CashKeeper®

Closed and secure cash handling

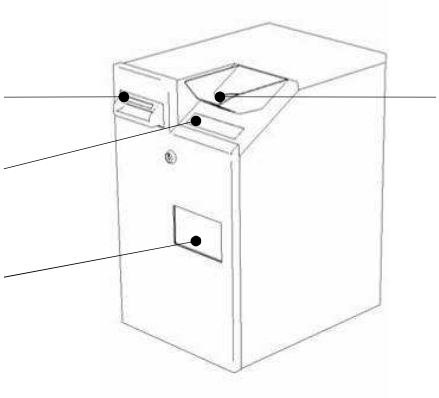
INTEGRATION MANUAL

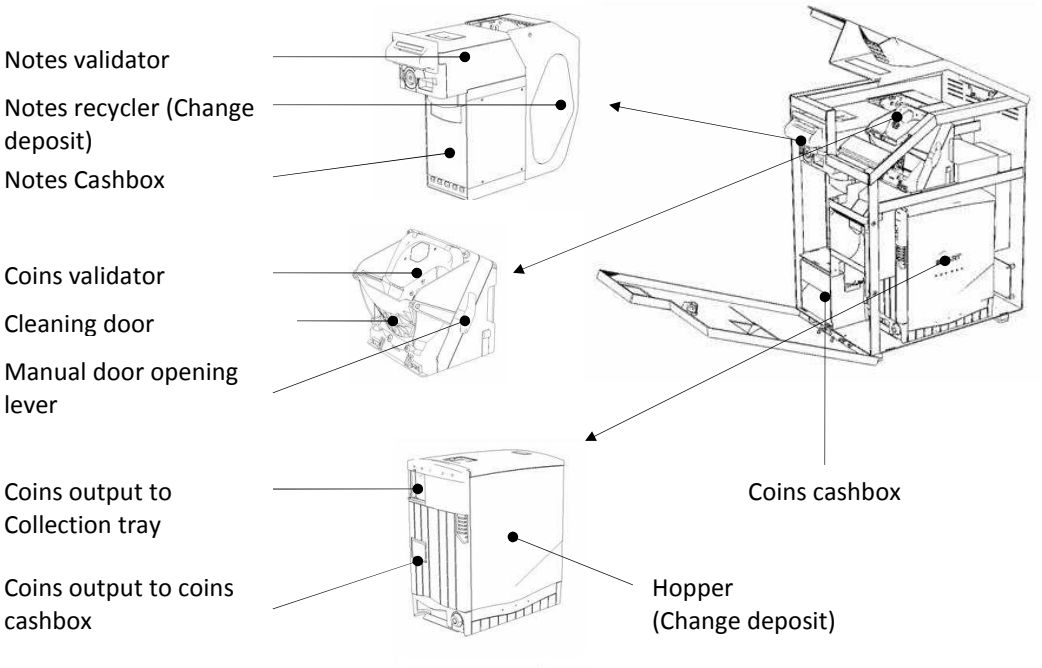
v. 1.0

Date	Modification	Resp	Level Doc	Version DLL	Version CSSI	Compatible	Protocol
18/03/12	Preliminary version	MIC	-	-			
	Official version (release)	JFC	OR	11.4.1		YES	
13/03/15	Added CriticalBehaviour	MIC					
5/10/15	Errors table updated. Added function GetLastLevels. Added function SetDateTime. Added function GetNetworkParams. Added function SetNetworkParams. Added function EmptyCashBoxEx. Added function GetCCDetails. Updated function GetAllProperties.	MIC		15.10.5	15.10.5	YES	
19/12/16	Updated function GetCurrentLevels	MIC					
25/5/17	Updated function GetLevels Addes section Special properties	MIC					

1. *CashKeeper* ® DESCRIPTION

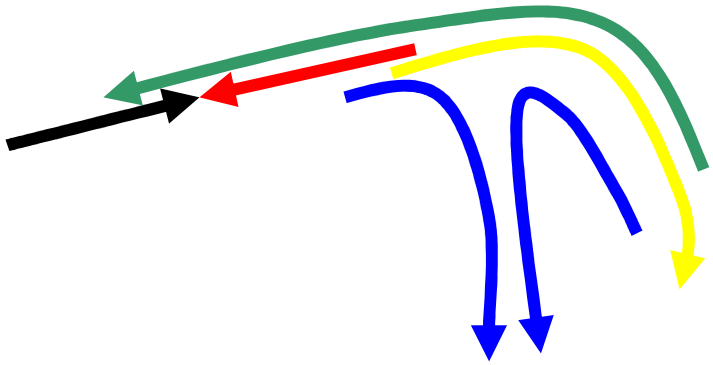
- **Elements description**



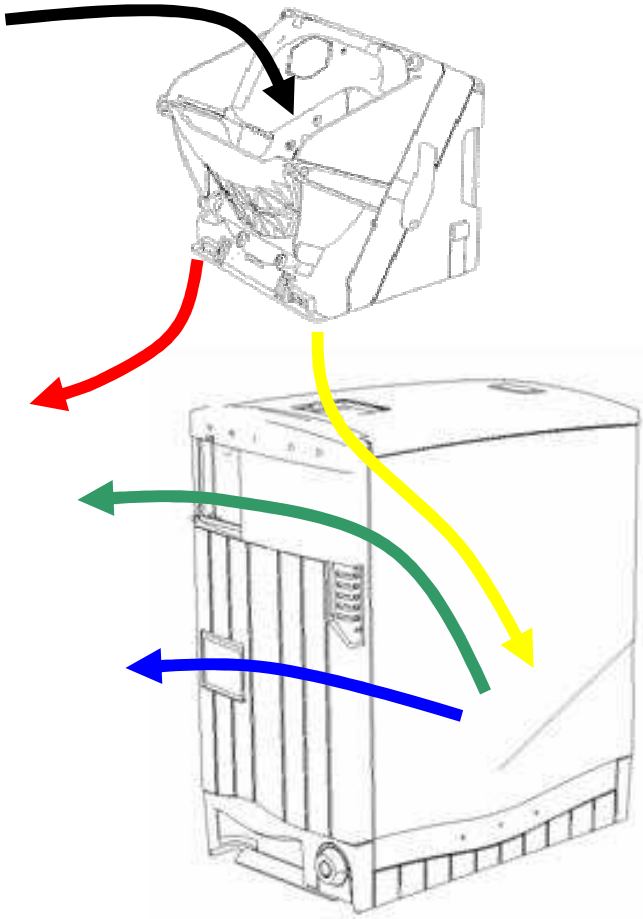


- **Banknotes and coins routes in CashKeeper ®**

Notes entrance
Change buffer storage
Output/pay notes
Collection cashbox storage
Rejected notes



Coins entrance
Change buffer storage
Output/pay coins
Collection cashbox storage
Rejected coins



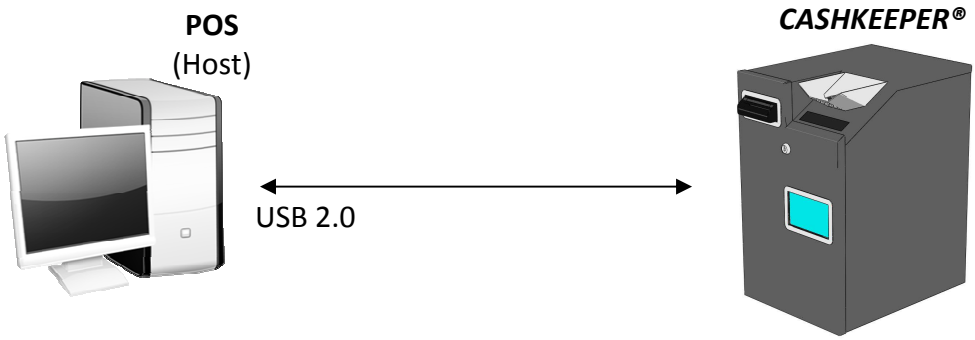
2. *CashKeeper* ® INTEGRATION CONCEPT

Due to the nature of the payment and collection processes, CashKeeper ® is a device that works **asynchronously**. There are different procedures that can be performed in a synchronous way, but those involving physical device operations (like collections, payments, sending change to cashboxes, etc...), work in a completely asynchronous manner.

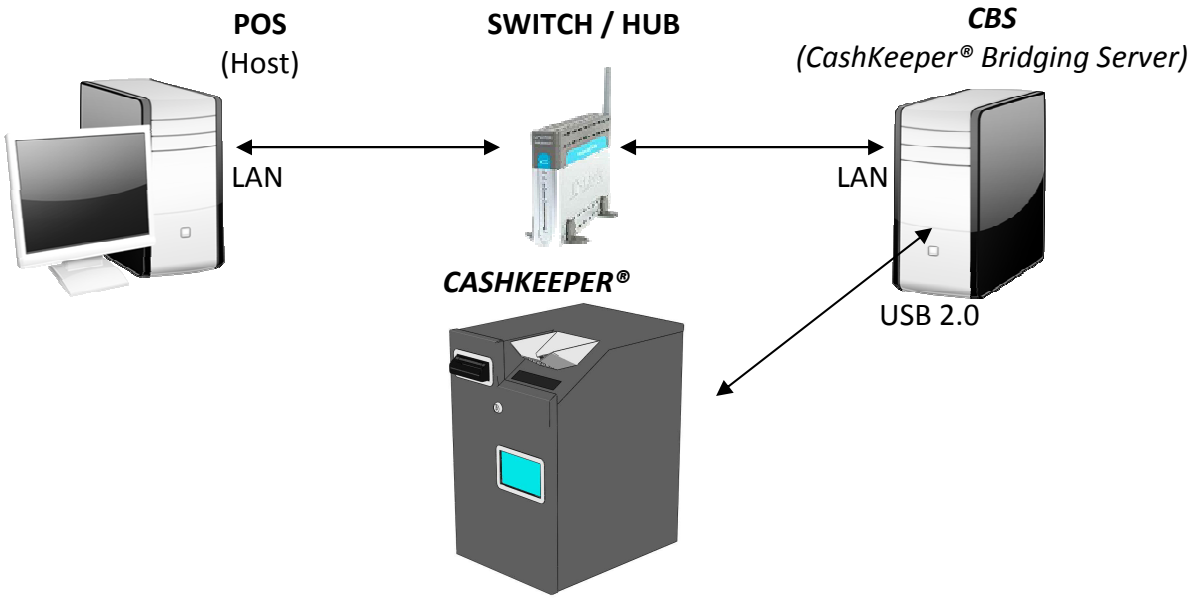
The integration of a CashKeeper ® device must be made through the communication with an interface called ***'CashKeeper® Secure Server Interface' (CSSI)***, protocol TCP/IP-based. To establish this communication with CSSI, Could be done through the tool ***Ckeeper.DLL*** (CashKeeper Control Tool), or through the 'direct method', attacking the CSSI directly via sockets (TCP/IP).

CSSI interface may be resident in the own POS terminal or can be used a middle 'bridge' between the POS terminal and the CashKeeper ® device, ***CashKeeper® Bridging Server' (CBS)***. CBS is a server that will manage all of its capacity and resources to control the communication between the POS terminal and the CashKeeper ®, through the CSSI.

- ***CSSI* LOCAL INTEGRATION**

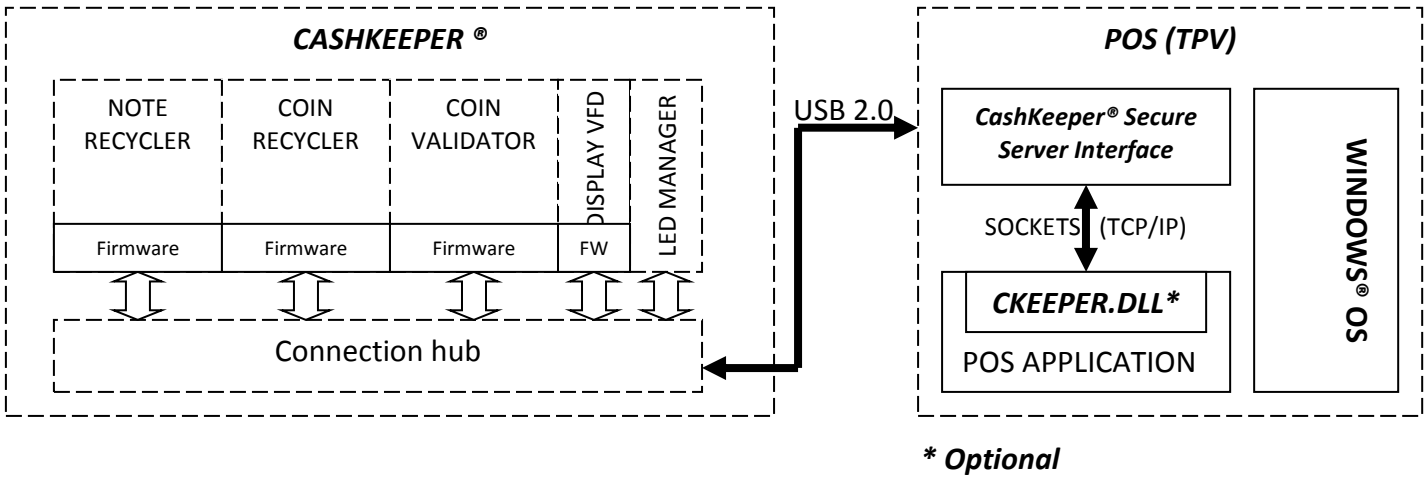


- ***CSSI* REMOTE INTEGRATION (with *CashKeeper® Bridging Server*)**



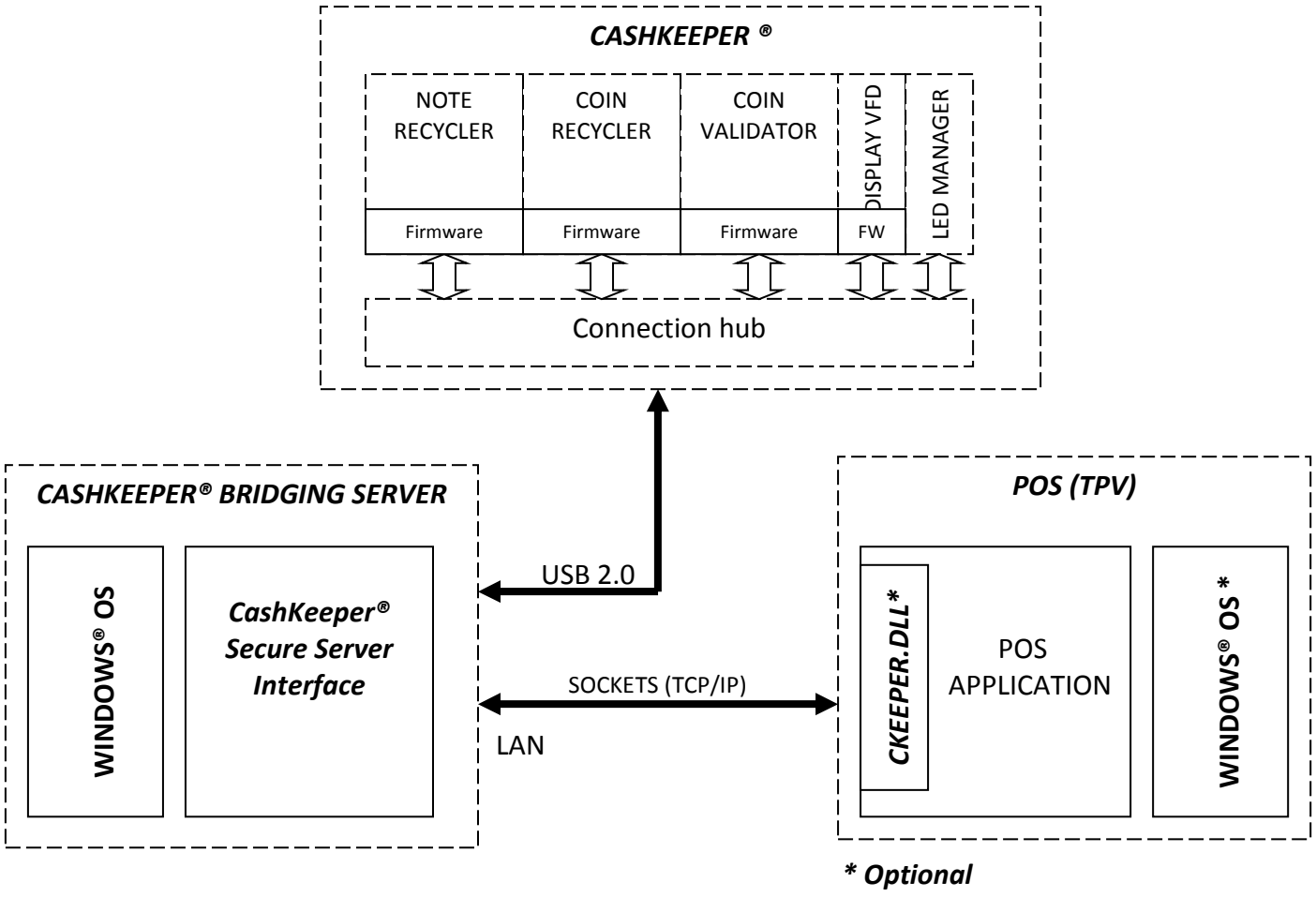
3. *CashKeeper* ® INTEGRATION DIAGRAM

- ***CSSI* LOCAL INTEGRATION**



** Optional*

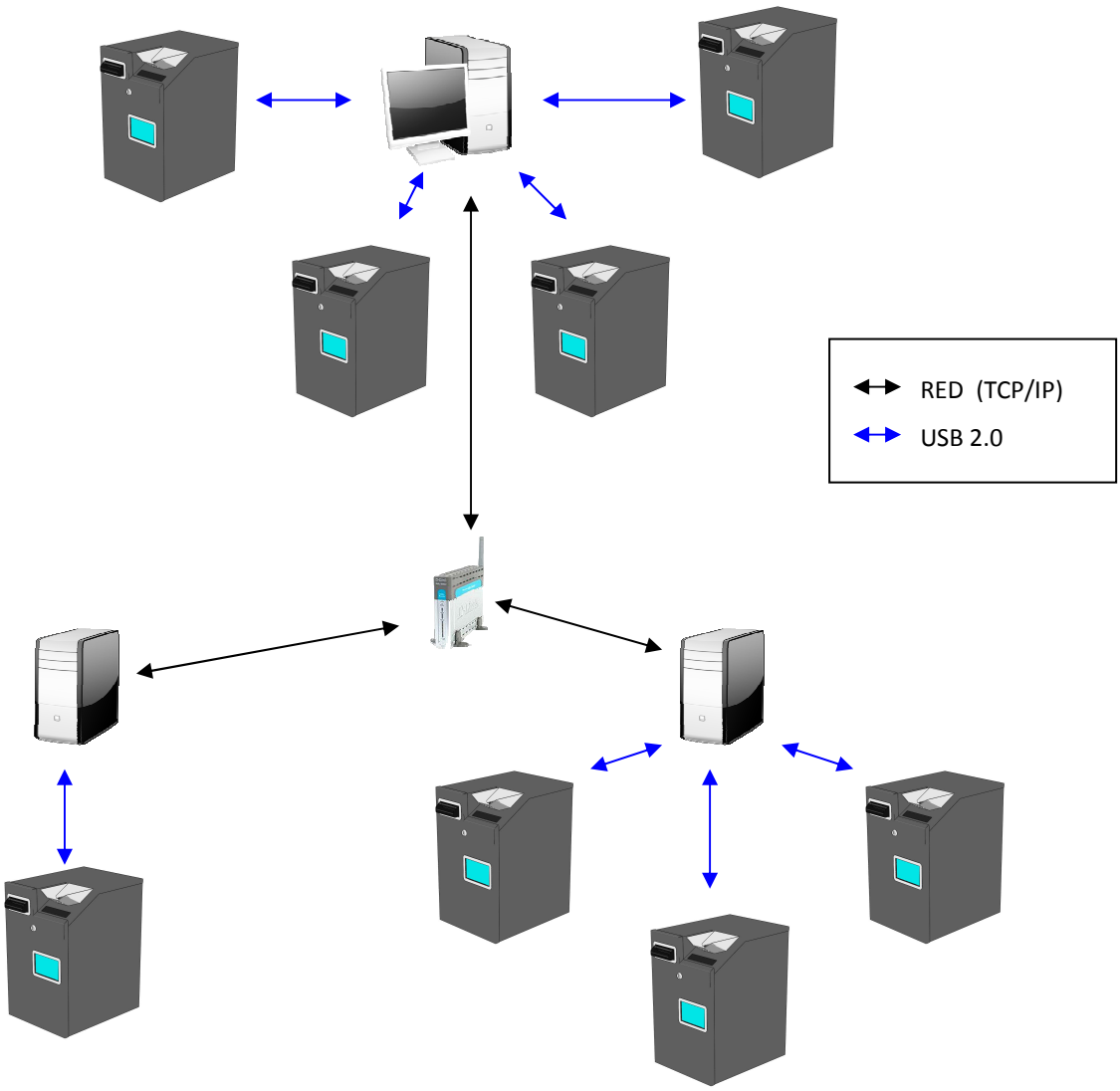
- ***CSSI* REMOTE INTEGRATION (with *CashKeeper® Bridging Server*)**



** Optional*

3.1. Multiple CashKeeper Integration

(protocol 4 or posterior)



4. CashKeeper Integration Methods

4.1. CKEasy

It is the easier and faster system integration, because it incorporates its own graphical interface. All calls are synchronous, which facilitates the use of different functionalities. Those functionalities, are very high level ones, like collections, payments, to tally, etc.. Also, it incorporates some calls without graphical support to allow the application to get the necessary data.

4.2. CKeeper

It's an events-oriented system that allows controlling CashKeeper in all its fullness, does not include graphical interface, enabling seamless integration with your application. It offers the same functionality as the sockets method, but as it is an ActiveX object, it makes it easier.

4.3. Sockets

It is the lower level integration system and which requires more complexity, although it allows you to be transparent to the OS used, the only requirement is that SO used can run with sockets. Nowadays, all.

5. Use CKEasy

5.1. Use

To use CKEasy, create an ActiveX Object EasyCashKeeper. Cause the connection process requires a few seconds. It is recommended to connect at initializing the application time (although the integrator can choose when connecting and in any case, it connects automatically when calling any function) and disconnecting when application exits. Once connected, use the function calls as desired. Attention, this library is not thread safe, so during call execution, others calls should be avoided.

5.2. Connection

If you connect to the CSSI located on your own computer, you only need to call the **Connect** method.

If you connect to the CSSI located on another computer, you need to inform the IP property with the ip address or the network name of the computer that has physically the CashKeeper attached. Then call the **Connect** method.

5.3. Disconnection

You only need to call Disconnect function.

5.4. Collection

For a collection the **Charge** function will be do the job, returning once completed the introduced amount, the returned change and if there has been any problem.

5.5. Payment

To make a payment the function to call is **Pay**, it will return the payment result (correct / incorrect).

5.6. Add change

AddChange function is specific to add change, displaying on the screen the lack/insufficiency levels depending on the different denominations configuration and allowing the introduction of banknotes and coins. Returns the amount introduced.

5.7. Give change

The **Change** function opens a screen that allows you to insert money and choose that way you want to be returned. It should be remind that in an installation with CashKeeper there is no money cash to give change for the tobacco machine, fairground, etc. Returns the entered and the paid amount.

5.8. Parameters

The **Configuration** function opens a screen where it can be defined some parameters which allows adapting the CashKeeper functioning to the client operation.

5.9. Cashbox

The **CashBoxControl** function allows the application to control the processes of end of day, configurate the acceptance/rejection of different denominations, empty cashboxes, completely empty available change to the collection cashboxes, set the initial amounts, etc. It includes several options without graphical interface, in the case that the application already have an own screen.

5.10. Functions without graphical interface

The **EmptyCashBox**, **GetAmounts** and **GetLevels** functions allow emptying drawers, obtaining the amount and getting the detail of the different denominations without any graphical interference.

5.11. Other parameters

BackColor and **InverseColor** are two properties that allow choosing to the integrator the forms colour to "improve" the CashKeeper integration with your application.

ErrorCode and **ErrorDescription** are two variables that contain code and description in case of error.

OnErrorDiscard this property allows software to define what will happened with the credited amount when an error occurs during charge operation.

5.12. Special Properties

This properties are read only, and can be read only when it is connected. When it is not connected, will return default values. Both properties helps applications to be currency aware.

ShowDecimals Application can read this property to know the number of decimals must be shown on amounts.

AmountFactor Application should read this property to know the multiply / divide factor between the Amounts in currency and the values must be used in operations or operations returns.

Ex. With EUR will return factor is 100.

If I want to charge/ pay 1,45€, I must send 1,45 * 100 = 145

When I add some change, machine will return 145, after apply the factor, (145 / 100) = 1,45

5.13. Functions reference

AddChange(PaidInValue As Int32)

Opens a screen that allows adding change to CashKeeper.

Output parameters

PaidInValue will contain the amount added to change

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

CashBoxControl(Mode As CBC_Modes, CoinsValueInCB As Int32, NotesValueInCB As Int32, RemainingChange As Int32)

Input parameters:

Mode possible values:

0-Normal: Open a form to see the CashKeeper levels and it allows changing the settings for initial, minimum and maximum values as well as to tally.

1- CBC_Disable_Config: Open a form to see the CashKeeper levels and allows to tally.

4- CBC_Disable_Actions: Open a form to see the CashKeeper levels and it allows changing the settings for initial, minimum and maximum values.

7- CBC_Blind: It shows a process screen (without data) and does the end of day process (send the remaining of initial change to cashbox and empty cashboxes).

15- CBC_Blind_By_Value: It shows a process screen (without data) and does the end of day process (send the remaining of initial change to cashbox and empty cashboxes). You must provide the amount you want to keep in change on RemainingChange parameter.

5: It is the combination of mode 1 and 4, it displays a screen with the CashKeeper data but doesn't take any action.

Output parameters

CoinsValueInCB: it will contain the amount of coins in cashbox in case of emptying the cashboxes.

NotesValueInCB: it will contain the amount of banknotes in cashbox in case of emptying the cashboxes.

RemainingChange: It will contain the amount of change that has CashKeeper.

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

Configuration()

Opens a screen that allows configuring various CashKeeper parameters.

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

Connect ()

Initiates the CSSI connection.

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

Change(PaidInValue As Int32, PaidOutValue As Int32)

Opens a screen that allows making denominations changes. For example 1 note €5 for 5 coins of €1.

Output parameters

PaidInValue: Entered amount

PaidOutValue: Returned amount

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

Charge(ValueInCents As Int32, PaidInValue As Int32, PaidOutValue As Int32)

Open a screen for the payment amount in cents

Input parameters:

ValueInCents : Amount to be received

Output parameters

PaidInValue: Entered amount

PaidOutValue: Returned amount

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

It is important to verify returned values. If function terminates OK, it allows you to detect "broken cents". If function fails, it allows you to detect pending payments or and incomplete amount.

Disconnect ()

Ends CSSI connection. This method never can have an error, so it does not return any value.

EmptyCashBox(Device As CKD_Devices, Value As Long)

Screenless function to empty the cashboxes.

Input parameters:

Device: Cashbox to empty possible values:

CKD_Coins = 0

CKD_Notes = 1

Output parameters:

Value: Amount containing the empty cashbox before emptying it.

Returns:

True if done correctly

False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

GetAmounts(Denoms As String, Quantities As String, Value As Int32)

Screenless function to get the exchange contains detail of CashKeeper.

Output parameters:

Denoms: Denomination list separated by commas.

Quantities: Quantities list of denominations separated by commas and respective to the denomination list.
Value: Exchange amount.
Returns:
True if done correctly
False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

GetLevels(LevelType As CKL_Levels, Denoms As String, Levels As String)

Screenless function to get the contain detail of CashKeeper. Depending on the LevelType parameter will return the initial, maximum, change or cashbox.
Input parameters:
LevelType: possible values:
CKL_Initial = 0
CKL_Max = 1
CKL_Current = 2
CKL_CashBox = 3
CKL_LevelStatus = 4
Output parameters:
Denoms: Denomination list separated by commas.
Levels: Quantities list of denominations separated by commas and respective to the denomination list. When LevelType is CKL_LevelStatus, levels will contains the traffic light code (0 = green, 1 = orange, 2 = red).
Returns:
True if done correctly
False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

Pay(ValueInCents As Int32, PaidOutValue As Int32)

Opens a screen that reports on the amount payment process in cents.
Input parameters:
ValueInCents: amount to pay in cents.
Output parameters:
PaidOutValue: Amount actually paid.
Returns:
True if done correctly
False if an error occurred, see properties *ErrorCode* y *ErrorDescription* for more information.

5.14. Properties reference

BackColor

Indicates the form background colour

InverseColor

Indicates the letters colour, it is automatically calculated as changed the BackColor property

IP

IP address where is the CSSI located. It is used at the connection time so it has to be reported before connecting.

OnErrorDiscard

When an error occurs during charge operation, could remain some credited amount on the machine. With this property, software can control if next charge operation takes care of the already credited amount or not.

6. USE *CASHKEEPER*

Procedures to operate with CashKeeper ®:

6.1. *CashKeeper® states*

To understand the CashKeeper processes operation, before we have to explain its states:

- ALL_CLOSED (0x00): Indicates that the communication ports are not open and there is no active communication with the device. It is the initial state. Attention, as it is a CashKeeper state, without being connected is impossible to know it. Therefore the integrator must control when is or not connected.

- IDLE (0X01): The system is in sleep mode. Communication with the device is active and can be performed any action. With the exception of some specific commands (used to make CashKeeper change of state), most of the functions must be done when CashKeeper is in this state.

- ENABLED (0X02): The system is in cash reception state.

- PAYING (0X03): The system is doing a payment (notes and/or coins). During this state, the system will trigger the **ValueOUT** event informing of the total amount paid.

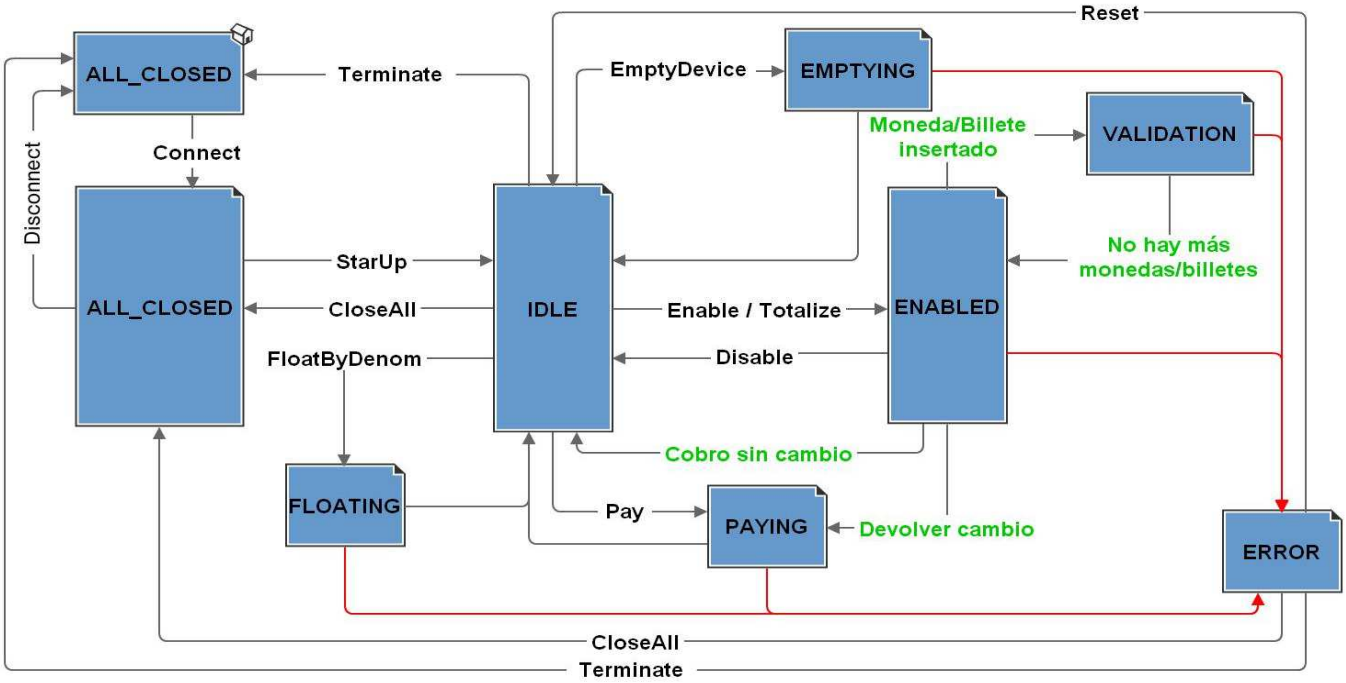
- FLOATING (0x04): The system is sending coins and/or banknotes to the cashboxes. During this state, the system will trigger the **ValueToCashBox** event reporting the total amount sent to cashboxes.

- EMPTYING (0X05): The system is emptying the coins and/or notes change recipients to the cashboxes. During this state, the system will trigger the **ValueToCashBox** event reporting the total amount sent to cashboxes.

- ERROR (0x06): The system is in an error state and is not possible to operate with it. The system acquires this status when a non-recoverable error occurs (note or coin jammed, etc.). The only exception would be the **Reset** function, which during its execution, acquires this status to prevent the execution of other commands.

- VALIDATION(0x07): The system is verifying the introduced coins/banknotes, this means that there is mechanical movement on the devices. It is in this state when **ValueIN** events occur corresponding to validated cash.

6.2. States and transition diagram



Legend diagram
Boxes: States
Black arrows: CashKeeper functions
Green arrows: external actions/desitions
Red arrows: Error situations.

6.3. Prior considerations

6.3.1. Accepted denominations

CashKeeper is able to accept and validate different denominations from different countries. Generally, it is set a currency (EUR, GBP, USD, etc.) as

principal (will be the only one that is paid), and there may be other currencies only accepted. For example, in border/touristic zone, could be interesting to accept dollars as well as the local currency.

6.3.2. Value data information

All the returned or informed value data, will be in cents mode and without decimals.
(f.e. 4,5 € → 450, 3.5£ → 350).

6.3.3. Function response

All CashKeeper functions return a 'boolean' type value as a result of the same. If the function can be done satisfactorily, it will be returned **true**. Otherwise, it will return **false** and you should refer to 'ErrorCode' and 'ErrorDescription' properties.

6.3.4. Payments in CashKeeper

CashKeeper is programmed to make payments with the lowest possible coins or banknotes number. Anyway, in some cases, to reduce payment times, the theoretical optimal denominations could not be used (f.e. if a payment of 10 Euro cents must be done having 10 cent coins available, it is possible that CashKeeper use 2 coins of 5 cents if the process of searching for the 10 cent coin takes too many time).

6.3.5. DISABLED event

Due to the CashKeeper processes nature, most of the functions that involve mechanical activity are asynchronous processes. To help the integrator know when a process is finished, and when can be sent new commands, the '**Disabled'** event is trigged to indicate the end of it.

6.3.6. Events

This library raises events one by one. Consecutives events won't be raised until previous event is not finished. So try to process events as fast as possible.

6.3.7. Characteristics

This library is not thread safe, so you must serialize functions calls.

6.4. Start-up *(synchronous process)*

[Involved functions: Connect, GetDevices , StartUp]

To the system start-up, a connection must be opened before with the active CSSI, using the **Connect** method **(***HostIP as string, HostPort as byte, OfficePort as byte, ConnectionType as byte, SecuritySeed as Int32***).** This should be passed the following parameters:

- **HostIP**: where to find the CSSI IP address or network name.
- **HostPort**: listening CSSI port for HOST connections.
- **OfficePort**: listening CSSI port for BACKOFFICE connections.

- **ConnectionType**: type of connection being made (Host or BackOffice)
- **SecuritySeed**: Security code for CSSI secure connections (higher than 100000 and lower than 999999).

6.4.1. Local CSSI

If CashKeeper is connected directly to the HOST machine, the CSSI will automatically start when the **CONNECT** method is called.

6.4.2. Remote CSSI

In the case that CashKeeper is connected to a remote PC (CBS or other), CSSI must be active previously before the **Connect** function is called.

If the Connect function result is satisfactory, the function will return true. Otherwise, it will return false and ErrorCode and ErrorDescription properties should be consulted to find out the cause of the failure.

Once it has been established the CSSI communication, we can check the ID of each CashKeeper connected to the computer where is the CSSI running. In the case of that it has been configured previously and we already know the configuration, this step can be ignored.

The next step would be to invoke the **StartUp** function (ConfigID as byte, Optional Device as Int32) to open the CashKeeper communication ports. Parameters:

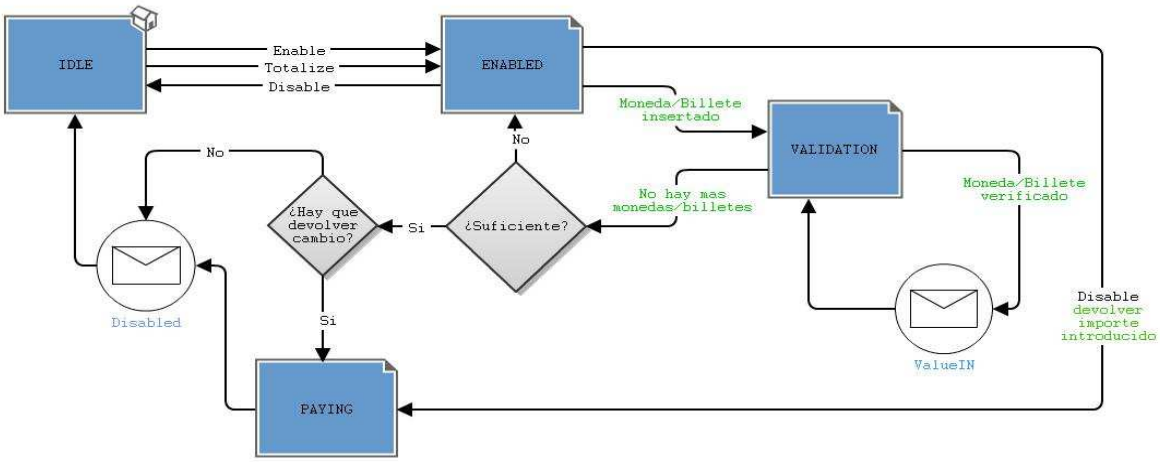
- **ConfigID** (Byte): Indicates the configuration to start. If it does not exist, it is automatically created. You can have as many settings as desired (256 maximum) with the same device. In the case that the same PC has connected more than one CashKeeper, the settings will indicate which of them is going to be connected.
- **Device** (Long): Device identificator. In the case of the first CashKeeper connection in a computer that's runs the CSSI and having more than one connected CashKeeper, this parameter becomes required and we need to indicate with which of them we want to connect.

Once has been performed satisfactorily the **StartUp** function, the CashKeeper state will turn from ALL_CLOSED(0x00) to IDLE (0x01).

6.5. Collection

[Involved functions: Enable, Totalize, Disable, AlternateOperation, DiscardOperation]

The main use of the device is the collection, Cashkeeper can have up to 10 payment operations open at the same time, although only one can be active. By default it works in operation zero. If in the middle of a charge (when it is in ENABLED (0x02) state), we want to change to another operation, can be done using **AlternateOperation** function. The state transition for a collection would be:



We can distinguish two types of colletions:

We know the amount in advance: Through the **Totalize** function, we inform to the CashKeeper that we wish to collect a certain amount and to automatically close the collection.

We want to activate the device, but we do not know the amount: Through the **Enable** function we move CashKeeper to ENABLED state to accept money. Once we know the amount, it has to be used the **Totalize** function to inform CashKeeper the amount to be collected.

If while in ENABLED state you want to abort the collection, the Disable command allows us to give back the introduced amount and return to IDLE state.

Once it is in VALIDATION state (it is validating the money introduced), it appears **ValueIN** event informing the validated amount till this moment and the total amount to collect, in case of being informed, and the current number operation.

Once finalized or cancelled the collection, CashKeeper will sent a **Disabled** event indicating if everything has gone ok and the amounts involved in the process or the error that occurred.

6.5.1. Different operations working

[*Involved functions*: AlternateOperation ValueIN, DiscardOperation]
CashKeeper is able to manage up to 10 collecting operations simultaneously. That means, with introduced cash, it can be 'parked' and another collection can be managed (up to 10). When the device is enabled for the first time, the active operation is the number 0 operation (zero). To be able to change and park the active operation, the **AlternateOperation** method must be invoked, where it will be indicated the operation to be activated.

When you call this method, the value of the current operation is stored and activated the new operation, leaving it as the current operation.

For reviewing at any time if there are any pending operation, the **ValueIN** can be invoked (¡ATTENTION!, not be confused with the ValueIN event), that will return the current amount entered in the specified operation. If the number of operation is not specified, the function will return the introduced amount of the current transaction.

There are times that we may want to "forget" operations with the consequent loss of the introduced amount, this can be done with **DiscardOperation** function.

6.6. Payments *(Asynchronous process)*

[Involved functions: Pay]

6.6.1. Payment of an amount

The **Pay** function should be used to make payments with CashKeeper. CashKeeper must be on IDLE state to start a payment or a payment test.

If CashKeeper is able to make the payment, the function will return true and if 'TestOnly' has been set to false will begin payment, changing the status of the device to PAYING. As coins and/or banknotes are being paid, it will appear ValueOUT event reporting the amount paid. In the same way, whenever it appears or disappears a note on the exit mouth, it will invoque the NoteHeldInBezel event informing us of the notes presence or absence. Once the payment has been completed, it will appear the Disabled event, indicating the process completion and leaving the CashKeeper state to IDLE.

6.6.2. Payment with specific denominations

[Involved functions: PaySpecific]
The **PaySpecific** function should be used to make payments with specific denominations.
The process and conditions that follows a specific payment are the same as a standard payment.

6.7. Emptying operations

6.7.1. Emptied of change *(Asynchronous process)*

There are two ways of exchange emptying to cashboxes.

6.7.1.1. Partial Emptying

[Involved functions: EmptyDeviceSpecific]

The most commonly used mode will be the change empty leftovers. That is, send to cashboxes coins or notes whose current quantity is above desired levels. Used to reset to the initial change.

To carry out this process, the function **EmptyDeviceSpecific** must be used, during the process, CashKeeper will inform us through the ValueToCashBox event.

NOTe: If any denomination present on the exchange is unreported, CashKeeper will send all the units of this denomination to cashbox. If any of the specified quantities of any of the denominations it is higher than the current level available instead, it will be left to the available level.

6.7.1.2. Total Emptying

[Involved functions: EmptyDevice]

To perform a total change emptying towards cashboxes, it must be used the **EmptyDevice** function indicating the device you want to empty.

Once starts the emptying process (partial or total), the device will go into FLOATING (0x04) or EMPTYING (0x05) state respectively and it will report the sent amount via ValueToCashBox event.

Once finished emptying (partial or total) will appear the Disabled event indicating the process completion and leaving the device into IDLE state.

6.7.2. Emptying cashboxes *(synchronous process)*

[Involved functions: EmptyCashBox]

Even having a cashbox removal detection system, the cashboxes emtying, as being a completely manual task and can be done with the device off, CashKeeper should be informed that that task has been made. **EmptyCashBox** function must be used.

6.8. Change refill

To fulfil of change effectively the device, it will be necessary the use of **ActivateRefillMode** function. Once this function is invoked, on the next **Enable** function (a Totalize when CashKeeper is in IDLE state involve an enable to all purposes) and only on the next, all notes introduced will be stored as change. In the case that this could not be possible, those will be sent back. Referring to the notes, there are two exceptions, the first is that the note is deteriorated, CashKeeper is not able to send it back and the second, it is done when the notes recycler is full.

6.9. Collecting information from the cash system

To use all the CashKeeper potential, it must be known the amount stored on the cashboxes and change recyclers.

6.9.1. Cash available for change

[Involved functions: GetCurrentLevel]

To know the levels available for change, **GetCurrentLevel** is the function it has to be used.

The levels information of CashKeeper is given in units of the denomination, not in quantities (25 coins of 2 cents, not 50 cents).

6.9.2. Cash on cashboxes

[Involved functions: GetCashBoxLevel]

To know the levels on the cashboxes, you must use the function **GetCashBoxLevel**, but remember, as CashKeeper can operate with different currencies, it is impossible to give the "money" on the cashboxes, so it gives the quantity, the denomination and the currency of the cashbox content.

6.10. The 'broken' cent

[Involved functions: GetBrokenCents]

In some currencies, CashKeeper is not able to manage all the amounts/denominations. Even the system limitation managing some denominations CashKeeper can pay all the amounts except some VERY SPECIFIC, in these cases, what it does is to pay one more cent. The application can manage it invoking the **GetBrokenCents** function.

For example, in EUROS, CashKeeper cannot manage 1 cent denomination. Even the system limitation of containing 1 cents coins, it exists Only two amounts absolutely impossible to be paid: 0,01 € y 0,03 €.

There is also another possibility, it exists the *BCMaxCoins* and *BCMinValue* properties that allow reduce the number of coins used at the expense of "give away" a cent from time to time.

For example: In a butcher's shop, the amounts tend to be high and at the same time widely dispersed (not rounded). This is translated into an important consumption of small coin.

If we establish BCMinValue = 1000 y BCMaxCoins = 4. This means that in any collection of an amount greater or equal to 10 Euros in which the return of the change involves more than 4 coins, the system consider if returning one more cent will reduce the number of coins, if it is really reduced, It will give back an extra cent and the broken cent will be established.
Let's suppose a 10,01 € operation, we pay with a 20,00€ note and a 2 cents coin to get back a 10,00€ note and 1 cent
In a standard situation, the system would return: 1 x 5€, 2 x 2€, 1 x 0.50€, 2 x 0.20€, 1 x 0.05€, 3 x 0.02€ (total amount 10.01€)
With the described configuration the system would return: 1 x 10€, 1 x 0.02€ and point 1 cent of brokenness.

6.11. Working with more than one currency

CashKeeper allows you to mix notes of different currencies collection (EUR, GBP, USD, etc.). Change is always returned in the main currency. The steps to follow to use this function are:

- Have a unit ready to work with different currencies.
- "Enable" the currency, we must provide the change from the main currency by using the **SetCCChange** function**.** This value is not maintained in the configuration, so it is necessary to be informed whenever we initiated CashKeeper.
- Enable the different denominations of this currency.

6.12. Barcode

CashKeeper is able to read barcodes on the CK900/v references; those must be "Interleaved 2 of 5" and be centred on the paper (for more details, see annex 10). To enable the reading, just inform to the **BarCodeLength** property the characters number of the code.

Once CashKeeper reads a barcode, appears BarCodeRead event informing about the code and we must answer to the event with RejectPending function in the case of refusing the ticket or with AcceptPending function in the case of sending the ticket to the cashbox. In this last case, CashKeeper will send a BarCodeStored event indicating that it has been stored properly.

6.13. Other functions

6.13.1. Cleaning the coin validator

CleanBulk function performs a coin validator cleaning. Depending on the Complete parameter, the process can take up to 7 seconds.

6.13.2. Modify the exchange coins counters

With **ForceCoinLevel** function, it can be modified the change coins counters. It is been used on situations of coins recycler substitution.

6.13.3. Upgrade Firmware devices

With **CheckSmartFirmwareFile** function**,** allows verifying if the upgrade file is compatible with the device we want to upgrade, as well as indicating the version.

With **GetFirmWareVersion** function we can obtain the versions that we have currently installed on the devices.

UpdateSmartFirmware function performs the real upgrade.

WARNING: Updating the firmware of a component is critical as well as long runtime functionality. Ensure the stability of the system before proceeding to update a firmware because in case of failure (power drop, disconnection, etc.) may cause that the component is completely UNUSABLE and UNRECOVERABLE.

6.14. System shut down

[Involved functions: CloseAll, Disconnect, Terminate]

We can distinguish two situations:

When the CSSI is located on the same computer that is running the application usually CSSI is being closed and the involved function is **Terminate**.

When it refers to different computers, we should leave the CSSI open (CSSI can not be booted on a remote computer), for this the devise is released with **CloseAll** function and then releases the connection with **Disconnect** function.

6.15. System parametrization

6.15.1. Denominations inhibition

[Involved functions: SetInhibitState, GetInhibitState

CashKeeper allows you to define within the recognized denominations, which will be accepted, which will be accepted but not used on payments and which will be rejected. For this, we have **GetInhibitState** function to obtain actives inhibitions and **SetInhibitState** function to establish active inhibition.

6.15.2. Change protection

CashKeeper uses two systems to protect us from possible failures of change.

6.15.2.1. Low level alerts

The system has a **LowLevel** event that will inform us when any of this denomination is below recommended levels. It can be distinguished between coins and notes.

Coins have a common parameter for all of them (it is important to try to maintain a balanced level of coins) **CoinsLowLevel** property, it also can appear (even still above the level) if any denomination is in a big disadvantage respect to the rest.

Notes are exclusively controlled by **SetLowLevelNotes** function, it which is specified the level for each denomination.

6.15.2.2. Note Level Protection

Note Level Protection is a complex protection system for change notes. The system seeks to control returned change transactions with high value notes.

¿How does it work? This protection system is divided in 4 steps:

1. Note value recognition
   If the value of the ticket introduced in operation is below the value reported on NLPStartValue property, the operation will proceed normally. Otherwise proceed to the next step.

2. Total transaction validation
   If the total value of the transaction has not been informed (through 'Totalize(…)' function), the note is returned. If the total has been informed, proceed to next step.

3. Change enough
   It is being determinate the theoretical change that should be returned in case of complete the operation at this time. If necessary change is below the limit of **NLPPercentValue** property the process continues, otherwise proceed to the next step.

4. Change to deliver out of limits. If the **NLPAutoProtect** property is in true, the note will be rejected, otherwise, it will appear **ProtectedValueNote** event. Once started this event, the note is temporary retained on the system until receiving a response indicating what to do with the note. The answer could be

'**AcceptPending**' to accept the note or '**RejectPending**' to be rejected. If no answer is received in 60 seconds, the note will be rejected.

6.15.3. 'Burglary' protection

CashKeeper has a protection system to prevent massive outflows by way of payment.

With ***MaxPayout*** y ***PayoutInterval*** properties it can be set a maximum amount of payment (*MaxPayout*) to be paid in a determined time interval (in minutes) (*PayoutInterval*).

F.e. Established MaxPayout = 10000 and PayoutInterval = 20, it means that in 20 minutes we can make as many payments as needed while the total sum of them does not exceed 100 Euros. Once you reach the 100 Euros or the amount of the next payment make to overcome the 100 Euros, the system will reject the payment with the error code nº 4.

6.15.4. Other parametres and properties

6.15.4.1. DisableAutoText (Int32) *('Appearance' property)*

'*DisableAutoText*' property will be useful to be able to control the display content or, leave that system autonomously controls these contents.

(NOTE: the available characters from VFD display will be 20 per line when using the standard letter size and 20 using the double font size)

To have access to specify the display contents it will be used **Display** function.

6.15.4.2. LightTime (Int32) *('Appearance' property and 'Lock' property)*

LighTime property, it allows to specify (in milliseconds) the time we wish that the light on coin cashbox, is switched on while making a payment (via change, coin reject or direct payment). The default value is 1000 (1 second).

6.15.4.3. RejectIfClosed *('Behavior' property)*

RejectIfClosed property will determine if the rejection gate of the coin validator is left in the open position, on disabled device (IDLE 0x01) or out of service (ALL_CLOSED 0x00).

6.15.4.4. CancelLowLevel *('Behavior' property)*

This property, will determine if the system, will send events when notes and/or coins are in low level.

6.15.4.5. CancelValueEvents *('Behavior' property)*

Like the previous, CancelValueEvents will determine if the system will send events ValueIN, ValueOUT y ValueToCashBox when receiving incomes, making payments or sending the amount to the cashboxes.

6.15.4.6. ConfigID *('ID' property)*

ConfigID returns the active configuration (the one informed on **Startup** command). It could be used to distinguish between devices when we want to control more than one device simultaneously.

6.15.4.7. LogDisable

Enables or disables logging of incoming notes and coins.

- Not implemented -

6.15.4.8. CoinCashBoxDetect

This property will allow us to enable or disable presence detection of coins cashbox before performing operations with the device. Defect value 'False'=0.

6.15.5. Change the language

It is possible to change the language that CashKeeper® deals with the user. There are two predefined languages that sets in which languages will be sent all the errors descriptions and warnings, as well as the text shown on the CashKeeper® display when is set on automatic mode.

The two predefined languages are Spanish (ESP) and English (ENG). By default the system is set up in Spanish. To modify it, **SetLanguage** function must be used with the wished language code.

By the way, the system has a functionality to modify as you want, the text shown on the CashKeeper® display when it is set on automatic mode. To do this, **SetDisplayText** function should be used. The message code you wish modify must be indicated and the new text. This modification is not permanent, so every time that the system is started-up it must be specified again.

6.16. Error resolution

Some of the CashKeeper errors are critical and they limit or end the system use. In these cases, it is recommended to use **Reset**, **CloseAll/StartUP** functions to solve the problem and reset the service because those errors, they involve the disconnection of one of the devices (coin validator, notes validator, etc.) and consequently the loss of communication with the Host.

To avoid conflicts with Host communications, before disconnecting any device, communication ports should be closed through **CloseAll** function and, once reset the device and the power, use **StartUp**.

On the other hand, some errors (specifically identified) will require to send the **Reset** command once has been solved the problem.

6.17. More than one simultaneous device management

It is possible to manage more that one CashKeeper® devices simultaneously and connected to the same computer.
The first step would be identify how many and which are they. For this purpose, exists **GetDevices** function that gives a devices list (separated by coma) connected at the same time that is shown on each display.
Once identified the device to be connected, we could connect with **StartUp** function specifying the device ID and then we will normally operate.
Each CashKeeper that want to work simultaneously needs its own CKeeper object as well as its own TCP/IP connectors.

7. CKeeper integration difference between Android and Windows versions

7.1. Function output parameters

The main difference of using the library 'CKeeper for Android', are the output parameters.

All the functions that had parameters by reference (Output values), now return an object of type 'CkResult'. This class only contains the Response attribute. Read-only property that reports the output parameters of last executed function.

You can access each parameter thought get method and parameter name ('.get ("parameter name")').

The name parameter is related to executed function, thus being a value predefined by the library 'ckeeper.dll' explained above.

The parameter name of the value passed by reference of a function of the library 'ckeeper.dll', matches the name of the parameter of the '.get' method of the 'CKeeper for Android' library.

Example:

Function in ckeeper.dll:

```
Function GetBrokenCents(ByRef Value As Long, ByVal resetValue As Boolean)
        As Boolean
```

In Android:

```
Public CkResult getBrokenCents(boolean resetValue) { ...}
ckSocket = ckConfiguration.getSocket();
CkResult ckRes = ckSocket.getBrokenCents(false);

Value val = ckRes.get(”Value”);
```

8. USE *CashKeeper®with Direct Method*

Below are the procedures for dealing CashKeeper ® with the direct method:

8.1. Prior considerations

The operations in the direct method are almost identical to the CKeeper method, so if in any doubt, see chapter 6. It is basically distinguished by some changes in properties only available Keeper method and some methods only available in the direct method.
In the annexes 3, 5 y 6 you will find the list of functions, properties, and events with their corresponding code.

8.2. Messages format

8.2.1. Commands

The commands structure sent to the CashKeeper ® service (CSSI) must have the following format:

Begin with the character '$' ascii 36.
Followed by the message code.
If you have parameters, these will be preceded by a vertical bar '|' ascii 124
End by the character '#' ascii 35.

Example
Send the command **Terminate**.
$48#

Send a **Disable** with the parameter payback in 1
$9|1#

8.2.2. Answers

The responses structure to commands sent to the service has the same format

Examples:
The sequence of command and response to enable device (**Enable** 14 command) would be:

We send:
← *$14#*
We get:
… with affirmative response
→ *$14|1#*
… with negative response
→*$14| 0|37| Device not available #*

(in negative responses, the parameters that follow the response are the error code and error description)

Sequence of command and response to request for the available quantity (**GetCurrentLevel** 19) in change of the 1.00€ and 2.00 € denominations
We send:
←$19|100,200#
We get:
→$19|1|30,25#

(It indicates that we have 30 coins of €1 and 25 of €2).

8.2.3. Events

The events structure depends on each one of the events, following the standard format established for all communications:

The following example of STATECHANGE event indicates that we are in **ENABLED** state
→$106|2#

8.3. Start-up *(synchronous process)*

For the system star-up, the implementation of the service must first be started (CKeeper.exe) and open a communication channel with the service (CSSI). So before, it should be decided if the type of communication is to have complete control over the device (HOST connection type) or, only, for consultation and configuration functions (OFFICE connection type). The difference lies in the port you have to deal, because there is a specific port connection for each of the types. Once established the communication it must be negotiate the key to access to the service.

8.3.1. Negotiate the access key to the service

(NOTE: This is the only procedure that does not follow the standard of communication COMMAND → RESPONSE COMMAND)

To negotiate the access key to the service, once established communication with the service, the command 47 must be sent **(StartCNT)** to indicate to the service the beginning of the negotiation:
←$47#

At the same time, the system service will response with the command 57 **(CNT)** to associate an additional parameter, that will be the basis for the access key:
→ $57|number#

Now, with the number that the service provided us, we will have to calculate the key, adding the value of the 'Security Seed' parameter (the same that we set up in the service implementation). F.e. 140806.
Access Key = 140806 + number (let's suppose number = 1620168189)
Access Key = 1620308995

We will now send the access key to the service, with **command 57 (CNT)**:
←*$57|**1620308995**#*

If the value of the key accords with which the service has been estimated (knowing the 'Security Seed' or 'Seed security' value previously), the service will answer with the command 58 **(CNT_OK)**: and the communication version number.
→*$58|5#*

Otherwise the service will be limited to closing the communication.

Summary of steps to begin communicating with the service (CCSI):

1. Run the service application (CKeeper.exe)
2. Open the sockets communications channel (TCP/IP protocol) on the port HOST type dedicated connections or on the port OFFICE type dedicated connections. By default, the port HOST type is **8001**, for OFFICE type connections is **8002**.
3. Negotiate the access key. By default the service 'Security Seed' parameter is established in **100001**.

Once communication is established and negotiated the access key to the service, we can initiate communication with the CashKeeper ® device. For this purposes the command 39 **(START_UP)** must be called:
←*$39|0#*

If the answer is satisfactory, the CashKeeper state will move from ALL_CLOSED (0x00) to IDLE (0x01). This state change will be communicated by STATE_CHANGE event (code 106).

8.4. Shut down the system

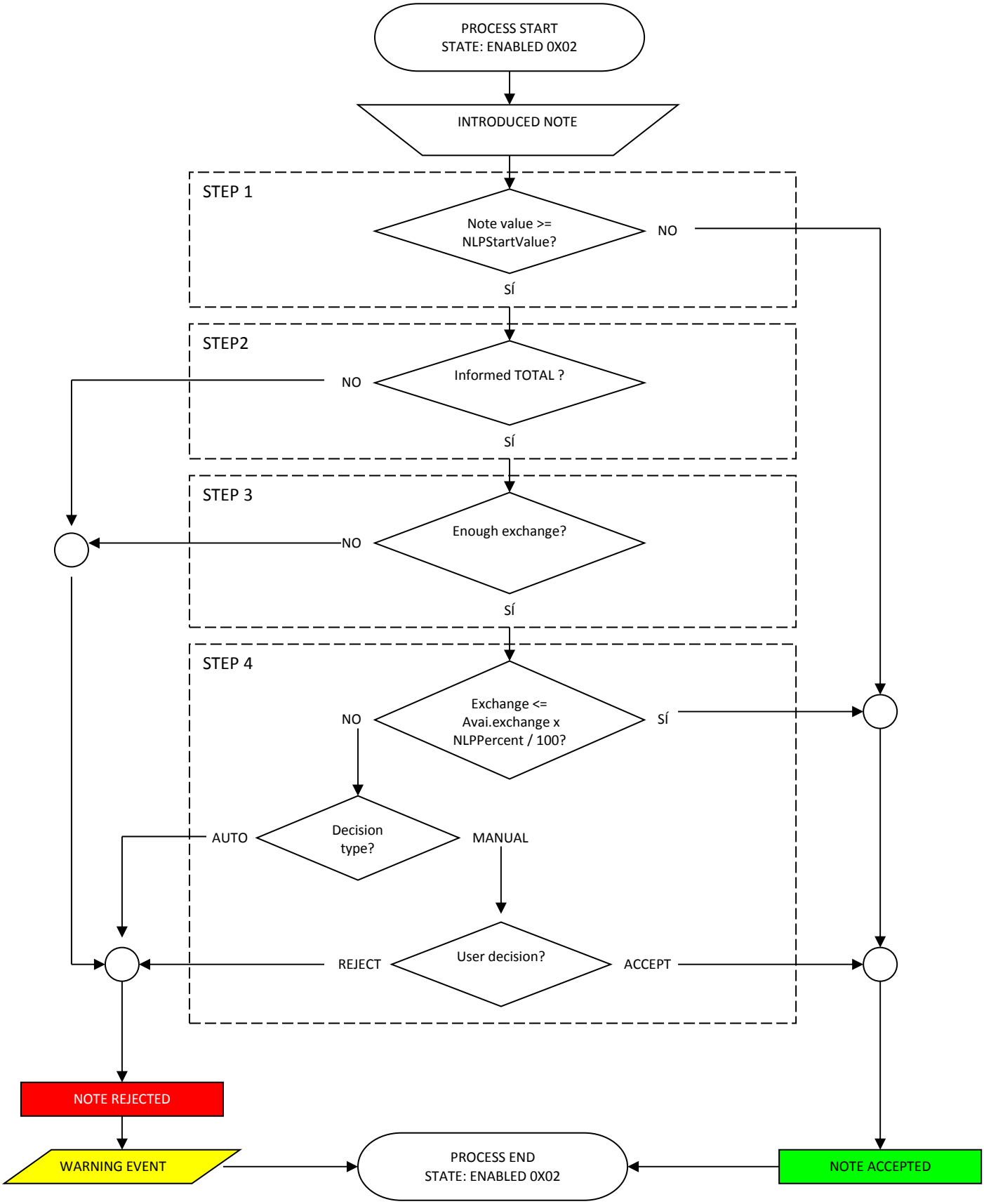
[Involved commands: 7 – CloseAll, 48 - Terminate,
To close communications with CashKeeper and close the service should be used **Terminate** (48) command. It will close the communication ports with CashKeeper leaving it in ALL_CLOSED (0x00) state and then it will close CSSI.
To close communications with CashKeeper, and keep the service on for a posterior connection, it should be used **CloseAll (7)** command. This will only close the CashKeeper communication ports leaving it on ALL_CLOSED (0x00) state.

WARNING: The service is prepared for, in case of a loss of comunication with the client (socket), it will close communications with the CashKeeper ® device whatever its state.

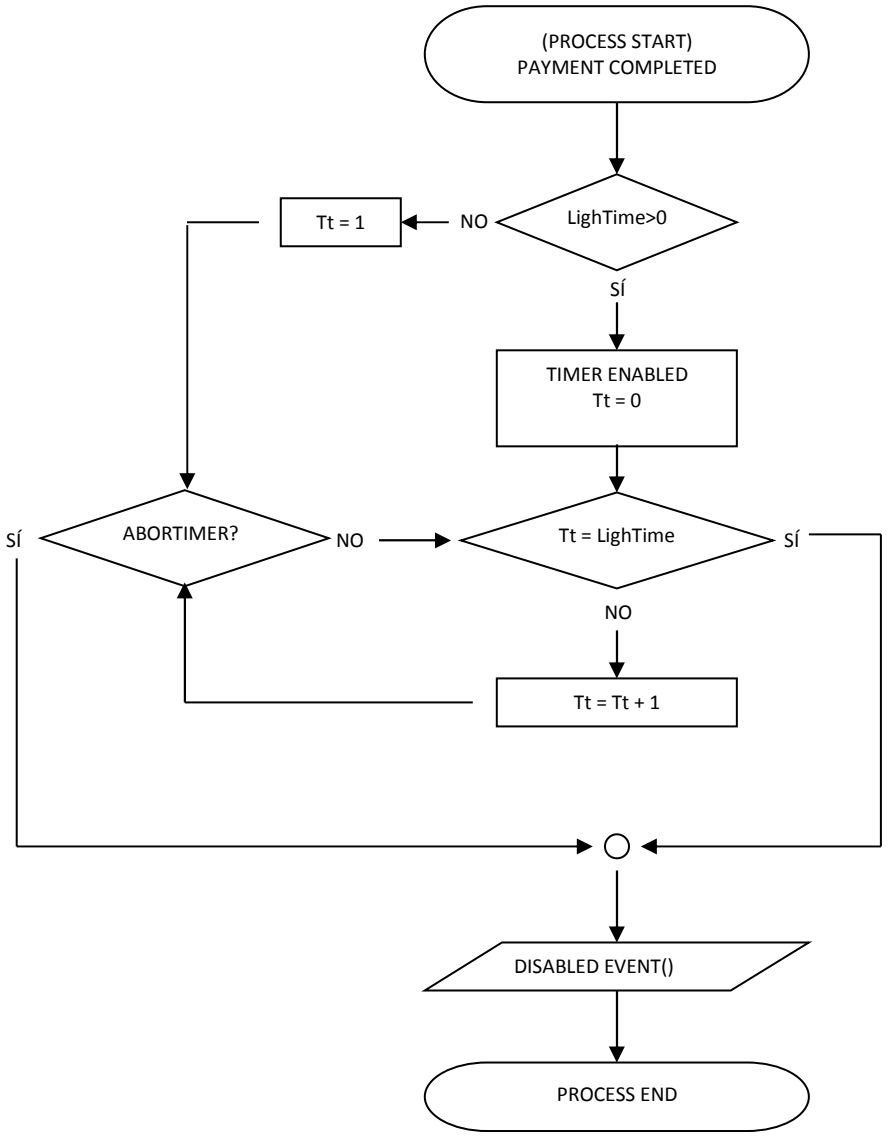
NOTE: Even and return an error from the CloseAll (7) command, the service will seek to close communication with the device and, therefore, the state change to ALL_CLOSED (0x00) will always be effective.

ANNEX 0. Images

- **Figure 1**



- **Figure 2**


(PROCESS START)
PAYMENT COMPLETED
LighTime>0
NO
Tt = 1
SÍ
TIMER ENABLED
Tt = 0
Tt = LighTime
SÍ
NO
ABORTIMER?
SÍ
NO
Tt = Tt + 1
DISABLED EVENT()
PROCESS END

ANNEX 1. Constants description

LC_DeviceType:

- *0 = LCDT_900*
- *1 = LCDT_1000*

LC_Smart_Devices:

- *0 = LCS_CoinHopper*
- *1 = LCS_NotePayout*

LC_CashBox_State:

- *0 = LCCBS_OK*
- *1 = LCCBS_Almost_Full*
- *2 = LCCBS_Full*

LC_Smart_Devices:

- *1 = LCSD_CoinHopper*
- *2 = LCSD_NoteFloat*
- *3 = LCSD_NotePayout*

LC_Logical_Devices:

- *1= LCLD_CoinValidator*
- *2 = LCLD_CoinDispenser*
- *3 = LCLD_NoteRecycler*

LC_CashBox:

- *1= LCCB_Coins*
- *2 = LCCB_Notes*
- *5 = LCCB_All*

LC_ConnTypes:

- *0 = HostMachine*
- *1 = BackOffice*

Idiomas:

- *1 = Español*
- *2 = English*

ANNEX 2. Errors and warnings

ERRORS

Code	Message
1	Uninitialized communication
2	Device busy. The command cannot be processed
3	It is not possible to enable the device
4	Security lock (MaxPayout exceeded)
5	Fraud attempted
6	Device out of service
7	Coin jammed in output device
8	Note jammed on secure zone
9	Note jammed on insecure zone
10	Note jammed
11	Device/s unconnected
12	Error on loading configuration data
13	Error in output data
14	Sockets Communications error
15	Error COM port opening
16	Error COM port closing
17	Error on devices reset
18	Unexpected device reset detected
19	Ports configuration error
20	Error initializing devices
21	Exceeded limit on coin cashbox
22	Stacker full
23	Payment refused by the operator
24	The value must be between 100.000 and 9.999.999
25	Enable devices error
26	SSLCash not found. Unable to initiate communication with CashKeeper
27	Exceeded time to find the necessary coins for the payment. Retry the payment
28	Error when verifying the note value for payment. Retry the payment.
29	Recovered communication error
30	Communication with device error
31	Internal device error
32	There is no enough amount for the payment
33	Notes CashBox removed
34	Device_ID value not supported
35	Time limit exceeded for the command execution
36	Device not enabled. The command cannot be processed
37	Device not available. The command cannot be processed
38	Not supported mode
39	Not supported value
40	Not supported values: number of denominations and values distinct
41	The device is already processing this command
42	An unexpected error occurred when executing the command

43	Incorrect device drivers
44	Error reading EEPROM
45	There are no notes and coins entry logs
46	ReadOnly Property
47	Config doesn't exist. You must specify witch device must be setup
48	Specified Device doesn't exist
49	Device already initialized
50	Generic system error
51	CountryCode is not supported
52	Note path open
53	Invalid Path or No enought privileges
54	Coin CashBox removed
55	Coin recycler calibration error
56	Use of reserved characters (# | $) on DISPLAY
57	Country Code not supported
58	Note held in bezel
59	Coin recycler is over his capacity, realize payments or do a full or partial empties.
60	Coin recycler top lid is open
61	Device type can not be detected
62	Coin validator trashdoor can't be closed
63	Firmware file corrupted.
64	Firmware file doesn't much device type.
70	Time of payment exceeded
90	Communication with CashKeeper error (CSSI)
92	Exceeded time limit for CashKeeper response (CSSI)

WARNINGS

Code	Message
1	Possible jam on notes cashbox
2	Unknown coin accepted
3	The last fundraising operation is not completed successfully
4	Error detected in coin validator. Possibility of non-counted coins
5	Recoverable fraud attempt
6	Note dispensed at power up
8	Note refused due to system security: Change limit exceeded
9	Note held in bezel
10	Note refused due to system security: Total value not informed
11	Note refused due to system security: Not enought change
12	Coins recycler full
13	Coins cashbox full
14	Coins cashbox is 90% of its capacity
15	Coin recycler top lid is open
16	The trashdoor of the coin validator is trapped and cannot be closed
17	Coin jam in validation area or validation sensor needs cleaning
18	Coin sensor blocked in coin validator

19	Coin validator, disk jammed
20	Extra notes are going to be stacked
22	Notes cashbox full
23	Notes cashbox is 90% of its capacity
24	Emptying Note Recycler to keep unit working
25	Notes cashbox has been removed
26	Coins cashbox has been removed
32	Error saving amounts of cashbox
33	Error return automatic exchange: [reason]
34	Note Recycler full, following notes will go to CashBox
35	Unexpected error in note reading
36	Pay out cancelled by the operator
50	Possible anomaly detected. Verify note cashbox contents
55	Coin recycler calibration error
108	Note jammed in safe area
109	Note jammed in unsafe area
110	Note jammed
111	Note Reader not working
112	Coin validator is not working

ANNEX 3. Function list. Definition

NOTE: All values related to denomination values or amounts should be sent and returned in cents.

- **(0) AbortTimer**
  It sends an order to cancel the 'expected' time established for the collection of change (*LighTime* property).

- **(3) AcceptPending**
  After trigging one of the *ProtectedValueNote*, *MaxCoinsWarning*, *BarCodeRead* event, should respond to the event with *AcceptPending* to accept or *RejectPending* to reject.

- **(67) ActivateRefillMode() as boolean**
  This method enables 'exchange filling' mode only during the next 'Enable()' (either direct or indirect using'Totalize'). This method will cause ALL incoming notes are sent to the Exchange store or returned.

- **(4) AlternateOperation (*Operation as Byte*) as boolean**
  Toggles the current operation with other, leaving aside the values of the current operation. The number of the active operation when the system starts is '0'.

- **(6) CleanBulk (*Complete as* boolean) as boolean**
  It starts a cleansing cycle for the coins entrance device.

  Complete:
  - false (short cleansing cycle)
  - true (long cleansing cycle)

- **(7) CloseAll ( ) as boolean**
  It closes the channels of communication with the device. Although closure procedure could result in an error, shall be considered, in all levels, that the channels have been closed.

- **(5) CheckLevels (LowLevelActive As boolean, HopperFull As boolean, CoinCashBoxState As LC_CashBox_State, NoteCashBoxState As LC_CashBox_State) as boolean**
  It performs a general check of exchange notes and coins levels and of the cashboxes levels, returning values on each of the parameters:

Parámetro	Valor	Significado
LowLevelActive	False	There are no denominations in alarm for low level
LowLevelActive	True	There are denominations in alarm state by low level. Check levels through **GetCurrentLevel** function
HopperFull	False	Coins recycler level OK
HopperFull	True	Coins recycler full. Must be emptied at least partially.
CoinCashBoxState	LCCBS_OK	Coins cashbox level OK
CoinCashBoxState	LCCBS_AlmostFull	Coins cashbox level up to 90%
CoinCashBoxState	LCCBS_Full	Coins cashbox level exceeded. Must be emptied.
NoteCashBoxState	LCCBS_OK	Notes cashbox level OK
NoteCashBoxState	LCCBS_AlmostFull	Notes cashbox level up to 90%
NoteCashBoxState	LCCBS_Full	Notes cashbox level exceeded. Must be emptied.

- **(55) CheckSmartFirmwareFile (FullPathFile as string, Device_ID as LC_Smart_Devices, FirmwareVersion as string, DataSet as string) as Boolean**
  It checks the firmware version and dataset contents on the specified file in '*FullPathFile'*, as the compatibility with '*Device_ID*' specified. Should be noted that path is related to the CSSI location.

- **(9) Disable(PayBack As Boolean) As Boolean**
  It disables the device leaving him in a rest state (not suitable for cash entry) returning, in the case, the amount.

Parámetro	Valor	Significado
PayBack	False	It does not return the entered amount.
PayBack	True	It returns the entered amount.

- **(52) DiscardOperation(Operation as byte) as Boolean**
  It discards the indicated operation, leaving the pending operation values to 0 (zero). WARNING! This function does not cause the refund of the introduced amount.

- **(68) DiscardPayOperation() as Boolean**
  It discards the interrupted payment operation. WARNING! This function does not cause any refund.

- **Disconnect()**
  It disconnects the CashKeeper connection. If the state is different to ALL_CLOSED, CSSI will execute automatically **CloseAll**.

- **(10) Display(*Line1 as String, Line2 as String, UseBigFont*) as Boolean**
  It sends a message to the VFD display. Each of the lines cannot contain more than 20 characters. In case of UseBigFont (double height text) is set to true, the text contained in Line2 will not be considered.
  NOTE: The characters '#', '|' and '$' are reserved and its use is restricted

- **(11) EmptyCashBox(*Device_ID as LC_CashBox*) as Boolean**
  Reset (=0) the value counter stored on the device cashbox identified as *Device_ID*. It should be used whenever cashboxes are emptied.
- **(11) EmptyCashBoxEx(CashBox As LC_CashBoxes) As Boolean**
  Extends function (11) EmptyCashBox, Allow user to empty Notes Cash Box one by one. (Only works on CK1000)

- **(12) EmptyDevice(*Device_ID as LC_ CashBox*) as Boolean**
  Empty completely the change to the device cashbox identified as *Device_ID*.

- **(13) EmptyDeviceSpecific(*Denoms as String, NumberToKeep as String*) as boolean**
  It sends to cashbox the amount corresponding to each of the denominations specified in the '*Denoms*' list (separated by commas) to leave as change the corresponding values specified on the '*NumberToKeep*' list (separated by commas respectively).

- **(14) Enable() as Boolean**
  It enables the device to let it in cash receiving state.

- **(15) ForceCoinLevel(Value as Integer, Level as Integer, AddToCurrentLevel As Boolean) as Boolean**
  The coins storage and delivery device, in exceptional cases it can be massively charged by top via (without using the coin input device). Only in these cases, it will be necessary to use this method, to inform the amount of each denominations introduced on the device.

  WARNING: An irresponsible use of this function can lead to problems in the coins count and payments.

- **(66) GetAllProperties**
  Returns all the properties in the following order:
  BCMAXCOINS
  P_BCMINVALUE
  CANCELLOWLEVEL
  CANCELVALUEEVENTS
  CONFIGID
  COINCASHBOXDETECT
  DISABLEAUTOTEXT
  LIGHTTIME
  LOGDISABLE
  MAXCOINS
  MAXPAYOUT
  MINFASTIN
  NLPAUTOPROTECT
  NLPPERCENTVALUE
  NLPSTARTVALUE
  NOTELEVELPROTECTION
  PAYOUTINTERVAL
  REJECTIFCLOSED
  COINSLOWLEVEL
  DEVICETYPE
  BARCODELENGTH
  CRITICALBEHAVIOR

- **(16) GetBrokenCents(*Value as INT32, ResetValue as Boolean)* as boolean**
  It stores in *Value* the amount of cents more that have been returned in change due to rounds to the rise in values of 1 and 3 cents. ResetValue parameter, indicates if is wanted to reset the counter.

- **(17) GetCashBoxLevel(CountryCode As String, Values As String, Levels As String) As Boolean**
  It returns on Levels variable, the denominations levels on cashboxes specified on Values.
  CountryCode: Currency which will be values referenced (EUR, GBP, USD, etc.).
  Values: Denomination value, several can be specified separated by comma.
  Levels: Amount of the requested denomination, in case of several, it returns a list of amounts separated by comma in the same order as the requested.

- **(77) GetCCChange(CountryCode As String, Change As Int32) As Boolean**
  On the *Change* variable, it returns the amount of the currency exchange *CountryCode* into the main currency.

- **(86) GetCCDetails(CC As String, Multiplier As int32, Decimals As int32) As Boolean**
  Return multiplier and the number of decimals to be displayed. This function allows us to make our application aware of working currency.
  Ex:
  Euro, multiplayer is 100, as amounts are cents of Euro, the number of decimals is 2.
  Chilean peso, el multiplier is 1, as they don't use decimals, the number of decimals to display is 0.

- **(78) GetCCDenominations(CountryCode As String, Coins As String, Notes As String) As Boolean**
  Returns the values list of the different currency denominations, deposited in Coins the list of the coins values separated by comma and in *Notes* the list of the notes values separated by comma.
- **(24) GetCounters(*CoinCounter as Int32, NoteCounter as Int32*) as Boolean**
  On *CoinCounter* and *NoteCounter* values, it returns the value of absolute counters (total input number of units) of coins and notes from device manufacture.

- **(38) GetCountryCodes(MainCC As String, OtherCC As String) As Boolean**
  On *MainCC* variable, it returns the main currency code (EUR, GBP, MXN, USD, etc.) and on ***OtherCC*** variable, the list of other supported currencies separated by comma.

- **(19) GetCurrentLevel(Values as String, Levels as String) as Boolean**
  On *Levels* variable, it returns the Exchange quantity contained on the denominations specified on *Values* variable separated by comma.
  Attention. When a device is in error, device contents could be not shown.

- **(72) GetDevices(DeviceList As String) As Boolean**
  Returns the list of CashKeeper IDs connected to the computer where it resides the CSSI.

- **(54) GetFirmwareVersion(Device as LC_Logical_Devices, FirmwareVersion as string, Dataset as string) as Boolean**
  On *Firmware Version and Dataset* variables, it returns the firmware version and the actual dataset from the device specified on Device.

- **(21) GetInhibitState(CountryCode As String, Values As String, Inhibits As String) As Boolean**
  On *Inhibit* variable, it returns a list of states of acceptance or inhibition of the specified denominations list in *Values* of the currency *CountryCode*.

  Inhibit = 0 : the denomination is accepted.
  Inhibit = 1 : the denomination is accepted but not paid.
  Inhibit = 2 : the denomination is rejected.

- **(76) GetLastIN(CountryCodes As String, AmountsOrDenom As String, Detail As Boolean, Operation As Byte) As Boolean**
  Gets the form of payment used in the last payment made returning the list in *AmountsOrDenom, Detail* parameter is used to select the mode being true detailed and false accumulated.
  Accumulated: Returns the total amount in each currency used in payment.
  Detailed: Returns the list of denominations used in the payment. If more than 50 denominations, returns the accumulative value and modify the *Detail* parameter.

- **(79) GetLastLevels(ConfigID As Byte, Denoms As String, Qtys As String, CCs As String) As Boolean**
  Obtains last known levels. Useful to get last known levels in case of error during start-up.

- **(8) GetLowLevelNotes(Values As String, Levels As String) As Boolean**
  On *levels* parameter, it returns the minimum level list (separated by comma) of the denominations list provided on *Values* parameter.

- **(82) GetNetworkParams(HostName As String, DHCPEnabled As Boolean, IP As String, Gateway As String, Mask As String, DNS1 As String, DNS2 As String, MasterPort As int32, OfficePort As int32, Seed As int32) As Boolean**
  Returns network parameter values.

- **(23) GetMaxLevel(Values as String, Levels as String) as Boolean**
  On *levels* parameter, it returns the maximum level list (separated by comma) of the denominations list provided on *Values* parameter.

- **(74) GetUnikeID (ID as string) as Boolean**
  On *ID* parameter, it returns a unique identifier for CashKeeper® device.

  NOTE: This identifier may change depending on the hardware updates

- **(29) Pay(Value as Int32, TestOnly as Boolean) as Boolean**
  Starts a cash exit process specified by the *Value* parameter. The denominations distribution in coins and notes is done automatically.

  If *TestOnly* variable is set into TRUE, only will be a test done if it is possible to pay this amount.

- **(30) PaySpecific(Denoms as String, NumberOf as String, TestOnly as Boolean) as Boolean**
  Starts a cash exit process with the denominations specified on 'Denoms' (separated by commas) in the quantities of each specified in 'NumberOf' (separated by commas).

  If TestOnly variable is set into TRUE, only will be a test done if it is possible to pay this amount.

- **(87) PaySpecificEx(Denoms as String, NumberOf as String, TestOnly as Boolean) as Boolean**
  Starts a cash exit process with the denominations specified on 'Denoms' (separated by commas) in the quantities of each specified in 'NumberOf' (separated by commas).

  If TestOnly variable is set into TRUE, only will be a test done if it is possible to pay this amount.

  This function difers from the previous one on the way this function uses MaxPayout and PayoutInterval properties.

- **(32) RejectPending()**
  After starting any of *ProtectedValueNote*, *MaxCoinsWarning*, *BarCodeRead* events, should respond to the event with *AcceptPending* to accept or with *RejectPending* to reject.

- **(33) Reset( ) as Boolean**
  Reset the device and restart.

- **(26) ResetCounters(Device_ID as LC_CashBox) as Boolean**
  Reset the coins or notes absolute counter.

- **(75) SetCCChange(CountryCode As String, Change As Int32) As Boolean**
  Set the specified currency exchange in *CountryCode* related to the main currency. The operation to get the payment is ((*Change* * Note value) / 1000) without decimals.

- **(81) SetDateTime(Year As Integer, Month As Integer, Day As Integer, Hour As Integer, Minute As Integer, Second As Integer) As Boolean**
  Set Date and Time on devices equipped with Smart CK Board (models CK900e and CK1000). On USB model, this function returns always true.

- **(71) SetDisplayText(Code as byte, NewText as string) as Boolean**
  Modifies the automatic display text identified with *Code* variable for the text sent in *NewText*.

  I.E.
  Originally the17 text code for the display contents the text 'TO BE PAID:', If we would like to modify this text to, i.e., 'TOTAL :', we should invoque the function…

  SetDisplayText (17, 'TOTAL :')

- **(36) SetInhibitState(CountryCode As String, Values As String, Inhibits As String) As Boolean**
  Set the inhibition state (acceptance permission) of the specified denominations on *CountryCode* / *Values*.

  Inhibit = 0 : accepted denomination
  Inhibit = 1 : accepted denomination, but not used on payments.
  Inhibit = 2 : rejected denomination

- **(70) SetLanguage(Idioma As Idiomas) As Boolean**
  Set the defect language used by CashKeeper® to communicate via text messages and display. The available languages are Spanish (ESP) and English (ENG).

- **(69) SetLogPath(NewPath as String) as Boolean**
  Set the path of the LOG file storage to the indicated route on permanent basis. This path must be relative to the device where it is running the CSSI.

- **(27) SetLowLevelNotes(Values As String, Levels As String) As Boolean**
  Assigns the minimum level by each denomination of note from which the system began to warn of low level.
  Values parameter will be informed with one or more values separated by comma of the denominations we want to be informed and *Levels* parameter will contain as many values separated by comma as values has *Values* parameter.

- **(35) SetMaxLevel(Values As String, Levels As String) As Boolean**
  Assigns the maximum level by each coins/note denomination from which the system will start sending to the cashbox by excess of change. On CK1000, the maximum notes level is forced to 26.
  Values parameter will be informed with one or more values separated by comma of the denominations we want to be informed and *Levels* parameter will contain as many values separated by comma as values has *Values* parameter.

- **(83) SetNetworkParams(HostName As String, DHCPEnabled As Boolean, IP As String, Gateway As String, Mask As String, DNS1 As String, DNS2 As String, MasterPort As int32, OfficePort As int32, Seed As int32) As Boolean**
  Set network parameters.

- **(39) StartUp(Configuration As Byte, Device as Int32) As Boolean**
  It is the first method that should be called once initiated connection. Opens communication ports, loads configuration data and initializes the device.
  Configuration: The configuration identifier that we want to start, in case there is no one, it is been done.
  Device: Optional parameter, it indicates device we want to start with that configuration. It is only mandatory if the computer that is running the CSSI has more than one CashKeeper physically connected.
  As a result in the direct method, it returns the same as **GetAllProperties** parameter, because the most common to start, is to find out the configuration that is loaded.

- **(51) TargetValue(Value As Long, Operation As Byte) As Boolean**
  Gets the Target of the operation in ordered.

- **(48) Terminate**
  This method is equivalent to run **CloseAll** and **Disconnect**, with the particularity that if the CSSI this on the same machine as the application, it closes the CSSI too.

- **(42) Totalize(Total_Value as Int32, AutoClose as Boolean) as Boolean**
  Sets the total operation to inform the customer the value that should be effective in the current operation.

  If AutoClose parameter is set to TRUE, once is detected that there is enough amount, it returns the change automatically. If IDLE (0x01) state is being invoqued, the device is automatically enabled.

- **(56) UpdateSmartFirmware(FullPathFile as string, Device_ID as LC_Smart_Devices) as Boolean**
  Check and update the firmware of the specified device with the indicated update file.

- **(43) ValueIN(Value as Int32, Operation as Byte) as Boolean**
  Returns the total value entered in the operation specified in the *Value* parameter. 255 operation is equivalent to the current active operation.

- **(44) ValueOUT(Value as Int32) as Boolean**
  Returns the total value paid in the *Value* parameter.

- **(45) ValueToCashBox(Value as Int32]) as Boolean**
  Returns the total value sent to cashbox in the *Value* parameter.

ANNEX 4. List of function by SYNCHRONOUS or ASYNCRHONOUS.

List of function by SYNCHRONOUS Synchronous functions are those which the returned result implies the completion of the function task, so there won't be any wait of any event release to complete this process.

Function	Command	States in which it is possible to run it
AbortTimer	0	0x01, 0x02, 0x07
AcceptPending	3	When you receive *ProtectedValueNote, MaxCoinsWarning, BarCodeRead* event.
ActivateRefillMode	67	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
AlternateOperation	4	0x01, 0x02
CheckLevels	5	0x01
CheckSmartFirmwareFile	55	0x01
CleanBulk	6	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
CloseAll	7	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
CNT*	57	0x00
CNTOK*	58	0x00
DISCARD_PAY_OPERATION *	68	0x01
DiscardOperation	52	0x01
Display	10	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
EmptyCashBox	11	0x01
Enable	14	0x01, 0x02
ForceCoinLevel	15	0x01
GET_PROPERTY*	63	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GET_PROTOCOL_VERSION*	62	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GETALLPROPERTY	66	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetBrokenCents	16	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetCashBoxLevel	17	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetCCChange	77	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetCCDenoms	78	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetCounters	24	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetCountryCodes	38	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetCurrentLevel	19	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetDevices	72	0x00
GetFirmwareVersion	54	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetInhibitState	21	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetLastIN	76	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetLowLevelNotes	8	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetMaxLevel	23	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
GetUnikeId	74	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
RejectPending	32	When you receive *ProtectedValueNote, MaxCoinsWarning, BarCodeRead* event.
Reset	33	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07

ResetCounters	26	0x01
SET_PROPERTY*	60	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
SetCCChange	75	0x01
SetDisplayText	71	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
SetInhibitState	36	0x01
SetLanguage	70	0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
SetLogPath	69	-todos- (0x00 a 0x07)
SetLowLevelNotes	27	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
SetMaxLevel	35	0x01
SetRefillMode	67	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
START_CNT*	47	0x00
StartUp	39	0x00
State	40	-todos- (0x00 a 0x07)
TargetValue	51	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
UpdateSmartFirmware	56	0x01
ValueIN	43	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
ValueOUT	44	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
ValueToCashBox	45	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07

* Only available in direct method

List of function by ASYNCHRONOUS

Asynchronous functions are those which the returned result do NOT imply the end of the task's function. The returned result only implies the availability of the system to start the task, as well as checking that the parameters specified in the function are syntactically correct.

For the completion of these functions, it must always expect the Disabled(…) event.

Function	Command	States in which it is possible to run it	Meaning of correct answer
Disable	9	0x02	Proceed to disable the system (system IDLE 0x01)
EmptyDevice	12	0x01	It is possible to make the emptying
EmptyDeviceSpecific	13	0x01	It is possible to make the emptying
Pay	29	0x01	It is possible to make the payment
PaySpecific	30	0x01	It is possible to make the payment
Totalize	42	0x01, 0x02, 0x07	It is possible to indicate the total operation amount

ANNEX 5. List of properties

- **(20) BarCodeLength**
  Specifies the size of the barcode can be read by CK900. It must be pair and between 6 and 24 digits, if you do not want to activate it, it can be assigned zero-length.

- **(0) BCMaxCoins**
  It specifies the minimum number of coins to allow the extended BrokenCent.

- **(1) BCMinValue**
  It specifies the minimum amount to allow the extended BrokenCent.

- **(2) CancelLowLevel**
  It allows disabling events from below minimum levels (Not recommended).

- **(3) CancelValueEvents**
  It allows disabling value events (ValueIN, ValueOUT, CashBoxValue).

- **(5) CoinCashBoxDetect**
  It specifies if is controlled the coins cashbox settled.

- **(18) CoinsLowLevel**
  It specifies the minimum coins Lumber under which it will begin to notify low levels warning (LowLevel event).

- **(4) ConfigID**
  Only reading property that returns the active configuration identification (used on StarUp function).

- **(*) ConnectionType**
  Only reading property that returns the active connection (used on Connect function).

- **(21) CriticalBehavior**
  This property, control the behaviour of the coin recycler when a critical level of any coin is detected. We consider a coin is in critical level, when his level is really disadvantage from the rest of coins level. Example: All coins has a level 200 and one of them has level 30.
  0 = we send to Cash Box the coins with a level higher than 40 until critical level disappears. This process is done while the unit is working, so it is going to take some time, there is enough time to refill the coin that is in critical level.
  1 = free pay mode will be used. That means payments will use coins in the order they appear, this cause to use more coins than usual and contributes to lower the level of the higher levels.
  2 = Ignore. This state is ignored, this allow user for extreme configurations, bad use can make payments more time consuming and cause more errors.

- **(*) CurrentOperation**
  Only reading property that returns the active operation (by default it is zero, but it can be change with AlternateOperation function).

- **(19) DeviceType**
  Only reading property that returns the type of device connected.

  Possible values:
  - 0 - CK900
  - 1 - LCDT_1000

- **(6) DisableAutoText**
  It indicates whether the CashKeeper display works autonomously or not.

- **(*) ErrorCode**
  Only reading property that informs of the error code occurred in the last executed function.

- **(*) ErrorDescription**
  Only reading property that informs of the error description occurred in the last executed function.

- **(*) HostIP**
  Only reading property that returns the IP reported in Connect command.

- **(*) HostPort**
  Only reading property that returns the Host port in Connect command.

- **(7) LightTime**
  Indicates the time that the coins output led will be open when coins exit occurs whatever the cause is. In addition, it allows controlling the delay when the Disabled event ends compared to the real ending.
  In milliseconds.

- **(8) LogDisable**
  It indicates whether it is deactivated or not log for subsequent diagnostics. (Partially implemented, HIGHLY recommended to always have it false).

- **(9) MaxCoins**
  It indicates from how many coins on a payment, the system will tell us that we are going to spend more coins.

- **(10) MaxPayout**
  Indicates the maximum authorized amount in payments for a time interval defined on PayoutInterval property. Regardless of whether one, two, or thousand payments have been done during that interval of time.
  WARNING!!!. Exchange payments are not counted as payments and that includes the negative amounts Totalize.

- **(11) MinFastIn**
  Applies only in CK900. It indicates, on a single operation, the number of notes from which must be sent directly to cashbox instead of to note recycler, in order to accelerate the notes insertion.

- **(12) NLPAutoProtect**
  It indicates whether the notes change protection decides autonomously or ask the user.

- **(13) NLPPercentValue**
  It indicates the percentage change in notes to protect.

- **(14) NLPStartValue**
  It indicates from note value will act the protection of change.

- **(15) NoteLevelProtection**
  It indicates if notes protection is active.

- **(*) OfficePort**
  It indicates the port connection for type OFFICE connections.

- **(16) PayoutInterval**
  It indicates the period in payment protection.

- **(17) RejectIfClosed**
  It indicates if in stand by state, the coins introduction door will be in a position of coins rejection.

- **(*) SecuritySeed**
  Safety number for connection must have 6 digits.

- **(*) State**
  It indicates CashKeeper state.

* Only available in Keeper method

ANNEX 6. List of events

- **(114) BarcodeRead**(Code as String)
  It starts each time the notes validator reads a barcode. This event must be answered. AcceptPending function sends the barcode note to cashbox (discount vouchers, gift voucher, etc.). RejectPending function gives back the barcode note (suitable for cards, etc.).

- **(115) BarcodeStored**(Code as String)
  It starts when a barcote note is stored into cashbox correctly.

- **(100) Disabled** (ErrorCode As Int32, ErrorDescription As String, CurrentValue As Int32, TargetValue As Int32, State As Byte, Operation As Byte)
  It starts at the end of any asynchronous operation. In Error case, ErrorCode parameter will be non-zero, ErrorDesciption will contain the error description, CurrentValue will indicate the incoming/paid amount, TargetValue indicates the objective value, State the kind of operation and Operation of the current operation.

- **(102) LowLevel** (ValuesInfo as String, LevelsInfo as String)
  It starts at the end of a cash exit operation (via change or cashbox) if any denomination is below *CoinsLowLevel* in the case of coins, and in the case of notes, of LowLevel value configurated in **SetLowLevelNotes** function.

  ValuesInfo(string): String containing denominations corresponding to the event.
  LevelsInfo(Int32): String that contains the current levels of each denominations informed in 'ValuesInfo'.

- **(103) MaxCoinsWarning** (ValuesInfo as String, NumberInfo as String)
  It starts each time that the quantity of coins to be used in a payment (directly or by refund of change) is bigger than the maximum established in *MaxCoins* property.

- **(113) NoteHeldInBezel** (NotePresent as Boolean)
  It starts each time that a note appears on the mouth validator (either by notes payment or entry rejection)

  NotePresent (boolean): It is reported the status of the note
  True: present note
  False: retired note

- **(104) ProcessInterrupted** (State as Byte, Value as Int32, Target as Int32, Operation as Byte)
  It starts at CashKeeper initialization if it has been closed in the middle of a process. Imagine that the light is going out in the middle of a transaction. At the

beginning it will indicate if at that time it was doing an operation, which was the introduced amount, the amount wanted to charge and operation. In the case of payment, the amount wanted to pay, and the paid amount.

State (byte): It is reported the status of the device at the time of the interruption.
Value (Int32): It is reported the value (input or output) which was reached before the interrupt.
Target (Int32): It is reported the objective value (input or output) which should be reached at the time of the interruption.
Operation(Byte): Reports the operation number that could not be completed.

- **(105) ProtectedValueNote** (Value as Int32, PayoutNeeds as Int32, TotalChange as Int32, Operation As Byte)
  It starts if … (see 6.15.2.2)

  When *ProtectedValueNote* starts, all cash input systems are paralyzed waiting to the response of the user using *AcceptPending* o *RejectPending* methods.

- **(106) StateChange** (State as Byte, OldState as Byte)
  It starts whenever the system state changes. Indicating the new and the previous state.

- **(107) ValueIN** (CurrentValue as Int32, Target as Int32, Operation As Byte)
  It starts whenever a note or coin is inserted into the system and *CancelValueEvents* property contains false.

- **(108) ValueOUT** (CurrentValue as Int32, Target as Int32, Operation As Byte)
  It starts in the cash output actions (via Exchange or return) and *CancelValueEvents* property contains false.

- **(109) ValueToCashBox** (CurrentValue as Int32, Target as Int32, Operation As Byte)
  It starts in the cash output actions to cashbox and *CancelValueEvents* property contains false.

- **(110) Warning** (Code as Int32, Description as String)
  It starts when it appears non-critical situations but that require knowledge of the user. After the event is released, the device will try to return to the state previous to the 'Warning' event.

ANNEX 7. Manual parameters configuration of connection to the service (CSSI)

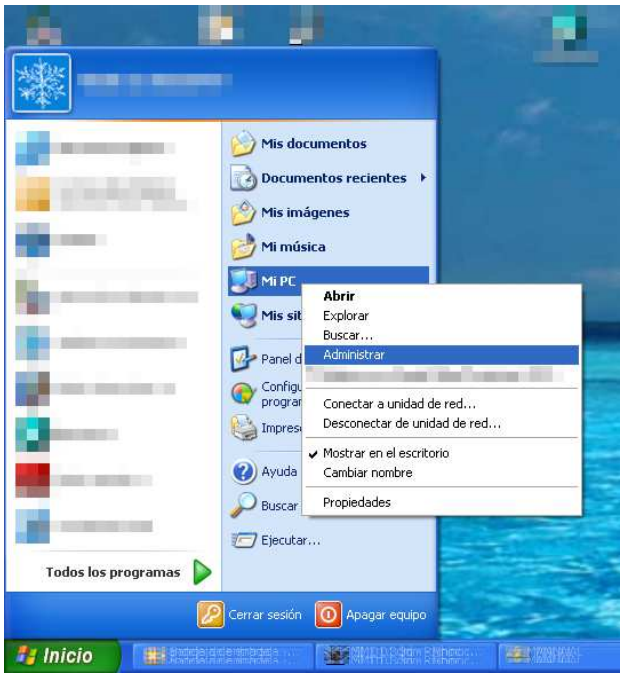
When the application runs on the same computer that is physically connected to CashKeeper, the configuration of the connection parameters to the CSSI is performed automatically at the moment of calling '***Connect(…)***' method. If the CSSI (as well as the physical connection to CashKeeper) is located in a dedicated machine, the configuration of the connection parameters must be done manually following those steps:

Create a direct access to the CSSI (CKeeper.exe) located by default in the system directory ("c:\windows\system32" or "C:\windows\syswow64") passing as parameter the Host port, Office port, Seed. Having approximately the following form:
"C:\windows\system32\CKeeper.exe 8001,8002,100001"

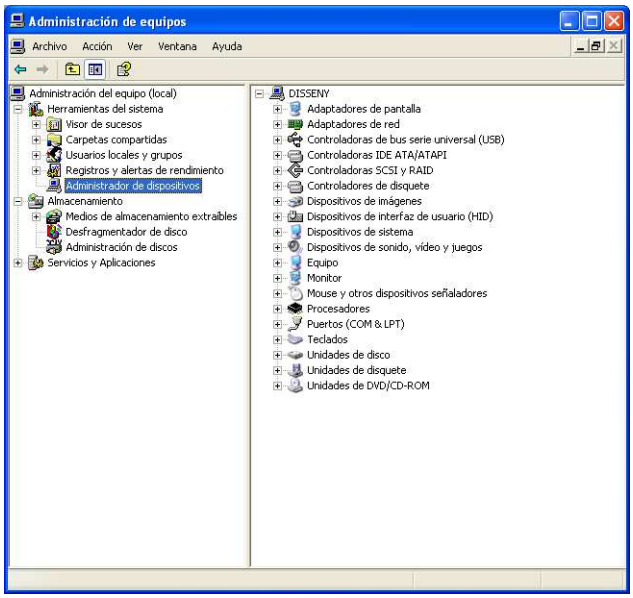
ANNEX 8. Configuration of the USB port (S.O. WINDOWS®) power management

To ensure that communication with CashKeeper is always available, on PC / TPV / Server connected to the device, it must disabled the option of power automatic disconnection from the USB port that WINDOWS ® operating systems have configured by default.

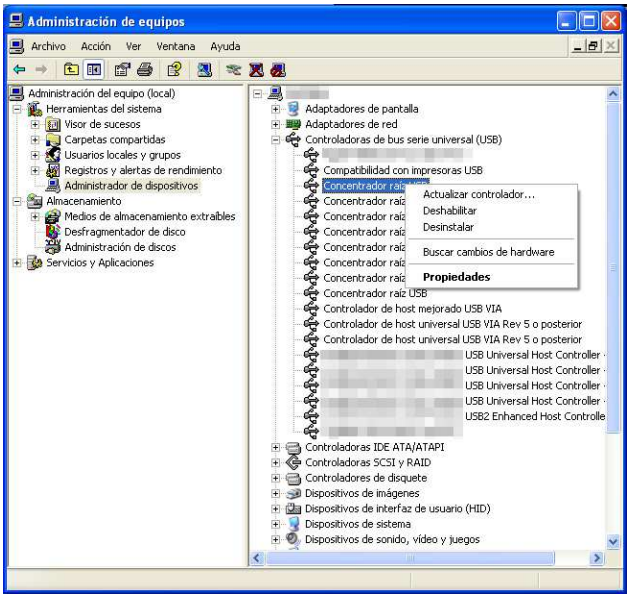
In order to proceed, it must acceded to the 'Device Administrator' (MyPC, right button, Administrate)



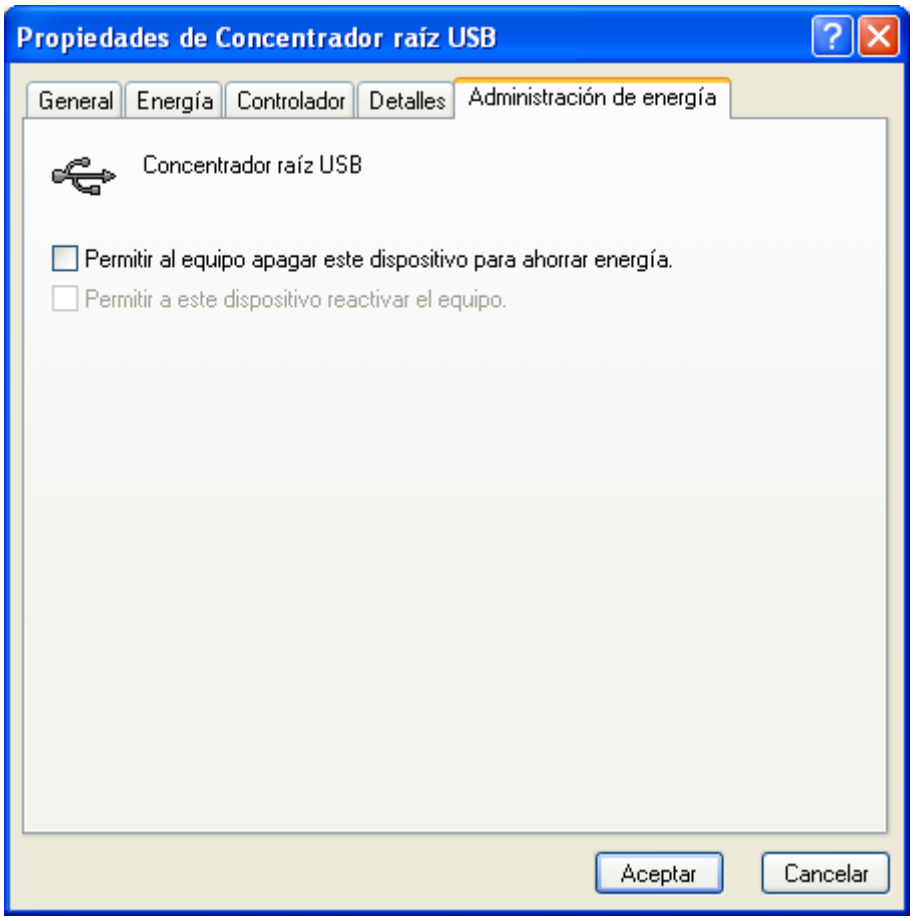
'Equipment Administrator' will be opened and, there, we will select 'Device Administrator'.



Already with the 'Device Administrator' active, it must be selected the hub USB root of our system, right button, 'Properties'.



Finally, in the properties window, select the tab 'Power Administration' and UNCHECK the option 'Allow the computer to turn off this device to save power'.


Propiedades de Concentrador raíz USB
General
Energía
Controlador
Detalles
Administración de energía
Concentrador raíz USB
Permitir al equipo apagar este dispositivo para ahorrar energía.
Permitir a este dispositivo reactivar el equipo.
Aceptar
Cancelar

ANNEX 9. Codes and text on the display relation in automatic mode

Código	Texto ESP	Text ENG
1	CASHKEEPER *(protegido)*	CASHKEEPER *(protected)*
2	Iniciando Sistema	System startup
3	Configurando Sistema	Configurating
4	INTRODUCIR IMPORTE	INSERT COINS/NOTES
5	ERROR EN EL SISTEMA	SYSTEM ERROR
6	--------------------	--------------------
7	Actualizando sistema	Updating system
8	FUERA DE SERVICIO	OUT OF SERVICE
9	Limpiando validador	Cleaning validator
10	...Espere...	...Wait...
11	- NO DISPONIBLE -	- NOT AVAILABLE -
12	Comprobando estado	Checking state
13	Cerrando sistema....	Closing system.....
14	Cerrando puertos....	Closing ports.....
15	....OCURRIó UN ERROR	..... ERROR
16	(20 espacios)	(20 blanks)
17	A PAGAR:	TOTAL:
18	PAGADO :	PAID :
19	IMPORTE INTRODUCIDO:	TOTAL PAID:
20	Terminal ocupado	Device blocked
21	por el operador	by the operator
22	¡ATENCIÓN!	¡WARNING!
23	-BILLETE ATASCADO-	-NOTE JAMMED-
25	-TERMINAL BLOQUEADO-	-DEVICE BLOCKED-
26	A DEVOLVER:	TO DISPENSE:
27	DEVUELTO :	DISPENSED :
28	- RECOGER IMPORTE -	- YOUR CHANGE -
29	Euro	Euro
30	Terminal detenido	Device Temporary
31	Temporalmente	Stopped
32	Terminal activo	Active Terminal

ANNEX 10. Barcode tickets format

Coding	Interleave 2 of 5 (ITF)
Bars width	Minimum: 0.5mm Maximum: 0.6mm
Contrast ratio	2:1
Number of characters	Minimum: 6 Maximum: 24

Dimensions

Orientation	Vertical
A	Minimum: 65mm Maximum: 82mm
B	Minimum: 120mm Maximum: 156mm
C	Minimum: 10mm
D	Minimum: 35mm
E	Minimum: 10mm