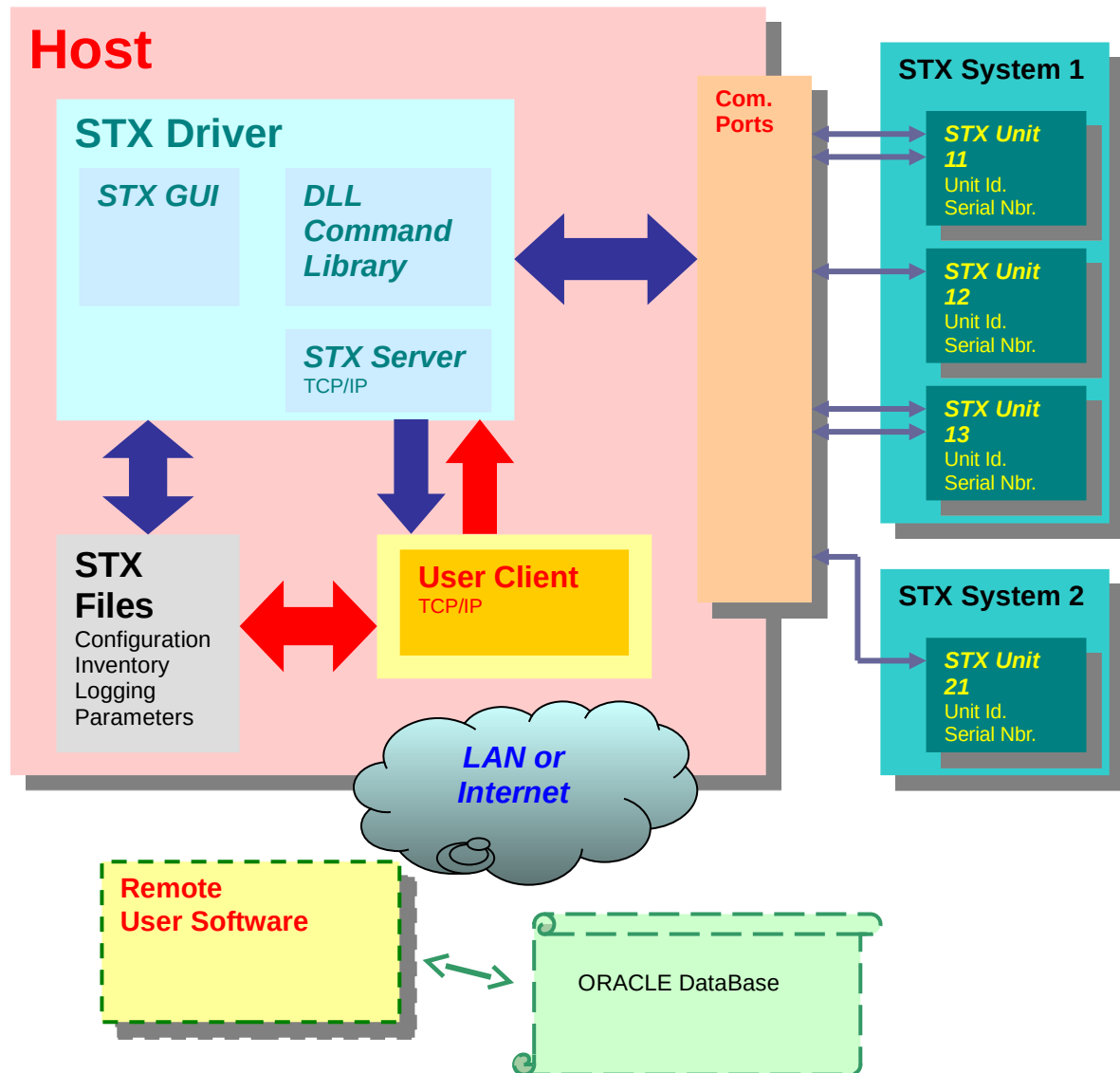
The picture above shows that operation by remotely located software is also possible through a LAN or an Internet connection.

2.2 Driver Architecture

The STX Driver includes the following main elements

- STX DLL Command Library
- STX Server
- STX GUI

2.3 STX Driver File System

The File System includes the files used by the STX driver. There are the following files discussed below

- STX Driver Set-up File
- STX Configuration Files
- STX Inventory Files
- STX Parameter Files

2.3.1 STX Driver Set-up File

```
[TCP]
port=3333
eventPort=3334

[paths]
StxMainFolder=

[mainFramePlacement]
max=0
left=456
right=1212
top=14
bottom=1161
```

2.3.2 STX Configuration Files

The STX configuration files include the following files

- STX System Files
- STX Unit Files

2.3.2.1 STX System Files

The STX System Files list the units which are members of a system. Units are grouped to a system if the members of the system are depending on each other. A system may also include one unit only, Note that a STX unit is always apart of a system

The example below shows a cascaded system consisting of a Base Unit, an Interstore Handling (Cascader) and an Extension Unit.

```
[system]
SystemName=Storage

[Unit]
Unit1=Unit1.lni
Unit2=Unit2.lni
Unit3=Unit3.lni
```

In the STX System File the files of the STX units within the system are listed. In the above example the STX Unit Files names are "unit1.lni".. unit3.lni.

2.3.2.2 STX Unit Files

The STX Unit Files list configuration properties for each unit.

```
[unit]
UnitComPort=1
UnitBCRPort=3
UnitName=AnyName
UnitId=AnyId1

[Settings]
AutoInventory=1
AutoAccessTimeOut=0
PlateTrace=1

[Climate]
climateTemperature=-20.0
climateHumidiy=0.0
ClimateCo2=0.0
ClimateN2=0.0
ClimateO2=0.0


[CassettesConfiguration]
UseCassConfTable=1
1-20=22,788
21-40=28,788

[Partitions]
A=1-20
B=21-40
```

2.3.2.3 Cassetts configuration table

For using different cassettes within one unit it's possible to add new section "CassettesConfiguration" in Unit conf file.

This section consists of such parameters:

1) "UseCassConfTable"

Sets whether Cassettes Configuration Table will used for plates operation.

UseCassConfTable=1 - Use Cassettes Configuration Table

UseCassConfTable=0 - Don't use Cassettes Configuration Table

2) List of Cassettes Configuration (Cassettes Configuration Table)

Each key is an ID of Cassette or range of Cassettes. (e.g. 1 or 1-5)

Value for the key is number of Levels and handler Z-Pitch value which depends on height of plate. The values separated by coma.

e.g.

```
1-5=22,788
6=4,3769
7=10,1713
```

The first line: five Cassettes (from 1 to 5) which have 22 levels for 17-mm plates (MPT). The second line: one Cassette ID=6 has 4 levels for 103-mm plate. The third line: One Cassette ID=7 has 10 levels for 42 mm plate (DWP).

Please use follow lookup table for define Z-Pitch value:

Plate mm	Z-Pitch
17	788
42	1713
11	82
21	959
26	1131
66	2467
103	3769
5	377
15	719
57	2158
53	2020
24	1028

Example.

```
[CassettesConfiguration]

UseCassConfTable=1
1-5=22,788
6=4,3769
7=10,1713
```

2.3.2.4 Partitions configuration table

Partition configuration table is stored in section: partitions.

Each key of this section is a name of a partion the values of this key are number of cassette or a range of cassettes which form to this partition.

e.g

```
[Partitions]
A=1-20
B=21-39
SafeZone=40
```

The partition A consists of 20 cassettes (from 1 to 20), partition B consists of 19 cassettes (from 21 to 39), partition SafeZone has only one cassette - 40.

2.3.3 STX Inventory File

The STX Inventory file holds information about the content of an STX system. The STX Inventory File is a text format file where one line contains the information for one location. The number of lines in the STX Inventory File is therefore equal to the number of locations.

The properties of each column are described below. For customer use there are

• BcrReading

- IdCustomer
- Partition
- ppSensor

The remaining columns are internally used by the StxDriver. In most applications there is no need to make use of these entries by the customer software.

2.3.3.1 BcrReading

The BcrReading column contains BarCode Readings read by the internal barcode reader from the labware located inside a unit. When addressing a plate for transport the BcrReading may be used for identification of the desired plate. The CbrReading fields are filled by the STX unit. Do not write into this column. The BcrReading field is updated by the STX Driver after

- Change of plate location using STX MovePlate command
- Inventory Function
- Retrieving a plate from the unit

The BcrReading field is cleared when a plate is removed from the unit or when the ppSensor does not detect a plate during inventory.

The Stx Driver does not check for unique identification of plates. In cases of multiple identification the first plate found in the list is taken for access.

2.3.3.2 IdCustomer

Plates may be addressed by BcrReading or IdCustomer for transport operation. In cases where no barcode is used or available the user may enter any identification tag when loading a plate using PlateMove command. The IdCustomer field accepts ASCII text format characters. The STX driver stores the IdCustomer into the IdCutomer fields. This Identification may later be used for addressing or identifying a plate. The IdCustomer fields may be understood virtual Barcode Labels assigned to a plate at the time of entry into the system.

The IdCustomer field is updated by the STX Driver

- On entry of a plate into a unit
- After change of plate location using STX MovePlate command
- When ppSensor does not detect plate during Inventory

The IdCustomer field is cleared by the STX driver when a plate is removed from the unit or when no plate is detected by the ppSensor during Inventory.

The Stx Driver does not check for unique identification of plates. In cases of multiple identification the first plate found in the list is taken for access.

2.3.3.3 Partition

Each stacker (slot, row) may be assigned to a Partition. The Partition field contain the Partition information a plate belongs to. Partitions are defined in the StxUnit files. The Partition a plate is stored is specified by the MovePlate command.

2.3.3.4 PpSensor

The ppSensor fields contain the reading results of the ppSensor during Inventory. The ppSensor fields are updated by the StxDriver after performing an Inventory function.

2.3.3.5 IdLic

IdLic is reserved for internal use by the StxDriver

2.3.3.6 IdSys

The IdSys field contains the System Identification a plate is physically located in.

2.3.3.7 IdUnit

The IdSys field contains the UnitIdentification a plate is physically located in.

2.3.3.8 Slot

The IdSys field contains the Slot number a plate is physically located in.

2.3.3.9 Level

The IdSys field contains the Level number a plate is physically located in.

2.3.3.10 Row

The IdSys field contains the Row number a plate is physically located in.

3 List of new Commands

Each command is a set of characters, consisting of name and parameters. All command have at least one parameter, that is ID of device and ended with Carriage Return ASCII 0Dh (CR).

The ID of each device is assigned in a configuration file (Section - "Unit", Key - "UnitId"). All parameters are enclosing in brackets and separated by coma. Each command is prompted by a Response string. Response is an ASCII string sent by device.

In case of syntax error the SoreX TCP/IP Server returns "E1", "E2" or "E3"values.

"E1"+CR - invalid command or command was sent without CR.

"E2"+CR - Invalid ID of a device configuration files was not loaded and Server can't recognise any ID

"E3"+CR - Invalid parameter or error of parsing a command (type cast error etc.).

Each response is ended with (each response consists of) two characters: Carriage Return ASCII 0Dh (CR) + Line Feed ASCII 0Ah (LF).

3.1 Main Commands

3.1.1 STX2Activate

"STX2Activate(ID)" – Opens Serial Communication and Initialises the StoreX (opens the PLC connection, initialises the handling, reads StoreX system constants).

Parameter:

ID – Identifier of a device.

Return values: the reply consists of one or two characters (separated by semicolon) and CR+LF.

The first character is the device initialisation state:

1 - Communication is opened and device is initialised.

-1 - Error of opening Serial Port.

-2 - Error of opening Serial Port (serial Port is already opened).

-3 - No communication.

-4 - Communication Error.

-5 - System Error (System Error Flag is true).

-6 - User Door is opened (or cannot read User Door status).

-7 – User Door is unlocked or cannot lock a User Door (available if door lock option present).

The second character depends on Barcode Reader presence and shows it initialisation state :

1 - BCR Serial Port is successfully opened.

-1 - Error of opening BCR port.

-2 - Wrong value of BCR port.

3.1.2 STX2Deactivate

"STX2Deactivate(ID)" - Closes serial communication through the active Serial Port. This function also closes Serial communication for Barcode Reader.

Parameter: ID – Identifier of a device.

Return values: CR+LF.

3.1.3 STX2Reset

"STX2Reset(ID)" - Reset the StoreX after the error. Puts the StoreX in the idle state. The user should call the STX2Reset method after any error of the machine. The user should call STX2Activate again to continue operations, or press manually the "Reset" button of the machine.

Parameter: ID – Identifier of a device.

Return values: CR+LF.

3.1.4 STX2ReadActualClimate

"STX2ReadActualClimate(ID)" – Returns actual climate values separated by semicolon.

Parameter: ID – Identifier of a device.

Return values: climate values separated by semicolon and CR+LF.

The first is a value of temperature in °C, the second is a value of relative humidity in percent, the third is a value of CO_2 concentration in percent, the fourth is a value of N2 concentration in percent.

3.1.5 STX2WriteSetClimate

"STX2WriteSetClimate(ID,T,H,CO2,N2)" – sets the climate values.

Parameter:

ID – Identifier of a device.

T - target temperature in °C.

H - target relative humidity in percent.

CO2 - the target CO2 concentration in percent.

N2 - the target N2 concentration in percent.

Return values: CR+LF.

3.1.6 STX2ReadSetClimate

"STX2ReadSetClimate(ID)" – Returns the target of climate values separated by semicolon.

Parameter: ID – Identifier of a device.

Return values: target climate values separated by semicolon and CR+LF.

The first is a target value of temperature in °C, the second is a target value of relative humidity in percent, the third is a target value of CO_2 concentration in percent, the fourth is a target value of N2 concentration in percent.

3.1.7 STX2ActivateShaker

"STX2ActivateShaker(ID,Speed)" - Writes shaker speed settings value and switches shaker on.

Parameter:

ID – Identifier of a device.

Speed - Shaker speed (range 1...50).

Return values: CR+LF.

3.1.8 STX2DeactivateShaker

"STX2DeactivateShaker(ID)" - Switches shaker off.

Parameter: ID – Identifier of a device.

Return value: CR+LF.

3.1.9 STX2ReadSetShakerSpeed

"STX2ReadSetShakerSpeed(ID)" - Returns shaker speed value,

Parameter: ID – Identifier of a device.

Return values: Shaker speed value or "-1" in case of an error and CR+LF.

3.1.10 STX2SwapIn

"STX2SwapIn(ID)" - Rotates the swap station on 180 degree.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

1 - Operation has been completed.

-1 - Error.

3.1.11 STX2SwapOut

"STX2SwapOut(ID)" - Rotates the swap station back to home position..

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

1 - Operation has been completed.

-1 - Error.

3.1.12 STX2Lock

"STX2Lock(ID)" - Locks the door and reads User Door Switch

Parameter: ID – Identifier of a device.

Return values: User door status and CR+LF.

"1" - User's door is opened.
"0" - User's door is closed.
-1 - Error.

3.1.13 STX2UnLock

"STX2UnLock(ID)" - Unlock the door.

Parameter: ID – Identifier of a device.

Return values: CR+LF.

3.1.14 STX2AbandonAccess

"STX2AbandonAccess(ID)" – Abandons to perform the prior Load Plate Operation.

Parameter: ID – Identifier of a device.

Return values: CR+LF.

3.1.15 STX2ContinueAccess

"STX2ContinueAccess(ID)" – Continues to perform the prior Load Plate Operation

Parameter: ID – Identifier of a device.

Return values: CR+LF.

3.1.16 STX2GetSysStatus

"STX2GetSysStatus(ID)" – Returns value of Status Register (DM202).

Parameter: ID – Identifier of a device.

Return value: DM202 (Status Register) or "-1" in case of an error and CR+LF.

Bit	Comment
00	System Ready
01	Plate Ready
02	System Initialized
03	XferStn status change
04	Gate closed
05	User door
06	Warning
07	Error
08	
09	
10	
11	
12	
13	
14	
15	

3.1.17 STX2ServiceReadBarcode

"STX2ServiceReadBarcode(ID,Slot,Level)" - Reads the barcode of a plate at specified location.

Parameters:

ID – Identifier of a device.

Slot - plate slot position.

Level - plate level position.

Return values: Result of operation and CR+LF.

"BCRError" - Barcode reader is not initialised.

"InitError" - StoreX is not initialized.

"Device Status Error" - There is no Barcode on the Plate.

"Device not Ready" - There is no Barcode on the Plate.

"No Plate" - There is no Plate at the specified position.

"No Barcode" - There is no Barcode on the Plate.

3.1.18 STX2Inventory

"STX2Inventory(ID,InvFileName,PP,BCR)" - Implements inventory of entire unit, the result is saved in the file– InvFileName. If the name of the file is not assigned, it will be generated automatically. The name of the file consists of a Serial number of device, date (e.g. "3298_A7010101.inv", where last two digits are number of the file).

Parameters:

ID – Identifier of a device.

FileName - name of file for saving results of inventory.

PPD – {0,1} sets whether Plate Present Detector will be used for Inventory.
1 - Inventory with Plate Presents Detector.
0 - Inventory without Plate Presents Detector.

BCR – {0,1} sets whether Barcode Reader will be used for Inventory.
1 - Inventory with Barcode Reader.
0 - Inventory without Barcode Reader.

Return values:

1 – STX2InventoryCassets operation is started.

-1 - Device is not initialised.

-2 - Previous long operation is not finished.

-3 – Device is Not Ready.

-4 – Device Status Error.

The result of the STX2Inventory is a file which consists of ten columns separated by coma.

The first is value of a Barcode (<null> - No Barcode or Barcode Reader is switched off), the second column is a IdCustomer value, the third column is a name of Partition, the fourth column is a value of Plate Present Detector (1 - plate is present; 0 - plate is not present or Plate Present Detector is off), the fifth column is a serial number of Plate, the sixth column is an Identifier of a System which assigned in a configuration file System.lni (Section - " System", Key - " SystemId"), the seventh column is an Identifier of a device, the eighth column is number of Cassette, the ninth column is a value of Level, the tenth column is a number of Row (reserved for the future).

3.1.19 STX2ServiceMovePlate

"STX2ServiceMovePlate(

      SrcInstrID,SrcPos,SrcSlot,SrcLevel,TransSrcSlot,SrcPlType,
      TrgInstrID,TrgPos,TrgSlot,TrgLevel,TransTrgSlot,TrgPltType)"

Moves a plate from position SrcPos to position TrgPos. This operation allows to move the plate within the bound of one device or between the cascader system.

Parameters:

SrcInstr - Identifier of a source Device.

SrcPos - Source position {1-TransferStation, 2-Slot-Level
           Position,3 – Shovel, 4-Tunnel, 5-Tube Picker}.

SrcSlot - plate slot position of source.

SrcLevel - plate Level position of source.

TransSrcSlot – number of a transport slot of a source device. It is obligatory for moving a
           plate between devices Base, Extended in Cascader. This Parameter is even for
           Extended Device and odd for Base Device.
           In case of Tube Picker this parameter defines a Source Plate Position on a Tube
           Picker Device {0,1}
           0-Target Position.
           1 - Source Position.

SrcPlType - Type of plate of source position {0-MTP, 1-DWP,
           3-P28}.

TrgInstr - Identifier of a target device.

TrgPos – Target position {1-TransferStation, 2-Slot-Level
           Positioan, 3 – Shovel, 4-Tunnel, 5-Tube Picker}.

TrgSlot - plate slot position of target.

TrgLevel - plate level position of target.

TransTrgSlot – number of a transport slot of a Target Device. It is obligatory for moving a
           plate between devices Base, Extended in Cascader. This Parameter is even for
           Extended Device and odd for Base Device.
           In case of Tube Picker this parameter defines a Target Plate Position on a Tube
           Picker Device {0,1}
           0-Target Position.
           1 - Source Position.

TrgPltType - Type of plate of source position {0-MTP, 1-DWP, 3-P28}.

Return value: result of operation and CR+LF.

1 - Function STX2ServiceMovePlate has been completed.

Returning values because of parameters error:

-1 – Previous long operation is not finished.

-2 - One of input parameters is not a valid integer value.

-3 – A Source or a Target Device is not specified or not initialised.

-4 - One or more of devices is not defined in a system.

-5 – One of transport slot is not specified.

-6 - Wrong value of a target transport slot.

-7 - Wrong value of a source transport slot.

-8 – Wrong value of a source position.

-9 – Wrong value of a target position.

Returning values because of internal device error:

This value consists of more than one character and separated by semicolon. First character returns Identifier of a Device where the error has been taken place and last character indicates a step of the error.

-ID;1 - Error during LoadPlate operation, device "ID".

-ID;2 - Error during UnloadPlate operation.

-ID;3 - Error during PickPlate operation.

-ID;4 - Error during PlacePlate operation.

-ID;5 - Error during SetPlate operation.

-ID;6 - Error during GetPlate operation.

-ID;7 – Device is not Ready

-ID;8 – Device Status Error

3.2 Additional Commands

3.2.1 STX2PartitionInventory

"STX2PartitionInventory(ID,InvFileName,PartitionID, PP,BCR)" - Implements inventory of defined Partitiont, the result is saved in a file - InvFileName. If the name of the file is not assigned, it will be generated automatically. The name of the file consists of a Serial number of device, date (e.g. "3298_A7010101.inv", where last two digits are number of the file).

Parameters:

ID - Identifier of a device.

FileName - name of file for saving results of inventory.

PartitionID - Name of a Partition. Information about partitions is stored in Unit Configuration file section - "Partitions".

PPD - {0,1} sets whether Plate Present Detector will be used for Inventory.
1 - Inventory with Plate Presents Detector.
0 - Inventory without Plate Presents Detector.

BCR - {0,1} sets whether Barcode Reader will be used for Inventory.
1 - Inventory with Barcode Reader.
0 - Inventory without Barcode Reader.

Return values:

1 - STX2InventoryCassets operation is started.

-1 - Device is not initialised.

-2 - Previous long operation is not finished.

-3 - Barcode Reader has not been Initialised.

-4 - Unknown Partition. Partition has not been defined in Unit Configuration file.

-5 - There is no Cassettes in a Partition. Unit configuration file Error section - "Partitions". The range of a Partition has not been configured properly.

-6 – Device is Not Ready.

-7 – Device Status Error.

The result of the STX2Inventory is a file which consists of ten columns separated by coma.

The first is value of a Barcode (<null> - No Barcode or Barcode Reader is switched off), the second column is a IdCustomer value, the third column is a name of Partition, the fourth column is a value of Plate Present Detector (1 - plate is present; 0 - plate is not present or Plate Present Detector is off), the fifth column is a number of Line, the sixth column is an Identifier of a System which assigned in a configuration file System.Ini (Section - " System", Key - " SystemId"), the seventh column is an Identifier of a device, the eighth column is number of Cassette, the ninth column is a value of Level, the tenth column is a number of Row (reserved for the future).

For using this command each partition has to be described in Unit configuration file section - "Partitions".

e.g.

[Partitions]
A=1-2
B=3-6
Test=7

Partition "A" has two cassettes 1 and 2. Partition "B" has four cassettes 3, 4, 5, and 6. Partition "Test" consists of one cassette - 7.

3.2.2 STX2IsOperationRunning

"STX2IsOperationRunning(ID)" - Checks whether previous long operation is running. The user can check whether long operations like STX2ServiceMovePlate, STX2Inventory are running by means of usage this method.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"1" - Previous long operation is still running

"0" - Previous long operation is completed or cancelled (or Device is Error).

"-1" – Error.

3.2.3 STX2ReadErrorCode

"STX2ReadErrorCode(ID)" - Interrogates Error Flag and Error code of System.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"0" - No Errors.
If returns the System status code then see descriptions of error codes.

"-1" – Error.

3.2.4 STX2SoftReset

"STX2SoftReset(ID)" - Implements Soft Reset command.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"1" – Command STX2SoftReset has been completed.

"-1" – Error.

3.2.5 STX2ReadUserDoorFlag

"STX2ReadUserDoorFlag(ID)" - Reads whether door is opened.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"1" - User's door is opened.

"0" - User's door is closed.

"-1" – Error.

3.2.6 STX2ReadShovelDetector

"STX2ReadShovelDetector(ID)" - Reads whether plate is present on the Shovel. If Plate Shovel Detector is not assigned in Unit configuration file section Sensor Configuration value PlateShovelSensor=1 this function returns value "0" by default.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"1" - Plate presents on the Shovel.

"0" - Plate doesn't present on the Shovel.

"-1" – Error.

3.2.7 STX2ReadXferStationDetector1

"STX2ReadXferStationDetector1(ID)" - Reads whether plate is present on the Transfer Station. If Plate Station Detector is not assigned in Unit configuration file section Sensor Configuration value PlateXferStSensor1=1 this function returns value "0" by default.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"1" - Plate is on the Transfer Station.

"0" - Plate is not on the Transfer Station.

"-1" – Error.

3.2.8 STX2ReadXferStationDetector2

"STX2ReadXferStationDetector2(ID)" - Reads whether plate is present on the second Transfer Station. If Second Plate Station Detector is not assigned in Unit configuration file section Sensor Configuration value PlateXferStSensor2=1 this function returns value "0" by default.

Parameter: ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"1" - Plate is on the Transfer Station.

"0" - Plate is not on the Transfer Station.

"-1" – Error.

3.2.9 STX2ManualAccess

"STX2ManualAccess(ID,Cassette)" - Rotates carousel and sets defined Cassette in front of the User Door for manual access. Carousel offset value has to be assigned in Unit configuration file section Carousel Configuration, key ManualAccessOffset.

e.g.

[Carousel Configuration]
ManualAccessOffset=3.

Parameters:

ID - Identifier of a device.

Cassette - Cassette ID for manual access.

Return values: Result of operation and CR+LF.

"1" - Operation has been finished.

"0" - Device is not initialised.

"-1" - Error (Error Flag is 1).

"-2" - A wrong value of slot.

"-3" - User's door is opened.

"-4" - Previous long operation is not finished.

3.2.10 STX2BeeperOn

"STX2BeeperOn(ID)" – Turns Bepper Alarm on.

Parameter: ID – Identifier of a device.

Return values: CR+LF.

3.2.11 STX2BeeperOff

"STX2BeeperOff(ID)" – Turns Bepper Alarm off.

Parameter: ID – Identifier of a device.

Return values: CR+LF

3.2.12 STX2ReadBarcodeAtTransferStation

"STX2ReadBarcodeAtTransferStation(ID)" - Reads the barcode of a plate at Transfer Station. Please, make sure that a firmware of your device is compatible with this function.

Parameters:

ID – Identifier of a device.

Return values: Result of operation and CR+LF.

"BCRError" - Barcode reader is not initialised.

"InitError" - StoreX is not initialized.

"No Barcode" - There is no Barcode on the Plate.

4 List of new Commands (overview)

Command	Parameters	Result
STX2Activate(ID)	ID – Identifier of a device	Result{x or x;y}+CR+LF
STX2Deactivate(ID)	ID – Identifier of a device	CR+LF
STX2Reset(ID)	ID – Identifier of a device	CR+LF
STX2ReadActualClimate(ID)	ID – Identifier of a device	Result {T;H;CO2;N2}+CR+LF
STX2WriteSetClimate(ID,T,H,CO2,N2)	ID – Identifier of a device T – target temperature in °C H – target relative humidity in percent CO2 – the target CO2 concentration in percent N2 – the target N2 concentration in percent	CR+LF
STX2ReadSetClimate(ID)	ID – Identifier of a device	Result {T;H;CO2;N2}+CR+LF
STX2ActivateShaker(ID,Speed)	ID – Identifier of a device Speed - Shaker speed	CR+LF
STX2DeactivateShaker(ID)	ID – Identifier of a device	CR+LF
STX2ReadSetShakerSpeed(ID)	ID – Identifier of a device	Resulr {Shaker speed or -1} +CR+LF
STX2SwapIn(ID)	ID – Identifier of a device	Result {1,-1}+CR+LF
STX2SwapOut(ID)	ID – Identifier of a device	Result {1,-1}+CR+LF
STX2Lock(ID)	ID – Identifier of a device	Result {1,0,-1}+CR+LF
STX2UnLock(ID)	ID – Identifier of a device	CR+LF
STX2AbandonAccess(ID)	ID – Identifier of a device	CR+LF
STX2ContinueAccess(ID)	ID – Identifier of a device	CR+LF
STX2GetSysStatus(ID)	ID – Identifier of a device	Result {Status registr or -1} +CR+LF
STX2ServiceReadBarcode(ID,Slt,Lvl)	ID – Identifier of a device Slt – plate slot position Lvl – plate level position	Barcode operation result + CR+LF
STX2Inventory(ID,FileName,PP,BCR)	ID – Identifier of a device FileName - name of file for saving results of inventory PPD – sets whether Plate Present Detector will be used for Inventory BCR – sets whether Barcode Reader will be used for Inventory	Result {1,-1,-2,-3,-4} CR+LF Inventory File
STX2PartitionInventory(ID,FileName,PartitionID,PP,BCR)	ID – Identifier of a device FileName - name of file for saving results of inventory PartitionID - Name of a Partition. PPD – sets whether Plate Present Detector will be used for Inventory BCR – sets whether Barcode Reader will be used for Inventory	Result {1,-1,-2,-3,-4,-5,-6,-7} CR+LF Inventory File
STX2ServiceIsPlateAtLocation(ID,Slt,Lvl)	ID – Identifier of a device Slt – plate slot position Lvl – plate level position	Result:{1,0,-1,-2}+CR+LF
STX2ServiceMovePlate(SrcInstr,SrcPos,SrcSlt,SrcLvl, TransSrcSlt,SrcPlType, TrgInstr,TrgPos,TrgSlt,TrgLvl, TransTrgSlt,TrgPltType)	ID – Identifier of a device SrcInstr – Identifier of a source Device SrcPos – Source position SrcSlt – plate slot position of source SrcLvl – plate Level position of source TransSrcSlt – number of a transport slot of a source device SrcPlType – Type of plate of source position TrgInstr – Identifier of a target device TrgPos – Target position TrgSlt – plate slot position of target TrgLvl – plate level position of target	Result: {x or –dev;x}+CR+LF

	TransTrgSlt – number of a transport slot of a Target Device TrgPltType – Type of plate of source position	
STX2IsOperationRunning(ID)	ID – Identifier of a device	Result {1,-1,0}+CR+LF
STX2ReadUserDoorFlag(ID)	ID – Identifier of a device	Result {1,-1,0}+CR+LF
STX2ReadErrorCode(ID)	ID – Identifier of a device	Result {-1,0, Error Code} +CR+LF
STX2ReadShovelDetector(ID)	ID – Identifier of a device	Result {1,-1,0}+CR+LF
STX2ReadXferStationDetector1(ID)	ID – Identifier of a device	Result {1,-1,0}+CR+LF
STX2ReadXferStationDetector2(ID)	ID – Identifier of a device	Result {1,-1,0}+CR+LF
STX2ManualAccess(ID,Cassette)	ID - Identifier of a device. Cassette - Cassette ID for manual access	Result {1,0,-1,-2,-3,-4} +CR+LF
STX2BeeperOn(ID)	ID – Identifier of a device	CR+LF
STX2BeeperOff(ID)	ID – Identifier of a device	CR+LF
STX2ReadBarcodeAtTransferStation(ID)	ID – Identifier of a device	Barcode operation result + CR+LF

5 StoreX Network Communication

5.1 Client Socket example (C#)

5.1.1 Client Socket class

```csharp
public class StateObject
{
        public Socket workSocket = null;
        public const int BufferSize = 256;
        public byte[] buffer = new byte[BufferSize];
        public StringBuilder sb = new StringBuilder();
}

public class AsynchronousClient
{
        private Socket client;

        // The port number for the remote device.
        //private const int port = 3336;

        // ManualResetEvent instances signal completion.

        private ManualResetEvent connectDone =  new ManualResetEvent(false);
        private ManualResetEvent sendDone =     new ManualResetEvent(false);
        private ManualResetEvent receiveDone = new ManualResetEvent(false);

        // The response from a remote StoreX Server.
        private String response = String.Empty;


        public Socket getSocket()
        {
                return client;
        }


        public void StartClient(String STXIPAddress, int STXport)
        {

                // Connect to a remote device.
                try
                {

                        // Establish the remote endpoint for the socket.
                        IPHostEntry ipHostInfo = Dns.Resolve(STXIPAddress);
                        IPAddress ipAddress = ipHostInfo.AddressList[0];
```

```csharp
                IPEndPoint remoteEP = new IPEndPoint(ipAddress, STXport);

                // Create a TCP/IP socket.
                client = new Socket(AddressFamily.InterNetwork,
                                        SocketType.Stream, ProtocolType.Tcp);

                // Connect to the remote endpoint.
                client.BeginConnect(remoteEP,
                                        new AsyncCallback(ConnectCallback), client);
                connectDone.WaitOne();

        }
        catch (Exception e)
        {
                Console.WriteLine(e.ToString());
        }

}

public void StopClient()
{
        // Release the socket.
        client.Shutdown(SocketShutdown.Both);
        client.Close();
}

private void ConnectCallback(IAsyncResult ar)
{

        try
        {
                // Retrieve the socket from the state object.
                Socket client = (Socket) ar.AsyncState;

                // Complete the connection.
                client.EndConnect(ar);

                Console.WriteLine("Socket connected to {0}",
                                client.RemoteEndPoint.ToString());

                // Signal that the connection has been made.
                connectDone.Set();
        }
        catch (Exception e)
        {
                Console.WriteLine(e.ToString());
        }
```

```csharp
        }

        private void Receive(Socket client)
        {

                try
                {
                        // Create the state object.
                        StateObject state = new StateObject();
                        state.workSocket = client;

                        // Begin receiving the data from the remote device.
                        client.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
                                        new AsyncCallback(ReceiveCallback), state);
                }
                catch (Exception e)
                {
                        Console.WriteLine(e.ToString());
                }

        }

        private void ReceiveCallback( IAsyncResult ar )
        {

                if (!this.client.Connected) return;

                try
                {
                        StateObject state = (StateObject) ar.AsyncState;
                        Socket client = state.workSocket;

                        int bytesRead = client.EndReceive(ar);

                        if (bytesRead > 0)
                        {

                        // There might be more data, so store the data received so far.

                                state.sb.Append(Encoding.ASCII.GetString(state.buffer,0,
                                        bytesRead));

                                // Get the rest of the data.

                                client.BeginReceive(state.buffer,0,StateObject.BufferSize,
                                                0,new AsyncCallback(ReceiveCallback), state);
```

```csharp
                        response = ncoding.ASCII.GetString(state.buffer,0,bytesRead);

                        Console.WriteLine("Received from STX: "+response);
                        receiveDone.Set();
                }
                else
                {

                        // All the data has arrived; put it in response.
                        if (state.sb.Length > 1)
                        {
                                response = state.sb.ToString();
                        }

                        // Signal that all bytes have been received.
                        receiveDone.Set();
                }
        }
        catch (Exception e)
        {
                Console.WriteLine(e.ToString());
        }

}


public String Send(String data)
{

        response = "";

        // Convert the string data to byte data using ASCII encoding.
        byte[] byteData = Encoding.ASCII.GetBytes(data);

        // Begin sending the data to the remote device.
        client.BeginSend(byteData, 0, byteData.Length, 0,
                        new AsyncCallback(SendCallback), client);

        Console.WriteLine("Sent STRING to STX server: "+data);

        Receive(client);
        receiveDone = new ManualResetEvent(false);
        receiveDone.WaitOne();

        return response;

}
```

```csharp
        private void SendCallback(IAsyncResult ar)
        {
                try
                {

                        // Retrieve the socket from the state object.
                        Socket client = (Socket) ar.AsyncState;

                        // Complete sending the data to the remote device.
                        int bytesSent = client.EndSend(ar);

                        Console.WriteLine("Sent {0} bytes to server.", bytesSent);

                        // Signal that all bytes have been sent.
                        sendDone.Set();

                }
                catch (Exception e)
                {
                        Console.WriteLine(e.ToString());
                }
        }
} // End of Socket's client class
```

5.1.2 Using an Client Socket

```csharp
String STXId = "UNIT1";

String STXAddress = "192.168.1.1";

int STXPort = 3336;

AsynchronousClient STXSCK = new STX_TCP_Client.AsynchronousClient();

STXSCK.StartClient(STXAddress, STXPort);

String ResActivate = STXSCK.Send("STX2Activate("+STXId+")"+(char)13);

if ((ResActivate == "1;1") || (ResActivate == "1"))
{
        MessageBox.Show("Device is initialisd!", "1", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
else
{
        MessageBox.Show("Initialisation Error!", "1", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

}

STXSCK.StopClient();