

**METHOD AND APPARATUS FOR CONTROLLING A COMPUTING OR
ELECTRONIC DEVICE**

FIELD OF THE INVENTION

5 The present invention relates to a method and apparatus
for controlling a computing or electronic device, so that
software applications can be provided to such a device or
updated, particularly where such devices are remotely
located. These devices can be in the form, for example,
10 of point of sale devices (such as those referred to as
customer interaction devices) used at points of customer
interaction.

BACKGROUND OF THE INVENTION

15 Traditional customer interaction devices (CIDs), such as
point-of-sale ("POS") terminals used in payment
transaction processing, must be loaded with software to
operate in the required manner. This software controls,
for example, the presentation of messages on the CID
20 display, the printing of messages on receipts, and
interactions with a host system or server in the required
application message protocol.

Such implementations typically involve the use of compiled
25 program codes comprising relatively large software modules
of machine instructions. These modules are typically
executed directly by the hardware of the CID. However, in
the event a new version is required, such modules
generally take too long to load into the CID via a
30 telecommunications link owing to their size. Indeed,
traditional CIDs such as POS terminals communicate with
host systems using relatively slow data communications
links, making it particularly impractical to replace
software in the CID through telecommunication lines. Any
35 change in the CID functionality thus requires the physical
replacement of the CID, or the transportation to and
loading on-site of replacement software, which adds to

costs and the required labour (particularly when the CID network is spread over a large geographical area). This makes it impractical and costly for businesses using the CID (such as banks with their POS networks) to make

5 frequent changes to CID functionality, thereby limiting the ability of businesses to respond to new market conditions or offer new services and products in response to new conditions.

10 In addition, a CID for credit card payments commonly includes functionality for customer loyalty rewards, where a customer may at the time of a payment transaction be given a virtual or electronic coupon, or be permitted to use a previously given electronic coupon to offset the

15 instant purchase price. The rules by which the coupon is issued (for example, "if a customer purchases \$100 or more, he or she receives a \$20 coupon redeemable in the next 3 weeks") may have to be modified from time to time. For example, the aforementioned rule might be modified for

20 a limited time such that "for the month of August, purchases of \$100 or more attract a \$30 coupon plus an additional \$20 cash rebate for purchases above \$300", to coincide with a local festival or celebration falling over that period. Such a \$20 cash rebate would be redeemable

25 against subsequent purchases in the following two weeks and would have to be redeemed before the redemption of the \$30 coupon. It may not be practical to implement this new rule in the CID with existing CID software, for the reasons discussed above, owing to the short-term life of

30 the change.

In prior art approaches, one technique for providing some flexibility in the CID is to include parameters in the CID software, such that changes in requirements to some extent

35 can be effected by downloading relatively small blocks of information that merely alter those parameters. For example, in the example cited above where the requirement

is to change the coupon award from a \$20 coupon to a \$30 coupon, such an approach would comprise downloading a small data packet to the CID indicating the new value of the appropriate parameter is \$30 (rather than the previous \$20). However, if - as in the example above - the original CID software had not been designed to handle the additional rule referring to a bonus cash rebate (in that example, of \$20) for purchases above \$300 and redeeming the \$20 cash rebate prior to the redemption of the \$30 coupon in the next 2 weeks, then it would not be possible to effect the change by just downloading one or more parameters. Rather, a change in the CID software would be required, with the associated practical problems discussed above.

Another common problem faced by businesses with large numbers of CIDs that have been installed over several or many years is that the population of CIDs typically will comprise many different models and, more likely than not, be provided by different manufacturers. Consequently, software in different models and makes of CIDs will commonly not be compatible. It hence becomes necessary to provide new software for each new functionality multiple times, according to the diverse characteristics of the software used by each CID.

It is a common practice that manufacturers of CIDs equip each CID with a CID Operating System, and further provide a software development kit (SDK) which facilitate the development of application software for execution in the CID. The SDK may include a set of Application Program Interfaces (APIs) incorporated in the CID Operating System that facilitates the development of application programs. These APIs typically includes APIs for handling input and output through the various CID interface devices, including keyboards, display units, card acceptors, printers and host communication ports and modems. The

software developed to run in a particular CID can make use of these APIs provided in the CID's SDK. However, each manufacturer provides different SDKs with different APIs, so CID applications must typically be customised according to the specific CID model. Modification of the software for different CIDs typically involves re-writing the CID application software that makes use of the CID-specific APIs. A common technique of managing the supporting of multiple models of CID is to 'layer' the application software design such that the application software is divided into an 'application function layer' that contains any software that is CID model-independent; the application function layer makes use of another 'layer' of software, the 'CID model-specific layer' that incorporates APIs which are CID model-specific. Thus, it is possible to port the software from one CID model to another by modifying only the CID model specific layer; only a single version of the CID model-independent layer has to be maintained which need not be modified to accommodate different CID models. However, when the application function layer must be modified because new functionality is required, the entire application software module has to be replaced, as conventional software does not allow for partial software module replacement. Further, this replacement is cumbersome and costly owing to the necessity (discussed above) of physical visiting the individual CIDs to replace the software.

The cost of implementing new requirements in CID therefore increases, owing both to the implementation of software changes for a given requirement many times, each time for a different model of CID, and to the physical visits to each CID to load in the modified software, for new functions to be implemented.

A further problem associated with changing user requirements and processing needs relates to the smart

card and other similar personal data storage devices (PDSs), such as, contact or contactless smart cards, magnetic striped cards, chip bearing devices such as watches, key rings, keys, phones, small pocket devices, electronic wallets and the like used at the point of interaction with CIDs. It is common for PDSs to be used for multiple applications, for convenience and efficiency.

Existing techniques for managing multiple applications on PDS include the division of the data storage area in the PDS into 'directories' and 'files' as specified in the International Standards Organisation's specifications for integrated circuit cards (the ISO 7816 series of specifications), and the loading of 'applets' or software modules into PDSs where the PDSs support the loading of such applets, such as those specified by Visa International and Mastercard International. However, in many types of lower cost smart cards and other PDSs, loading of applets is not a supported option.

Such PDSs (i.e. non-applet PDSs) are typically used as secure data storage devices, and the memory of such PDSs can then be divided into 'directories' and 'files' for use by different applications. When a first application is no longer required, the memory allocated to the first application can be re-used by a second application. However, typical implementations limit the flexibility of the files and directories and makes it cumbersome to change and to share between authorised users effectively. Prior art techniques require an 'application directory' to be stored in the PDS, so that it can be determined what application data reside in which part of the PDS memory. The CID typically interrogates the 'directory' (such as that defined in ISO 7816 of the International Standards Organisation, or that disclosed U.S. Patent No. 6,449,684), in order to determine whether an application is loaded in the PDS. When an application ceases to be

effective (e.g. expires), the data previously stored in the PDSB remains in the PDSB.

SUMMARY OF THE INVENTION

5 According to a first aspect of the invention, there is provided a method for controlling a computing or electronic device having an operating system and digital processor, comprising:
providing said device with one or more scripts
10 each comprising one or more script lines;
providing said device with a program for interpreting said script lines;
wherein each of said script lines includes a respective one of a plurality of command tags, said
15 respective command tag being indicative of a respective one of a plurality of application software modules resident in said device, whereby one or more of said application software modules can be invoked by means of said scripts so that said device can be operated according
20 to said scripts.

Thus, application software modules typically comprise instructions to digital processor to carry out the desired processing indicated by the respective script line, and
25 the program acts as a virtual computer. While scripts as such are known in the prior art for a number of applications (see for example U.S. Patents Nos. 6,256,635, 6,167,567, 6,041,183, 6,275,868, 6,038,394 and 5,784,541), no prior art application employs scripts for the purpose
30 of implementing the primary application itself, nor does any implement a script interpreter and executor as a virtual computer. The approach of the present invention allows a minimum of overheads in interpretation of the script, unlike prior art interpreters which interpret
35 scripts and generates machine code in real-time, the machine code then being executed (cf. U.S. Patent No. 6,038,394). Other prior art applications of scripts have

generally involved using scripts to modify configuration and other data in the target devices and peripherals, the configuration and other data being part of some other primary application function (e.g. U.S. Patents Nos. 5 6,256,635 and 6,041,183). Some prior art approaches employ scripts for the management of the download of machine code software (e.g. U.S. Patent Nos. 6,167,567 and 5,931,909).

10 Preferably the script lines include one or more references to data to be passed to said application software modules when invoked.

Thus, the scripts can specify through references, whether 15 direct or indirect, the module to invoke and the data to use to invoke the module.

Preferably each of said command tags is indicative of respective application module by means of an intermediary 20 respective interface to said respective application module.

No prior art approach involves associating each line of script with an interface (such as an API) so that the 25 script points to the machine code to be executed.

Preferably the program for interpreting said script lines is invokable by means of at least one of said application software modules.

30 Preferably the program for interpreting said script lines comprises a virtual computer comprising machine code software.

35 Preferably the device comprises a customer interaction device or a point-of-sale terminal.

Preferably the device is operable to update said scripts or said script lines. More preferably the device is operable to update said scripts or said script lines by remote communication means.

5

Preferably the updating is based on version numbering whereby said program can determine by comparing a current version of a respective script or of one or more script lines and a proposed version to be loaded into said device, and to request said updating only if said proposed version is newer than said current version.

More preferably the updating comprises either a full replacement or a partial replacement, determined by comparing any proposed new script or script lines with current script or script lines to determine if a partial replacement is feasible or if a full replacement is required.

20 Preferably the device is adapted for use with personal data storage devices and memory usage within said personal data storage devices is performed on a time limited basis involving:

25 using a version number encoded in each respective personal data storage device for tracking memory allocation within said respective personal data storage device; and

controlling the use of multiple versions of said memory allocation so that memory in said respective personal data storage device can be redefined for different use over a period of time,

30 wherein said version number is automatically performed and embedded in said script lines such that said personal data storage devices are automatically updated from old to new memory allocations by said script lines.

Preferably the script lines can be usable in a plurality

of different models of computing or electronic devices.

According to a second aspect of the invention, there is provided a method of implementing a loyalty program
5 involving a method of controlling a remote computing or electronic device described above.

According to a third aspect of the invention, there is provided a method of implementing marketing program
10 involving a method of controlling a remote computing or electronic device described above.

According to a fourth aspect of the invention, there is provided a method of implementing a loyalty and marketing
15 program involving a method of controlling a remote computing or electronic device described above.

According to a fifth aspect of the invention, there is provided a method of implementing payment applications at
20 a customer interaction device involving a method of controlling a remote computing or electronic device as described above, wherein said remote computing or electronic device comprises said customer interaction device.

25 According to a sixth aspect of the invention, there is provided a method of implementing a combined payment application and loyalty program involving a method of controlling a remote computing or electronic device
30 described above.

According to a further aspect of the invention, there is provided a method of implementing a combined payment application and marketing program involving a method of
35 controlling a remote computing or electronic device described above.

According to a still further aspect of the invention,
there is provided a method of implementing a combined
payment application, loyalty program and marketing program
involving a method of controlling a remote computing or
5 electronic device described above.

According to yet a further aspect of the invention, there
is provided a computing or electronic device having an
operating system and digital processor, comprising:
10 one or more scripts each comprising one or more
script lines;
a program for interpreting said script lines;
wherein each of said script lines includes a
respective one of a plurality of command tags, said
15 respective command tag being indicative of a respective
one of a plurality of application software modules
resident in said device, whereby one or more of said
application software modules can be invoked by means of
said scripts so that said device can be operated according
20 to said scripts.

The invention also provides a system for generating
scripts for use with the method of controlling a computing
or electronic device described above.

25 Preferably the system also includes a script
administration module having a visual interface by means
of which a user can input requirements, wherein said
system is operable on the basis of said requirements to
30 generate a script or scripts for interpretation and
execution by means of said program.

The invention provides, in a further aspect, a system for
testing scripts adapted for use in the method of
35 controlling a computing or electronic device described
above, comprising a host computer provided with a version
of said program, said version being operable to interpret

and execute said scripts in a virtual replica of said device, so that errors in said scripts can be detected and corrected before the scripts are loaded into said device.

5 Thus, the present invention provides a means for managing the memory in PDSDs for use by different applications over a period of time, and to have this coordinated with corresponding CID functionality change to support the changing applications, and ensure that the PDSD and CID
10 work in concert over a lifetime of application changes.

With this invention, businesses are able to change CID functionality by downloading new functions into the CID even via slow data communications links, such as those
15 commonly used by credit card payment POS terminals through use of 'scripts', as scripts can be kept to small modules that do not take too long to download remotely, even via slow data communications links.

20 This allows businesses to effect new functionality in CIDs quickly and cost-effectively, with minimal cost incurred in deploying the new functions into CIDs which may be spread over large distances, and thus enjoy a shorter time to market and lower cost of implementation.

25 Further, this invention allows scripts to be common to all makes and models of CIDs, thus saving businesses effort in re-implementing different versions of software for different models and makes of CIDs for the same
30 functionality. The same script written to cater for any new requirement can be downloaded into CIDs of all models, thus reducing implementation effort. For example, traditional POS software would require the software modifications for new requirements to be re-implemented
35 for each model of POS. Using this invention, the script for new requirements would only have to be written once, and the same script could be deployed to all models of

CIDs implemented with the invention.

Where the application in the CID is designed to work with PDSDs, this invention allows the user to define the
5 application data in the PDSD on a time-limited basis, together with the corresponding application scripts to work with the application data. Thus, a given area of memory in the PDSD can be allocated for use by a first application during a given time period, after which period
10 a second application can be loaded in the CID to use the same area of memory in the PDSD. This clearly can continue indefinitely, with one application after another, for the lifetime of the PDSD's use.

15 Further, the memory areas in the PDSD used by a first application need not be exactly the same as the memory areas used by a second application initiated after the expiry of the first application. The second application may use more or less memory, which may include a part, the
20 whole or none of the memory area occupied by the first application. A first application may share memory with other applications running concurrently in time.

This invention therefore facilitates the re-use of memory
25 in PDSDs for time-limited applications and ensures that the use of memory in the PDSD and the application scripts in the CID are coordinated.

BRIEF DESCRIPTION OF THE DRAWING

30 In order that the invention may be more clearly ascertained, a preferred embodiment will now be described, by way of example, with reference to the accompany drawing, in which;

Figure 1 is a schematic diagram of a CID system
35 incorporating a Script Engine according to a preferred embodiment of the present invention.

DETAILED DESCRIPTION

According to a preferred embodiment of the present invention, there is provided a 'virtual computer' referred to as a Script Engine which is software implemented in a conventional programming language (for example, Assembler Language, C or Pascal or any other programming language) and compiled into machine instructions for direct interpretation and executed by the central processing unit of a CID.

10

The Script Engine is designed to interpret scripts, each of which comprises lines of instructions interpretable by the Script Engine; each line of script causes the Script Engine to invoke a corresponding software function through an API implemented in machine language in the Script Engine; the software function incorporates a specific processing algorithm and/or causes the CID hardware to perform a specific action, such as reading data from the keyboard or displaying data on the display unit.

20

Thus, the Script Engine comprises a main flow control program with a set of software functions that are invoked via a corresponding set of APIs. Each API corresponds to a software function and is invoked from a specific script command making up a line of script. The lines of the scripts indicate to the main flow control program, in a controlled sequence according to the order of the lines of script, which APIs to invoke and what data should be passed to the respective API. When an API is invoked, the machine code making up the software function corresponding to that API is executed by the CID computer hardware, thus carrying out the required processing or effecting the required CID hardware action.

35

A script line typically comprises some indicator (a 'Command Tag') indicating to the Script Engine which API should be invoked. Thus, each Command Tag corresponds to

an API, and the line of script containing the Command Tag includes data to be passed to the API to effect the required processing and/or action through execution of the machine code making up the software function associated with that API. Thus, by assembling appropriate lines of script, the script writer can control the processing performed by the CID and, further, change the CID functionality by changing lines of scripts, without changing the underlying CID software. By implementing a properly defined set of APIs and corresponding Command Tags that provides a sufficient set of functions, an entire application can be specified in the script, thus enabling the script to be an 'application software' in its own right. This is analogous to the support by a hardware processor unit of a set of 'machine instructions' that make up a conventional software program. The set of APIs, according to this embodiment, is equivalent to such 'machine instructions', and the Script Engine is analogous to the said hardware processor unit of a conventional computer.

Figure 1 is a schematic diagram of a CID system incorporating the Script Engine of this embodiment.

The CID 2 comprises hardware including a central processing unit 4, random access memory (RAM) 6, flash memory or Static RAM 8, a set of input/output (I/O) in the form of LCD display or displays 10, a PDSD acceptor 12 (in the form of a barcode reader, a magnetic stripe reader, a smart card reader or the like), a keyboard 14 by which the user, cashier or teller enters transaction details, a PIN pad 16 by means of which the customer's Personal Identification Number or PIN can be entered, a printer 18 by means of which receipts and other information can be printed, and a communications link 20 (typically through a dial-out modem) to Host system 22.

The CID 2 also includes software components including a CID Operating System loaded in SRAM or flash memory 8, and the Device Interface APIs 26 through which other application software in the CID 2 accesses and makes use of the hardware devices described above. Software in the CID 2 also includes application software for 'Other applications' 28 such as credit card payment software and the Script Engine software 30.

Computer memory storage has to be set aside in the RAM 6 and SRAM or flash memory 8 for a Script Store 32 where the scripts are kept, and a Data Store 34 for transactional data storage.

In this embodiment, the Script Engine 30 is implemented in the CID 2 in a conventional programming language such as C or Assembler Language, and compiled into machine code which is then loaded into the CID 2. The Script Engine 30 is implemented in two main layers: a first or Common Layer, comprising software that is independent of the CID model, and a second or CID-specific Layer, comprising software code that makes use of the CID model specific APIs (such as the Device Interface APIs 26 provided with the CID 2).

The Script Engine 30 includes an API exposed to the Other Applications 28 on the CID 2; this exposed API is the means by which the Other Applications 28 initiate execution of the Script Engine 30. Transaction data is sent to the Script Engine 30 via the exposed API for additional processing after or during processing by these Other Applications 28. A typical processing steps start with the Other Applications 28, in this case the credit card payment application, in control of the CID 2, instructing user to swipe the credit card through the magnetic stripe reader PDS acceptor 12 to initiate a payment. When the card details have been read from the

card, the payment application (i.e. one of the Other Applications 28) prompts for the amount to be entered through the keyboard 14. After the amount is entered, the payment application passes the card and transaction data
5 (viz. card details and amount) and control to the Script Engine 30. On being invoked through the API by the Other Applications 28, the Script Engine 30 interprets the scripts in order to decide what to do with the data received via the API. The script may, for example,
10 include instructions to check the card number to determine if this card is entitled to a discount or a gift coupon, etc, and instructions to display and or print relevant data to further prompt for user action or retrieve data from the host via host messages. Upon completing the
15 scripted actions, the Script Engine 30 returns control of the CID 2 to the payment application to complete the payment process.

The Scripts typically contain instructions such as:

- 20 • Display 'You have three \$10 coupons';
- Display 'Press Enter to continue'; or
- Keyboard RESULT.

The third of these script instructions tells the CID 2 to
25 wait for a keyboard input and place the result in the data storage area named RESULT.

Scripts are made up of 'Script Lines', each Script Line being a series of bytes. Script Lines are arranged in an
30 array in sequence; each Script Line is referenced by a Script Line Number. The syntax of the Script Line comprises a 'verb' or action denoted by a Tag, followed by the data or information with which the action is to be carried out. Each verb generally corresponds to an API.
35 Following each verb in the script line are data to be passed to the API, as well as condition handling specifiers that indicate the next action to be taken

depending on the particular data results arising from the execution of the API. For example, the condition handling specifiers could comprise a list of possible data values in a specified memory location arising from the execution
5 of the function invoked through the API, and a list of destination identifiers (Script Line Numbers) indicative of the next line of script to be executed, depending on the resulting value in the specified memory location.

10 The following script line example illustrates this:
'Keyboard RESULT, 1, 0, 1, 2, 3, 9; 50, 110, 140, 190, 240'. In this example, the Script Engine 30 is programmed to invoke the Keyboard API passing in the argument RESULT (the memory location to receive the data from the
15 keyboard) and the following argument 1 (the number of characters to accept from the keyboard). The five arguments following this ('0', '1', '2', '3' and '9') are the "Possible Data Conditions", while the last five arguments are the Destination Line Numbers. If the value
20 in RESULT is 0 (i.e. the first Possible Data Condition), the Script Engine will continue execution at script line number 50 (the first Destination Line Number). Similarly, if the value in RESULT is 1 (the second Possible Data Conditions) after the API has been invoked by the Script
25 Engine, the Script Engine will execute script line number 110, the second Destination Line Number.

The Script Line may be structured to always have the first n characters in the script line indicate the Command Tag
30 and thereby identify the API that the Script Engine is to invoke, followed by the arguments for the API, and instructions on how to locate the next Script Line. The following are examples of possible Script Lines in this embodiment.

35

1. Display lineNo, columnNo, displayData, onError

The Display command instructs the Script Engine 30 to

display the data in the memory location indicated by displayData on the display, at line and column numbers indicated by lineNo and columnNo respectively. The onError argument comprises a list of Script Line numbers indicating at which Script Line the Script Engine is to commence execution, depending on the error condition resulting from executing the Display command. This list may be empty (i.e. no Script Line specified), or may contain one or more Script Line numbers. Where more than one Script Line number is given, it is accompanied by a list of Error Codes, each Error Code corresponding to one Script Line. For example, if onError comprises '0,1,2,3; 45,60,75,90', the next Script Line to be executed is 45 if the Display API returns an error code of 0, the next Script Line will be 60 if the error code returned is 1, the next Script Line will be 75 if the error code returned is 2, and the next Script Line will be 90 if the error code returned is 3.

2. Keyboard numChars, inputArea, inputLength, dataType, timeout, onError

A Script Line with the Keyboard command instructs the Script Engine 30 to accept input through the keyboard, and to place the data in the memory location indicated by inputArea. The number of characters to accept is indicated by the inputLength argument. The API waits for a user input for the length of time indicated in the argument timeout. The argument in dataType indicates to the API the data should be numeric only, or alphabetic only, or may be either. The onError argument is as described above.

3. ChipCard outputArea, outputLength, inputArea, inputLength, onError

The ChipCard command instructs the Script Engine 30 to output the data in the memory location indicated by outputArea to the PDS acceptor 12 (where the PDS

acceptor 12 is in the form of a smart card reader and the expected PDS is a ChipCard), the number of bytes of data being indicated by outputLength. ChipCards are designed to always return response data to any input and, in this case, the response data from the ChipCard is placed in inputArea, and the number of bytes of data received from the ChipCard is placed in inputLength by the API and returned to the Script Engine 30. The onError argument is as described above.

10

4. Compute equation, resultSet, destinationSet, onError
The Script Engine incorporates an 'expression evaluator' function that evaluates a standard algebraic expression passed into it, and places the result of the evaluation in the memory location specified by the right-hand side argument in the expression.

The Compute command causes the Script Engine 30 to pass the algebraic expression contained in 'equation' to the Expression Evaluator, which carries out the computation specified in the right-hand side of the expression 'equation' and places the result in the memory location specified by the argument in the left-hand side of the expression in 'equation'.

25

Optional argument resultSet, if present, contains a list of one or more possible resulting values of evaluating the expression. For each value in the resultSet list, a corresponding Script Line Number is given in the destinationSet list. The next Script Line Number that the Script Engine executes is determined by the resulting value of evaluating the expression. For example, if the resultSet is '0,3,5,7' and the destinationSet is '45,60'90,105', then - if the result of the expression evaluation is 3 - the next ScriptLine Number to be executed is 60.

35

5. Print outputArea, outputLength, onError

The Print command causes the Script Engine 30 to pass the data in outputArea to the print function, which prints the data on the printer 18. The outputLength argument is used to let the print function know the amount of data to be printed. The purpose of the onError argument is as described above.

10 6. SendData outputArea, outputLength, inputArea, inputLength, destinationPort, timeOut, onError

The SendData command causes the Script Engine to pass the data in outputArea to the data communications function for transmission on the communications port identified by the destinationPort argument. The outputLength argument is used to let the data communications function know the amount of data to be transmitted. Any reply from the recipient is stored in the memory location specified by inputArea, with the length of data received stored in the memory location pointed to by inputLength argument. The timeOut argument is used to specify how long the data communications function should wait for a successful transmission and receipt of data before giving up and returning an error condition. The purpose of the onError argument is as previously described.

25

7. WaitEvent eventType, eventOccurred, inputArea, inputLength, timeOut, onError

The WaitEvent command allows the Script Engine 30 to fall into a wait mode, waiting for specified events to occur. These events include, for example, the insertion of a smart card into the PDSO acceptor 12, the swiping of a magnetic stripe card through the magnetic stripe card reader, or the pressing of a key on the keyboard 14. The argument eventType indicates the location that contains a list of types of events to wait for. The eventOccurred argument indicates the location where the code indicating the type of event that has taken place is stored, upon the

event taking place. Any resulting data from the event is placed in inputArea and the number of bytes of data in inputArea is kept in inputLength. The argument timeOut is used to indicate how long the wait for an event should be.

5 The onError argument is as described above.

Thus, the Script Engine 30 can be constructed to provide a wide range of APIs to perform any computing process or function, and the corresponding Command Tag and Script

10 Line can then be supported.

Scripts are stored in the memory area of the CID 2 set aside as the Script Store 32 managed by the Script Engine 30. This allows the Script Engine 30 to retrieve the

15 script for interpretation when required, as well as to update the script if new ones are received from the Host computer 22. Thus, although the CID application may be implemented using many lines of script which may make it impractical to download over the slow data communication

20 link available to the typical CID (and are hence typically loaded in the CID prior to installation of the CID in the field), it is practical to replace small blocks of the script. The script can be structured in small modules that can be replaced individually without the need to

25 replace the entire block of script.

Scripts can be written to effect 'housekeeping' actions, that is, the receiving of new scripts from the host 22 and updating them in the Script Store 32, storage and

30 retrieval of data in the transaction Data Store 34, exchanging data with the host system or systems 22, reading, verifying, and authenticating PDSs such as magnetic stripe, barcode or chip cards and updating PDSs, processing and computing data, displaying data based on

35 rules encoded in the scripts and printing receipts with contents based on rules in the script.

The Script Engine 30 is given control of the Data Store 34, in which Script Engine 30 stores data that can be subsequently sent to the Host System 22. This memory location is typically used as a transaction store or
5 transaction log.

According to this embodiment, there is also provided a host-based Script Administration Module (SAM) implemented in the host system 22 for the purpose of generating Script
10 Lines based on user input through a graphical user interface. The SAM provides the user with a set of GUI screens by means of which the user specifies the Command Tags, and the data required for the API corresponding to the Command Tag, and from this specification input, the
15 SAM then generates the Script Lines which can be downloaded into the Script Engine for execution. All Script Lines are developed in blocks or 'modules'. Each script module comprising a number of Script Lines has a version number. Each time a script module is modified a
20 new version number is assigned to the modified script module such that each script module is uniquely identified by the script module version number.

The SAM also includes a function by which the user defines and manages the memory area allocation and data types in
25 the PDSs on which the Script Lines generated using the SAM are to operate. Each definition of the PDS memory area allocation is tracked as a distinct version, such that each version of the PDS Data Definition (PDD), has a
30 specific starting date from which the definition is effective.

Each PDD version contains a set of data element definitions, such that each of the said data element
35 definition specifies the Starting Memory Location (SML, typically in byte offsets from the starting memory address location), a description of the type of data to be kept in

the location, and the number of bytes used up for this piece of data starting from the SML, as well as the start and end dates between which the data in this area is defined by this definition, and outside of which said

5 start and end dates the memory location is available for use by another application. The SAM is operable by the user to manage the areas of memory locations, allowing the user to pick areas which can be included in and/or excluded from a new PDD version, based on the availability

10 of the locations as defined in the existing versions of PDDs. For example, the SAM would prevent the user from including an area of PDS memory for use for a new data element if that area is already included as part of an existing data element definition and the end date for that

15 the definition has not past. The SAM can also show the user available areas of the PDS memory for a given period of time by checking the existing PDS versions for memory locations not included in any data element definitions that are valid during the period in question. After a PDD

20 version has been defined, the SAM allows the user to create script modules based on the selected PDD version. The resulting script is constrained to be valid only during the period that the data elements it refers to is valid according to the PDD. This constrain is imposed by

25 the SAM. The CID 2 does not allow execution of the script module outside the validity period of any data element that it is referring to, according to the PDD whose version number associated with the script module. (This is readily distinguishable from prior art approaches where

30 memory allocation for different applications is based on pre-allocated files or memory which are managed using 'directory' or 'index' areas of the PDS, such areas being set aside to indicate which application is using which files or which parts of the PDS memory. Such approaches

35 using static indices or directories are relatively fixed and once allocated cannot be changed easily and synchronised with CID applications.)

During the development of scripts using the SAM, the user selects the version number of the PDD to be associated with the script module to be developed. The SAM
5 automatically allows the user to develop scripts which manipulate data from PDSs of the same version as the selected PDD version number. Each PDS has the version number encoded in it. The SAM allows the user to specify the rules for handling PDSs containing data recorded
10 according to older versions of the PDD, such rules being encoded in the Script Lines generated by the SAM.

There are a number of ways that this embodiment can handle older versions of the PDD:

- 15 (a) by ignoring old version data and overwriting the memory location with new data specified in the Script Lines generated by the SAM;
- (b) by saving the old data in a transaction record in the CID 2 for transmission to the host computer system 22; or
- 20 (c) by altering the old data according to some processing rule encoded in the Script Lines and transferring the old data to new memory locations specified in the new PDD version.

25 These rules can be automatically generated by the SAM during definition of a new version of the PDD. Upon such a conversion from an older to a newer version, the PDS is updated with the new PDD version number, which enables scripts generated with the newer version of the PDD to
30 work with the PDS.

This allows unused memory areas of PDSs to be allocated or used memory areas of PDSs to be reallocated for use by new applications as and when the new applications are
35 introduced into the CID 2. The Script Lines implementing new applications can be generated to include data transformation in the PDS from earlier versions to the

new version. The memory can be allocated for limited periods of time, thus allowing them to be reused as and when required for new applications. This technique for allocating memory allows PDS memory to be managed and
5 changed dynamically without concerns that the CID applications will not be able to cope with different versions of PDSs as applications change over time.

The inclusion of data definitions and data version
10 management algorithms within scripts to coordinate and manage different versions of PDSs with the application scripts provides an effective means of version control and multi-application management such that the PDS can be used with various and changing application requirements
15 with a minimum of memory overheads in the PDS, and allows memory usage in the PDS to be dynamically allocated and reallocated on demand in a synchronised and coordinated way.

20 It is also possible to manage Script Line module versions in the CID 2, so that each CID is able to receive from the host computer the correct Script Line module version that it should receive. The Script Engine 30 in the CID 2 has embedded in it a program logic to determine that if the
25 Script Line modules already loaded in the Script Engine 30 are not the same as that the Script Line module version that the host 22 is notifying it of, the Script Engine 30 automatically makes a request to the host computer 22 to download the latest Script Line module version. The
30 download contains instructions as to the Script Lines to be replaced, inserted or removed. This form of download reduces the number of scripts to be downloaded and thus makes downloads practical over slow communications links.

35 A version of Script Engine 30 is also incorporated in the SAM running on host computer 22, for the purpose of validating the Script Lines generated from the Script

Administration module. This Script Engine runs in the host computer system 22 in a virtual CID with all the virtual equivalents of the CID devices including display or displays, keyboard, printer, card reader or readers, PIN pad, etc., and is used to interpret and execute the Script Lines just as the Script Engine 30 in the CID 2 would. The user is thereby assisted in detecting erroneous Script Lines and developing error-free Script Lines for download to remote CIDs.

10

TYPICAL APPLICATIONS

This embodiment is of particular application in, for example, personalised marketing, customer loyalty and general customer relationship management (CRM). In these areas the interaction with a customer at the CID 2 is made more effective and relevant through use of scripts in the CID that can be modified and implemented promptly and efficiently at remote locations through use of data communications which may be limited in speed and which would normally preclude remote change of CID software.

20

Example 1

CID 2 has a payment and loyalty application loaded in which customers are awarded with 20% discount on every 3rd transaction that exceeds \$50.

25

As the shop's anniversary is in September, retailer decides to offer a special bonus to all customers whose birthday falls in September.

30

The scripts are modified and downloaded into the CID to check on each customer's birthday during a payment transaction: on every payment, the host replies with a 'flag' that indicates whether customer qualifies. Script then prints and displays the appropriate message if the customer qualifies.

35

To address customer privacy concerns, customer may be asked at the start of the payment transaction if customer wishes to participate: if customer says no, script does not check on customer's birthday.

5

Example 2

A mobile phone with Java technology support is loaded with Script Engine. The mobile phone allows a user to access a bank's services and enjoy various benefits according to the customer's overall relationship with bank.

10

A new scheme is introduced, which offers selected customers special cash vouchers in the month of December for any unit trust trades done over the mobile phone.

15

The script to implement this scheme is downloaded into the mobile phone over the mobile telephone network, and is automatically deactivated after December to make way for new scripts implementing new schemes.

20

Example 3

The Script Engine running in the CID accesses each customer's PSD to compute and offer to each customer gifts and other forms of rewards based on the customer's profile and transaction history.

25

A new scheme is introduced, where customers who are visiting the store for the second time in the week are given a bonus.

30

A replacement script block is downloaded into the CID to effect this change, such that the script algorithm now makes an additional check on the PSD data for the relevant transaction history information to determine the customer's entitlement under the scheme, and to update the PSD with information of the current visit.

35

Example 4

A Script Engine is integrated into e-wallets issued to customers of the bank for use in PCs over the Internet. The Script Engine is used to execute secure scripts which
5 have been digitally signed, and downloaded to the PC on demand via web pages where they can be used in conjunction with PDSs in the user's PC in order to effect financial calculations and determine special offers, prices or other entitlements. Owing to its integration with the e-wallet,
10 scripts are more secure and can access PDS via PDS readers connected to the PC. Scripts can be dynamically selected and downloaded to the user depending on the web page and the customer actions and profile.

15 Modifications within the scope of the invention may be readily effected by those skilled in the art. It is to be understood, therefore, that this invention is not limited to the particular embodiments described by way of example hereinabove.

CLAIMS:

1. A method for controlling a computing or electronic device having an operating system and digital processor,
5 comprising:
 providing said device with a script comprising one or more script lines, each script line comprising at least one predefined command tag;
 providing said device with a software program for
10 interpreting said script lines;
 wherein said software program is operable to invoke a plurality of software functions in said device and each of said command tags corresponds to a respective one of said software functions, whereby said software
15 program can be controlled to invoke in said device one or more of said software functions by means of said script.
2. A method as claimed in claim 1, including providing said script with user variable data elements associated
20 with said command tags and invoking the respective software functions in a manner dependent on said data elements.
3. A method as claimed in either claim 1 or 2, including
25 providing said software program in the form of a virtual computer.
4. A method as claimed in claim 3, wherein said virtual
30 computer comprises machine code software.
5. A method as claimed in any one of the preceding claims, wherein said device comprises a customer interaction device or a point-of-sale terminal.
- 35 6. A method as claimed in any one of the preceding claims, wherein said device is operable to receive update data by remote communications means from a remote host

computer and, in response, to update said script or said script lines.

5 7. A method as claimed in claim 6, including tracking in said host computer a version identifier of a script previously received by and installed in said device.

10 8. A method as claimed in claim 6, including comparing any proposed new or updated script or script lines proposed to be installed in said device with a current script or script lines, and determining whether a partial replacement is feasible or if a full replacement is required.

15 9. A method as claimed in claim 8, including updating said current script or script lines according to such a determination.

20 10. A method as claimed in claim 1, including providing said device with a plurality of scripts, each comprising one or more script lines, each comprising at least one predefined command tag.

25 11. A method as claimed in any one of the preceding claims, wherein said device is adapted for use with personal data storage devices and memory usage within said personal data storage devices is performed on a time limited basis involving:

30 using a version number encoded in each respective personal data storage device for tracking memory allocation within said respective personal data storage device; and

35 controlling the use of multiple versions of said memory allocation so that memory in said respective personal data storage device can be redefined for different use over a period of time,

wherein said version number is automatically

performed and embedded in said script lines such that said personal data storage devices are automatically updated from old to new memory allocations by said script lines.

- 5 12. A method as claimed in any one of the preceding claims, wherein said script lines can be usable in a plurality of different models of computing or electronic devices.
- 10 13. A method of implementing a loyalty program involving a method of controlling a remote computing or electronic device as claimed in any one of the preceding claims.
- 15 14. A method of implementing marketing program involving a method of controlling a remote computing or electronic device as claimed in any one of claims 1 to 12.
- 20 15. A method of implementing a loyalty and marketing program involving a method of controlling a remote computing or electronic device as claimed in any one of claims 1 to 12.
- 25 16. A method of implementing payment applications at a customer interaction device involving a method of controlling a remote computing or electronic device as claimed in any one of claims 1 to 12, wherein said remote computing or electronic device comprises said customer interaction device.
- 30 17. A method of implementing a combined payment application and loyalty program involving a method of controlling a remote computing or electronic device as claimed in any one of claims 1 to 12.
- 35 18. A method of implementing a combined payment application and marketing program involving a method of controlling a remote computing or electronic device as

claimed in any one of claims 1 to 12.

19. A method of implementing a combined payment application, loyalty program and marketing program
5 involving a method of controlling a remote computing or electronic device as claimed in any one of claims 1 to 12.

20. A computing or electronic device having an operating system and digital processor, the device comprising:
10 digital memory for receiving a script comprising one or more script lines, each script line comprising at least one predefined command tag;
a software program for interpreting said script lines;
15 wherein said software program is operable to invoke a plurality of software functions in said device and each of said command tags corresponds to a respective one of said software functions, whereby said software program can be controlled to invoke in said device one or
20 more of said software functions by means of said script.

21. A device as claimed in claim 20, wherein said script includes user variable data elements associated with said command tags so that the respective software functions can
25 be invoked in a manner dependent on said data elements.

22. A device as claimed in claim 20, wherein said digital memory is operable to receive a plurality of scripts, each comprising one or more script lines, each script line
30 comprising at least one predefined command tag, wherein said software program is controllable to control said device according to one or more of said scripts.

23. A system for generating scripts for use with the
35 method of any one of claims 1 to 19.

24. A system as claimed in claim 23, including a script administration module having a visual interface by means of which a user can input requirements, wherein said system is operable on the basis of said requirements to
5 generate a script or scripts for downloading to said software program, and interpretation and execution by means of said software program.

25. A system for testing scripts adapted for use in the
10 method of any one of claims 1 to 19, comprising a host computer provided with a version of said software program, said version being operable to interpret and execute said scripts in a virtual replica of said device, so that errors in said scripts can be detected and corrected
15 before the scripts are loaded into said device.

26. A method of composing a software application, comprising:
decomposing an intended functionality of said
20 software application;
expressing said functionality in terms of existing software functions or modules controllable by a software script engine; and
assembling a script comprising one or more script
25 lines, each script line comprising at least one predefined command tag corresponding to a respective one of said software functions, whereby said script engine can be controlled to invoke in a hardware device one or more of said software functions by means of said script, said
30 script constituting said software application.