

hand**Point** *Retail*



handPoint Retail 5
Script Guide

License Agreement

Disclaimer

This document may contain technical inaccuracies or typographical errors. Periodically, changes are made to the information herein; these changes will be incorporated in revisions of this documents, or addendums made available on our website. handPoint does not accept liability for the use or misuse, direct or indirect, of this product. Users must accept to be responsible for results obtained by software by handpoint. All conclusions and decisions made based on the use of this product are at the responsibility of the user.

Important!

Read this carefully before initiating installation procedure

Opening and installing handPoint Retail, in full or in part, indicates that you have signed, agreed, and accepted the handPoint License Agreement that is part of the purchasing contract.

If you do not agree with the terms spelled out in the contract, please refrain from installing the product and return the product, in its entirety, to handPoint or the appropriate retailer.

The Software

The fact that you have access to this user manual means that you have duly purchased a copy of handPoint Retail or you have downloaded an evaluation version of either, and thereby accepting software license agreement.

The Documentation

No part of this publication may be reproduced, transmitted, transcribed, stored in a public retrieval system, or translated into any language or computer language, in any form, or by any means, electronic, mechanical, optical chemical, manual, or otherwise, without the prior consent of handPoint.

Trademarks

handPoint Retail is a registered trademark of handPoint, Ltd. Symbol is a trademark of Symbol, Inc. Pocket PC 2000, Pocket PC 2002, Microsoft Windows NT, Windows 2000 and Windows XP are trademarks of Microsoft, Inc

Table of Contents

License Agreement	1
THE SCRIPTING LANGUAGE	4
1.1 LOOKUP	5
1.1.1 MASTER	5
1.1.2 PROD	5
1.1.3 PRE	6
1.1.4 INPUT	6
1.1.5 BARCODE	6
1.1.6 EAN128	7
1.1.7 EAN128PF	7
1.1.8 ACTION	8
1.1.9 PERFORMLOOKUP	8
1.1.10 LOOKUP	8
1.1.11 TOTAL	9
1.1.12 COUNT	9
1.1.13 DEVICEID	9
1.1.14 ACCUMULATEFIELD	10
1.1.15 DEVICE	10
1.1.16 USER	10
1.1.17 PAY	11
1.1.18 SUMMARISE	11
1.2 VARIABLES	12
1.2.1 TEXT	12
1.2.2 ASSIGN	12
1.2.3 VAR	12
1.2.4 PASSIGN	13
1.2.5 PVAR	13
1.2.6 MASSIGN	13
1.2.7 MVAR	14
1.2.8 METASTATEASSIGN	14
1.2.9 SASSIGN	15
1.2.10 SVAR	15
1.2.11 CHECKLISTASSIGN	15
1.2.12 CHECKLIST	16
1.2.13 METASTATE	14
1.3 CALCULATIONS	16
1.3.1 MUL	16
1.3.2 DIV	16
1.3.3 ADD	17
1.3.4 SUB	17
1.3.5 INC	17
1.3.6 DEC	17
1.3.7 ROUND	18
1.3.8 FLOOR	18
1.3.9 CEIL	18
1.3.10 EQ	19
1.3.11 LT	19
1.3.12 AND	19
1.3.13 OR	20
1.3.14 NOT	20
1.4 LOGIC	20
1.4.1 IF	20
1.4.2 WHILE	21
1.4.3 EMPTY	21
1.5 DATE AND TIME	21
1.5.1 DATEFORMAT	21

1.5.2	TIMEFORMAT.....	22
1.5.3	DATE.....	23
1.5.4	TIME.....	23
1.5.5	TIMESTAMP.....	24
1.5.6	FILESTAMP.....	24
1.5.7	DATEOFFSET.....	24
1.6	STRINGS.....	25
1.6.1	STRCMP.....	25
1.6.2	STRSTR.....	25
1.6.3	STRCAT.....	25
1.6.4	STRLEN.....	26
1.6.5	LEFT.....	26
1.6.6	MID.....	26
1.6.7	SUBSTRING.....	27
1.6.8	RIGHT.....	27
1.6.9	ENDSWITH.....	27
1.6.10	STARTSSWITH.....	27
1.6.11	LPAD.....	28
1.6.12	RPAD.....	28
1.6.13	STREQ.....	29
1.6.14	STRLT.....	29
1.6.15	SECTION.....	29
1.6.16	REPLACE.....	30
1.7	MISCELLANEOUS.....	30
1.7.1	NUMBER.....	<i>Error! Bookmark not defined.</i>
1.7.2	PAYMENT.....	30
1.7.3	CURRENCYROUNDTO.....	<i>Error! Bookmark not defined.</i>
1.7.4	CURRENCYPRECISION.....	<i>Error! Bookmark not defined.</i>
1.7.5	TRANSFORMATION.....	30
1.7.6	READBLOCK.....	31
1.7.7	READBLOCKS.....	31
1.7.8	WRITEBLOCK.....	32
1.7.9	BLOCKCOUNT.....	32
1.7.10	DELETEBLOCK.....	32
1.7.11	ALERT.....	33
1.7.12	SLEEP.....	33
1.7.13	XPDAPRINTERTEST.....	33
1.7.14	XPDAPRINTBLOCKS.....	33

The Scripting Language

Objectives:

The objective of handPoint Scripting Language (HSL) is to generate dynamic strings without requiring advanced programming skills.

An important feature of HSL is its capability to manipulate strings making use of scanned, entered or static information from input files.

Syntax:

The scripts consist of a string inside angel brackets <>. The string inside angel brackets contains the name of the script and is possibly followed by a parameters or one or more argument scripts.

Upper and lower case alphabetic characters are to be treated identically in script names. Thus, any of the following may represent the <TIMESTAMP> script:

<TIMESTAMP> <TimeStamp> <timestamp> <TIMESTAmP> <TIMESTAMP>

However, within parameters and return values of scripts taken as arguments case is generally important. In particular, in jobs the field name "price" is different from the field name "Price".

An argument script to another script is a script which is evaluated before the enclosing script is fully evaluated so the outcome of the argument script becomes the input to the outer script. This behaviour allows nesting of scripts and is particularly useful in evaluation of calculation scripts and the more complex scripts.

The general syntax of a handPoint Retail script is of the form:

<script-name>, <script-name=parameter> or <script-name<argument-script><argument-script>>

Note that this is the general syntax. A particular script might require a parameter or one or more an argument scripts or require the parameter string to be a predefined string. If the option is not taken, the appropriate default is often implied.

The general syntax of handPoint Retail script using BNF notation is given here and these definitions are use below when the syntax of individual scripts are defined.

```
<script> ::= "<" <script-name> [<parameter> | {<argument-script>}] ">"
<script-name> ::= <string>
<parameter> ::= "="<string>
<argument-script> ::= <script>
<string> ::= <c> | <c> <string>
<c> ::= any one of the 128 ASCII characters, but not any <special>
<special> ::= "<" | ">" | "=" | the control characters (ASCII codes 0 through 31 inclusive and 127)
<float-string> ::= ["-"] <int-part> [ "." <decimal-part> ]
<int-part> ::= <digit> | <int-part> <digit>
<decimal-part> ::= <int-part>
<digit> ::= any one of the ten digits 0 through 9
```

Script types:

The scripts can be divided into 7 types:

- 1.1 Lookup scripts
- 1.2 Variable scripts

- 1.3 Calculation scripts
- 1.4 Logic scripts
- 1.5 Date and Time scripts
- 1.6 String scripts
- 1.7 Miscellaneous

Each type and its syntax will be described in detail and examples of scripts will be given below.

1.1 Lookup

Lookup scripts are available in Auto buttons and in fields of the job modules.

Lookup fields are used to look up specific values. PROD scripts use values from the current job. PRE script uses values from a job comments file that is attached to the current job. Note that if you want to use the value of another field in the script for the auto buttons, it is important that that value is generated prior to the field containing the auto buttons screen.

1.1.1 MASTER

Syntax:

```
<master-script> ::= "<MASTER=" <parameter> ">"
<parameter> ::= "ID" | "BARCODE" | <master-field-name>
<master-field-name> ::= <string> The name of a defined field in the current master
```

Examples:

```
<MASTER=ID>
<MASTER=Price>
```

Description:

The master binds to a record when it finds a product and that record remains current until the product is saved or the handling of the product is cancelled then it is set to an empty record. The MASTER script is typically used to fetch a static information from the master by specifying the name of the master field. The MASTER field name is case sensitive but the special "ID" parameter isn't.

Return value:

This script returns the value of a field in the current product record of the indexed master. If the job field name doesn't exist the empty string is returned.

Note:

The "BARCODE" parameter is deprecated. It returns the first field in the barcode record of the current index record of the indexed master. The current index record is empty if the barcode was not used to find the product in that case it returns an empty string.

1.1.2 PROD

Syntax:

```
<prod-script> ::= "<PROD=" <parameter> ">"
<parameter> ::= "DESC" | "EMPTY" | "PROID" | <job-field-name>
<job-field-name> ::= <string> The name of a defined field in the current job
```

Examples:

```
<PROD=ID>
<PROD=Quantity>
```

Description:

The job binds to a record when a product is opened and it remains until the product is saved or the handling of the product is cancelled then it is set to an empty record. The PROD script is typically used to fetch information entered by the user in calculations and data processing by specifying the name of the job field. The job field name is case sensitive but the special parameters aren't.

Return value:

This script returns the value of a field in the current product record of the job. If the job field name doesn't exist the empty string is returned.

Note:

The "DESC", "EMPTY", "PRODID" parameters are deprecated.

"DESC" is case insensitive and returns the same value as <MASTER=Description> use it instead.

"EMPTY" is case insensitive and returns an empty string "" use <TEXT=> instead.

"PRODID" is case insensitive and returns the same value as <PROD=ID> use it instead.

1.1.3 PRE

Syntax:

<pre-script> ::= "<PRE=" <parameter> ">"

<parameter> ::= "USER" | <job-comment-field-name>

<job-comment-field-name> ::= <string> The name of a defined field in the current job comment

Examples:

<PRE=ID>

<PRE=StoreNumber>

Description:

The job comment binds to a record when a job comment is selected and remains until the job is closed or a new job comment is selected. The PRE script is typically used to fetch static information from a job comment to be used in calculations or data processing by specifying the name of the job comment field. The job comment field name is case sensitive but the special parameter "USER" isn't.

Return value:

This script returns the value of a field in the current record of the current job comment. If the job comment field name doesn't exist the empty string is returned.

Note:

The "USER" parameter is deprecated. "USER" is case insensitive and returns the current user name.

1.1.4 INPUT

Syntax:

<input-script> ::= "<INPUT>"

Example:

<INPUT>

Description:

This script is typically used to find products by actions over network. It can also be used to get the raw barcode as scanned but consider using <BARCODE> if you want to get the barcode as it is in the master.

Return value:

This script returns the value of a string either entered or scanned.

1.1.5 BARCODE

Syntax:

```
<barcode-script> ::= "<BARCODE" ["=" <parameter>] ">"  
<parameter> ::= <digit> | <parameter> <digit>  
<digit> ::= any one of the ten digits 0 through 9
```

Example:

```
<BARCODE>  
<BARCODE=2>
```

Description:

The master binds to a record when it is used to find a product by barcode and that record remains current until the product is saved or the handling of the product is cancelled then it is set to an empty record. The BARCODE script is typically used to fetch static information that applies to a particular barcode from the master by specifying the index of the master index field. The fields are enumerated from zero. Field number 0 is the barcode field. Field number 1 is the productID and then there can be optional fields following.

If the field number is not given then this script is used to get the raw barcode as scanned. If the product is not scanned but selected in some other way the <BARCODE> script returns "".

Return value:

<BARCODE> script returns the unprocessed barcode or "" if the product was not scanned.
<BARCODE="index"> script returns the value of a field in the current index record of the indexed master.

Note:

The reason the field number is used instead of the index field name as in the other scripts is that the index fields do not have a name in the current version of handPoint Retail.

1.1.6 EAN128

Syntax:

```
<barcode-script> ::= "<EAN128" <AI> <formatted-string>">"  
    <AI > ::= <script>  
<formatted-string> ::= <script>
```

Example:

```
<EAN128<TEXT=02><BARODE>>
```

Description:

The UCC/EAN 128 barcode is made up of key-value pairs that is displayed in the form (key)value(key)value... The EAN128 script takes a string that is formatted in that way and returns the value that is assigned to the supplied key.

Return value:

<EAN128<TEXT=02><TEXT=(02)123465789(37)132>> script returns the value that is assigned to the '02' key and that is '123465789'. If <formatted-string> is not a EAN128 barcode the whole string is returned.

1.1.7 EAN128PF

Syntax:

```
<barcode-script> ::= "<EAN128PF" <AI > <formatted-string>">"  
    <AI > ::= <script>  
<formatted-string> ::= <script>
```

Example:

```
<EAN128PF<TEXT=310><BARODE>>
```

Description:

Like the EAN128 script, the EAN128PF works on EAN128 formatted string. The difference on these scripts is that the EAN128PF script does not return the value attached to the key value but the postfix of the key. The key in EAN 128 barcode is often postfixed with informations about the value part. For example the '310' key is meant for weight informations and then the postfix is a digit indicating the ordinal of the decimal point in the data field.

Return value:

<EAN128PF<TEXT=310><TEXT=(01)123465789(3101)132>> script returns the postfix of the '310' key and that is '1'. Then we know that the weight displayed in this barcode is 13,2.

1.1.8 ACTION

Syntax:

```
<barcode-script> ::= "<ACTION"<name-parameter>[<return-result>]">"  
<return-result> ::= <script>  
<name-parameter> ::= <script>
```

Example:

```
<ACTION<TEXT=SMTPAction><TEXT=1>>  
<ACTION<TEXT=PrintAction>>
```

Description:

The ACTION script takes one predefined action module and executes its execution part. The action is loaded by name which is taken from <name-parameter> and executed and if it returns any result it is possible to make the ACTION script return that result.

Return value:

<ACTION<TEXT=SMTPAction><TEXT=1>>. This script returns the result that the SMTPAction action module will return.

<ACTION<TEXT=SMTPAction>>. This will return an empty string even though SMTPAction will return some result because <return-result> switch is missing.

1.1.9 PERFORMLOOKUP

Syntax:

```
<performlookup-script> ::= "<PERFORMLOOKUP" {<parameter>} ">"  
<parameter> ::= <script>
```

Example:

```
<PERFORMLOOKUP=DollarExchangeRates>  
<PERFORMLOOKUP<MASTER=DiscountID><PRE=DiscountClass>>  
<PERFORMLOOKUP<STRCAT<STRCAT<PROD=ID><TEXT=_>><PRE=DiscountClass>>
```

Description:

This script is used to perform a lookup in the current lookup master by searching for a key given as a parameter to the script. If the key is found in the lookup master it binds to that record and it remains current until a new lookup is performed or until the product is saved or the handling of the product is cancelled then it is set to an empty record.

Return value:

This script always returns an empty string.

Note:

Number of index keys must be set in lookup master module and that number must match number of parameters entered into the PERFORMLOOKUP script.

1.1.10 LOOKUP

Syntax:

```
<lookup-script> ::= "<LOOKUP=" <parameter> ">"  
<parameter> ::= <master-field-name>  
<master-field-name> ::= <string>
```

Example:

```
<LOOKUP=EuroExchangeRate>  
<LOOKUP=Discount>  
<LOOKUP=Discount>
```

Description:

The lookup master binds to a record when the PERFORMLOOKUP script is used to find a record by a key given as a parameter to the PERFORMLOOKUP script. The LOOKUP script is typically used to fetch static information from the lookup master by specifying the name of the lookup master field. The lookup master field name is case sensitive.

Return value:

This script returns the value of a field in the current lookup record of the lookup master. If the lookup master field name doesn't exist the empty string is returned.

1.1.11 TOTAL

Syntax:

```
<total-script> ::= "<TOTAL=" <parameter> ">"  
<parameter> ::= <job-field-name>  
<job-field-name> ::= <string> The name of a defined field in the current job
```

Example:

```
<TOTAL=Price>  
<TOTAL=Quantity>
```

Description:

This script calculates the sum or total value of a specific field in the job. The field must be included in data handling iteration.

Return value:

This script returns sum or total value of a specific field.

1.1.12 COUNT

Syntax:

```
<count-script> ::= "<COUNT>"
```

Example:

```
<COUNT>
```

Description:

This script calculates the number of products in the output file.

Return value:

This script returns the number of products in the output file.

1.1.13 DEVICEID

Syntax:

```
<device-script> ::= "<DEVICEID>"
```

Examples:

<DEVICEID>

Description:

Fetches the serial number for the current device and returns that as a string

Return value:

Returns the serial number for the current device

1.1.14 ACCUMULATEFIELD

Syntax:

<accumulatefield-script> ::= "<ACCUMULATEFIELD" <field-name> ">"
<field-name> ::= <script>

Examples:

Description:

Sums up a specified field in the output.

Return value:

1.1.15 DEVICE

Syntax:

<device-script> ::= "<DEVICE="<parameter>">"
<parameter> ::= "ID"

Examples:

<DEVICE=ID>

Description:

The <DEVICE=ID> script gets the name of the device.

Return value:

Returns the name given to the device.

1.1.16 USER

Syntax:

<user-script> ::= "<USER=" <parameter> ">"
<parameter> ::= "ID" | "Description" | "OFFSET"

Examples:

<USER=ID>

<USER=Description>

Description:

The USER script is used to access information about the current selected user. ID is the first field of the record, Description the second field and OFFSET is a value obtained from "User Offset" field in User Dialog.

Return value:

Returns information about the selected user.

1.1.17 PAY

Syntax:

<pay-script> ::= "<PAY=" <parameter> ">"

<parameter> ::= <script>

CurrentStoreAndForwardQueueSize
GetTxnCashBackAmount
SigStatus
StorePayPluginVersion
TxnAcqSequenceNumber
TxnAcqTerminalID
TxnAID
TxnAmount
TxnApplicationLabel
TxnApplicationPANSequence
TxnAuthMsg
TxnAuthorisationCode
TxnCardSchemeName
TxnCashBackMaximum
TxnCashBackMinimum
TxnCryptogramValue
TxnCVM
TxnCvv2
TxnDeclinedReason
TxnEFTSequenceNumber
TxnEntryType
TxnExpiryDate
TxnIsChipCard
TxnIssueNumber
TxnIssueNumberLength
TxnMerchantID
TxnPan
TxnPanX
TxnResult
TxnResultString
TxnStartDate
TxnTrack2Data
TxnTransactionNumber
TxnType
TxnVoidReason

Examples:

<PAY=TxnEntryType>

Description:

Retrieves the value of the specified transaction attribute/field.

Return value:

1.1.18 SUMMARISE

Syntax:

<summarise-script> ::= "<SUMMARISE" <statements> ">"

<statements> ::= <script> | <script> <statements>

Examples:

Description:

Return value:

1.2 Variables

Variables scripts are used to create variables and assign values to them. Using variables makes scripts often easier to handle and allows complex tasks to be performed very easily.

1.2.1 TEXT

Syntax:

```
<text-script> ::= "<TEXT=" <parameter> ">"  
<parameter> ::= <string>
```

Examples:

```
<TEXT=_>  
<TEXT=100>
```

Description:

The TEXT script is typically used in conjunction with other scripts where a argument-script is required. Using the TEXT script allows easy entry of static strings in such cases.

Return value:

The TEXT script simply returns the parameter given to it.

1.2.2 ASSIGN

Syntax:

```
<assign-script> ::= "<ASSIGN" <variable-name> <variable-value> ">"  
<variable-name> ::= <script>  
<variable-value> ::= <script>
```

Examples:

```
<ASSIGN<TEXT=StartTime><TIME>>  
<ASSIGN<TEXT=TotalPrice><MUL<PROD=Quantity><PROD=Price>>>
```

Description:

The ASSIGN script assigns a value to the variable name both given as arguments to the script. The assigned value can then later be accessed by the VAR script. Both the variable name and the variable value are case sensitive.

Assign script is relevant while product is being processed. E.g. when a product is being scanned and saved.

Return value:

The script always return the empty string.

1.2.3 VAR

Syntax:

```
<var-script> ::= "<VAR=" <variable-name> ">"  
<variable-name> ::= <string>
```

Examples:

```
<VAR=StartTime>  
<VAR=TotalPrice>
```

Description:

The VAR script is used to get the value of a variable previously set by the ASSIGN script.

Using variables makes scripts often easier to handle and allows complex tasks to be performed very easily. The variable name is case sensitive.
VAR script is relevant while product is being processed. E.g. when a product is being scanned and saved.

Return value:

The VAR script returns the value of a variable previously set by the ASSIGN script. If the variable has not been set the script returns an empty string.

1.2.4 PASSIGN

Syntax:

```
<assign-script> ::= "<PASSIGN" <variable-name> <variable-value> ">"  
<variable-name> ::= <script>  
<variable-value> ::= <script>
```

Examples:

```
<PASSIGN<TEXT=StartTime><TIME>>  
<PASSIGN<TEXT=TotalPrice><MUL<PROD=Quantity><PROD=Price>>>
```

Description:

The PASSIGN script assigns a value to the variable name both given as arguments to the script. The assigned value can then later be accessed by the PVAR script. Both the variable name and the variable value are case sensitive. The PVAR is persistent and is not cleared on program exit or when the device is synced.

Return value:

The script always return the empty string.

1.2.5 PVAR

Syntax:

```
<var-script> ::= "<PVAR=" <variable-name> ">"  
<variable-name> ::= <string>
```

Examples:

```
<PVAR=StartTime>  
<PVAR=TotalPrice>
```

Description:

The PVAR script is used to get the value of a variable previously set by the PASSIGN script. Using variables makes scripts often easier to handle and allows complex tasks to be performed very easily. The variable name is case sensitive. The PVAR is persistent and is not cleared on program exit or when the device is synced.

Return value:

The PVAR script returns the value of a variable previously set by the PASSIGN script. If the variable has not been set the script returns an empty string.

1.2.6 MASSIGN

Syntax:

```
<assign-script> ::= "<MASSIGN" <variable-name> <variable-value> ">"  
<variable-name> ::= <script>  
<variable-value> ::= <script>
```

Examples:

```
<MASSIGN<TEXT=StartTime><TIME>>
```

<MASSIGN<TEXT=TotalPrice><MUL<PROD=Quantity><PROD=Price>>>

Description:

The MASSIGN script assigns a value to the variable name both given as arguments to the script. The assigned value can then later be accessed by the MVAR script. Both the variable name and the variable value are case sensitive. The MVAR is persistent and is not cleared on program exit but it is cleared when the device is synced. The MVAR is specific to a particular output file.

Return value:

The script always return the empty string.

1.2.7 MVAR

Syntax:

<var-script> ::= "<MVAR=" <variable-name> ">"
<variable-name> ::= <string>

Examples:

<MVAR=StartTime>
<MVAR=TotalPrice>

Description:

The MVAR script is used to get the value of a variable previously set by the MASSIGN script. Using variables makes scripts often easier to handle and allows complex tasks to be performed very easily. The variable name is case sensitive. The MVAR is persistent and is not cleared on program exit but it is cleared when the device is synced. The MVAR is specific to a particular output file.

Return value:

The MVAR script returns the value of a variable previously set by the MASSIGN script. If the variable has not been set the script returns an empty string.

1.2.8 METASTATEASSIGN

Syntax:

<metastateassign-script> ::= "<METASTATEASSIGN" <state> <action> ">"
< state > ::= "Success"|" Error"|" Edited"|"New"|<script>
<action> ::= <script> (action name as defined in the Job under Output Actions)

Examples:

<METASTATEASSIGN<TEXT=Success> <Text=SaveAction>>

Description:

The METASTATEASSIGN script assigns a state to the metastate of the specified action.

Return value:

Nothing.

1.2.9 METASTATE

Syntax:

<metastate-script> ::= "<METASTATE" <action> ">"

Examples:

<METASTATE=SaveAction>

Description:

Returns the metastate of the specified action

Return value:

Returns the metastate of the specified action

1.2.10 SASSIGN

Syntax:

```
<assign-script> ::= "<SASSIGN" <variable-name> <variable-value> ">"  
<variable-name> ::= <script>  
<variable-value> ::= <script>
```

Examples:

```
<SASSIGN<TEXT=RegisterTotal><TEXT=100.00>>
```

Description:

The SASSIGN script assigns a secured (encrypted) value to the variable name both given as arguments to the script. The assigned value can then later be accessed by the SVAR script. Both the variable name and the variable value are case sensitive. The SVAR is persistent and is not cleared on program exit or when the device is synced.

Return value:

The script always returns the empty string.

1.2.11 SVAR

Syntax:

```
<var-script> ::= "<SVAR=" <variable-name> ">"  
<variable-name> := <string>
```

Examples:

Description:

The SVAR script is used to get the secured (encrypted) value of a variable previously set by the SASSIGN script.

Using variables makes scripts often easier to handle and allows complex tasks to be performed very easily. The variable name is case sensitive. The SVAR is persistent and is not cleared on program exit or when the device is synced.

Return value:

The SVAR script returns the value of a variable previously set by the SASSIGN script. If the variable has not been set the script returns an empty string.

1.2.12 CHECKLISTASSIGN

Syntax:

```
<assign-script> ::= "<CHECKLISTASSIGN" <checklist-field-name> <checklist-value>  
">"  
<checklist-field-name> ::= <script>  
<checklist-value> := <script>
```

Examples:

```
<CHECKLISTASSIGN<TEXT=Color><TEXT=Color=RED>>  
<CHECKLISTASSIGN<TEXT=QuantityLeft><TEXT=0>>
```

Description:

Sets a field in a checklist to the specified value

Return value:

Returns the empty string

1.2.13 CHECKLIST

Syntax:

```
<checklist-script> ::= "<CHECKLIST=" <checklist-field-name> ">"  
<checklist-field-name> ::= <string> The name of a defined field in the current  
checklist
```

Examples:

```
<CHECKLIST=ID>  
<CHECKLIST=Quantity>
```

Description:

Retrieves a value from a checklist field.

Return value:

The value stored in the specified checklist field.

1.3 Calculations

Calculation scripts are used to calculate values from information from different fields. The basic arithmetic functions adding, multiplying, subtracting and dividing can be used.

1.3.1 MUL

Syntax:

```
<mul-script> ::= "<MUL" <value> <value> ">"  
<value> ::= <script> that returns a <float-string>
```

Examples:

```
<MUL<TEXT=2><TEXT=3>>  
<MUL<TEXT=1.125><PROD=Price>>
```

Description:

The MUL script used to multiply numbers. Both argument scripts must return a string that can be interpreted as a number.

Return value:

Returns the multiplication of the argument scripts. The MUL script returns a string that can be interpreted as a number.

1.3.2 DIV

Syntax:

```
<div-script> ::= "<DIV" <value> <value> ">"  
<value> ::= <script> that returns a <float-string>
```

Examples:

```
<DIV<TEXT=3><TEXT=2>>  
<DIV<PROD=Price><TEXT=5>>
```

Description:

The DIV script is used to divide numbers. Both argument scripts must return a string that can be interpreted as a number. Remember that it is not possible to divide by zero;

Return value:

Returns the division of the argument scripts. The DIV script returns a string that can be interpreted as a number or NaN if division by zero is attempted.

1.3.3 ADD

Syntax:

```
<add-script> ::= "<ADD" <value> <value> ">"  
<value> ::= <script> that returns a <float-string>
```

Example:

```
<ADD<TEXT=3><TEXT=2>>  
<ADD<PROD=Price><TEXT=5>>
```

Description:

The ADD script is used to add numbers. Both argument scripts must return a string that can be interpreted as a number.

Return value:

Returns the addition of the argument scripts. The ADD script returns a string that can be interpreted as a number.

1.3.4 SUB

Syntax:

```
<sub-script> ::= "<SUB" <value> <value> ">"  
<value> ::= <script> that returns a <float-string>
```

Example:

```
<SUB<TEXT=3><TEXT=2>>  
<SUB<PROD=Price><TEXT=5>>
```

Description:

The SUB script is used to subtract numbers. Both argument scripts must return a string that can be interpreted as a number.

Return value:

Returns the subtraction of the argument scripts. The SUB script returns a string that can be interpreted as a number.

1.3.5 INC

Syntax:

```
<inc-script> ::= "<INC" <parameter> ">"  
<parameter> ::= <script>
```

Examples:

```
<INC<TEXT=1>> (will return 2)
```

Description:

Returns a given parameter after adding one to it.

Return value:

The incremented value of a parameter.

1.3.6 DEC

Syntax:

```
<dec-script> ::= "<DEC" <parameter> ">"
```

<parameter> ::= <script>

Examples:

<DEC<TEXT=1>> (will return 0)

Description:

Returns a given parameter after subtracting one from it

Return value:

The decremented value of a parameter.

1.3.7 ROUND

Syntax:

<round-script> ::= "<ROUND" <value> [<num-decimals>]">

<value> ::= <script> that returns a <float-string>

<num-decimals> ::= <script> that returns a <int-part>

Example:

<ROUND<DIV<PROD=Price><TEXT=5>>>

<ROUND<MUL<PROD=Price><TEXT=0.85>>><TEXT=2>>

Description:

The ROUND script rounds the value to the given number of decimals. If number of decimals is not supplied zero is assumed.

Return value:

Returns the value rounded to the number of decimals.

1.3.8 FLOOR

Syntax:

<floor-script> ::= "<FLOOR" <value> [<num-decimals>]">

<value> ::= <script> that returns a <float-string>

<num-decimals> ::= <script> that returns a <int-part>

Example:

<FLOOR<DIV<PROD=Price><TEXT=5>>>

<FLOOR<MUL<PROD=Price><TEXT=0.85>>><TEXT=2>>

Description:

The FLOOR script returns the largest integral value not greater than the value given. If number of decimals is not supplied zero is assumed.

Return value:

Returns the largest integral value not greater than x with the given number of decimals.

1.3.9 CEIL

Syntax:

<ceil-script> ::= "<CEIL" <value> ">"

<value> ::= <script> that returns a <float-string>

Example:

<CEIL<DIV<PROD=Price><TEXT=5>>>

<CEIL<MUL<PROD=Price><TEXT=0.85>>><TEXT=2>>

Description:

The CEIL script returns the smallest integral value not less than the given value. If number of decimals is not supplied zero is assumed.

Return value:

Returns the largest integral value not greater than x with the given number of decimals.

1.3.10 EQ

Syntax:

```
<eq-script> ::= "<EQ" <value> <value> ">"  
<value> ::= <script> that returns a <float-string>
```

Examples:

```
<EQ<TEXT=2><TEXT=3>>  
<EQ<TEXT=1.125><PROD=Price>>
```

Description:

The EQ script is used to compare numbers numerically. Both argument scripts must return a string that can be interpreted as a number.

Return value:

Returns the "1" if the values are equal else "0" is returned.

1.3.11 LT

Syntax:

```
<lt-script> ::= "<LT" <value> <value> ">"  
<value> ::= <script> that returns a <float-string>
```

Examples:

```
<LT<TEXT=2><TEXT=3>>  
<LT<TEXT=1.125><PROD=Price>>
```

Description:

The LT script is used to compare numbers numerically. Both argument scripts must return a string that can be interpreted as a number.

Return value:

Returns the "1" if the value of the first argument is less than the value of the second argument else "0" is returned.

1.3.12 AND

Syntax:

```
<and-script> ::= "<AND" <value> <value> ">"  
<value> ::= <script> that returns a <int-string>
```

Examples:

```
<AND<TEXT=0><TEXT=1>>  
<AND<TEXT=1><PROD=InStock>>
```

Description:

The AND script performs the logical AND operation. Both argument scripts must return a string that can be interpreted as an integer. "" and "0" denote the logical false value. Numbers containing decimal point are evaluated based on their integer part only. All other strings denote the logical true value.

Return value:

Returns the "1" if both values are true else "0" is returned.

1.3.13 OR

Syntax:

```
<or-script> ::= "<OR" <value> <value> ">"  
<value> ::= <script> that returns a <int-string>
```

Examples:

```
<OR<TEXT=0><TEXT=1>>  
<OR<TEXT=1><PROD=lnStock>>
```

Description:

The OR script performs the logical OR operation. Both argument scripts must return a string that can be interpreted as an integer. "" and "0" denote the logical false value. Numbers containing decimal point are evaluated based on their integer part only. All other strings denote the logical true value.

Return value:

Returns the "1" if either value is true else "0" is returned.

1.3.14 NOT

Syntax:

```
<not-script> ::= "<NOT" <value> ">"  
<value> ::= <script> that returns a <int-string>
```

Examples:

```
<NOT<TEXT=0>>  
<NOT<TEXT=1>>
```

Description:

The NOT script performs the logical NOT operation. The argument script must return a string that can be interpreted as an integer. "" and "0" denote the logical false value. Numbers containing decimal point are evaluated based on their integer part only. All other strings denote the logical true value.

Return value:

Returns the "1" if value is false else "0" is returned.

1.4 Logic

Logic scripts can be used to process information based on the values of certain expressions. If the evaluated expression (the first expression) has a value of 0 or the empty string, the script will show the outcome of the third expression if it is available else the second value is evaluated.

1.4.1 IF

Syntax:

```
<if-script> ::= "<IF" <test-value> <consequent-value> [ <alternate-value> ] ">"  
<test-value> ::= <script>  
<consequent-value> ::= <script>  
<alternate-value> ::= <script>
```

Description:

This script is typically used when desired result depends on a test value. The if script is used to return a string based on the value of test-value expression.

Return value:

The IF script returns either the consequent-value or the alternative value based on the result of the test value script. If the test-value is any other string than "0" or the empty string "" the consequent value is returned otherwise the alternate value is returned.

1.4.2 WHILE

Syntax:

```
<while-script> ::= "<WHILE" <conditional-expression> <statement> ">"  
<conditional-expression> ::= <script>  
<statement> ::= <script>
```

Examples:

```
<EMPTY  
<ASSIGN<TEXT=counter><TEXT=10>>  
<WHILE<VAR=counter>  
<EMPTY  
<...>  
<ASSIGN<TEXT=counter><DEC<VAR=counter>>>  
>  
>  
>
```

Description:

Executes a statement until a particular condition is false (i.e. equal to 0).

Return value:

The combined string output from all iterations of <statement>.

1.4.3 EMPTY

Syntax:

```
<empty-script> ::= "<EMPTY" <statements> ">"  
<statements> ::= <script> | <script> <statements>
```

Examples:

```
<EMPTY  
<IF...>  
<IF...>  
>
```

Description:

Can be used to create logical sections that do not return a value.

Each argument script is executed in order.

Return value:

Nothing.

1.5 Date and Time

There are several scripts available to show date and time. These scripts can be used everywhere in Retail (in Auto buttons and script fields) and are not limited to specific types of jobs or module types. The scripts show the current date or the current time in a format you select.

1.5.1 DATEFORMAT

Syntax:

```
<dateformat-script> ::= "<DATEFORMAT=" <dateformat> ">"
```

<format-char> ::= d|dd|ddd|dddd|M|MM|MMM|MMMM|yy|yyyy
<dateformat> ::= <format-char> | <format-char><string>

Examples:

<DATEFORMAT=yyyy-mm-dd>
<DATEFORMAT=dd.mm.yyyy>

Description:

The DATEFORMAT script is used to set the formatting of the returned string of subsequent DATE scripts. The format set by the DATEFORMAT script remains until either the format is set again by another DATEFORMAT script or until all scripts in the string that the DATEFORMAT script is part of has been fully evaluated possibly longer in certain contexts. The dateformat is not case sensitive.

List of format characters:

d the day as number without a leading zero (1-31)
dd the day as number with a leading zero (01-31)
ddd the abbreviated localized day name (e.g. 'Mon'..'Sun').
dddd the long localized day name (e.g. 'Monday'..'Sunday').
M the month as number without a leading zero (1-12)
MM the month as number with a leading zero (01-12)
MMM the abbreviated localized month name (e.g. 'Jan'..'Dec').
MMMM the long localized month name (e.g. 'January'..'December').
yy the year as two digit number (00-99)
yyyy the year as four digit number (1752-8000)
All other input characters will be ignored.

Return value:

The script always returns the empty string.

1.5.2 TIMEFORMAT

Syntax:

<timeformat-script> ::= "<DATEFORMAT=" <timeformat> ">"
<format-char> ::= h|hh|m|mm|s|ss|t|tt|z|zzz|AP|ap
<timeformat> ::= <format-char> | <format-char><string>

Example:

<TIMEFORMAT=hh:mm ap>
<TIMEFORMAT=hh:mm:ss>

Description:

These expressions may be used for the time:

h the hour without a leading zero (0..23 or 1..12 if AM/PM display)
hh the hour with a leading zero (00..23 or 01..12 if AM/PM display)
m the minute without a leading zero (0..59)
mm the minute with a leading zero (00..59)
s the second without a leading zero (0..59)
ss the second with a leading zero (00..59)
z the milliseconds without leading zeroes (0..999)
zzz the milliseconds with leading zeroes (000..999)
AP use AM/PM display. AP will be replaced by either "AM" or "PM".
ap use am/pm display. ap will be replaced by either "am" or "pm".
All other input characters will be ignored.

The TIMEFORMAT script is used to set the formatting of the returned string of subsequent TIME scripts. The format set by the TIMEFORMAT script remains until either the format is set again by another TIMEFORMAT script or until all scripts in the string that the TIMEFORMAT script is part of has been fully evaluated possibly longer in certain contexts. The timeformat is not case sensitive.

Return value:

The script always returns the empty string.

1.5.3 DATE

Syntax:

```
<date-script> ::= "<DATE" [ "=" <dateformat> ] ">"
<format-char> ::= d|dd|ddd|dddd|M|MM|MMM|MMMM|yy|yyyy
<dateformat> ::= <format-char> | <format-char><string>
```

Example:

```
<DATE>
```

Description:

Used to get the current date in the correct format. If format parameter is omitted then format is set by current DATEFORMAT format string. If either is set then default format is set; yyyy-MM-dd.

List of format characters:

d	the day as number without a leading zero (1-31)
dd	the day as number with a leading zero (01-31)
ddd	the abbreviated localized day name (e.g. 'Mon'..'Sun').
dddd	the long localized day name (e.g. 'Monday'..'Sunday').
M	the month as number without a leading zero (1-12)
MM	the month as number with a leading zero (01-12)
MMM	the abbreviated localized month name (e.g. 'Jan'..'Dec').
MMMM	the long localized month name (e.g. 'January'..'December').
yy	the year as two digit number (00-99)
yyyy	the year as four digit number (1752-8000)

All other input characters will be ignored.

Return value:

Returns the current date in the format set by the DATEFORMAT script.

1.5.4 TIME

Syntax:

```
<time-script> ::= "<TIME" [ "=" <parameter> ] ">"
<format-char> ::= h|hh|m|mm|s|ss|t|t|z|zzz|AP|ap
<timeformat> ::= <format-char> | <format-char><string>
```

Example:

```
<TIME>
```

Description:

Used to get the current time in the correct format. If format parameter is omitted then format is set by current TIMEFORMAT format string. If either is set then default format is set; hh:mm:ss.

These expressions may be used for the time:

h	the hour without a leading zero (0..23 or 1..12 if AM/PM display)
hh	the hour with a leading zero (00..23 or 01..12 if AM/PM display)
m	the minute without a leading zero (0..59)
mm	the minute with a leading zero (00..59)
s	the second without a leading zero (0..59)

ss the second with a leading zero (00..59)
z the milliseconds without leading zeroes (0..999)
zzz the milliseconds with leading zeroes (000..999)
AP use AM/PM display. AP will be replaced by either "AM" or "PM".
ap use am/pm display. ap will be replaced by either "am" or "pm".
All other input characters will be ignored.

Return value:

The TIME script returns the current date in the format set by the TIMEFORMAT script.

1.5.5 TIMESTAMP

Syntax:

```
<timestamp-script> := "<TIMESTAMP>"
```

Example:

```
<TIMESTAMP>
```

Description:

The TIMESTAMP is a script that is used to output the current date and time. The TIMESTAMP format is: YYYY-MM-DDThh:mm:ss, which can for example produce 2002-06-17T14:27:46. This format is in compliance with the ISO standard iso8601 date format. The format is invariant of the format set by DATEFORMAT or TIMEFORMAT

Return value:

Returns current date and time in ISO format iso8601.

1.5.6 FILESTAMP

Syntax:

```
<timestamp-script> := "<FILESTAMP>"
```

Example:

```
<FILESTAMP>
```

Description:

FILESTAMP is a script that can be used to output the current date and time, in a format that can be used as a filename. The <FILESTAMP> format is: YYYYMMDD.hhmmss, which can for example produce 20020617.142746.

Return value:

Returns current date and time similar format to the ISO format but without redundant separators.

1.5.7 DATEOFFSET

Syntax:

```
<dateoffset-script> := "<DATEOFFSET" <offset> ">"  
<offset> := <script> that returns a <float-string>
```

Examples:

```
<DATEOFFSET<TEXT=1>>  
<DATEOFFSET<PROD=DateOffset>>
```

Description:

Calculates the date with the give offset in days from the current date.

Return value:

Returns the date with the given offset from the current date in the format set by the DATEFORMAT script.

1.6 Strings

String scripts manipulate text and numbers. String scripts can be used to search for specific strings or strings that are in specific locations.

1.6.1 STRCMP

Syntax:

```
<strcmp-script> ::= "<STRCMP" <string-argument> <string-argument> ">"  
<string-argument> ::= <script> that returns a <string>
```

Example:

```
<STRCMP<TEXT=Coke><TEXT=Pepsi>>  
<STRCMP<TEXT=CokeID><Prod=ID>>
```

Description:

The STRCMP compares strings.

Return value:

It returns "0" if the strings are equal and only if the strings are equal. If the first argument precedes the second it returns "-1" else it returns "1". The strings are compared lexically so that the comparison is based exclusively on the numeric Unicode values of the characters and is very fast, but the ordering is not always what a human would expect.

1.6.2 STRSTR

Syntax:

```
<strstr-script> ::= "<STRSTR" <haystack> <needle> ">"  
<haystack> ::= <script> that returns a <string>  
<needle> ::= <script> that returns a <string>
```

Example:

```
<STRSTR<TEXT=Banana><TEXT=an>>
```

Description:

The STRSTR script finds the first occurrence of the substring needle in the string haystack.

Return value:

The STRSTR script returns the substring from the to the first occurrences of the needle, or an empty string if the substring is not found. If the needle is empty the whole haystack is returned.

1.6.3 STRCAT

Syntax:

```
<strcat-script> ::= "<STRCAT" <destination> <source> ">"  
<destination> ::= <script> that returns a <string>  
<source> ::= <script> that returns a <string>
```

Example:

```
<STRSTR<TEXT=Banana><TEXT=an>>
```

Description:

The STRCAT script appends the source string to the destination string.

Return value:
The resulting string of the concatenation is returned

1.6.4 STRLEN

Syntax:
<strlen-script> ::= "<STRLEN" <string-argument> ">"
<string-argument> ::= <script> that returns a <string>
Example:
<STRLEN<TEXT=Banana>>
<STRLEN<PROD=Price>>

Description:
The STRLEN script calculates the length of the string given as the argument to the script. Empty strings have zero length.

Return value:
The STRLEN function returns the number of characters in the argument.

1.6.5 LEFT

Syntax:
<left-script> ::= "<LEFT" <string-argument> <length> ">"
<string-argument> ::= <script> that returns a <string>
<length> ::= <script> that returns a <int-part>
Example:
<LEFT<TEXT=Banana><TEXT=3>>

Description:
The LEFT script creates the substring from the start of string of the given length.

Return value:
Returns a substring that contains the len leftmost characters of the string. The whole string is returned if len exceeds the length of the string.

1.6.6 MID

Syntax:
<mid-script> ::= "<MID" <string-argument> <index> <length> ">"
<string-argument> ::= <script> that returns a <string>
<index> ::= <script> that returns a <int-part>
<length> ::= <script> that returns a <int-part>
<int-part> ::= <digit> | <int-part> <digit>
<digit> ::= any one of the ten digits 0 through 9

Example:
<MID<TEXT=Coca Cola><TEXT=2><TEXT=4>>

Description:
The MID script creates the substring that contains length characters starting at position index.

Return value:
Returns a string that contains the length characters of this string, starting at position index.

Returns an empty string if the string is empty or index is out of range. Returns the whole string from index if index+len exceeds the length of the string.

1.6.7 SUBSTRING

Syntax:

```
<substring-script> ::= "<SUBSTING" <string-argument> <index> <length> ">"  
<string-argument> ::= <script> that returns a <string>  
<index> ::= <script> that returns a <int-part>  
<length> ::= <script> that returns a <int-part>
```

Example:

```
<SUBSTING<TEXT=Coca Cola><TEXT=2><TEXT=4>>
```

Description:

Identical to MID see MID

Return value:

Identical to MID see MID

1.6.8 RIGHT

Syntax:

```
<right-script> ::= "<RIGHT" <string-argument> <length> ">"  
<string-argument> ::= <script> that returns a <string>  
<length> ::= <script> that returns a <int-part>
```

Example:

```
<RIGHT<TEXT=Coca Cola><TEXT=4>>
```

Description:

The RIGHT script creates the substring of the given length to the end of string.

Return value:

Returns a substring that contains the len rightmost characters of the string. The whole string is returned if len exceeds the length of the string.

1.6.9 ENDSWITH

Syntax:

```
<endswith-script> ::= "<ENDSWITH" <haystack> <needle> ">"  
<haystack> ::= <script> that returns a <string>  
<needle> ::= <script> that returns a <string>
```

Example:

```
<ENDSWITH<TEXT=Coca Cola><TEXT=Cola>>  
<ENDSWITH<TEXT=Coca Cola><TEXT=Coke>>
```

Description:

The ENDSWITH script checks if the haystack ends with the needle.

Return value:

Returns "1" if the haystack ends with the needle else it returns "0".

1.6.10 STARTSSWITH

Syntax:

```
<startswith-script> ::= "<" <script-name> <haystack> <needle> ">"  
<script-name> ::= "STARTSSWITH"
```

<haystack> ::= <script> that returns a <string>
<needle> ::= <script> that returns a <string>

Example:

```
<STARTSSWITH<TEXT=Coca Cola><TEXT=Cola>>  
<STARTSSWITH<TEXT=Coca Cola><TEXT=Coke>>
```

Description:

The STARTSSWITH script checks if the haystack starts with the needle.

Return value:

Returns "1" if the haystack ends with the needle else it returns "0".

1.6.11 LPAD

Syntax:

```
<lpad-script> ::= "<LPAD" <string-argument> <length> [<padding-char> <truncate>] ">"  
<string-argument> ::= <script> that returns a <string>  
<length> ::= <script> that returns a <int-part>  
<padding-char> ::= <script> that returns a <c>  
<truncate> ::= <script> that returns a "0" or "1"
```

Examples:

```
<LPAD<TEXT=Date:><TEXT=10>>  
<LPAD<PROD=ID:><TEXT=10><TEXT=0><TEXT=1>>
```

Description:

Modifies string to be of specified length by inserting the fill character in front of the string.

If truncate is "0" and the length of the string is more than width, then the returned string is unmodified. If truncate is "1" and the length of the string is more than width, then the resulting string is truncated to the given length. padding-char is optional and defaults to space " ". truncate is also optional and default to "1".

Return value:

Returns a string of specified length that contains the fill character followed by the string.

1.6.12 RPAD

Syntax:

```
<rpad-script> ::= "<RPAD" <string-argument> <length> [<padding-char> <truncate>] ">"  
<string-argument> ::= <script> that returns a <string>  
<length> ::= <script> that returns a <int-part>  
<padding-char> ::= <script> that returns a <c>  
<truncate> ::= <script> that returns a "0" or "1"
```

Examples:

```
<RPAD<TEXT=Date:><TEXT=10>>  
<RPAD<PROD=ID:><TEXT=10><TEXT=0><TEXT=1>>
```

Description:

Modifies string to be of specified length by appending the fill character to the string.

If truncate is "0" and the length of the string is more than width, then the returned string is unmodified. If truncate is "1" and the length of the string is more than width, then the resulting string is truncated to the given length. padding-char is optional and defaults to space " ". truncate is also optional and default to "1".

Return value:

Returns a string of the specified length that contains the string padded by the fill character.

1.6.13 STREQ

Syntax:

```
<streq-script> ::= "<STREQ" <value> <value> ">"  
<value> ::= <script> that returns a <string>
```

Examples:

```
<STREQ<TEXT=Coke><TEXT=Pepsi>>  
<STREQ<TEXT=Coke><PROD=Name>>
```

Description:

The STREQ script is used to compare string lexically.

Return value:

Returns the "1" if the strings are equal else "0" is returned.

1.6.14 STRLT

Syntax:

```
<strlt-script> ::= "<STRLT" <value> <value> ">"  
<value> ::= <script> that returns a <string>
```

Examples:

```
<STRLT<TEXT=Coke><TEXT=Pepsi>>  
<STRLT<TEXT=Coke><PROD=Name>>
```

Description:

The STRLT script is used to compare strings lexically.

Return value:

Returns the "1" if the value of the first argument is less than the value of the second argument else "0" is returned.

1.6.15 SECTION

Syntax:

```
<section-script> ::= "<SECTION" <stringarg> <sep> <start> [<stop> [<flags>]] ">"  
<stringarg> ::= <script>  
<sep> ::= <script>  
<start> ::= <script>  
<stop> ::= <script> (optional)  
<flags> ::= <script> (optional, default is 0)
```

Examples:

```
<SECTION<TEXT=field1,field2,field3,field4><TEXT=,><TEXT=3>>  
  (will return "field3,field4")  
<SECTION<TEXT=field1,field2,field3,field4><TEXT=,><TEXT=2><TEXT=3>>  
  (will return "field2,field3")  
<SECTION<TEXT=field1,field2,field3,field4><TEXT=,><TEXT=1><TEXT=1>>  
  (will return "field1")  
<SECTION<TEXT=field1,field2,field3,field4><TEXT=,><TEXT=-1>>  
  (will return "field4")  
<SECTION<TEXT=field1,field2,field3,field4><TEXT=,><TEXT=-3><TEXT=-2>>  
  (will return "field2,field3")
```

Description:

Returns a section (or a field) from a string, based on the input given.

<sep> is the field separator

<start> specifies the position of the first field to return

<stop> specifies the position of the last field to return

<flags> is a bit mask, which affects the behaviour of the script

0 – default – empty fields are counted, compare is case sensitive, leading and trailing separators are not included

1 – skip empty fields

2 – include leading separator

4 – include trailing separator

8 – separator compare is case-insensitive

Return value:

A sub-string containing fields from an input string.

1.6.16 REPLACE

Syntax:

```
<replace-script> ::= "<REPLACE" <source-string> <search-string> <replace-string>
">"
<source-string> ::= <script>
<search-string> ::= <script>
<replace-string> ::= <script>
```

Examples:

<REPLACE<TEXT=I am a cat><TEXT=cat><TEXT=dog>> (returns the string "I am a dog")

Description:

Searches for occurrences of <search-string> in <source-string> and replaces with <replace-string>

Return value:

The new string, which has the replaced values.

1.7 Miscellaneous

Miscellaneous scripts are scripts that do not easily fall into any of the other categories, in essence they have simply yet to be classified.

1.7.1 PAYMENT

Syntax:

```
<payment-script> ::= "<PAYMENT=" <parameter> ">"
<parameter> ::= <script>
```

Examples:

Description:

Returns a value that has been generated in a payment transformation.

Return value:

Returns a value that has been generated in a payment transformation.

1.7.2 TRANSFORMATION

Syntax:

```
<transformation-script> ::= "<TRANSFORMATION=" <parameter> ">"
```

<parameter> ::= <script>

Examples:

Description:

Returns a value that has been generated in a table transformation.

Return value:

Returns a value that has been generated in a table transformation.

1.7.3 READBLOCK

Syntax:

```
<readblock-script> ::= "<READBLOCK" <file-name> <block-index> [<file-type>] ">"
<file-name> ::= <script>
<block-index> ::= <script>
<file-type> ::= <script> optional parameter that can take the values:
    structured (default if nothing is given)
    btree
    memory
    rawmemory
    sstructured (secured and structured)
```

Examples:

```
<READBLOCK<TEXT=File><TEXT=99>>
<READBLOCK<TEXT=File><TEXT=10><TEXT=rawmemory>>
```

Description:

Reads a record from a file.

Return value:

1.7.4 READBLOCKS

Syntax:

```
<readblocks-script> ::= "<READBLOCKS" <file-name> [<block-start-index> [<block-
stop-index> [<file-type>]]] ">"
<file-name> ::= <script>
<block-start-index> ::= <script>
<block-stop-index> ::= <script> the read includes block at block-stop-index
<file-type> ::= <script> optional parameter that can take the values:
    structured (default if nothing is given)
    btree
    memory
    rawmemory
    sstructured (secured and structured)
```

Examples:

```
<READBLOCKS<TEXT=File><TEXT=99><TEXT=101>>
<READBLOCKS<TEXT=File><TEXT=10><TEXT=-1><TEXT=rawmemory>>
```

Description:

Reads a specific number of records from a file.

Setting <block-stop-index> to -1 will read from the <block-start-index> to the end of file.

Return value:

1.7.5 WRITEBLOCK

Syntax:

```
<writeblock-script> ::= "<WRITEBLOCK" <file-name> <block-index> <data> [<file-type>] ">"  
<file-name> ::= <script>  
<block-index> ::= <script>  
<data> ::= <script>  
<file-type> ::= <script> optional parameter that can take the values:  
    structured (default if nothing is given)  
    btree  
    memory  
    rawmemory  
    sstructured (secured and structured)
```

Examples:

Description:

Writes a record to the specified file

Return value:

1.7.6 BLOCKCOUNT

Syntax:

```
<blockcount-script> ::= "<BLOCKCOUNT" <file-name> [<file-type>] ">"  
<file-name> ::= <script>  
<file-type> ::= <script> optional parameter that can take the values:  
    structured (default if nothing is given)  
    btree  
    memory  
    rawmemory  
    sstructured (secured and structured)
```

Examples:

Description:

Returns the number of records in a file

Return value:

The number of records

1.7.7 DELETEBLOCK

Syntax:

```
<deleteblock-script> ::= "<DELETEBLOCK" <file-name> <block-index> [<file-type>] ">"  
<file-name> ::= <script>  
<block-index> ::= <script>  
<file-type> ::= <script> optional parameter that can take the values:  
    structured (default if nothing is given)  
    btree  
    memory  
    rawmemory  
    sstructured (secured and structured)
```

Examples:

Description:
Deletes a record from a file

Return value:
Nothing.

1.7.8 ALERT

Syntax:
<alert-script> ::= "<ALERT" <message-parameter> ">"
<message-parameter> ::= <script>

Examples:
<ALERT<TEXT=This is a message>>

Description:
Displays a message dialog to the user

Return value:
Nothing.

1.7.9 SLEEP

Syntax:
<sleep-script> ::= "<SLEEP" <sleep-duration> ">"
<sleep-duration> ::= <script> specifies a sleep duration in milliseconds

Examples:

Description:
Makes the script halt for the specified number of milliseconds.

Return value:
Nothing.

1.7.10 XPDAPRINTERTEST

Syntax:
<xpdaprintertest-script> ::= "<XPDAPRINTERTEST>"

Examples:
<XPDAPRINTERTEST>

Description:
Test whether the printer can print or not.

Return value:
1 – if the printer can print
0 – if the printer cannot print

1.7.11 XPDAPRINTBLOCKS

Syntax:
<readblocks-script> ::= "<XPDAPRINTBLOCKS" <file-name> [<block-start-index>
[<block-stop-index> [<file-type>]]] ">"
<file-name> ::= <script>
<block-start-index> ::= <script>
<block-stop-index> ::= <script> the read includes block at block-stop-index
<file-type> ::= <script> optional parameter that can take the values:

structured (default if nothing is given)
btree
memory
rawmemory
sstructured (secured and structured)

Examples:

Description:

Prints a specific number of records from a file.

Setting <block-stop-index> to -1 will read from the <block-start-index> to the end of file.

Return value:

Nothing