+

WMLScript Programmer's Manual

Manual Version: 5.0, Enero 2022
Latest TPVBrowser cover in this manual: 3.24

TPV Solutions S.A. de C.V.
Seneca 43
Col. Polanco, C.P. 11550, México City

1 Table of Contents

2 WMLScript

2.1 What is WMLScript?

- WMLScript is a language used for light web pages.
- WMLScript is a light version of the JavaScript language.
- WMLScript is not embedded in the WML pages. WML pages only contain references to script and URLs.
- WMLScript is compiled into byte code before it is loaded into the terminal.
- TPV Solutions WMLScript is an integral component of the WAP specification plus several libraries to address the payment industry needs.

2.2 What is WMLScript used for?

- WMLScript is used to validate user input.
- WMLScript is used to generate message boxes and dialog boxes locally, to view variable messages.
- WMLScript is used to perform math and string operations
- WMLScript is used to perform file operations.

2.3 How to name your WML Script files

WMLS source file can have any name you want but make sure the extension is WMLS.

Please take into consideration that each different POS operating systems have their own limitations for the length of the file name. Ingenico's system has a limit of 12 characters for Unicap systems and 15 characters for Telium systems. VeriFone and Hypercom have bigger limits, nevertheless to assure your programs work on all systems, make sure you name your files to be maximum 12 characters long.

Please consider that a compiled file will have the extension ".WMLSC" (6 characters including the period) so your file names should not be bigger than 6 characters.

For example, "RECEIPT.WMLS" once compiled will have the name "RECEIPT.WMLSC" which has 13 characters and thus will not work on Unicap systems!

Update: At the time of this version of the manual, unicap systems is discontinued, so if you do not intend to work with this family of terminals, you may safely use up to 15 characters names.

Note: the length limitation rule applies for all the files in your program, including BMP, WBMP, WML, fonts, etc. Please take this into consideration.

Additionally, while some terminal systems work with lower and/or upper cases file names, some payment terminal required uppercase names. To make sure your system works with all payment terminal, it is recommended to use only upper cases file names.

2.4 How to call a WMLScript from a WML page

WMLScripts are not embedded in WML pages. The WML page only contains referents to script URLs.

For example:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
        <card id="no1" title="Go to URL">
                <do type="options" label="Go">
                        <go href="CHECK.WMLSC#go_url('TEST.WML')"/>
                </do>
        </card>
</wml>
```

The line above contains a reference to a WMLScript. The script is in a file called **CHECK.WMLSC**, (which is a compiled file of the source CHECK.WMLS), and the name of the function is **go_url**.

Here is the file **CHECK.WMLS**:

```
extern function go_url(the_url) {
        Dialogs.display("HELLO WORLD",5);
        if (String.length(the_url) >0) {
                WMLBrowser.go(the_url) //going somewhere else…
        }
}
```

Note that the function is using the extern keyword. When using this keyword, the function can be called by other functions or WML events outside the WMLS file. To keep a function private, drop the extern keyword.

In WMLScript, a large number of functions have been provided as part of a library. Functions are classified into libraries based on their functionality and purpose.

This manual explains each library and their functions.

3 How to compile a WMLS Script file

TPV Solutions provides a WMLS file compiler called WMLSC.EXE.

Place a copy of the WMLSC.EXE compiler in the directory where your source is located and at the DOS prompt write the following (assuming your wmls source code is named INDEX.WMLS):

C:\Development\MyProject\Version 1.0\WMLS>wmlsc INDEX.WMLS

The complier will show the result of the operation:

Compilation success!
output file: INDEX.WMLSC
Size: 5164 byte

In case of error, the compiler will show the line where the error was found, for example:

INDEX.WMLS:173: syntax error
INDEX.WMLS:345: unknown library function `UTILS.cleareyboard'

Fix the errors and the try the compilation again until the compiler shows you "Compilation success!" message.

The result of the compilation will be a file in the same directory with the same name but with the extension WMLSC

For example, INDEX.WMLS will produce INDEX.WMLSC

If you have several WMLS files that need to be compiled, you may use a batch file to compile all of them at once. For example:

```
@ECHO OFF
@echo ******************************************************
@echo *******          Compiling WMLS files          ********
@echo ******************************************************
WMLSC          WMLS\CONFIG.WMLS

WMLSC          WMLS\INDEX.WMLS
WMLSC          WMLS\ISO.WMLS
WMLSC          WMLS\USERS.WMLS
echo Process completed.
pause
```

What you need to load in the terminal are the WMLSC files. Do not download the WMLS files as they will not be use by the Browser and will only take terminal's memory unnecessarily PLUS those are your SOURCE Code!

4 Main enviromental variables:

Environmental variables are control parameters that dictates how TPVBrowser will behave or give information about TPVBrowser environment. Their control scope is global within the system.

These variables can be access with WBMLbrowser.setEnv and WMLBrowser.getEnv commands.

Read only variables should not be set by the script, only read. Setting the variables will produce errors as generally this type of variables are set by the browser only once at system startup

Variable's names should be type exactly as they appear below

MX_QUIET,

When 0, displays on the bottom line of the screen the process of pages interpretation. For example, "processing WML", "executing script", etc.

When 1, process is muted.

MX_STATUSBAR

When 0, the Browser does not show time & date, power and connection status on the top of the screen and is up to the WML programmer to build and show this information if needed. This is useful if GUI requirements dictates a different way to show this information and its icons.

When 1, everytime a clearScreen command is executed, the Browser shows time & date, power and connection status on the top of the screen. This information will also be updated periodically when the system is idle.

This is convenient as requires no programming effort from the WMLScript programmer to show this essential information.

MX_CARRIER, MX_SIM, MX_IMEI, MX_FWVER, MX_FWSUBVER

Read only variables – when the system is working in GPRS mode, these variables give the programmer the name of the Carrier, the SIM number, the radio's IMEI number and version/subversion of radio firmware.

MX_TERMINAL_MODEL

Read only variable – Contains the model name the system is currently running on. TPVBrowser currently supports:
"O37XX", "O3600", "Vx510", "Vx510D", "Vx520", "Vx520T", "Vx520C", "Vx510G", "Vx610", "Vx570", "Vx670","Vx675", "Vx680","VxC680", "Vx690", "V5", "iWL220", "iWL250", "iWL280", "iSC250", "iSC280", "iCT220", "iCT250", "MOVE2500","400M", "200T"

MX_SHARED_CONN

When 1, it signals the Browser that a communication channels is already open and can be use.
When 0, it signals the Bowser it needs to open a communication channel before transmitting.

MX_SSL_ENABLED, SSLPOLICY, #CETSLL, CAFILE, SSLCERT, SSLPW, SSLKEYFILE

MX_SSL_ENABLED is the "master switch" to turn TLS (or SSL) on or off.

When 1, TLS/SSL is activated.
When 0, communications are done without TLS/SSL.

In verifone systems, policy 99 is turned on by default, to disable it, set **SSLPOLICY** to 1

TLS/SSL Version

Use **#CETSLL** variable to set the version of TLS/SSL the system will use.

0 – SSL 3.0
1 – TLS 1.0
2 – TLS 1.1
3 – TLS 1.2

Server authentication public certificates.

Server´s middle and root certificates should be place in file **CA.PEM**, which should be located in RAM in the same group where the system will be running. For systems other than Verix family, group and RAM notes does not apply.

Note: if for any reason the server certificates need to have a different name, the name can be indicated with the variable **CAFILE**

Client authentication public certificates.

To enable Client TLS/SSL set the following environment variables.

SSLCTE to 1, it is possible to set it to 2 if you only need client authentication with no server authentication.
SSLCERT – name of the TLS/SSL certificate file (should be place on GID 15)
SSLPW – password of a p12 cert
SSLKEYFILE – name of the private key file (should be place on GID 15)

MX_HOME
Name of the entry point file. It should be a WML file, for example F:INDEX.WML
This variable should be included in the file MBROWSER.ENV as part of the system initialization download files.

---

track1, track2, track3
Contains data of each track, once a card is swiped.
trackerr
This variable has 3 bytes in negate logic indicating presence of track, for example "001" means track 1 and 2 are present in variables track1 and track2. Track 3 is not present.

EMVFallBack,

When a card chip is read with error, this variable will be 1 and is up to your script program to allow fallback or not.

MX_MIN_BAT_PERC

Controls the minimum acceptable percentage of the battery. A percentage value of battery below this indicator triggers a "Recharge Terminal" message on the status screen (when MX_STATUSBAR is 1)

MX_PREPRINT , MX_PREPRINTED

When MX_PREPRINT is set to 1 (with WMLBrowser.setVar), the system will NOT immediately print nor handled any printer operations, such open, close, setwidth, etc. immediately when requested. All instructions will be saved in a spooler fashion and will be executed either when Printer.processPrePrint is invoked or when the system is waiting for a host response.

This is useful to perform pre-print operations when the system is waiting a transaction response. The functionality helps to minimize the host response waiting period to the user.

MX_PREPRINTED is set to 1 when all store printer commands have been executed. Use WMLBrowser.getVar to read it.

Please refer the Printer library for more details.

#MX_DUMPMB

Number of bytes to dump when the browser is in QA Mode. Default value is 600. See Quality Assurance mode (QA mode) for more detail.

5 Main programming recommendations

Avoid using WMLBrowser.setEnv when not really needed.

setEnv stores persistent variables, setVar stores volatile variables (explained in detail in the WMLBrowser library section).

Using setEnv for variables that are not needed between power cycles will degrade execution performance unnecessarily as each setEnv forces the Browser to perform I/O operations. Use setVar instead when possible.

After WMLBrowser.go, use return.

A common error is to assume the code passes control immediately when executing a WMLBrowser.go statement. This is not the case.

A programmer can execute the .go statement as a first instruction in a function and then execute the function's task. The Interpreter will not go anywhere until a return statement is found.

This also means that if several go statements are "executed", ONLY the last go statement will be taken in count.

Check signal level before doing a GPRS or WiFi transmission.

Signal levels below a given percentages (about 15%) have a small chance to succeed. Check the signal level to avoid reversals and communication problems before attempting a transaction. The actual minimum percentage criteria is up to the script programmer and final customer decision.

Check battery levels.

The Browser will check battery levels for you when MX_STATUSBAR is "1" and send appropriate messages to plug the terminal or turn the terminal off when battery is to low base on MX_MIN_BAT_PERC configuration variable. But if you turn this variable off (0), it is your responsibility, at script level, to do it. The minimum recommended value for most terminals is 12% and for some more advanced (more power consuming) terminals like Vx680 is 25%. Failure to check this may result in incomplete transactions and system errors or systems resets as more power is required in printing and transmition operations.

MX_MIN_BAT_PERC usage:
Controls the minimum acceptable percentage of the battery. It defaults to 15 (percent) if not defined or defined with a lower than 15% value. If defined bigger than 99, the system takes 99. When a terminal is not plugged into a power outlet and reached the indicated level, it will send a warning message asking the user to plug the terminal toa power outlet. If the system reaches this value minus 5%, the system will turn the terminal off.

Most recent terminals like the Vx690, VxC680 and Vx675 use a battery that do not report the charge percentage. They report the voltage value and is up to the system to use this value and estimate a charge percentage. Currently, TPVBrowser uses this table for this model of terminals.

Five Value battery icon (Remaining Capacity)	API Return Value (Remaining Capacity)	Battery Voltage (V) idle mode	Battery Voltage (V) charging mode
4 bars (75%~100%)	100%	V≥3.96	V≥4.06
3 bars (50%~74%)	70%	3.82≤V<3.96	3.92≤V<4.06
2 bars (25%~49%)	40%	3.76≤V<3.82	3.86≤V<3.92
1 bars (0%~24%)	10%	3.61≤V<3.76	3.71≤V<3.86
Blank (0%)	0%	V<3.61	V<3.71

Initialize variables.

When defining local variables, initialize them with default values to let the Browser know in advance which type of data they will handle (integers, floats, strings). In some cases, failure to follow this recommendations leads to undesired results.

```
var longVar=0.00;
var intVar=0;
var stVar="";
```

Watch the size of WML and WMLSC files.

WML programming is intended for a collection of small WML and WMLSC files. Do not try to write your complete program in one single file, besides not being the intention of the system, it won't run.

Limit your files to a maximum of 6.0 Kbytes each (once compiled).

There is no predefined limit within the browser on the number of files your system can have. The limit depends on the operating system of a terminal, but it is usually bigger than 512 files.

6 Quality Assurance mode (QA version)

TPVBrowser is released in two modes, production and debugging mode.

In debugging mode, the system will display "QA Version" on the screen. This message cannot be turned off to avoid deploying a debugging version in production.

QA mode version is equivalent to production version plus the capability to turn on several traces or logs on demand for the programmer and QA engineers to debug their programs or the browser.

Note: A QA version browser will run significatively slower than a production release even if no logs and turned on.

All debugging variables can be turned on with WBMLBrowser.putEnv or by setting the variables in config.sys in Verix systems or equivalent in Ingenico and engage systems.

A variable is off when not defined or defined with zero value. Other values do the following

1 – logs will be printed on the terminal printer. This is general not recommended as logs can be very long.

2 – logs will be sent thru the serial port of the terminal at 9.6k, 8-N-1, use any serial monitor program to receive the data.

3 – logs will be sent via the USB/serial port of the terminal at 115k, 8-N-1, use any serial monitor program to receive the data.

Note: The only exception applies to logs of the serial communications library. This log will always be printed when the value of its corresponding variables is set to 1, 2 or 3.

The log entry has the following format:

- Date and time
- Optional Browser name (useful when more than one browser instance is running on the terminal)
- Variable that generated the entry
- Function name of the browser
- Line within the function
- Description message

Example:

02/08 13:26:14 #CET mx_stopComm,341, media=Ethernet

Some entries result in a dump of data and will take several lines enclose between "BEGIN OF DUMP" and "END OF DUMP" entries, for example:

```
02/08 13:27:07 TPVB2:HTTPDT inSendViaCE,3810,---------- BEGIN OF DUMP ----------
02/08 13:27:07 TPVB2:HTTPDT inSendViaCE,3810,Data to send - 1200 bytes (showing 600 bytes)
02/08 13:27:07 TPVB2:HTTPDT inSendViaCE,3810,504F5354202F57656254505641646D69 POST /WebTPVAdmi
02/08 13:27:07 TPVB2:HTTPDT inSendViaCE,3810,6E2F6D73674D6F6E69746F7220485454 n/msgMonitor HTT
02/08 13:27:07 TPVB2:HTTPDT inSendViaCE,3810,502F312E310D0A557365722D4167656E P/1.1..User-Agen
          .         .         .         .         .         .         .         .         .
          .         .         .         .         .         .         .         .         .
          .         .         .         .         .         .         .         .         .
02/08 13:27:07 TPVB2:HTTPDT inSendViaCE,3810,3038300D0A4163636570743A20746578 080..Accept: tex
02/08 13:27:09 TPVB2:HTTPDT inSendViaCE,3810,---------- END OF DUMP ----------
```

Data dumps will be limited by default to 600 bytes of data. If more (or less) data is needed to be dumped, the variable **#MX_DUMPMB** can by set to the number of bytes that is required (1 to 99999). Use WMLBrowser.putEnv to set this variable.

Following the available log variables:

VARIABLE	DESCRIPTON
#AEST	shows technical steps for AES processing, no actual data
#BASE64T	show BASE64 technical details
#BATT	shows battery handling details
#BTRACE	dialup operations details.
#CET	IP communications details. May not show actual data being send or transmitted.
#CETSSL	TLS negotiation technical details, no actual data
#CTT	show contactless operation detail on verifone terminals
#CTTI	show contactless driver configuration detail
#CTTT	show contactless tags details on verifone terminals
#CTIT	show contactless operation detail on Ingenico terminals
#ELEMT	show detail on elementAt operations
#EMVT	shows technical details on EMV processing
#EXECT	shows script execution technical details
#FILET	show technical detail on file processing, no actual data
HTTPDT	show transmited and received data via HTTP
#HTTPT	show general HTTP assembly, headers
#IMGT	show detail on image processing, no actual data
#KEYT	show technical details on keyboard and touch screen processing
#LANGT	show technical details on language handling
#MSGT	show technical detail on screen message processing, no actual data
#PXMLT	shows technical details on XML parsing
#RSAT	shows RSA technical details

#SERIALT	show technical detail on serial communications, no actual data
#STRACE	show Browser start sequence
#WMLT	show memory handling detail on WML file processing, no actual data
#WMLST	show memory handling detail on WMLSC file processing, no actual data
#WRTTRACE	show Welcome Real Time implementation details
#X509T	show X509 handling (not for TLS)
#ZIPT	show details on zip processing, no actual data

7 CONSOLE Library

This Library is intended mainly to help the developer debug the code.

To show formatted messages on the screen please refer to the Dialog library.

To print formatted messages to the printer, please refer to the Printer library.

7.1 Function print(String message)

This function shows the *message* passed as parameter and do not inserts a new line character after the message

Parameter:
String message – text to print (no format added).

Returns:
No significant value.

Example:

```
Console.print("In loop, iteration:");
```

7.2 Function printLn(String message)

This function shows the message passed as parameter and inserts a new line character alter the message.

Parameter:
String message – text to print (no format added other than line feed).

Returns:
No significant value.

Example:

```
Console.printLn("In loop, iteration "+ String.toString(i));
```

8 CRYPTO Library

This library handles single DES encription.

This library is provided for compatibility porposes. To handle 3DES functions XOR DES and XOR 3DES please refer to UTILS.DES function.

8.1 Function decrypt(String data, String sessionKey)

This function decrypts the parameter *message* with the DES algorithm.

This function assumes *message* is an ascii hex-encoded.

Parameters:
String Data - The size of the message must be less than or equals to 1024 bytes.
String *sessionKey* - size of the key should be 16 bytes string corresponding to an 8 bytes key hex-encoded.

Returns:
string with the double length of the returned value.

Example:
var stClearValue = "";
var stKey = "123456789ABCDEF0";

stClearValue = Crypto.decrypt (stEncriptedValue,Key);

8.2 Function encrypt(String data, String sessionKey)

This function encrypts the parameter *message* with the DES algorithm.

Parameters
String data – information to encrypt. Data size must be less than or equal to 512 bytes.
String *sessionKey* – key to be use to encrypt. Size is the 16 bytes string corresponding to an 8 bytes key hex-encoded.

Returns
This function returns an ASCII string hex-encoded with the double length of the *message* parameter.

Example:
var stEncriptedValue = "";
var clearValue = "data to encrypt";
var stKey = "123456789ABCDEF0";

stEncriptedValue = Crypto.encrypt (clearValue, stKey

8.3 Function xor(String data1, String data2)

This function receives two hex-encoded ASCII strings and performs XOR operation between each pair of bytes.

Parameters:
String data1 – hex-encoded ASCII data to be XOR'ed with data2
String data2 – hex-encoded ASCII data to be XOR'ed with data1

Returns:
The hex-encoded ASCII string resulting of the xor between their binary values.

Example:

```
var sx= Crypto.xor ("313233","414243");
```

performs the following:

```
0x31 xor 0x41 = 0x70
0x32 xor 0x42 = 0x70
0x33 xor 0x43 = 0x70
```

Returns: "707070"

9 DIALOGS Library

This library deals with display output. It is capable of handling menus and formatted input for the user.

This library handles color or black & white displays in a transparent way to the WMLScript programmer.

9.1 General color parameters.

The following environmental variables control the general look and feel of the applications and should be set by the WML programmer and the begining of the program execution.

This parameters will be ignored if the system is runing in b&w terminal, nevertheless it is advisable for the WML programmer to always set their value to avoid having different versions of the script for different terminals.

FONT COLORS DEFINITION

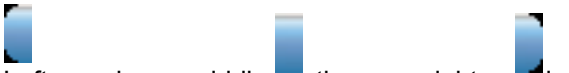
Value given in the description are examples only but should be 5-6-5 RGB codes.

- For general font color
  WMLBrowser.putEnv("#FCOLOR","65535"); //white
- For title font color
  WMLBrowser.putEnv("#TCOLOR","32768"); //red
- For menu options font color
  WMLBrowser.putEnv("#MCOLOR","0"); //black
- For clock font color
  WMLBrowser.putEnv("#CCOLOR","0"); //black

BOTTON, TEXT BOXES & TITLE GRAPHICS DEFINITION

System needs to know the graphic file names that will be used to build bottons, text boxes and titles. The WML programmer can acomplish this by setting environmental value with WMLBrowser.putEnv instruction as well.

Bottons, text boxes and titles are made of 3 parts: The left marging, the middle section and the right marging, for example (using graphics for a botton):


Left marging middle section right margin

When building a botton, text box or title, the system will search for the graphic files indicated in variables mentioned below and build the botton or title be using one left image, N middle section images (depending on the length of text to be included on the botton) and one right image

Botton's left margin is especified by the variable #LBxx
Botton's middle section is especified by the variable #MBxx
Botton's right margin is especified by the variable #RBxx

Text box's left margin is especified by the variable #LTXTxx
Text box's middle section is especified by the variable #MTXTxx
Text box's right margin is especified by the variable #RTXTxx

Title's left margin is especified by the variable #LTxx
Title's middle section is especified by the variable #MTxx
Title's right margin is especified by the variable #RTxx

For Vx680/Vx690 xx should be 80 (none for textboxes) for Vx675/iWL250/iCT250/C680 xx should be 75

Sizes for "80 type" graphics should be 10 x 26 pixels (width x height) for "75 type" graphics are 8 x 19 pixels.

For example, botton's components for 75 type graphics. File names can be anything, these are examples only. Remember Ingenico's file names are limited to 15 characters.

```
WMLBrowser.putEnv("#RB75","F:RB75.BMP");
WMLBrowser.putEnv("#MB75","F:MB75.BMP");
WMLBrowser.putEnv("#LB75","F:LB75.BMP");
```

Textbox's components for 75 type graphics

```
WMLBrowser.putEnv("#RTXT75","F:RTXT75.BMP");
WMLBrowser.putEnv("#MTXT75","F:MTXT75.BMP");
WMLBrowser.putEnv("#LTXT75","F:LTXT75.BMP");
```

Title's components for 75 type graphics

```
WMLBrowser.putEnv("#RT75","F:RT75.BMP");
WMLBrowser.putEnv("#MT75","F:MT75.BMP");
WMLBrowser.putEnv("#LT75","F:LT75.BMP");
```

And…

botton's components for 80 type graphics

```
WMLBrowser.putEnv("#Rb80","F:RB80.BMP");
WMLBrowser.putEnv("#Mb80","F:MB80.BMP");
WMLBrowser.putEnv("#Lb80","F:LB80.BMP");
```

Textbox's components for 80 type graphics (please note no 80 is use for textboxes)

```
WMLBrowser.putEnv("#RTXT","F:RTXT80.BMP");
WMLBrowser.putEnv("#MTXT","F:MTXT80.BMP");
WMLBrowser.putEnv("#LTXT","F:LTXT80.BMP");
```

Title's components for 80 type graphics

```
WMLBrowser.putEnv("#RT80","F:RT80.BMP");
WMLBrowser.putEnv("#MT80","F:MT80.BMP");
WMLBrowser.putEnv("#LT80","F:LT80.BMP");
```

Programmers should set up all variables decribed above to ensure their system will work correctly despite the hardware the program is downloaded into.

9.2 Function alert(String message)

This function clears the screen and shows the *message* passed as parameter and waits for the user key press (any key).

Parameters:
String message – prompt to show the user.

Returns:
This function returns an empty string (no significant return value).

Example:

```
Dialogs.alert("Incorrect Amount Entered!");
```

Note: The programmer may set (with setVar) the variable "MX_INVERSEFONT" with "1" if the text is to be displayed in inverse mode (white letters on dark background).

9.3 Function center(String message, integer device)

This function adds enough spaces to the left of the message accordingly to the requested device.

This function is aware of the dimensions of the display the system is running on and the current configuration of the printer (double or normal character width)

Parameters:
String message – data to be left padded with enough spaces to be centered for the requested device
Integer device – 1 for display, 2 for printer.

Returns:
The padded message

Example:

```
var centeredMessage="";

centeredMessage = Dialogs.center("PLEASE WAIT",1) ;
Dialogs.display(centeredMessage,5);
```

Returns (and show on the screen) for a Vx610/Vx510/iWL220 " PLEASE WAIT "
Returns (and show on the screen) for a Vx675/iWL250 " PLEASE WAIT "

9.4 Function colorBG(String Logo, int color, int changeInRed, int changeInGreen, int changeInBlue)

This function will paint the specified logo starting at the top of the display and then color the display with the specified color degrading the color from top to bottom in the Red, green and/or blue components accordingly to the parameters.

This function is meaningful with color terminals and is ignored when the system is running on b&w terminals.

Parameters:
String logo – a BMP graphic (bank or merchant logo for example)
Integer color – an RGB color code which will be use to color the display.
Integer changeInRed – 0 or 1. When 1, the color will be degraded from top to bottom in the red component.
Integer changeInGreen – 0 or 1.When 1, the color will be degraded from top to bottom in the green component.
Integer changeInBlue – 0 or 1. When 1, the color will be degraded from top to bottom in the blue component.

Returns:
Non relevant integer value.

Example:

```
Dialogs.colorBG("BANKLOGO.BMP",0xFFFF,0,0,0);
```

Note: BMP file vary from terminal brand to terminal brand. In Verifone it is 16 bit 5-6-5 (R-G-B) format for Ingenico is 24 bit 8-8-8 format

---

9.5 Function confirm(String message, String option1, String option2)

This function shows the *message* passed as parameter and two options of responses. This function

Parameters:
String message – prompt to show the user
String option1 – name of first option
String option2 – name of second option

Returns:
True (1) if the option 1 was chosen or false (0) if option 2 was chosen.

Example:

```
Dialogs.confirm("is correct?", "YES", "NO");
```

The call returns 1 if option YES was selected, 0 otherwise.

9.6 Function display (String message, Int line)

This function displays a *message* on the terminal screen in the line specified by *line* parameter.

Parameters:
String message – data to show the user
Integer line – number of the line on which message will be displayed

Returns:
No significant string value.

Example:

```
Dialogs.display("Hello World!", 4);
```

Notes:

The programmer may set (with setVar) the variable "MX_INVERSEFONT" with "1" if the text is to be displayed in inverse mode (white letters on dark background.

By default, this function uses font15x16.bmp for Vx675, VxC680, Vx520, Vx690, all other terminals use FONT10x20.bmp. This default can be override setting (with putEnvSec) variable "#MX_FONT".

9.7 Function displayMultiLine (String message, int CharsPerLine,int Pixels, int col, int row)

This function displays and justifies a long *message* on the terminal screen in the line specified by *col and row parameters*.

Parameters:
String message – data to show the user
ChasPerLine – maximum number of characters allowed in a single line, this can be less than actual screen width.
Pixels – if 1, following coordinates are to be interpreted in pixels, if 0, coordinated are to be interpreted in characters.
Col – column where message display will start.
Row – row where message display will start

Returns:
No significant int value.

Example:
Dialogs.displayMultiLine("This is an example of a very long text message containing some instructions for the user",20,1,100,50);

This is an example of
a very long text
message containing
some instructions for
the user
Notes:

By default, this function uses font15x16.bmp for Vx675, VxC680, Vx520, Vx690, all other terminals use FONT10x20.bmp. This default can be override setting (with putEnvSec) variable "#MX_FONT".

9.8 Function extendedPrompt(String type,String titles,String Default, Int Row, Int Min, Int Max, Int showError, String errorLogo)

This function handles user input.

Input can be of alphanumeric, password, numeric, currency and IP addresses types. It also handles variable decimal places for numeric and currency types.

Calling this functions clears the screen. In color terminals, the system will display color title and draw a textbox. In b&w terminal, the system will show the title in inverse text.

Please verify that title and textbox components are set up properly by following the recommendations at the beginning of the Dialogs library section.

Parameters:
String type – possible values – "A" for alphanumeric, "P" for alphanumeric passwords (masked input with asterics), "N[n]" for numeric, "$[n]" for currency, "I" for IP addresses. [n] is the number of decimal places, for numeric default is zero, for currency is two.

String titles – message for title and prompt separated by semi colon

String Default – default value to be shown

Integer Row – number of the row where the prompt starts

Integer Min – minimum length

Integer Max – Maximum length

Integer showError – 0 or 1. when 1 and input is done (user hit ENTER) the system will validate length. If length is outside min and max parameters, the system will display an error messages indicating expected minimum and maximum and will display the errorLogo, if specified.

String – errorLogo – optional file name for a logo to be dispayed along with an error message. System will use .BMP extension for color terminals and .wbmp extension for B&W terminals.

Other indirect parameters:
Environmental variable #ITO – time out in miliseconds to wait for the user, default is 30000 (30 secs)
Environmental variable #EMTO – time to display the error message in miliseconds, default is 1300 (1.3 secs)

Returns:
String – value entered by the user or "#A#" if user hit CLEAR or time out expires.

Example:
```
Var stAmount="";
stAmount = Dialogs.extendedPrompt("$2", "SALE;ENTER AMOUNT", "0.00" ,3,4,12,
1, "error.bmp");
```

The system is going to accept monetary amount with 2 decimal places with a default initial value of "0.00". The prompt is to be placed on row 3. The minimum length if 4 characters (counting the period) and a maximum of 12 characters (counting any commas). It will display and error message if user hits ENTER with an incorrect input length and will also show the graphic "error.bmp", if found.

Function errorMessage(string title, string message, integer initialPosition, int clear screen, integer beepType, integer waitTime)

Parameters
Title – string to be shown at the top of the screen
Message – string to be shown at the initialPosition
initialPosition – 0- top of screen, 1-middle of the screen, 2-bottom of the screen
clearScreen – 1- clear screen before showing the message. 0-do not clear screen
beepType – 1 normal beep, 2 – error beep
waitTime – milliseconds to wait before returning control.

Returns
Not significant integer value

Example:
Dialogs.errorMessage("ERROR PROCESSING","PLEASE TRY AGAIN",2,1,2,1000);

9.9 Function InsertBotton(int Title, int TextX, int TextY, int GraphicX, int GraphicY, String text, int Value)

This funtion draws a botton on the screen.

The botton reacts to touch if the terminal is touchscreen capable and parameter "Value" is non-zero.

Please verify that botton components are set up properly by following the recommendations at the beginning of the Dialogs library section.

Parameters:
Integer Title – 0 or 1. If 1 the graphic drawn will be a title type, not a botton (and will not react to touch)

Integer TextX – Column in characters to use when running in b&w terminals (no graphic will be drawn, just the text).

Integer TextY – row in characters to use when running in b&w terminals (no graphic will be drawn, just the text).

Integer GraphicX – Column in pixels to use when running in color terminals

Integer GraphicY – row in pixels to use when running in color terminals

String Text – Message to be shown inside the botton or title (in b&w terminals text to be shown without any graphic)

Integer Value – Value to return by UTILS.getKey() function if this graphic is touched. If zero, the botton will not react to touch.

Returns:
Zero if the graphic was not drawn (out of memory)
Non zero (graphic handler) if the graphic was drawn (or text displayed in b&w terminals).

Example:

```
Dialogs.insertBotton(0,10,7,84,280," ALFA",143);
```

9.10 Function insertIcon(String filename, int GraphicX, int GraphicY, int Value)

This funtion draws a graphic on the screen. This function works only on color terminals and is ignored if called in b&w terminals.

The graphic reacts to touch if the terminal is touchscreen capable and parameter "Value" is non-zero.

Note: to draw general graphics in b&w terminals or in color terminals without the need to react to touch, please use the function UTILS.displayImage().

Parameters:
String – filename – name of the graphic file. A BMP file.
Integer GraphicX – Column in pixels.
Integer GraphicY – row in pixels.
Integer Value – Value to return by UTILS.getKey() function if this graphic is touched. If zero, the graphic will not reach to touch.

Returns:
Zero if the graphic was not drawn (out of memory)
Non zero (graphic handler) if the graphic was drawn.

Example:

Dialogs.insertIcon("logo.bmp", 84, 280,150);

9.11 Function language(int index, int center)

This function retrieves the text message associated with the indicated index value from a language file. Text is center base on the value of the second parameter.

Parameters:
Index – message index value to be searched inside a language file.
Center – 0, 1 or 2. 0- do not center text, 1 - center text for screen, 2 - center text for printing. Please refer to Dialogs.Center function for a more detail description of the center functionality.

Note:
Language file name is made of the concatenation of a prefix + "_WMLS.TXT"

Programmers can specify the prefix of the language file name with putEnv and the variable "#MX_LANGUAGE". Default is "SPA"

9.12 Function menu(String titles, integer count, string options, string graphicName, integer timeout, integer default)

This functions displays a menú from which the user may select an option. Up to 9 items can be display on each menu.

Parameters:
String titles – Text to be use as the menu's name. Up to 3 lines may be use for this, separated by semicolon (";")
Integer count – Total number of options of the next parameter
String options – Text of each of the options separated by semicolon. Up to 9 options can be use per menu. Be careful to keep the total length of each option no longer than 17 to make sure they will display correctly on all terminal brands and models.

String graphicName – optional graphic filename. Independently of the extension indicated, on monochromatic terminals the extension will be wbmp. In case of color terminals, the name will be appended with .bmp
Integer timeout – in seconds, if 0 it will default to 30 seconds.
Integer default – Number of the option returned by default (if user did not pick any option).

Returns:
The function returns 1 if first option is selected, 2 if second option is selected, etc.
Note: if user presses CLEAR or the functions times-out, the function will return the indicated default or 0 if no default was indicated.

Example:

Dialogs.menu("TIP;",4," NONE ; 10% ; 15% ; 20% ;","TIPLOGO.BMP",45,0);

Notes:
The function will display the options with a prefix of sequential numbers from 1 to 9, so the user may use the keyboard to directly select the option. If running on terminals with touchscreen capability, the user may touch the option directly.

The programmer may set (with putEnv) the variable "MX_CENTERMENU" with "1" if the options are to be displayed centered or "2" if options will be left justify. "0" or absent, options are right justified by default.

The programmer may set (with putEnv) the variable "MX_ARROWS" with "1" to display arrows in "down up" order. If "0" or absent arrows will be displayed "up down" order. This variable work only with 5100, 7780 and 7910 terminals. All other terminals ignore this variable and displays arrows in "down up" order.

The programmer may set (with putEnv) the variable "MX_CASE" with "U" to show all options in uppercase, "L" in lower case or "P" in proper case (first letter capitalized all other lower case). As default, options will be displayed as originally specified.

9.13 Function prompt(String message, String defaultInput)

This function shows the *message* passed as parameter and prompts for user input. This function assigns the input value typed by the user at the parameter.

Parameters:
String message – prompt to show the user.
String *defaultInput* contains a standard start value to be shown to the user.

Returns:
String – value entered by the user.

Example:

```
var stDefault = "04400" ;
Dialogs.prompt("Enter zip code", stDefault);
```

Please refer to Dialogs.extendedPrompt for formatted input handling.

9.14 Function relativeDisplay(String message, int initialPosition, int offsetFromIticialPosition, int horizontalAligment)

Shows the given message on the screen at the given position.

Parameters
Message – String text to be shown.
initialPosition – 0 – top of the screen, 1 – center of the screen, 2-botton of the screen
offsetFromInitialPosition – number of lines after top or before botton. When using center, this parameter can be negative or positive. Zero to avoid offset.
HorizontalAligment 1-horizontal center on the screen, 0 shows at left.

Returns
Not significant integer valur

Example

```
#define SCREEN_BOTTOM 0
Dialogs.relativeDisplay("hello world",SCREEN_BOTTOM,0,1);
```

9.15 Function removeIcon(String filename, int GraphicX, int GraphicY, int Value)

This funtion restore the background erasing the graphic specified in the filename. It is the opossite funtionality to insertIcon().

In cases where the programmer does not want to clear the screen but just to delete an icon, this function can be call.

This function is ignored in b&w terminals.

Parameters:
Should be the same values use when the graphic was created using insertIcon.
String – filename – name of the graphic file. A BMP file.
Integer GraphicX – Column in pixels.
Integer GraphicY – row in pixels.
Integer Value – Value to return by UTILS.getKey() function if this graphic is touched. Do not indicate a different value than the originally used with insertIcon as the touch screen index table uses this value to remove the entry form the list.

Returns:
Always returns integer 1.

Example:

If icon was created with Dialogs.insertIcon("logo.bmp", 84, 280,150);

It can be removed from screen with Dialogs.removeIcon("logo.bmp", 84, 280,150);

9.16 Function setBG(String Logo, int color, int changeInRed, int changeInGreen, int changeInBlue)

This function set the parameters for background screen to be display each time UTILS.clrscr() is invoke. This function has no immediate action on the display, only when clrscr() is called.

Instead of just clearing the screen when UTILS.clrscr() is called on color terminals, the system will paint the specified logo starting at the top of the display and then color the display with the specified color degrading the color from top to bottom in the Red, green and/or blue components accordingly to the parameters.

This function is meaningful with color terminals and is ignored when the system is running on b&w terminals.

Parameters:
String logo – A BMP graphic file name (bank or merchant logo for example)
Integer color – An RGB color code which will be use to color the display.
Integer changeInRed – 0 or 1. When 1, the color will be degraded from top to bottom in the red component.
Integer changeInGreen – 0 or 1.When 1, the color will be degraded from top to bottom in the green component.
Integer changeInBlue – 0 or 1. When 1, the color will be degraded from top to bottom in the blue component.

Returns:
No relevant integer value.

Example:

```
Dialogs.setBG("banklogo.bmp",0xFFFF,0,0,0);
```

9.17 Function show(String message)

This function shows the *message* passed as parameter on the last row of the display and continues execution.

Parameters:
String message – text to show on the last line of the displays.

Returns:
This function returns an empty string (no significant return value).

Example:

```
Dialogs.show("Dialing, please wait...");
```

Note: The programmer may set (with setVar) the variable "MX_INVERSEFONT" with "1" if the text is to be displayed in inverse mode (white letters on dark background

9.18 Function showSignature(String filename, integer, row)

This function shows on screen the previously captured signature (on signature capture capable terminals)

Parameters:
String filename – name of the compressed signature file returned by utils.getHandSignature function.
Row – Starting row in pixels

Returns:
The Size of the signature in bytes. If Value>0 if the file was uncompressed and therefore shown on the screen.

Example:

```
Dialogs.showSignature("mysign.dat",100);
```

9.19 Function showStatus(String message)

This function is synonymous to function show() above.

10 EMVBrowser Library (contact cards)

Following the functions that deal with an EMV transaction.

Function are shown in the order they are tipically used during a transaction.

10.1 Function isCardInserted()

This function checks for the presence of a card in the main chip reader.

Parameters:
None.

Returns:
1 if the card is inserted, 0 otherwise.

Example:
```
EMVBrowser.closeEMV("");
while (EMVBrowser.isCardInserted() == 1) {
    UTILS.clrscr();
    EMVBrowser.closeEMV(Dialogs.center("REMOVE CARD",1));
    WMLBrowser.beep(3,2);
    UTILS.pause(1000);
}
```

note: To start a transaction it is recommeded that the programmer uses UTILS.getKey() to check for the presence of a chip card.

10.2 Function initEMV()

This function initializes the chip reader. This function should be called before doing every EMV transaction (after detecting insertion of a card).

Parameters:
None.

Returns:
On successful initialization, the function returns 0.

-1 if card has not selectable application

Other value, chip error.

Example:
```
var init = 1;
init = EMVBrowser.initEMV();   // if init is 0, initialization was ok.
```

10.3 Function setTag(string Tag, string Value)

This function changes the value of a TAG in the EMV structure.

Parameters:
String Tag – ascii name of the tag to change.
String Value - ascii value to asign to 'Tag'.

Returns:
No relevant string value.

Example:
EMVBrowser.setTag(TagNumber,TagValue);

Note: To change the currency code do not use this function. Please use WMLBrowser.putEnv("#CURRENCY",yourValue) before calling WMLBrowser.readTag function.

10.4 Function readTag(Int amount, String message, Int CashAmount)

This function generates the first cercificate of an EMV transaction.

Parameters:
Unsigned long *amount – amount of the* transaction.
String *message* - message to show while the EMV engine is doing the task
Unsigned long *CashAmount – cash back amount of the transation.*

If doing a refund, accordengly to current rules, the ARQC most NOT be generated. Please set the following variable to abort the operation just before the ARQC is done.

WMLBrowser.putEnv("#EMVAbort","1");

Returns:
Besides function return, please read LastEmvError environment variable like this:

EMVErr=WMLBrowser.getVar("LastEmvError");

If function returns 0 or EMVErr is 215 or EMVErr is 400:
If refund and #EMVAbort is "1", continue.
Else EMV error - Decline

If function returns 102, 107 or -999, transaction aborted.

If EMVErr is 110 – chip error

If EMVErr is 130 or function returns -2 – card is blocked

If EMVerr is 1 and TVR xx xx xx xx 2x – PIN exceeded

Besides the available tag collection (see function getTag). The following environment variables are populated:

- **EMVApplicLabel**: Application label for example "Mastercard".
- **EMVPAN**: Personal Account Number.
- **EMVPANSeqNum**: Security number .
- **EMVService**: Service code .
- **EMVCHName**: Card holder.
- **EMVExpDate**: Expiration date.
- **track2**: Track 2.
- **EMVCardType**: type of reading ("emv").
- **EMVFallBack**: fallback (0).
- **EMVCrypto**: Cryptogram of the transaction, if successful reading**.**
- **EMVAid**: AID.
- **CardDecision,** which may have the following values:
  - **1002.-** Go online
  - **1005 –** Approved offline.
  - **Any other value** – EMV declined.

10.5 Function getTag(int tag)

This function allows the user to extract the values of a particular EMV tag.

Parameters:
Integer tag – Index accordingly to the following table.

Example:
var tag = "";
tag = EMVBrowser.getTag(5); //Will return tag 57 (track 2 equivalent data)

TAG	DESCRIPTION	TAG	DESCRIPTION
1	TAG_42_ISSUER_ID_NUM	2	TAG_4F_AID
3	TAG_50_APPL_LABEL	4	TAG_52_CMD_TO_PERFORM
5	**TAG_57_TRACK2_EQ_DATA**	6	TAG_5A_APPL_PAN
7	TAG_5F20_CARDHOLDER_NAME	8	TAG_5F24_EXPIRY_DATE
9	TAG_5F25_EFFECT_DATE	10	TAG_5F28_ISSUER_COUNTY_CODE
11	TAG_5F2A_TRANS_CURCY_CODE	12	TAG_5F2D_LANG_PREFERENCE
13	TAG_5F30_SERVICE_CODE	14	TAG_5F34_APPL_PAN_SEQNUM
15	TAG_5F36_TRANS_CURR_EXP	16	TAG_5F50_ISSUER_URL
17	TAG_5F53_INT_BANK_ACC_NO	18	TAG_5F54_BANK_ID_CODE
19	TAG_5F55_ISSUER_COUNTRY_CODE	20	TAG_5F56_ISSUER_COUNTRY_CODE
21	TAG_5F57_ACCOUNT_TYPE	22	TAG_APPL_TEMPL_61
23	TAG_FCI_TEMPL_6F	24	TAG_ISUER_SCRPT_TEMPL_71
25	TAG_ISUER_SCRPT_TEMPL_72	26	TAG_DIR_DISCR_TEMPL_73
27	TAG_RESPMSG_FMT1_TEMPL_77	28	TAG_RESPMSG_FMT2_TEMPL_80
29	TAG_81_AMOUNT_AUTH	30	TAG_8200_APPL_INTCHG_PROFILE
31	TAG_83_CMD_TEMPLATE	32	TAG_84_DF_NAME
33	TAG_86_ISSUER_SCRIPT_CMD	34	TAG_87_APPL_PRIORITY_IND
35	TAG_88_SFI	36	TAG_89_AUTH_CODE
37	TAG_8A_AUTH_RESP_CODE	38	TAG_8C_CDOL1
39	TAG_8D_CDOL2	40	TAG_8E00_CVM_LIST
41	TAG_8F_AUTH_PUBKEY_INDEX	42	TAG_90_ISS_PUBKEY_CERT
43	TAG_91_ISS_AUTH_DATA	44	TAG_92_ISS_PUBKEY_REM
45	TAG_93_SIGNED_SAD	46	TAG_94_AFL
47	TAG_9500_TVR	48	TAG_97_TDOL
49	TAG_98_TC_HASH	50	TAG_99_PIN_DATA
51	TAG_9A_TRAN_DATE	52	TAG_9B00_TSI
53	TAG_9C00_TRAN_TYPE	54	TAG_9D_DDF_NAME
55	TAG_9F01_ACQ_ID	56	TAG_9F02_AMT_AUTH_NUM
57	TAG_9F03_AMT_OTHER_NUM	58	TAG_9F05_APPL_DISC_DATA
59	TAG_9F04_AMT_OTHER_BIN	60	TAG_9F06_APPL_ID
61	TAG_9F07_APPL_USE_CNTRL	62	TAG_9F08_APP_VER_NUM
63	TAG_9F09_TERM_VER_NUM	64	TAG_9F0B_CRDHLDRNAME_EXT
65	TAG_9F0D_IAC_DEFAULT	66	TAG_9F0E_IAC_DENIAL
67	TAG_9F0F_IAC_ONLINE	68	TAG_9F10_ISSUER_APP_DATA
69	TAG_9F11_ISSSUER_CODE_TBL	70	TAG_9F12_APPL_PRE_NAME
71	TAG_9F13_LAST_ONLINE_ATC	72	TAG_9F14_LC_OFFLINE_LMT
73	TAG_9F15_MER_CAT_CODE	74	TAG_9F16_MER_ID
75	TAG_9F17_PIN_TRY_COUNTER	76	TAG_9F18_ISSUER_SCRIPT_ID
77	TAG_9F1A_TERM_COUNTY_CODE	78	TAG_9F1B_TERM_FLOOR_LIMIT
79	TAG_9F1C_TEMR_ID	80	TAG_9F1D_TERM_RISKMGMT_DATA
81	TAG_9F1E_IFD_SER_NUM	82	TAG_9F1F_TRACK1_DISC_DATA
83	TAG_9F20_TRACK2_DISC_DATA	84	TAG_9F21_TRANS_TIME
85	TAG_9F22_CERT_PUBKEY_IND	86	TAG_9F23_UC_OFFLINE_LMT
87	TAG_9F26_APPL_CRYPTOGRAM	88	TAG_9F27_CRYPT_INFO_DATA
89	TAG_9F2D_ICC_PIN_CERT	90	TAG_9F2E_ICC_PIN_EXP
91	TAG_9F2F_ICC_PIN_REM	92	TAG_9F32_ISS_PUBKEY_EXP
93	TAG_9F33_TERM_CAP	94	TAG_9F34_CVM_RESULTS
95	TAG_9F35_TERM_TYPE	96	TAG_9F36_ATC
97	TAG_9F37_UNPRED_NUM	98	TAG_9F38_PDOL
99	TAG_9F39_POS_ENTRY_MODE	100	TAG_9F3A_AMT_REF_CURR
101	TAG_9F3B_APPL_REF_CURR	102	TAG_9F3C_TRANS_REF_CURR
103	TAG_9F3D_TRANS_REFCURR_EXP	104	TAG_9F40_TERM_ADTNAL_CAP
105	TAG_9F41_TRANS_SEQ_COUNTER	106	TAG_9F42_APPL_CURCY_CODE
107	TAG_9F43_APPL_REF_CURR_EXP	108	TAG_9F44_APPL_CURR_EXP
109	TAG_9F45_DATA_AUTH_CODE	110	TAG_9F46_ICC_PUBKEY_CERT
111	TAG_9F47_ICC_PUBKEY_EXP	112	TAG_9F48_ICC_PUBKEY_REM
113	TAG_9F49_DDOL	114	TAG_9F4A_SDA_TAGLIST
115	TAG_9F4B_DYNAMIC_APPL_DATA	116	TAG_9F4C_ICC_DYNAMIC_NUM
117	TAG_9F4D_LOG_ENTRY	118	TAG_9F4E_MERCHANT_NAME_LOCN
119	TAG_9F4F_LOG_FORMAT	120	TAG_BF0C_FCI_DATA
121	TAG_FCI_PROPTRY_TEMPL_A5	122	TAG_9F53_TRAN_CURR_CODE
123	TAG_AEF_TEMPL_70	124	TAG_9F55_ISSUER_SCRPT_RESULT

10.6 Function processTransact(String code, String field55, Int numberCode)

This function sends the approval code and scripts to the card and then the second generate AC is done.

Parameters:
String code – approval code (tipical "00")
String field55 – Scripts receved form the host (issuer)
Integer Number code - not used, send 0.

Returns:
150.- Transaction approved**.**
151.- Transaction declined.

10.7 Function closeEMV(String Message)

This function should be called when the card operation is finished, regardless of the outcome.

Parameter:
String message – text to be display if the card is still inserted (tipically "REMOVE CARD")

Returns:
Always 1.

Example:
```
EMVBrowser.closeEMV("");
while (EMVBrowser.isCardInserted() == 1) {
    UTILS.clrscr();
    EMVBrowser.closeEMV(Dialogs.center("REMOVE CARD",1));
    WMLBrowser.beep(3,2);
    UTILS.pause(1000);
}
```

11 EMVBrowser Library (contactless cards)

11.1 Function mx_initEMVContactless ()

This functions initialices all parameters needed to perform a contactless transactions.

The program should call this function once or when a configuration parameter changes only. It is not recomended to call this function just before every single transaction as it may take several seconds to execute.

On execution, the function will look for 2 files. The key file MYKDEMV.TXT and the general parameters file CTLSEMVCONFIG.DAT

MYKDEMV.TXT - The file is text based and needs to have the following format:

Field Name	Description	Example
Key number	A sequential number from 01 to NN	01
Field separator		;
RID	Key RID	A000000003
Field separator		;
Length	Key length in hexa in bytes	80
Field separator		;
Date	Key Expiration date	311235
Field separator		;
Exponent	Key Exponent	03
Field separator		;
Key	Key data	N bytes of data
Field separator		;
Hash	Key hash value in hex	40 bytes
Field separator		;

The file can be downloaded in RAM or FLASH in verifone. In Ingenico in HOST. If the file does not exists, the function will continue with the next file.

CTLSEMVCONFIG.DAT – This is a hexadecimal file with the following structure:

Field Name	Description	Length	Example
CTLSEMVTermCap	CTLS Terminal Capabilities	20	0xE868C8
CTLSEMVAddTermCap	CTLS additional term capabilities	20	0x6000F0F001
CTLSEMVTermCountryCode	CTLS Country code	20	0x0170
CTLSEMVFloorLimit	CTLS Floor limit	20	0x00000000
CTLSEMVTermType	CTLS Terminal Type	20	0x22
CTLSEMVTransCurrCode	Currency code	20	0x170
CTLSEMVTTQ	CTLS TTQ	20	0xA6000000
CTLSEMVTACOther	CTLS TAC Other	20	0xFC50A888000
CTLSEMVTACDef	CTLS TAC Default	20	0xFC50A80000
CTLSEMVTACDenial	CTLS TAC Denial	20	0x0000000000
CTLSEMVCVMLimit	CTLS CVM Limit Value	20	0x000000025001

CTLSEMVTransactionLimit	CTLS tx limit value	20	0x999999999999

The first record is intended as default, the second record is intened for Mastrcard, the third record is intended for VISA, the fourth record is intended for AMEX.

If the file does not exists, the system will configure the parameters with defaults values.

Returns:
-1 if terminal does not support contactless, 0 otherwise.

11.2 Function mx_readContactless (string graphicfile, string amount, string OKmsg, string swipeMSG, string promptMsg)

This function will prompt the user to tap the contactless card for a transacction.

The function currently supports paypass 2.1 and paywave 2.1.1

The funcion will wait upto 30 seconds for the card to be tapped.

Parameters:
String graphicFile – optional, The contactless logo, if not specified (“”), a default logo will be shown
String amount – The purchase amount, formated with commas and piriods
String OKmsg – the “thank you” message displayed after card is read. Defaults if “” is “Gracias”
String swipeMsg – the “please swipe card” message if needed, default if “” is “Deslice Tarjeta”
String promptMsg – the “please tap your card” message, default if “” is “Aproxime Tarjeta”

Returns:
-1 if terminal is not CTLS capable or error
0 if no card was tapped
>0 if card was read, use MX_GETCONTACTLESSTAG function to get card data

11.3 Function mx_getCtlsTag(integer tag)

Please refer to function 9.5 getTag as the functionality is the same.

12 FLOAT Library

12.1 Function addFloat (string value1, string value2, integer addition)

This function converts the first two string arguments to floating-point number and then adds them or subtracts value2 from value1 depending on the value of the third parameter.

Parameters:
String value1 – text representation of a float number
String value2 – text representation of a float number
Integer addition – if 1, the function will add the first 2 value, of 0, will subtract value2 from value1

Returns:
A string representation of the result.

Example:
```
var v;
v= Float.addFloat("1,110.30","1.20",1);   //v="1,111.50"
v= Float.addFloat("10.30","0.20",0);      //v="10.10"
v= Float.addFloat("5.00","5.20",0);       //v="-0.20"
```

12.2 Function ceil (float value)

Rounds the value of the argument to the next bigger integer value.

Parameter:
Value – any floating-point value.

Returns:
Integer value.

Example:
```
var w;
w= Float.ceil(-2.7);      // w= -2
w= Float.ceil(2.4);       // x= 3
```

12.3 Function floor (float value)

Rounds the value of the argument to the next smaller integer value.

Parameter:
Value – any floating-point value.

Returns:
Integer value.

Example:
```
var w;
w= Float.floor(-2.7);       // w= - 3
w= Float.floor(2.4);        // x= 2
```

12.4 Function int (float value)

Returns the integer part of a float value.

Parameter:
Value – any floating-point number.

Returns:
The integer part of the given number, no decimal portion.

Example:
```
var v= -67.09;
var w;
w= Float.int(v);            // w= -67
w= Float.int(7.98);         // w= 7
```

12.5 Function maxFloat()

This function returns the maximum floating point value supported.

Parameter:
None.

Returns:
platform dependant.

Example:
```
var v;
v= Float.maxFloat();  //v= 3.402823466 E + 38  (example, platform dependent)
```

12.6 Function minFloat()

This function returns the minimum floating point value supported.

Parameter:
None.

Returns:
platform dependant.

Example:
```
var v;
v= Float.minFloat();       //V= 1.175494351 E – 38 (example, platform dependent)
```

12.7 Function pow (float value1, float value2)

Returns the result of a raising *value1* to the power of the *value2*.

Parameter:
Value1 – base
Value2 - exponent

Returns:
Float number as result.

Example:
```
var w;
w= Float.pow(2.5,2);   // w= 6.25
```

12.8 Function round (float value)

Rounds the number to the highest integer value

Parameter:
Value – any floating-point number.

Returns:
Rounded value of argument.

Example:
```
var v= Float.round(89.94);        // v= 90
var w = Float.round(-89.94);      // w= -89
var x= Float.round(-0.99);        // x=0
var y= Float.round(0.99);         // y=1
```

12.9 Function sqrt(Float value)

This function returns the square root of the given *value*.

Example:
```
var v= 81;
var w= Float.sqrt(v);  // w= 9
var x= Float.sqrt(39);  // x= 6.244997998398
```

13 HTTPClient Library

Following the description of the needed functions to perform HTTP requests.

If you need activate the SSL, please set the following environment variables like this:

MX_SSL_ENABLE = "1" (where 1 means activated and 0 deactivated)

To enable Client SSL set the following environment variables:

SSLCTE to "1"
SSLCERT – name of the ssl certificate file (should be place on GID 15)
SSLPW – password of a p12 cert
SSLKEYFILE – name of the private key file (should be place on GID 15)

13.1 Function addHeader(String headerName, String headerValue)

This function adds http headers to your request, you must have created an instance http before by calling httpPetition (described below).

Returns:
An integer value:
0.- If not successful
1.- If succesful

Example:

```
if(httpClient.httpPetition() == 1) {
        httpClient.addHeader("Accept-Language", "en-us");
        httpClient.addHeader("Cache-Control", "no-cache");
}
```

13.2 Function addParameter(String paramName, String paramValue)

This function adds parameters to a HTTP request, you must have created an instance http before with httpPetition().

Returns:
An integer value:
0.- If not successful
1.- If succesful

Example:

```
if(httpClient.httpPetition() == 1) {
        httpClient.addParameter("sernum", "xxx-xxx-xx");
        httpClient.addParameter ("type", "sale");
}
```

Note: value passed on a single addParameter function can not be bigger than 1024 bytes. Should you need to pass a bigger amount of data, do the following:

```
httpClient.addParameter("Bigdata", "first 1024 bytes");
httpClient.addParameter("", "another 1024 bytes");
httpClient.addParameter("", "another 1024 bytes");
httpClient.addParameter("", "another 1024 bytes");
```

you may call up to 4 addParameter like this for one parameter

13.3 Function addRequest(String request)

This function adds text to your request, you must have created an instance http.

Returns:
An integer value:
0.- If not successful
1.- If succesful

Example 1:

```
if(httpClient.httpPetition() == 1) {
    httpClient.addRequest("MY REQUEST");
}
```

Note: If order to add a request bigger than 1000 ascii bytes, make consecutive calls to this function. Up to 4,000 ascii bytes can be added to the request, maximum 1,000 ascii bytes at a time.

Example 2:

To add 3200 ascii bytes, you need to make the following consecutive calls.

```
httpClient.addRequest (first 1000 ascii bytes)
httpClient.addRequest (second 1000 ascii bytes)
httpClient.addRequest (third 1000 ascii bytes)
httpClient.addRequest (last 200 ascii bytes)
```

13.4 Function executeMethod(String Method, String url)

This function executes the http request and returns a text received, not including the response headers. The parameter Method can be GET or POST.

Example:

This example uses all of the HTTP available functions and then the executeMethod.

//INIT http structure by calling this function.

```
HttpClient.httpPetition();
```

//add headers and parameters as needed

```
HttpClient.addHeader("Accept-Language", "en-us");
HttpClient.addHeader("Content-Type", "plain/text");
HttpClient.addHeader("User-Agent", "Terminal;VeriFone;O3750");
HttpClient.addHeader("host", "www.myserver.com:8080");
HttpClient.addHeader("Cache-Control", "no-cache");
HttpClient.addHeader("Connection", "Keep-Alive");
HttpClient.addHeader("origin", ori);
HttpClient.addHeader("country", "484");
HttpClient.addHeader("region", "LAC");
HttpClient.addHeader("Content-length", String.toString(String.length(request)));
HttpClient.addParameter("parameter1","134550");

HttpClient.addRequest(request);
```

//call the executeMethod function.

```
response=HttpClient.executeMethod ("POST","https://"+URL or IP+":"+port+"/context");

if (String.length(response) >0) {
    Dialog.alert("This is the response");
    Dialogs.alert(response);
}
```

Note: call ISO.openConnectionIP before calling executeMethod to open de channel.

For example, to open a SSL channel for HTTPS, besides setting SSL variables described at the begging of this library, do the following.

ISO.openConnectionIP (URL or IP, "443")

To open a channel for http without SSL:

ISO.openConnectionIP (URL or IP, "80")

13.5 Function fileCurrentHTTP (String fileUrl)

This function returns an integer value with the total bytes of the file to download.

Example:
httpClient.fileCurrentHTTP("http:\\www.mysite.com\download\update.zip");

13.6 Function fileCurrentSizeHTTP(String fileUrl)

This function returns an integer value with the size in bytes of the file to download.
The *fileUrl* parameter is a string with file path to download.

Example:
httpClient.fileCurrentSizeHTTP("http:\\www.mysite.com\download\update.zip");

13.7 Function getFileHTTP(String fileUrl, String saveAs)

This function allows download a file from internet.
This functiion uses the HTTP 1.1 protocol which allows you to resume downloading from the point where it was interrupted if the download was not finished.

This function returns 200 if succeeds.

Example:

```
httpClient.getFileHTTP("http:\\www.mysite.com\download\update.zip");
if(WMLBrowser.getVar("httpStatus") == "200") {
        Dialogs.alert("Download success");
        UTILS:Unzip(saveAs);
}else {
        Dialogs.alert("Download error");
        UTILS.reset();
}
```

13.8 Function httpPetition()

This function creates an instance for http download. Returns an integer value with the following values:

0.- not successful
1.- succesful

Example:

```
httpClient.httpPetition();
```

13.9 Function resetFileVarsHTTP(String fileUrl)

This function resets the variables in the current download and delete the file.

Example:

```
httpClient.resetFileVarsHTTP("http:\\www.mysite.com\download\update.zip");
```

14 LANG Library

14.1 Function abs (numeric value)

Returns the absolute value of a given *numbe*r. If the number is of integer type, an integer value will be returned. If it is a floating-point type, a floating-point type will be returned.

Parameters:
Value - Is a numeric data.
Note: Avoid passing non-numeric values. Non-numeric value will result in undetermined results.

Returns:
Return the positive value or the input parameter.

Example:

```
var v= -5.8;
var w= Lang.abs(v);              // w= 5.8;

var v= 10;
var w= Lang.abs(v * (-2));       // w= 20;
```

14.2 Function float ()

This function returns a boolean value. If floating point is supported the function returns True. This function remains for compatibility purposes, currently, flote type is support with all terminal models.

Parameters:
None.

Returns:
Returns boolean value.

Example:

```
var v ;
v= Lang.float(); //v=1 (true)
```

14.3 Function isFloat (string value)

Returns an integer boolean value depending on the type evaluation of the parameter

Parameters:
1 If the *parameter* is of floating-point type. Otherwise 0 is returned.

Returns:
Returns boolean value.

Example:
```
var v;
v= Lang.isFloat("89.09");          //true
v= Lang.isFloat("-90.98");         //true
v= Lang.isFloat("alphanumeric");   //false
v= Lang.isFloat("10%");            //false
```

14.4 Function isInt (string value)

Returns an integer boolean value depending on the type evaluation of the parameter

Parameters:
True(1) If the *parameter* is of integer type. Otherwise false (0) is returned.

Returns:
Returns boolean value.

Example:
```
var v;
v = Lang.isInt("89.09");           //false
v= Lang.isInt("-90");              //true
v= Lang.isInt("alphanumeric");     //false
v= Lang.isInt("10%");              //false
```

14.5 Function max (numeric value1, numeric value2)

Returns the maximum value between *two numeric data*.

Parameters:
Value1 - Is a numeric data.
Value2 - Is a numeric data.
Note: Avoid passing non-numeric values. Non-numeric values will result in undetermined results.

Returns:
Returns a numeric value, integer or floating-point type.

Example:

```
var a= 5.1;
var b= 5.8
var x= Lang.max(a,b);          // x= 5.8
var x= Lang.max(28.5,8.9);     // x=28.5
var x= Lang.max(8.0,8);        // x=8.0, when equal value, first parameter format will be returned.
var x= Lang.max(8,8.0);        // x=8, when equal value, first parameter format will be returned.
```

14.6 Function maxInt ()

Returns the maximum integer value. It is used to find out the largest positive number that is supported.

Parameters:
None.

Returns:
Returns an integer type value.

Example:

```
var v ;
v= Lang.maxInt();          // v=2147483647
```

14.7 Function min (numeric value1, numeric value2)

Returns the minimum value between *two numeric data*.

Parameters:
Value1 - Is a numeric data.
Value2 - Is a numeric data.
Note: Avoid passing non-numeric values. Non-numeric values will result in undetermined results.

Returns:
Returns a numeric value, integer or floating-point type.

Example:

```
var a= 5.1;
var b= 5.8
var x= Lang.min(a,b);          // x= 5.1
var x= Lang.min(28.5,8.9);     // x=8.9
var x= Lang.min(8.0,8);        // x=8.0, when equal value, first parameter format will be returned.
var x= Lang.min(8,8.0);        // x=8, when equal value, first parameter format will be returned.
```

14.8 Function minInt ()

Returns the minimum integer value. It is used to find out the largest negative number that is supported.

Parameters:
None.

Returns:
Returns an integer type value.

Example:

```
var v ;
v= Lang.minInt();          // v=-2147483648
```

14.9 Function parseFloat (string value)

Returns a floating-point value defined by the *string parameter*.

Parameters:
Value1 - Is a string data.

Returns:
Returns a floating-point type value.

Example:

```
Var v;
var v= Lang.parseFloat("-89.08");          // v= 89.08
var v= Lang.parseFloat("8.09e-9 Mhz");     // v= 8.09e-9
var v= Lang.parseFloat("-4e9 Hz");         // v=-4.0e9
var v= Lang.parseFloat("Number: 8.90");    // v=0
var v= Lang.parseFloat("76");              // z=76.0
var v= Lang.parseFloat("9.8e mts");        // a= invalid
var v= Lang.parseFloat("9.8e- km/hr");     // b= invalid
```

14.10 Function parseInt (string value)

Returns an integer value defined by the *string parameter*.

Parameters:
Value - Is a string data.

Returns:
Returns an integer type value.

Example:

```
var v;
v= Lang.parseInt("-2°C");          // v=-2
v= Lang.parseInt("289 km/hr");     // v=289
v= Lang.parseInt("sale");          // v=0
v= Lang.parseInt("300.5");         // v=300
```

14.11 Function random (int value)

Returns an integer value with positive sign.

Parameter:
Value – integer value use as seed for randomized, if 0, current ticks value is use to randomized

Returns:
Returns an integer type value.

Example:
```
var v= 25;
var w= Lang.random (25); // w=any integer number.
```

14.12 Function seed (int value)

Initializes the random number generator (randomize)

Parameter:
Value – integer value use as seed for randomized, if 0, current ticks value is use to randomized

Returns:
Returns integer zero.

Example:
```
Lang.seed(1500);        //use number 1500 as seed
Lang.seed(0);           //use internal clock ticks as seed
```

15 STRING Library

15.1 Function length(String value)

Returns the length of the *string* parameter.

Parameter:
Value – any valid string from an empty string to upto 1024 bytes string

Returns:
The length of the argument string in integer form.

Example:

```
var v;
var w;
var a;

v="TPVBrowser";
w="Hello"+" World";
a= String.length("");      // a=0
a= String.length(w);       // a=11
a=String.length(v);        // a=10
```

15.2 Function isEmpty(String value)

This function returns true if the size of the *string* is zero and it returns false otherwise.

Parameter:
Value – Any valid string from an empty string to upto 1024 bytes string

Returns:
True is the string is empty, false if string is not empty

Example:

```
var isEmpty;
isEmpty = String.isEmpty("");              //isEmpty = true (1)
isEmpty= String.isEmpty("any value");   //isEmpty = false (0)
```

15.3 Function charAt(String value, Int index)

This function returns a character of the *string value*, depending of the specified *index* position.

Parameters:
Value – Any valid string from an empty string to up to 1024 bytes string
Index – position of the needed character.

Returns:
A string with the character found at position *index* inside string *value*.

Example:

```
var firstChar;
firstChar = String.charAt("Hello world!", 0);        //firstChar = "H"
```

15.4 Function subString(String stringData, Int startIndex, Int maxLength)

This function returns the string found at position *startIndex* and the length *maxLength*.

Parameters:
stringData – Any valid string from an empty string to up to 1024 bytes string
startIndex – position inside *stringData*, 0 is the first character
maxLength – maximum count of characters to read from *stringData, if number of available characters is less than indicated value, length will be maximum characters available.*

Returns:
A string starting at startIndex position of maxLength or less characters.

Example:
Var data;
data = String.subString("Hello world!", 6, 10); //data="world!"

15.5 Function find(String stringData*, String subString)*

This function returns the index of the first character in *stringData* that matches the *subString* passed as parameter.

Parameters:
stringData – Any valid string from an empty string to up to 1024 bytes string
substring – string to look for inside *stringData*.

Returns:
Position where searched *subString* starts, 0 points to first character.
-1 if searched *subString* is not found in *stringData*.

Example:
var position;
position= String.find("Hello world!", "lo"); //position = 3
position= String.find("Hello world!", "za"); //position = -1

15.6 Function replace(String stringData*, String oldString, String newString)*

This function returns a new string, resulting from replacing all occurrences of *oldString* found in *stringData* by the *newString.*

Parameters:
stringData – Any valid string from an empty string to up to 1024 bytes string
oldString – string to look for inside *stringData*.
newString – string to replace ocurrances of oldString

Returns:
A new string will all occurrences of oldString replaced by newString.

Example:
```
var editedString;
editedString = String.replace("Hello world!", "Hello", "Bye"); //editedString = "Bye world!"
editedString = String.replace("$1,450,321.00", ",", ""); //editedString = "$1450321.00"
editedString = String.replace(editedString, ".", ""); //editedString = "$145032100"
```

15.7 Function elements (String stringData, String separator)

This function returns the number of elements in the given *stringData* separated by the given *separator.*

Parameters:
stringData – Any valid string from an empty string to up to 1024 bytes string
separator – string use to separate elements in *stringData*

Returns:
Number of elements found, 0 if none.

Example:
```
var elements;
elements = String.elements("data;data;data;", ";");        // elements = 3
elements = String.elements("data;data;data;", ".");        // elements = 0 (no point found in string)
```

15.8 Function elementAt (String stringData, Int index, String separator)

This function searches an element into *stringData* that is located in the position given by *index* and is identified by *separator*.

Parameters:
stringData – String where search is going to take place
index – number of element in value. First element is 0, second is 1 and so on.
Separator – string to be used to identify each element.

Returns:
A string with the element text or empty string if not found.

Example:
```
var element;
element= String.elementAt("Hello world!", 0, " ");          //Element = "Hello"
element= String.elementAt("Hello world!", 2, " ");          //Element = "" (only elment 0 and 1 exist)
```

15.9 Function removeAt(String stringData, Int index, String separator)

This function returns a new string where the element and the corresponding *separator* with the given *index* are removed from the given *string*. If the parameter index is less than zero, then the first element is removed. If the parameter index is larger than the number of elements, then the last element is removed.

Parameters:
stringData – String where search is going to take place
index – number of element in *stringData*. First element is 0, second is 1 and so on.
Separator – string to be used to identify each element.

Returns:
A string with the element text removed, may result in an empty string.

Example:
```
var newString;
newString = String.removeAt("Hello world!", 0, " ");        //getNewString = "world!"
```

15.10 Function replaceAt(String stringData, Char replacement, Int index, String separator)

This function returns a new string with the element in *source* specified by *index* replaced with the *replacement* element. If the parameter index is less than zero, then the first element is replaced. If the parameter index is larger than the number of elements, then the last element is replaced.

Parameters:
stringData – String where search is going to take place
replacement – String to be written on top of element
index – number of element in *stringData*. First element is 0, second is 1 and so on.
Separator – string to be used to identify each element.

Returns:
A string with element *replacement* text replaced.

Example:
```
var NewString;
NewString= String.replaceAt("A E I O U", "Z", 4, " "); //NewString = "A E I O Z"
```

15.11 Function insertAt(String stringData, String newElement, Int index, String separator)

This function returns a new string with the *element* and the *separator* inserted at the position specified by *index* inside the *original stringData*. If the parameter index is less than zero, then zero is used as the index. If the parameter index is larger than the number of the elements, then the element is appended at the end of the string.

Parameters:
stringData – String where search is going to take place
newElement – String to be inserted at *index* position.
index – number of element in *stringData*. First element is 0, second is 1 and so on.
Separator – string to be used to identify each element.

Returns:
A string with the *newElement* text inserted.

Example:
```
var newString;
newString = String.insertAt("A E I O U", "Z", 5, " ");        //NewString = "A E I O U Z"
```

15.12 Function squeeze(String stringData)

This function returns a *string* where all the consecutive series of the white spaces, carry returns and line feeds, inside the *string* passed as parameter, are reduced to single space.

Parameters:
stringData – Original stringData

Returns:
A new string with spaces reduce to one per word.

Example:
```
var newString;
newString= String.squeeze("Total is   $1,000.00 \r\n Do you agree   ?");

//NewString = "Total is $1,000.00 Do you agree ?"
```

15.13 Function trim(String stringData)

This function removes all trailing and leading spaces in the given *stringData*.

Parameters:
stringData – Original text string.

Returns:
New string text without leading and trailer spaces.

Example:
```
var newString;
newString = String.trim("    Please Enter PIN   ");  //newString = "Please Enter PIN"
```

15.14 Function compare(String string1, String string2)

This function compares the two given arguments.

Parameters:
string1 – text data to be compared to string2.
string2 – text data to be compared to string1.

Returns:
if *string1* is less than *string2,* the function returns -1. If *string1* and *string2* are equal, the function returns zero. If *string1* is larger than *string2,* the function returns 1.

Example:

```
var result;
result = String.compare("Hello", "Hello");          // result = 0
result = String.compare("Hello", "Bye");            // result = 1
result = String.compare("Bye", "Hello");            // result = -1
```

15.15 Function toString(numericValue)

This function returns the numericValue value passed as parameter in text form
Note: see also addFloat functions to convert floats values with higher precision.

Parameters:
numericValue – numeric, integer or float value.

Returns:
The text representation of numericValue.

Example:
```
var stringData;
stringData= String.toString(3.1416);  //stringData = "3.1416"
```

15.16 Function isNumeric(String stringData)

This function checks if the string argument is a number

Parameters:
stringData – numeric, integer or float value.

Returns:
True if the *string* argument can be converted to a numeric value, otherwise, it returns false.

Example:
```
var isNumber = String.isNumeric("50");                  //isNumber = true
var isNumber = String.isNumeric("50.00");               //isNumber = true
var isNumber = String.isNumeric("Total is $50.00");     //isNumber = false
```

Function toUpper(String stringData)

This function returns stringData in upper case.

Parameters:
stringData – Text data

Returns:
stringData converted to upper case

Example:
```
var newString;
newString= String.toUpper("Total $ 5,000.00 USD");      //newString = "TOTAL $ 5,000.00 USD"
```

15.17 Function toLower(String stringData)

This function returns stringData in lowercase.

Parameters:
stringData – Text data

Returns:
stringData converted to lower case

Example:
var newString;
newString= String.toLower("Total $ 5,000.00 USD"); //newString = "total $ 5,000.00 usd"

15.18 Function padLeft(String stringData, char pad, int len)

This function left pads the string passed as argument.

Parameter:
stringData – original text string.
pad - character to use for padding.
len - final length of the string. The function has no effect if the input string is equal or longer than *len* bytes long.

Returns:
The padded text string.

Example:
var newString;
newString= String.padLeft("Hello!", ".", 9); //newString = "…hello!"

15.19 Function padOAEP(integer SHAType, integer finalStringLen, string stringData, integer stringDataLength, string SHAseed, integer SHAseedLength)

This function right pads the string passed as argument with the OAEP algorithm.

Parameter:
SHAType – 1, 2 or 256
finalStringLen – target Length of padded string.
StringData – original string data to be padded
stringDataLength – length of original data (of *StringData)*
SHAseed – original data to use in SHAx computation
SHAseedLength – length of SHAseed.

Returns:
The padded text string.

Example:
var newString;
newString= String.padOAEP(1,256, "data to be padded", 17, "12424",5);
//newString = data padded with OAEP algorithm.

15.20 Function padRight(String stringData, Char Pad, Int len)

This function pads to the right the string passed as argument.

Parameter:
stringData – original text string.
pad - character to use for padding.
len - final length of the string. The function has no effect if the input string is equal or longer than *len* bytes long.

Returns:
The padded text string.

Example:
var newString;
newString = String.padRight("Hello!", ".", 9); //newString = "hello!..."

15.21 Function formatCurrency(Data, Boolean useComma)

This function formats a string or numeric amount passed as parameter 1 into currency notation, separating the cents with comma or period mark, as well as the thousands.

Parameter:
Data – original amount in text or numeric string.
useComma – if 1, use comma for thousand and period for cents, if 0 use perid for thousand and comma for cents.

Returns:
A text string with formatted amount

Example:
```
var stringAmount;
stringAmount = String.formaCurrency("951700",1);  //stringAmount = "9,517.00"
stringAmount = String.formaCurrency("9,517.00",0);  //stringAmount = "9.517,00"
stringAmount = String.formaCurrency("9,51700",1);  //stringAmount = "9,517.00"
```

16 URL Library

16.1 Function isValid(String url)

This function returns a boolean value that signifies whether the given absolute or relative *url string* is of the correct syntax for a URL. A true is returned if the syntax is correct. A false is returned if the syntax is not correct.

Parameter:
url – a string to be validated for a valid universal resource locator

Example:

```
var isValid = URL.isValid("http://www.host.com/test#func()");
//isValid = true
var isValid = URL.isValid("../test#func()");
//isValid = true
var isValid = URL.isValid("test_://www.host.com/test");
//isValid = false
```

16.2 Function getScheme(Strin url)

This function returns the scheme used in the *url* passed as parameter. There is no resolution of relative Urls into absolute Urls.

Parameter:
url – a string to an universal resource locator

Example:

```
var scheme = URL.getScheme("http://www.host.com/test#func()");
// scheme = "http"
var scheme = URL.getScheme ("www.host.com/test#func()");
//scheme = ""
```

16.3 Function getHost(String url)

This function returns a string value with the host specified in the *url* parameter. If the host is not defined the function returns an empty string value.

Parameter:
url – a string to an universal resource locator

Example:

```
var host = URL.getHost("http://www.host.com/test#func()");
// host = "www.host.com"
```

16.4 Function getPort(String url)

This function returns an integer value with the port number specified in the *url* parameter. If the port number is not specified in the *url* parameter then returns a string empty.

Parameter:
url – a string to an universal resource locator

Example:

```
var port = URL.getPort("http://www.host.com:80/test#func()");
// port = 80
var port = URL.getPort ("w.mail@host.com/test#func()");
//port =""
```

16.5 Function getPath(String url)

This function returns the path specified in the *url* parameter.

Parameter:
url – a string to an universal resource locator

Example:

```
var path = URL.getPath("http://www.host.com/test#func()");
// path = "/test"
```

16.6 Function getParameters(String url)

This function returns the parameters specified in the *url* parameter of last path segment. If not exists any parameter the function returns an empty string.

Parameter:
url – a string to an universal resource locator

Example:

```
var param = URL.getParameters("http://www.host.com/test;1;2;3");
// param = "1;2;3"
var param=URL.getParameters ("http://www.host.com/test;1;2;3/test1");
// param = ""
```

16.7 Function getQuery(String url)

This function returns the query specified in the string *url string* parameter. If query not specified in the *url string* parameter then returns an empty string.

Parameter:
url – a string to an universal resource locator

Example:
var Query=URL.getQuery("http://www.host.com/test?id=1&x=2");
// Query = "id=1&x=2"

16.8 Function getFragment(String url)

This function returns the fragment part used in the *url*. The fragment of a *url* is the part that begins with the # character. If no fragment specified, then the function returns an empty string.

Parameter:
url – a string to an universal resource locator

Example:
var fragment = URL.getFragment("http://www.host.com/test#frag");
// fragment = "frag"

16.9 Function resolve(url String, String embeddedUrl)

This function returns an absolute URL that is formed by *embeddedurl* into *url string*.

Parameter:
url – a string to an universal resource locator
embeddedUrl – string with query or context to be added to the URL.

Example:
var newUrl = URL.resolve("http://www.host.com/", "test#frag");
// newUrl = "http://www.host.com/test#frag"
var newUrl = URL.resolve("http://www.host.com","c");
// newUrl = "http://www.host.com/c"
var newUrl = URL.resolve("http://www.host.com", "?q");
// newUrl = "http://www.host.com/?q"

16.10 Function escapeString(String)

This function replaces special characters in *string* with the corresponding hexadecimal escape sequence according to the rules of url escaping.

Parameter:
url – a string to an universal resource locator

Example:
var escape = URL.escapeString("http://host.com/wml/");
// escape = "http%3a%2f%2fw3host.com%2fwml%2f"

16.11 Function unescapeString(string)

This function replaces hexadecimal escape sequences in *string* with the corresponding character according to the rules of URL unescaping.

Example:
var unescape = URL.unescapeString("http%3a%2f%2fw3host.com%2fwml%2f");
// unescape = "http://host.com/wml/"

17 WMLBROWSER Library

17.1 Function getVar(String varName)

This function returns the value of the variable that matches the given *varName* parameter and that was stored with WMLBrowser.setVar or WMLBRowser.setEnv, in the current context of the WMLBrowser. If the variable does not exist then, an empty string is returned.

Parameter:
varName – Name of the variable.

Returns:
String value of the parameter or empty string if not found.

Example:
WMLBrowser.setEnv("myVariable","123");
Var data=**WMLBrowser.getVar**("myVariable"); // var="123"

17.2 Function setVar(String varName, String value)

This function stores the string value of the second argument into the variable indicated in the first argument.

Variables stored with this function are transient. They will be lost at next power cycle.

Parameters:
varName – variable name. Any alphanumeric string
value – value to be store in the first argument – any alphanumeric string

Returns:
returns a boolean value. If the given variable in the *varName* is initialized with the value given for *value*, returns true, otherwise it returns false. Function would fail only if the system runs out of memory.

Example:
WMLBrowser.setVar("myVariable","123");

17.3 Function setEnv(String varName, String value)

This function stores the string value of the second argument into the variable indicated in the first argument.

Variables stored with this function are persistent. Their value will prevail between power cycles

Parameters:
varName – variable name. Any alphanumeric string
value – value to be store in the first argument – any alphanumeric string

Returns:
returns a boolean value. If the given variable in the *varName* is initialized with the value given for *value*, returns true, otherwise it returns false. Function would fail only if the system runs out of memory.

Example:
WMLBrowser.setEnv("myVariable","123");

17.4 Function putEnv(String varName, String value)

This function stores the string value of the second argument into the variable indicated in the first argument.

Variables stored with this function are kept in the file config.sys of a verix system and are persistent. Their value will prevail between power cycles. In other systems, this function behaves like WMLBrowser.setEnv().

In verix systems, This function uses NAND memory which has a limit of several million write cycles and should not be use loosely. For normal variables use setEnv or setVar functions.

Parameters:
varName – variable name. Any alphanumeric string
value – value to be store in the first argument – any alphanumeric string

Returns:
returns a boolean value. If the given variable in the *varName* is initialized with the value given for *value*, returns true, otherwise it returns false. Function would fail only if the system runs out of memory.

Example:
WMLBrowser.putEnv("myVariable","123");

17.5 Function getEnv(String varName)

This function returns the value of the variable that matches the given *varName* parameter and that was stored with WMLBrowser.putEnv. If the variable does not exist then, an empty string is returned.

Parameter:
varName – Name of the variable.

Returns:
String value of the parameter or empty string if not found.

Example:
WMLBrowser**.putEnv**("myVariable","123");
Var data=**WMLBrowser.getEnv**("myVariable"); // var="123"

17.6 Function putEnvSec(String varName, String value)

This function stores the string value of the second argument into the variable indicated in the first argument.

Variables stored with this function are kept encoded (SECured) apart from variables stored with setVar, setEnv and putEnv and are persistent. Their value will prevail between power cycles.

This function uses NAND memory which has a limit of several million write cycles and should not be use loosely. For normal/general variables use setEnv or setVar functions.

Parameters:
varName – variable name. Any alphanumeric string
value – value to be store in the first argument – any alphanumeric string

Returns:
returns a boolean value. If the given variable in the *varName* is initialized with the value given for *value*, returns true, otherwise it returns false. Function would fail only if the system runs out of memory.

Example:
WMLBrowser.putEnvSec("myVariable","123");

17.7 Function getEnvSec(String varName)

This function returns the value of the variable that matches the given *varName* parameter and that was stored with WMLBrowser.putEnvSec. If the variable does not exist then, an empty string is returned.

Parameter:
varName – Name of the variable.

Returns:
String value of the parameter or empty string if not found.

Example:
WMLBrowser**.putEnvSec**("myVariable","123");
Var data=**WMLBrowser.getEnvSec**("myVariable"); // var="123"

17.8 Function go(String url)

This function loads the URL specified in the *url* parameter. If url String begins with f: or i:, the wml card will be search in the ram memory or in the flash memory.

This function only stores the URL. The system will not go to that URL immediately. The system will go to the URL when it reaches a return statement.

Parameter:
URL - string containing the Uniform Resource Locator to reference.

Returns.
Not significant integer value.

Example
WMLBrowser.go("http://www.host.com/test)");
WMLBrowser.go("f:myfile.wmlsc#test()"); //call function test() in file f:myfile.wmlsc

17.9 Function newContext()

This function allows clear all the variables associated with WML context and the navigation historic stack, except for the current card, before returning control to the calling function.

Parameter:
None.

Returns.
Not significant integer value.

Example:
WMLBrowser.newcontext();

17.10 Function beep(Int frequency, Int repetitions)

This function emits a sound on the output device, with the frequency specified by *freq*, and the repetitions specified by *the second parameter.*

Parameter:
Frequency – tone to be use for a 500 milisecond sound
Repetitions – number of tones to use.

Returns.
Not significant integer value.

Example:
WMLBrowser.beep(100, 5);

17.11 Function isConnected()

This function returns a boolean value depending on the physical media connection status.

This function does not refer to logical socket connections. It refers to physical connection to the network.

Parameter:
None.

Returns.
1 – if connected to the media (cellular, ethernet, wifi)
0 – not connected to the media.

Example:
```
If (WMLBrowser.isConnected()) {
        //proceed to open a socket
}
```

17.12 Function getSignalStrength()

This function returns an integer between 0 and 100 representing the signal strength of the current connection.

Parameter:
None.

Returns.
Number from 0 to 100. Ethernet media will return 0 or 100 only depending on its physical connection status.

Example:
```
If (WMLBrowser.getSignalStrength()<10) {
        //notify user of poor signal level
}
```

17.13 Function getBatteryLevel()

This function returns the battery level of the terminal POS. The result is given in the range from 0 to 100.

Parameter:
None.

Returns.
Number from 0 to 100. Systems with no battery attached will return 0.

Example:
```
If (WMLBrowser.getBatteryLevel()<10 && WMLBrowser.power()==0) {
        //ask user to plug terminal to power
}
```

17.14 Function power()

This function checks the terminal POS is connected to power.

Parameter:
None.

Returns.
1 if plugged to power outlet, 0 if unplugged.

Example:
```
If (WMLBrowser.getBatteryLevel()<10 && WMLBrowser.power()==0) {
        //ask user to plug terminal to power
}
```

17.15 Function requestSignal()

The system reads signal from the radio periodically (every 30 second or so when in idle). If you need to know the exact current signal level, before calling function getSignalStrength(), call this function to ask the system to get the signal level.

Calling this function may take up to 500 miliseconds.

Parameter:
None.

Returns.
Not significant integer value.

Example:
```
WMLBrowser.requestSignal();
If (WMLBrowser.getSignalStrength()<10) {
        //notify user of poor signal level
}
```

18 RECORDSTORE Library

TPVbrowser implements WML standard file stores as a collection of data file of variable length record plus an automatic index to make it possible to make fast searches.

Note: for short, standard, fix-length files; openFile, createFile collection of functions should be used instead of recordStore functions as less memory is required.

18.1 Function addFieldtoStoreBuffer(int fieldNumber, string data)

This function inserts the given data into the field number specified. If the field number is greater than the total number of fields, all needed empty field will be created. If the field numbe is less than the total number of filed, the new data will be inserted and the size fo the record will be adjusted automatically.

Parameters:
fieldNumber – number of field to add/replace.
data – information to be store in the indicated field. Data should be in string format.

Returns:
positive number if successful.

Example:
var autNumber="1234";
rc = RecordStore.addRecord(1,autNumber);

18.2 Function int addRecord(Int storeId, String record)

This function appends a new data record to the indicated file store. Data for the new record is provided in the function itself.

Parameters:
storeId – file handler returned by functions openStore
record – Text string to be appended. Record may be of variable length but it recommended to use only text strings.

Returns:
positive number if successful.

Example:

rc = RecordStore.addRecord(storeId, "street 11;Eddievedder;1111111111");

18.3 Function addRecordFromBuffer(int storeId)

This function appends a new data record to the indicated file. Data is taken from the internal store buffer.

Parameters:
storeId – file handler returned by functions openStore

Returns:
positive number if successful.

Example:
```
rc = RecordStore.addRecordFromBuffer(storeId);
```

18.4 Function clearStoreBuffer()

This function allocates and reset to nulls the internal buffer use by functions addFieldToStoreBuffer, addRecordFromStoreBuffer, setRecordFromBuffer,getRecordToBuffer and getFieldFromStoreBuffer.

This function should be called only once prior to any of the above functions.

Parameters:
None.

Returns:
Not meaningfull integer

Example:
RecordStore.clearStoreBuffer();

18.5 Function closeStore(Int storeId)

This function closes a store file

Parameters:
storeId – file handler returned by the function openStore.

Returns:
An integer grater or equal to zero when successful.

Example:
var result = RecordStore.closeStore(storeId);

18.6 Function createDatabase(String fileName, String nameSource)

Creates a new store name filename from a table received from the file nameSource-

Parameters:
fileName - name of the recordStore to be created.
nameSource - name of the source file.

Returns:
If successful, the function returns a positive number which is the handler of the new file

Example:
var storeId= createDatabase("myFile","sourceFile");

Function deleteRecord(Int storeId, Int recordId)

This function delete the indicated record from the store file.

Parameters:
storeId – file handler returned by functions openStore
recordId – Number of the record to be deleted.

Returns:
positive number if successful.

Example:
rc = RecordStore.deleteRecord(storeId, 4);

18.7 Function deleteStore(String name)

This function deletes a store file. The associated file index is also deleted.

The store file must be closed before calling to this function.

Parameters:
Name of the store file to be deleted.

Returns:
This function returns a positive integer if successful.

Example:
rc=RecordStore.deleteStore("myfile");

18.8 function MX_GETFIELDFROMSTOREBUFFER(int fieldNumber)

Gets the information from the indicated field from the internal store buffer.

Parameters:
Number of the field to fetch data from.

Returns:
This function returns the data found in the field in string format.

Example:
var data=RecordStore.getFieldFromStoreBuffer(4);

18.9 Function getNextRecordId(Int storeId, Int recordId)

This function returns the index of the next record if succeeds, if not returns invalid.

Parameters:
storeId – file handler returned by functions openStore
recordNumber – Record number to read and retrieve. Please note the first record is 0, the second is 1 and so on.

Returns:
A positive number means it is the index of the next record.

Example:
rc= RecordStore.getNextRecordId(storeId, 3);

18.10 Function getNumRecords(Int storeId)

This function gets the number of records in a given store file.

Parameters:
storeId – file handler returned by the function openStore.

Returns:
An integer grater or equal to zero

Example:

var records = RecordStore.getNumRecords(StoreId);

18.11 Function string getRecord(Int storeId, Int recordNumber)

This function reads a record from the indicated store file.

Parameters:
storeId – file handler returned by functions openStore
recordNumber – Record number to read and retrieve. Please note the first record is 0, the second is 1 and so on.

Returns:
Record data. Empty if record not found.

Example:
dataBuffer= RecordStore.getRecord(storeId, 0);

18.12 Function getRecordSize(Int storeId, Int recordId)

This function returns the size in bytes of the record if recordId exists.

Parameters:
storeId – file handler returned by functions openStore
recordNumber – Record number to read and retrieve. Please note the first record is 0, the second is 1 and so on.

Returns:
Length of the record. -1 if error.

Example:
var sz= RecordStore.getRecordSize(ag, record1);

18.13 Function getRecordToBuffer(int StoreId, Int recordNumber)

This function reads a record from the indicated store file. Data will be stored in the internal working buffer and will not be passed to the script

Parameters:
storeId – file handler returned by functions openStore
recordNumber – Record number to read and retrieve. Please note the first record is 0, the second is 1 and so on.

Returns:
A positive number if data was read.

Example:
rc= RecordStore.getRecord(storeId, 0);

18.14 Function getSize(Int storeId)

This function receives as parameter the handle (*storeId*) of the opened store and returns an integer with the data file size in bytes.

Parameters:
storeId – file handler returned by the function openStore.

Returns:
An integer grater or equal to zero

Example:
var S = RecordStore.getSize(storeId);

18.15 Function getTotalRecords(String fileName)

Gets the total number of records in an store file. This function opens and closes the recordStore automatically. In case the recordStore is already open this function closes the recordStore.

Parameters:
fileName - name of the recordStore to be created.

Returns:
Integer with the total of records stored.

Example:
```
var Total Records = RecordStore.getTotalRecords("myStoreFile");
```

18.16 Function int openStore(String name, boolean create)

This function opens a file in the data base. If the file does not exist, the function may create it if parameter *create* is TRUE.

Returns an integer with a handle of the open store, or -1 in case it fails to open or create the file.

Parameters:
Name – name of the file in the database. No extension should be used as system will create file *name*.dat and *name*.idx. On Ingenico terminals name should be no longer than 11 characters. Always use UPPERCASE to be compatible with Ingenico terminals.

Create – Boolean, if TRUE and file does not exist, the function will create the file.

Returns:
File handler with value zero or greater than zero if successful.

Example:

```
var storeId = RecordStore.openStore("myfile", true);
```

The system will open file myfile.dat and its index myfile.idx. If the file does not exist it will be created.

18.17 Function setRecord(Int storeId, Int recordNumber, String record)

Completely replaces data of a given record in a file store. **The size of the new record should be the same size of the previously existing record.**

Parameters:
storeId - handle of the recordStore to be use.
recordId - index of the record to be changed, the first record is index 0, the second is index 1, etc.
record – text data containing the new values to be recorded.

Returns:
If successful, the function returns the index of the record returns -1.

Example:
Var rc = RecordStore.setRecord(storeId, 4, "street 11;Eddievedder;5555555555");

18.18 Function setRecordFromBuffer(int StoreID, int recordNumber)

Completely replaces data of a given record in a file store. Data is taken from the internal store buffer.

Parameters:
storeId - handle of the recordStore to be use.
recordId - index of the record to be changed, the first record is index 0, the second is index 1, etc.

Returns:
If successful, the function returns the index of the record. If failure returns -1.

Example:
Var rc = RecordStore.setRecordFromBuffer(storeId, 4

18.19 Function findRecord(Int storedId, String key, Int pos, String separator, Int ordered)

In this function each register of the recordStore whose handle is *StoredId* will be processed as a sequence of fields separated by the *Separator*. This function searches a string *Key* located in the position *Pos* related to the *Separator*. If the corresponding field is a primary key of the store and the record store assigned to the handle *StoreId* is ordered by the desired field, the parameter *Ordered* may be true, allowing it to perform a binary search on the record store.

This function returns the record number and the content of that register.

Example:
RecId = RecordStore.findRecord (table, "5", 2, ";",false);
// searches the record which has value 5 after the second ";".

18.20 Function findRecordData(Int storedId, String key, Int pos, String separator, Int ordered)

In this function each register of the recordStore whose handle is *StoredId* will be processed as a sequence of fields separated by the *Separator*. This function searches a string *Key* located in the position *Pos* related to the *Separator*. If the corresponding field is a primary key of the store and the record store assigned to the handle *StoreId* is ordered by the desired field, the parameter *Ordered* may be true, allowing it to perform a binary search on the record store.

This function only returns the content of the register found.

Example:
RecId = RecordStore.findRecordData (table, "5", 2, ";",false);
// searches the record which has value 5 after the second ";".

18.21 Function mergeOrderedDB(Int storeId, Int source, Int pos, String separator)

This function appends the recordSotreA whose handle is *Source*, into the recordStroreB whose handle is *StoreId*. The parameters *Pos* and *Separator* are keeping for compatibility and future uses.

Example:
Merge DB = RecordStore.mergeOrderedDB(40,1,0,NULL)

19 PRINTER Library

19.1 Function open()

This function opens the printer.

Parameters:
None.

Returns:
if it succeeds, it returns a bigger than zero value

Example:
var result=Printer.open();

19.2 Function setWidthMode(Int mode)

This function set normal or double text width.

Parameters:
Mode – 1 for normal width, 2 for double width.

Returns:
Not significant.

Example:
Printer.setWidthMode(2);

19.3 Function setHeightMode(Int mode)

This function set normal or double text height.

Parameters:
Mode – 1 for normal height, 2 for double height.

Returns:
Not significant.

Example:
Printer.setHeightMode(2);

19.4 Function print(String text)

This function prints the content of the parameter *text*. This function returns true if the print succeeds, otherwise returns false.

Parameter:
Text – string of data to be printed.

Returns.
Number of characters printed.

Example:
Printer.pprint("Hello world!");

19.5 Function printLn(String text)

This function prints the content of the parameter *var* and adds a new line character after the print.

Example:
Printer.pprintLn("\n\n");

19.6 Function close()

This function closes the printer. This function should be call after any printing operation (receipt, report, etc) is done to assure all date in buffer gets printed.

Parameters:
None.

Returns:
not significant value.

Example:
Printer.pclose();

19.7 Function PDF417(string fileName)

This function prints a pdf 417 bidimensional bar code graphic.
Note: this functions opens and closes the printer. To print anything else after calling this function, please call Printer.open() function previously.

Parameters:
fileName – that to be represented on pdf 417 graphic will be gotten from this file.

Returns:
not significant value.

Example:
Printer.PDF417("I:TED.DAT");

19.8 Function int printLogo(String File, Int Aligment)

Prints a WBMP logo.
Note: At this and any print operation, TPVBrowser will check for paper present. If there is no paper it will pause until paper is provided sending a proper message on the screen. The system will resume 5 seconds after detecting paper, to give time to the user to close the printer cover.

Parameters:
File – name of the .WBMP file, extension should be specified.
Aligment - 0 left, 1 center, 2 right

Returns:
not significant.

Example:
Printer.pprintLogo("BANKLOGO.WBMP", 1)

19.9 Function setInverseMode(Int isInverse)

This function is employed to indicate to the printer that the following text should be displayed in format inverse.

Example:

```
Printer.popen();
Printer.psetInverseMode(1); // format inverse
Printer.pprint("This text is inverse!!!");
Printer.psetInverseMode(0); // format inverse
Printer.pprint("This text is not inverse!!!");
Printer.pclose();
```

19.10 Barcode

To print barcodes, use the print logo function, instead of a file name use the following as a parameter:

Printer.printLogo("BARCODE,type,height,value,showvalue",aligment);

Parameters:

- ***BARCODE***: is a constant string.
- ***type:*** EAN (EAN-8 or EAN-13 will be established by the length of 'value'), 2OF5 or 3OF9.
- ***height:*** in pixels, min is 20 and max is 100.
- ***value:*** value needed in bar code.
- ***showvalue:*** 0-does not print the value below bar code, >0 prints the value.
- ***Aligment:*** 0 - left, 1-centered, 2-right

Example:

```
Printer.open();
Printer.printLogo("BARCODE,EAN,30,044231189012,1", 1); //12 digits, it is EAN-13
UTILS.pause(1000); //if printing more than one bar code at once, allow some time between them.
Printer.printLogo("BARCODE,EAN,30,0442311,1", 1); //7 digits, it is EAN-8
UTILS.pause(1000); //if printing more than one bar code at once, allow some time between them.
Printer.printLogo("BARCODE,2OF5,50,0442311,1", 1);
Printer.close();
```

NOTES:

1) Rules must be followed in terms of length for EAN-8 or EAN-13 otherwise, the browser wont be able to print a bar code.

2) 3OF9 is a wider bar code and can hold a limited number of characters in the POS printers.

3) Before using this function, you most open the printer with Printer.open(), then invoke the function.

Do not forget to close the printer when you are done using it.

20 SYSTEM Library

20.1 Function currentTimeSecs()

This function returns a long value with the current POS time in seconds since 01/01/1970 00:00:00.

Example:

```
var tS = System.currentTimeSecs();
```

For example if the current POS time is 1/1/1970 00:00:01 the function returns 1.

20.2 Function datetime2Seconds(String YYYYMMDDhhmmss)

This function returns a long value with the number of seconds passed from 01/01/1970 00:00:00 to the value passed as parameter, the format of parameter must be "YYYYMMDDhhmmss".

Example:

```
var tS= System.datetime2Seconds("19700102020000");
var tS= 93600
```

20.3 Function seconds2datetime(Int seconds)

This function returns the date and time corresponding to the number of seconds passed as parameter, the value of return is with the format "YYYYMMDDHHMMSS".

Example:

```
var Sdt= System.seconds2Datetime (93600);
var Sdt= 19700102020000
```

20.4 Function isValidDate(String YYYYMMDDhhmmss)

This function returns a boolean value. If the value passed as parameter is a valid date returns true, otherwise returns false.

Example:

```
var  IVD= System.isValidDate(20000231000000);
// var IVD = false.
```

20.5 Function datetime()

This function returns the current POS date and time. The value returns with format "YYYYMMDDhhmmss".

Example:
var Sdt= System.datetime ();

20.6 Function currentTicks()

This function returns a number in milliseconds since the POS was started.

Example:
var nT = System.currentTicks( );

20.7 Function setTimePos(String Time)

This function sets the clock with the *Time* passed as parameter.The parameter has the format: ""YYYYMMDDhhmmss"

Example:
var Timeok = System.setTimePos(20110708105003)

20.8 Function getTerminalModel()

This function returns a string value with the terminal model

Example:
System.getTerminalModel(); // maybe V5, Vx510, M4230, I5100

20.9 Function getDateFormat(String date, String format)

This function returns a string value with the indicated format.

Parameters:
date – date to be formatted
format – string with the required format. Use the following codes as needed.

YYYY = year
MM = month
DD = Day
hh = hour
mm = minutes
ss = seconds

Returns:
A string with the formatted date.

Example:
System.getDateFormat(System.datetime(), "MM/YY"); // 08/11
System.getDateFormat(System.datetime(), "AA/MM/DD HH:MM:SS"); // 11/08/07 09:30:00
System.getDateFormat(System.datetime(),"DD MMM, AAAA hh:mm"); // 08 Jul, 2011 09:30

20.10 Function getTerminalComm()

This function returns a string value with the type of communications configured in the terminal.

Example:

System.getTerminalComm(); // ETHERNET, WIFI, GPRS, DIALUP, BLUETOOTH

20.11 Function vScreenSize()

Returns the vertical size of the screen in pixels

Parameters:
None

Returns:
Integer number, the number of pixels.

Example:
If (System.vScreenSize()==240) …

20.12 Function hScreenSize()

Returns the horizontal size of the screen in pixels

Parameters:
None

Returns:
Integer number, the number of pixels.

Example:
If (System.hScreenSize()==320) …

20.13 Function colorScreen()

Indicated if the running system supports color display.

Parameters:
None

Returns:
Integer number, 1 if color screen. 0- monocromatic screen

Example:
If (System.colorScreen()) …

20.14 Function touchScreen()

Indicates if the running system supports touch screen display.

Parameters:
None

Returns:
Integer number, 1 if touch screen is supported. 0- no tocuhscreen

Example:
If (System.touchScreen()) …

20.15 Function wifiCapable()

Indicates if the running system supports WIFI communications.

Parameters:
None

Returns:
Integer number, 1 if wifi is supported. 0- no wifi

Example:
If (System.wifiCapable()) …

20.16 Function ethernetCapable()

Indicates if the running system supports ethernet communications.

Parameters:
None

Returns:
Integer number, 1 if ethernet is supported. 0- no ethernet

Example:
If (System.ethernetCapable()) …

20.17 Function cellularCapable()

Indicates if the running system supports cellular communications.

Parameters:
None

Returns:
Integer number, 1 if cellular is supported. 0- no ethernet

Example:
If (System.cellularCapable()) …

20.18 Function pinpadCtlsMode()

Set the external pinpad to contactless mode (verifone only with pinpad 1000se)
This function may take up to 4 seconds to execute.

Parameters:
None

Returns:
Integer number, 1 if successful. 0- if failure

Example:

```
If (System.pinpadCtlsMode()) {
        //ask to tap a card
}
```

20.19 Function pinpadNormalMode()

Sets the external pinpad to PIN capture mode (verifone only with pinpad 1000se).
This function may take up to 4 seconds to execute.

Parameters:
None

Returns:
Integer number, , 1 if successful. 0- if failure

Example:

```
If (System.pinpadNormalMode()) {
        //ask for PIN
}
```

21 ISO Library

21.1 Function readPackager(String pathFile)

This function reads a list of fileds formats from the packager structure file use to build the ISO-8583 message. The parameter *pathFile* is the path of the file with the packager definition. This function must be executed before the *ISO.performISOTransaction* function.

The structure of the file is a list of formats per ISO8583 field as follows:

fieldNumber=format string --- field number goes from 0 to 128. Format string is as follows

IFB_NUMERIC;x	A BCD number of max length of x nibbles
IFB_LLNUM;x	A BCD number of max length x nibbles and a BCD length field of 2 nibbles.
IFB_CHAR;x	An alphanumeric string of exactly x bytes.
IFB_LLCHAR;x	An alphanumeric string of up to 99 bytes with a length BCD header of 2 nibbles
IFB_LLLCHAR;x	An alphanumeric string of up to 999 bytes with a length BCD header of 3 nibbles (padded to the left to 4 nibbles).
IFB_LLBINARY;x	String of bytes of any value (0x00 to 0xFF) of up to x bytes with a BCD header length of 2 nibbles.
IFB_BINARY;x	String of bytes of any value (0x00 to 0xFF) of exactly x bytes.

Example of the file:

52=IF_BINARY;8	field 52 is binary of 8 bytes
53=IFB_LLBINARY;48	field 53 is binary with header length of 2 nibbles up to 48 bytes
54=IFB_LLLCHAR;120	field 54 is alphanumeric with header length of 3 nibbles of up to 120 bytes.
55=IFB_LLLHEX;999	field 55 is hex format with a header length of 3 nibbles of up to 999 bytes.

Special cases:

Field 0 is use for the message type value (0400, 0200, etc)

0= IFB_NUMERIC;4	A BCD number of max length 4

Field 1 refers to bitMap

1=IFB_BITMAP;8 or 16	ISO8583 bitmap of 8 bytes (64 fileds) or 16 bytes (128 fields). Use with field 1 always.

Example of the function call.

ISO.readPackager("ISOFORMA.INI");

21.2 *Function* performISOTransaction *(String host, Int port, String channel, String header, String trailer, int Header_Lenght, int Include_Header)*

This function builds and ISO8583 message base on the values previously set in the WMLBrowser.setVar variables "F00" to "F128" and on the packager configuration file (previously defined with readPackager). The function then sends the message to the host and receives its response.

Parameters:
Host: string with the host name or host URL to send the message.
Port: integer with the port number.
***Channel*: C**hannel reflects how the message length is stored in the message. The available channels are: "ASCII", "RAW", "NAC" and "NCC".
***Header*:** Normally the Header is the TPDU included before the built packet.
Trailer:Trailer is located at the end of the built packet and normally is blank.
Header_Length: 1-To indicate if the length of the TCP HEADER is included inside the packet
Include Header: 1- To indicate if the TCP HEADER will be included

Connection timeout value is defined with WMLBrowser.putEnv("#MX_CONNECTTO","XX")
XX goes from 10 to 60 seconds. If not specified default is 10 seconds.

Response timeout value is defined with WMLBrowser.putEnv("#MX_IPTO","XXX")
XX goes from 1 to 120 seconds. If not specified default is 40 seconds.

Returns.
0 – operation ran ok
-1000, -1001 no conection to the socket
-1002 – Socket was connected, package could not be transmitted
-1003 – package transmitted but no response received after waiting period.

Example:
```
Var TPDU="6001010000 ;
rc=ISO.openConnectionIP(IP,port);
rc=ISO.performISOTransaction("","","NAC",TPDU,"",0,1);
```

Not IP address or Port is specified in this call ("","",…) as the sockets is previously open with ISO.openConnectionIP in this example. NAC is a constant within normal acquirer operations. TPDU, any valid TPDU for your organization. The ISO8583 will include and TCP length header (…0,1) but the header will not count its own length (…0,1)

21.3 *Function openConnectionIP(String host, int port)*

This function opens a socket to IP communications. Returns a positive integer value if successful.

Example:
```
ISO.openConnectionIP("www.mysite.com", 1600);
```

21.4 Function openConnectionDialup(String phone1, String phone2, String pbx)

This function opens a socket to the Dial Up communication. This function attempts to establish communication with the first parameter phone1, if it fails, attempts with the second parameter phone2. The parameter *pbx* must be the external line access code.
Returns a positive integer value if successful.

Example:
ISO.openConnectionDialup("551234567", "557654321", "9");

21.5 Function closeConnection()

This function closes a PREVIOUSLY OPEN connection. This function is used when is configured Dial Up communication.

Example:
ISO.closeConnection();

21.6 Function clearFields()

This function clear all variables employed by the ISO message.
This function must be call before transactMessage function, because this function sets the "F" fields with 0.

Example:
ISO.clearFields();

21.7 Function charToHex(String)

This function receives a String as input and returns another string with the Hexadecimal representation of the input string in ASCII.

Example:
Var strhex;
strhex = ISO.charToHex("123456");
srthex = "313233343536"

21.8 Function hexToChar(String hexa)

This function receives as input a Hexadecimal representation of an ASCII string and returns the ASCII string. The returning string will always have the half size of the source string. If the source string length is odd, the function will return invalid.

Example:
var result;
result = ISO.hextoChar("313233343536");
result = "123456"

21.9 Function hexToInt(string hexa)

This function receives a Hexadecimal representation of an Integer. The Integer in the script engine is four bytes long, so the Hexa parameter can be a string of 2, 4, 6 or 8 character long, representing the Integer bytes. The returning value is the Integer value or invalid if the conversion was not successful.

Example:
var Intnumber = hexToInt("ff");
// intnumber = 255

21.10 Function intToHex(Int Value, Boolean BigEndian)

This function receives an integer *Value* and returns a Hexadecimal representation of an integer. The Integer in the script engine is four bytes long, so the Hexadecimal can be a string of 2, 4, 6 or 8 character long, representing the Integer bytes. This representation can be as a "Big Endian" or "Little Endian", and this is formed in the Boolean parameter BigEndian (if working with a big endian representation, set it to true). This function returns invalid if the conversion was not successful.

Example:
hexastr = ISO.intToHex(10, true);
//hexastr = A

21.11 Function hexToBin(String hexa)

This function receives a Hexadecimal string and returns a binary representation.

Example:
binstr = ISO.hexToBin(A);
//binstr = 1010

21.12 Function intToChar(Int Number)

This function receives as parameter *Numbe*r and returns the character represented by that number based on its value in the ascii table. The value of *Number* can go from 0 to 255.

Example:
Var CharacterValue = ISO.intToChar(64);
// CharacterValue = @

Var CharacterValue = ISO.intToChar(65);
// CharacterValue = A

21.13 Function getBuiltMsgWsaexp(String List, String Channel, String Header, Int Header_length, Int Include_Header)

This function sends an ISO-8583 message based on the packager configuration. The packager is configured with readPackager function.

List**:** string with the list of fields to send. The fields are named by the rule "f<filedId>" and separated by ';' character
Channel**: C**hannel reflects how the message length is stored in the message. The available channels are: "ASCII", "RAW", "NAC" and "NCC".
Header**:** Normally the Header is the TPDU included before the built packet.
Header_Length:To indicate if the length of the TCP HEADER is included inside the packet
Include Header:To indicate if the TCP HEADER will be included

Example:

```
Send the fields 00, 03, 04, 11, 41
First setter the variables:
Rule f<filedId>
WMLBrowser.setVar("f00", "800");
WMLBrowser.setVar("f03","000000");
WMLBrowser.setVar("f11","111111");
WMLBrowser.setVar("f41","60000004");

ISO.transactMessage( "f0;f3;f11;f41;","NAC","6011112222", 2,1)
```

21.14 Function getResponseMsgWsaexp(String Package, Int Header)

This function receives in the parameter *Package* an ISO-8583 message based on the packager configuration, and the parameter *Header* indicating if the message includes header.
This function breaks the message into variables from f0 to f64, corresponding to the configured passed message.

22 Reversal handling

22.1 Function savePotentialReversal(string filename)

Saves all the previously define ISO8583 fields (variables f00 to f128) into a file with the indicated name.

Parameter:
Filename – String, name of the file to be use to store the reversal data. Should be no more than 12 characters.

Returns:
No meaningful integer value

Example:
ISO.savePotentialReversal("reverse");

22.2 function ISO.restoreReversalData(string filename)

Build ISO8583 message base on the data stored in file which name is passed as parameter. This file should had previously been created with savePotentialReversal.

Parameter:
Filename – String, name of the file to be used to build the reversal message.

Returns:
No meaningful integer value

Example:
ISO.restoreReversalData ("reverse");

22.3 Function ISO.deletePotentialReversal(string filename)

Deletes the reversal file indicated in the parameter. Do not use deleteFile to delete a reversal file, please use this function.

Parameter:
Filename – String, name of the file to be deleted.

Returns:
No meaningful integer value

Example:
ISO.deletePotentialReversal ("reverse");

23 Serial Library

23.1 Function openSerial(String port, Char speed, String config)

This function opens the serial port specified. This function receives three parameters:
port indicates the port that will be open and this parameter must be a string of 4 characters ASCII.
speed indicates the port speed, this parameter must be a string of one character ASCII, and it´s possible values are:

- "2" : 1200 bps
- "3" : 2400 bps
- "4" : 4800 bps
- "5" : 9600 bps
- "6" : 19200 bps
- "7" : 38400 bps
- "8" : 57600 bps
- "9" : 115200 bps

config indicates the serial configuration, and this parameter must be a string of 3 characters ASCII. These characters indicate the following, data, parity and stop bits: **8N1, 7E1, 7N1, 701, 8E1, 801**.

This function returns an integer value with the handle of the open port.

Example:

```
var port = Serial.open("COM1", "2", "8N1");
//port = 33 //maybe
```

23.2 Function read(Int portHandle)

This function reads from the specified serial port.
The parameter *port* contains the *port handle* that was obtained by serial.open function.
This function returns a hexadecimal string like representation of the byte read.

Example:

```
Serial.read(port);
```

23.3 Function write(Int handlePort, String data)

This function writes the data to the specified serial port.
The parameter *handlePort* contains the handle of serial port, and the *data* parameter is an ASCII string to be written to the serial port.
This function returns an integer value with the number of bytes written.

Example:
Serial.write(port, "Hello");

23.4 Function writeHexa(Int handlePort, String data)

This function writes the data to the specified serial port.
The parameter *handlePort* contains the handle of serial port, and the *data* parameter is a hexadecimal representation string to be written to the serial port.

This function returns the number of bytes written.

Example:
Serial.writeHexa(port, "31323334");

23.5 Function close(Int handlePort)

This function closes the specified serial port. The parameter *handlePort* contains the handle of serial port. This function returns 0 if is successful.

Example:
var isClose = Serial.close(port);
//isClose = 0 //maybe

23.6 Function transactHexa(String port, String portConfig, Int protocol, String send, String readLen, integer timeOut)

This function opens the *port*, writes the bytes represented by *send* parameter and waits up to *timeOut* milliseconds for a reply of *readLen*/2 bytes long.

Parameters:

- **port:** is a 4 character long ASCII string indicating the port.
- **portConfig:** is a 3 character long ASCII string indicatind the following: data, parity and stop bits.
- **protocol:** : Define the protocol to use as. 1 if it is going to use "STX" and "ETX" or 0 if using "SI" and "SO"..
- **send:** is the hexadecimal representation of the byte to be sent.
- **readLen:** is the expected size of the hexadecimal representation of the reply.
- **timeOut:** is the number of milliseconds to wait for the reply.

This function returns an ASCII string with the hexadecimal representation of the bytes read.

Example:
Serial.transactHexa("COM1", "8N1", 1, "015244", 14, 2000)

23.7 Function waitRead(Int handler, Int waittime)

This function waits for incoming data considering the handler of the serial connection during a defined wait time expressed in milliseconds.

Example:

Serial.waitRead(ag,2000)

23.8 Function transacthexaexserial ((String port, String config, Int protocol, String message, Char incomingbytes, Int timeout, Int delay)

This function sends information through the serial port and controls the output displayed for a certain model of terminal.

Parameters:

- **port:** Defines port number.
- **config**: First byte defines speed and the port configuration as defined in 16.1.
- **protocol**: Define the protocol to use as. 1 if it is going to use "STX" and "ETX" or 0 if using "SI" and "SO".
- **message**: String to be sent.
- **incomingbytes**: Number of bytes to receive.
- **timeou**t: time out to wait for the response expressed in milliseconds.
- **delay**: the waiting time between characters sent.

Example:

Serial.transactHexaexserial("COM1", "8N1", 1, "015244", 14, 2000,10)

24 Utils Library

24.1 Function pause(Int time)

This function adds a pause determinated by the parameter *time*. This parameter must be specified in milliseconds.

Example:
Utils.pause(3000); // 3 seconds

24.2 Function getKey(Int time)

This function awaits for an event like a key pressed, a card swiped or an inserted card. Waits for *param1* seconds expressed in milliseconds. The function returns 0 if the time expired without any event.

Parameter:
Time – integer, time to wait the user in miliseconds (1000 = 1 sec)

Returns:
the following integer code or zero if nothing was done after timeout.

Key pressed	Returned Value
97	"a"
98	"b"
99	"c"
100	"d"
122	"f1"
123	"f2"
124	"f3"
125	"f4"
48	"0"
49	"1"
50	"2"
51	"3"
52	"4"
53	"5"
54	"6"
55	"7"
56	"8"
57	"9"
13	"0d"//enter
15	"AL" //alpha
27	"1B"
35	"23"
42	"*"
8	"BS"
900	"SC"; //smart
901	"MC"; //mag card

24.3 Function getKeyAndCtls (int time)

Acts like function getKey described above plus it wait for a card to be tap as well.

Parameter:
Time – integer, time to wait the user in miliseconds (1000 = 1 sec)

Returns:
Same values as getKey plus 902 if a contactless card is read.

24.4 Function clrscr()

This function clears the screen of the terminal.

On color terminals, this functions draws the graphic and colors the screen accordengly the parameters sent to the system by the calling of the Dialogs.setBG function.

On color terminals, this functions removes all the icons and bottons touch screen variables.

Parameters:
None – see also Dialogs.setBG function for color terminals.

Returns:
No significant interger.

Example:
Utils.clrscr();

24.5 Function Unzip(String fileName)

This function unpacks a zip a file, The file is determinated by *fileName* parameter.

Example:
Utils.unzip("update.zip");

24.6 Function reset()

This function resets the terminal POS.

Example:
```
if(WMLBrowser.isConected() == false)
        Utils.reset();
```

24.7 Function clearKeyboard()

This function clears the keyboard buffer.

Example:
Utils.clearKeyboard();

24.8 Function openConnectionHTTP(String address, Int port)

This function opens a socket connection. The parameter *address* defines the address while the parameter *port* defines the port.

Example:
Utils.openConnectionHTTP("187.141.6.51",8080);

Returns:
0 – if connected, otherwise function fail to connect.

24.9 Function closeConnectionHTTP()

This function closes the existing open connection.

Example:

```
Utils.closeConnectionHTTP();
```

24.10 Function getCheckSum(String fileName)

This function calculates the crc16 of the name that is included in *fileName* and returns an hexadecimal value.

Example:
Utils.getCheckSum("test.txt"); //0x1021

24.11 Function openFile(String filename, Int mode)

This function opens a file and uses *filename* as the name of the file to open while *mode* defines the action to perform.

The mode valid values are:

1 only for reading and
2 for reading and writing.

This function does not create the file if this does not exist.

Example:
Utils.openFile("invoices",2);

24.12 Function readFile(Int storeId, Int initial_pos)

This function reads 512 bytes or less. The parameter *storeId* is the handler while *initial_pos* is the position where it will start to read.

Example:
Utils.readFile(ag,3);

24.13 Function writeFile(Int storeId, Int begin, String include, Int type)

This function writes an string to a specified file. The parameter *storeId* defines the handler, *begin* defines where to start writing the string, *include* defines the string to write, *type* defines if 0 replaces the FF by 00 and if defined as 1 writes the string as it is.

Example:
Utils.writeFile((ag,5, "street 22;Eddievedder;2222222222",1);

24.14 Function closeFile(Int storeId)

This function closes the file. The handler is referenced by storeId.

Example:
Utils.closeFile(ag);

24.15 Function sizeFile(Int storeId)

This function gets the size of the file. The handler is defined by *storeId*. Cannot exceed 65K.

Example:
Utils.sizeFile(ag);

24.16 Function createFile(String fileName)

This function creates a file. *FileName* is used as the name of the file. It is created for read and write.

Example:
Utils.createFile("NewTransacctions");

24.17 Function displayImage(String imageFile, Int posRaw, Int posColumn)

This function displays an image. *ImageFile* defines the name of the file that contents the image, *posRaw* defines the raw while *posColumn* defines the column. Start position is 1. This function accepts wbmp or bmp files.

Example:
Utils.displayImage("logo.wbmp",3,5);

24.18 Function enabledHotKey()

This function enables the "*" key and calls VMAC. Only used for VeriFone.

Example:
Utils.enableHotKey();

24.19 Function deleteFile(String fileName)

This function deletes a file. The name of the file is defined by *fileName*. The file must be already closed to be deleted.

Example:
Utils.deleteFile("test.txt");

24.20 loadMasterDoubleKey(String Key, Integer index)

This function injects the specified Key parameter into the index slot inside the cryptographic key storage of the terminal.

Parameters:
Key: is a 32 byte length string containing the key to be injected.
Index: Location to use, valid values are 0,2,4 and 6.

Example:
UTILS.loadMasterDoubleKey("12345678901234567890123456789012",0);

24.21 Function DES(Integer Type, String key, String data)

This function encrypts or decrypts data.
This function returns the encrypted or decrypted string in Hex formatted string.

Parameters:

- ***Type***: Is an integer with one of the following values
  - ✓ 2: DESXOR encryption with single-length key
  - ✓ 3: DESXOR decryption with single-length key
  - ✓ 4: DESXOR encryption with double-length key
  - ✓ 5: DESXOR decryption with double-length key
  - ✓ 6: DESXOR encryption with triple-length key
  - ✓ 7: DESXOR decryption with triple-length key
  - ✓ 8: DES encryption with single-length key
  - ✓ 9: DES decryption with single-length key
  - ✓ 12: TDES encryption with double-length key
  - ✓ 13: TDES decryption with double-length key
  - ✓ 14: TDES encryption with triple-length key
  - ✓ 15: TDES decryption with triple-length key

- ***key***: String that holds the 8, 16 or 24 byte key
- ***data***: String to encrypt or decryp.

Example

```
UTILS.DES(8,"12345678","TBROWSER");     // returns "493e8e6e3ba21869"
```

NOTE: if you need the returned value in ASCII, use ISO.hexToChar()

```
UTILS.DES(9,"12345678",ISO.hexToChar("493e8e6e3ba21869"));
```
returns "8466827987836982", where
ISO.hexToChar("8466827987836982") is "TBROWSER"

24.22 Secure3DES(String data, Integer Index, String Workingkey)

Encrypts data using a secure key already loaded into the cryptographic chip of the terminal.

Parameters:

Data – ASCII data to be encrypted
Index – location of the master key to use (0,2,4 or 6)
Working Key – optional 32 byte encrypted key (under the master key) to be used as encryption key. If not use pass ""

Example:
UTILS.secure3DES("12345678",0,"");

Returns:
Encrypted data "12345678" with masterkey loaded in area 0, no working key is used.

UTILS.secure3DES("12345678",0,"12345678901234567890123456789012");

Returns encrypted data "12345678" with masterkey loaded in area 0, using the working key specified as follows:

Working key gets uncrypted using the master key in area 0 and the result is the key with which data 12345678 will be encrypted.

24.23 Function DESBinaryInput(String data)

Despite the fact the DES function receives data to be encrypted/unencrypted in ASCII, it internally converts the data to binary.

In WML passing nulls is not possible in natural form, so to assure all data including nulls gets to the DES function, we first need to escape the nulls by inserting a "/" (0x2F) characted in front of them. This way the Browser knows the combination 0x2F00 is a null.

Note: if original data contains 0x2F, it will be escaped to 0x2F2F.

Parameters:
data – ASCII data to input to the UTILS.DES function

UTILS.DESBinaryInput("653000122F995312"); //this secuence has a apair of ASCII characters that when converted to binary will be null (00) and "/" (2F)

Returns "65302F00122F2F995312", where the 00 and 2F are now escaped and ready to be input to the DES or Secure3DES functions.

24.24 Function openTokenFile(string fileName)

Opens a file that is expected to be a text list of tokens (or variables) with the format decribed below. The file is kept open for subsequent reads by UTILS.readTokenFile function. After reading is done, the file must be close with UTILS.closeTokenFile function.

Expected file format:

tokenVariable1 = [“]value[“] [cr] [LF]
tokenVariable2 = [“]value2[“] [cr] [LF]
…
tokenVariableN = [“]valueN[“] [cr] [LF]

Note: only ONE token file can be open at a time.

Parameter:
fileName – any name for the file

Returns:
0 if file could not be open, 1- the file is open. No handler is return as the file should be read and close with funtions readTokenFile and closeTokenFile.

Example:

```
If (UTILS.openTokenFile(“F:PARAMS.DAT”)) {
        UTILS.readTokenFile(“variable”,1,”storageVariable”);
        UTILS.closeTokenFile();
}
```

Example of file PARAMS.DAT:

VAT= 16.5
STORENAME =”MY STORE”
Street = My Street 123

24.25 Function readTokenFile(string Token, int tokenNumber, string storageVariable)

Reads a token value from previously open token file and stores the value in secure variable storageVariable or in a file name storageVariable if the value is more than 100 bytes long.

Parameters:
Token – a string representing the name of the token variable.
tokenNumber – a number indicating which token we need in case there is more than one token with the same name.

Returns.
1 – token was read ok. 0-token was not found
Token's value will be stored, if up to 100 bytes, in the variable storageVariable which can be read with WMLBrowser.getEnvSec("storageVariable").
If token value is bigger than 100 bytes, it will be stored in a file name as storageVariable.

Example.

Content of PARAM.DAT:

```
VAT= 16.5
STORENAME ="MY STORE"
Street = My Street 123
STORENAME =my other store
RSAKEY="727a817bc3728e82882f238828b737154419302836….(256 bytes)"
```

```
If (UTILS.openTokenFile("F:PARAMS.DAT")) {
        UTILS.readTokenFile("STORENAME",1,"Store1");
        UTILS.readTokenFile("STORENAME",2,"Store2");
        UTILS.readTokenFile("RSAKEY",1,"MYKEY");
        UTILS.closeTokenFile();
}

Printer.open();
Printer.print(WMLBrowser.getEnvSec("Store1")); //→ Store 1
Printer.print(WMLBrowser.getEnvSec("Store2")); // → Store 2
Printer.print(WMLBrowser.getEnvSec("MYKEY")); // → empty string… should be read with readFile
Printer.close();

var file=UTILS.openFile("MYKEY",2);
var key=UTILS.readFile(file,0);
UTILS.closeFile(file);
Printer.open();
Printer.print(key); //→ ="727a817bc3728e82882f238828b737154419302836….(256 bytes)"
Printer.close();
```

24.26 Function closeToken()

This function Frees memory use to store the token file.

Note: you must call this function before opening another token file. Failure to do so will result in allocated memory to be left allocated and most probably cause a system crash over time.

Parameter:
None.

Returns:
Not significant integer value.

Example:
UTILS.closeTokenFile();

25 Socket Library

If you need activate the SSL, please set the following variables like this:

mx_ssl_enable = 1 (where 1 means activated and 0 deactivated)
mx_ssl_cln_auth = 0

25.1 Function openSocket(String IP_address, Int port_number)

This function receives like parameter the IP address or URL and the port number. The function returns de handler.

Example:
var fh=Socket.openSocket("187.141.6.51",8080);

25.2 Function closeSocket(Int storeId)

This function receives the handler (storIed) to be closed.

Example:
Socket.closeSocket(fh);

25.3 Function sendSocket(String Data, Int Size)

This function receives the Data to be sent and the length of the data. This function returns the number of bytes sent.

Example:
Socket.sendSocket(Data, size);

25.4 Function receiveSocket(Int waitTime)

This function waits for socket information, the time specified in *waitTime* parameter, must be set in seconds.

Example:
Socket.receiveSocket(10);

25.5 Function receiveSocketInFile(Int Seconds, String file_name, Int bytes)

This function deletes the file specified in *file_name* and creates a new file with the same name, in this new file will receive the data. The parameter *seconds* sets the time out, and the parameter *bytes* sets the length of the file. The function returns the number of bytes and the file is closed.

Example:
var file1 = Socket.receiveSoketInFile(30,"test.txt",1000);

26 Webservice Library.

26.1 Function initEnv()

Before any clients can be created, some supporting structures must be created. This is done with the initEnv function.

Example:
WebService.initEnv();

26.2 Function setEndPoint(String URL)

Once the environment is properly initialized, the programmer can create a client to consume a service. Two configuration items are possible: the service endpoint (or "address"), and the SOAP action (or "indication of intent"").

Example:
WebService.setEndPoint(" http ://192.168.1.145:8088 / mockKCMXSoapBinding ");

26.3 Function setSoapAction(String URL)

Once the environment is properly initialized, the programmer can create a client to consume a service. Two configuration items are possible: the service endpoint (or "address"), and the SOAP action (or "indication of intent"").

Example:
WebService.setSOAPAction(" http :/ / microsoft . com / webservices / GetPrimeNumbers ");

26.4 Function initClient()

This function creates the client

Example:
WebService.initClient();

26.5 Function nameSpaceCreate(String URL, String prefix)

Namespaces are used to avoid name collisions in Web Services. Both a URI and a prefix are needed to declare a namespace.

Parameters:
uri namespace URI
prefix namespace prefix

Example:
var bean = WebService.namespaceCreate("http://beans", "bean");

26.6 Function elementCreate(parent, localname, ns)

This function creates an axiom element based on the following parameters:

- ***parent :*** parent of the element node to be created. Can be blank .
- ***localname:*** local name of the element. Cannot be NULL.
- ***ns:*** namespace of the element. can be NULL. If the value of the namespace has not already been declared then the namespace structure ns will be declared and will be freed when the tree is freed. If the value of the namespace has already been declared using another namespace structure then the namespace structure ns will be freed.

Example:
var CVV2 = WebService.elementCreate(consultaTarjeta, "CVV2", bean);

After executing this statement, CCV2 is a reference to an element locally named "CCV2", a child of the element referenced by consultaTarjeta and within the namespace "bean". In case of failure, elementCreate's return value is "".

26.7 Function elementSetText(text, element)

This function sets an element´s text.

Example:
WebService.elementSetText("02/16",fechaVenc);

26.8 Function clientSendReceive(request)

The clientSendReceive function is used to send the request (given by its first argument) over the network and returning the corresponding response. This function returns the server response.

Example:
var respuesta = WebService.clientSendReceive(consultaTarjeta);

If the transport phase meets with failure (e.g., server down, no connection, etc), the returned value is empty (not valid). This can be tested as follows:
if (WebService.isEmpty(respuesta)) .... // Handle empty response.

26.9 Function getFirstChild(node)

To process the response we need to navigate the children and siblings of the response nodes in order to read the values of the variables of interest.

Example:

```
var resConsulta_node = WebService.getFirstChild(respuesta);
var resultado_node = WebService.getFirstChild(resConsulta_node);
var value_node = WebService.getFirstChild(resultado_node);
var saldoAlDia_node = WebService.getNextSibling(value_node);
var pagoMin_node = WebService.getNextSibling(saldoAlDia_node);
var fechaLimPago_node = WebService.getNextSibling(pagoMin_node);
var payFree_node = WebService.getNextSibling(fechaLimPago_node);
var movimientos_node = WebService.getNextSibling(payFree_node);
var movimiento_node = WebService.getFirstChild(movimientos_node);
```

26.10 Function getNodeType(Int node)

This function receives the handle of a node and returns an integer value.
The return values are:

Return Value	Node Type
0	Invalid node type
1	AXIOM document type
2	AXIOM element type
3	AXIOM doctype type
4	AXIOM comment type
5	AXIOM attribute type
6	AXIOM namespace type
7	AXIOM processing instruction type
8	AXIOM text type
9	AXIOM data source, represent a serialized XML fragment with a stream

26.11 Function freeTree(node)

Once the response has been processed, the environment must be freed before another web service may be consumed. The following functions serve that purpose:

Example:
WebService.freeTree(node); / frees the given node

26.12 Function freeClient()

Once the response has been processed, the environment must be freed before another web service may be consumed. The following functions serve that purpose:

Example:
webservise.freeClient(); / frees the client

26.13 Function freeEnv()

Once the response has been processed, the environment must be freed before another web service may be consumed. The following functions serve that purpose:

Example:
webservice.freeEnv(); / frees the environment

26.14 Function getNextSibling(Int node)

This function receives the handle of a node and returns an integer value. If the handle does not point to a valid node, or if the node pointed by the handle does not have a further sibling, this function returns -1. Otherwise the function returns a handle to the next sibling of the node.

To process the response we need to navigate the children and siblings of the response nodes in order to read the values of the variables of interest.

Example:

```
var resConsulta_node = WebService.getFirstChild(respuesta);
var resultado_node = WebService.getFirstChild(resConsulta_node);
var value_node = WebService.getFirstChild(resultado_node);
var saldoAlDia_node = WebService.getNextSibling(value_node);
var pagoMin_node = WebService.getNextSibling(saldoAlDia_node);
var fechaLimPago_node = WebService.getNextSibling(pagoMin_node);
var payFree_node = WebService.getNextSibling(fechaLimPago_node);
var movimientos_node = WebService.getNextSibling(payFree_node);
var movimiento_node = WebService.getFirstChild(movimientos_node);
```

26.15 Function getNodeText(node)

This function returns the given node´s text value.

Example:
var ctext = WebService.getNodeText(cnode);

26.16 Function getLocalName(node)

This function returns the element local name.

A node's localname (that name unqualified by namespace uri or prefix) can be obtained as follows:

Example:
webservice.var cname = WebService.getLocalName(cnode);

26.17 Function childrenIteratorCreate()

This function creates the node children iterator

In some cases an iterator is of great help for this purpose:
childrenIteratorCreate(node) returns an iterator that allows navigating the given nodes children.

Example:

```
var i = 1;
var children_iter =
WebService.childrenIteratorCreate(WebService.getFirstChild(movimiento_node));
while (WebService.childrenIteratorHasNext(children_iter)) {
var cnode = WebService.childrenIteratorNext(children_iter);
var cname = WebService.getLocalName(cnode);
var ctext = WebService.getNodeText(cnode);
Dialogs.display(cname + ":" + ctext, i);
i = i + 1;
}
```

26.18 Function childrenIteratorHasNext(children_iter)

This function checks if there are more nodes to process.

In some cases an iterator is of great help for this purpose:
childrenIteratorHasNext(children_iter) indicates whether there are more children to visit in the iterator.

Example:

```
var i = 1;
var children_iter =
WebService.childrenIteratorCreate(WebService.getFirstChild(movimiento_node));
while (WebService.childrenIteratorHasNext(children_iter)) {
var cnode = WebService.childrenIteratorNext(children_iter);
var cname = WebService.getLocalName(cnode);
var ctext = WebService.getNodeText(cnode);
Dialogs.display(cname + ":" + ctext, i);
i = i + 1;
}
```

26.19 Function childrenIteratorNext(children_iter)

This function returns the next node in iterator

Example:

```
var i = 1;
var children_iter =
WebService.childrenIteratorCreate(WebService.getFirstChild(movimiento_node));
while (WebService.childrenIteratorHasNext(children_iter)) {
var cnode = WebService.childrenIteratorNext(children_iter);
var cname = WebService.getLocalName(cnode);
var ctext = WebService.getNodeText(cnode);
Dialogs.display(cname + ":" + ctext, i);
i = i + 1;
}
```