Vista Sales Server

Software Interface Specification

Version 4

Printed on 22 July, 2009

Contents

About Vista

Vista Entertainment Solutions develops software for the Cinema Exhibition industry. The Vista software system consists of a number of integrated products that cover almost all aspects of managing and operating cinemas. The product line is scalable so as to be suitable to exhibitors who run from one cinema to hundreds of cinemas.

The Vista Point of Sale and Vista BackOffice (base Vista) provide all Cinemas level function for Box Office and Concessions. At least one installation of Base Vista is required for all Vista customers. All other modules are optional.

The optional modules are:

- Web Ticketing - a customisable system that enables ticket sales on the Internet along with display of show times and movie information.
- IVR Ticketing System- an automated touchtone phone booking system.
- Vista Kiosk - a customisable ATM ticketing system that features touch screen and state of the art multimedia technology for remote ticket sales either on or off-site.
- Call Center - provides a central web based application for booking and selling seats across a circuit of cinemas.
- MobilePOS - utilises a Pocket PC based PDA's to sell tickets and concessions while connected to the Vista system via a wireless network.
- Vista Signs - manages configured animated messages on cinema signs including LED, TV Monitors and Plasma.
- Vista Projection - controls the export of cinema show-time schedules to automated projection systems.
- Vista Air Conditioning - provides an interface between base Vista and the air conditioning system to regulate air circulation and temperature depending on head count information stored in the Vista database.
- HeadOffice - provides central maintenance of key cinema data, uploading of cinema performance data to HeadOffice, a film settlements system and a business intelligence system for analysing circuit wide performance.
- CashDesk - a companion product for Vista BackOffice for cinemas that wish to have higher levels of cash and treasury control within the cinema.
- Employee Scheduling - provides a graphical employee roster system at cinema locations, along with a HeadOffice module that consolidates all roster information.
- Film Programming and Scheduling - a companion product to HeadOffice. It is a system for planning and booking films across a circuit from a central location. The booking system generates best fit schedules to download to the cinema.
- Vouchers and Gift Cards - a companion product to Vista HeadOffice that controls the ordering, stocking, transfer, and redemption of coupons, vouchers and passes.
- Loyalty - a customer relation management program for the creation, maintenance and evaluation of loyalty programs.

Copyright Notice

Vista Entertainment Solutions Ltd
PO Box 8279, Symonds St,
Auckland, New Zealand.
Ph: +64 9 984 4570
Fax: + 64 9 379 0685
Website: http://www.vista.co.nz

Overview of Vista Sales Server

The Vista Sales Server component is an out-of process ActiveX executable which provides an interface for the ticket sales process to Vista cinema software. One is to be located on every Vista server to provide a sales service. This component insulates the client (IVR, Internet, Kiosk etc) from the specifics of the ticketing system, and defines a limited set of functionality.

While it could be located on the client application server, with database activity carried out via ODBC, locating the component on each cinema server provides the following benefits:

- it defines a specific boundary between different software providers - each work with their own hardware
- cinema software can be upgraded a cinema at a time without affecting the remote clients ability to continue processing transactions at cinemas operating various versions.

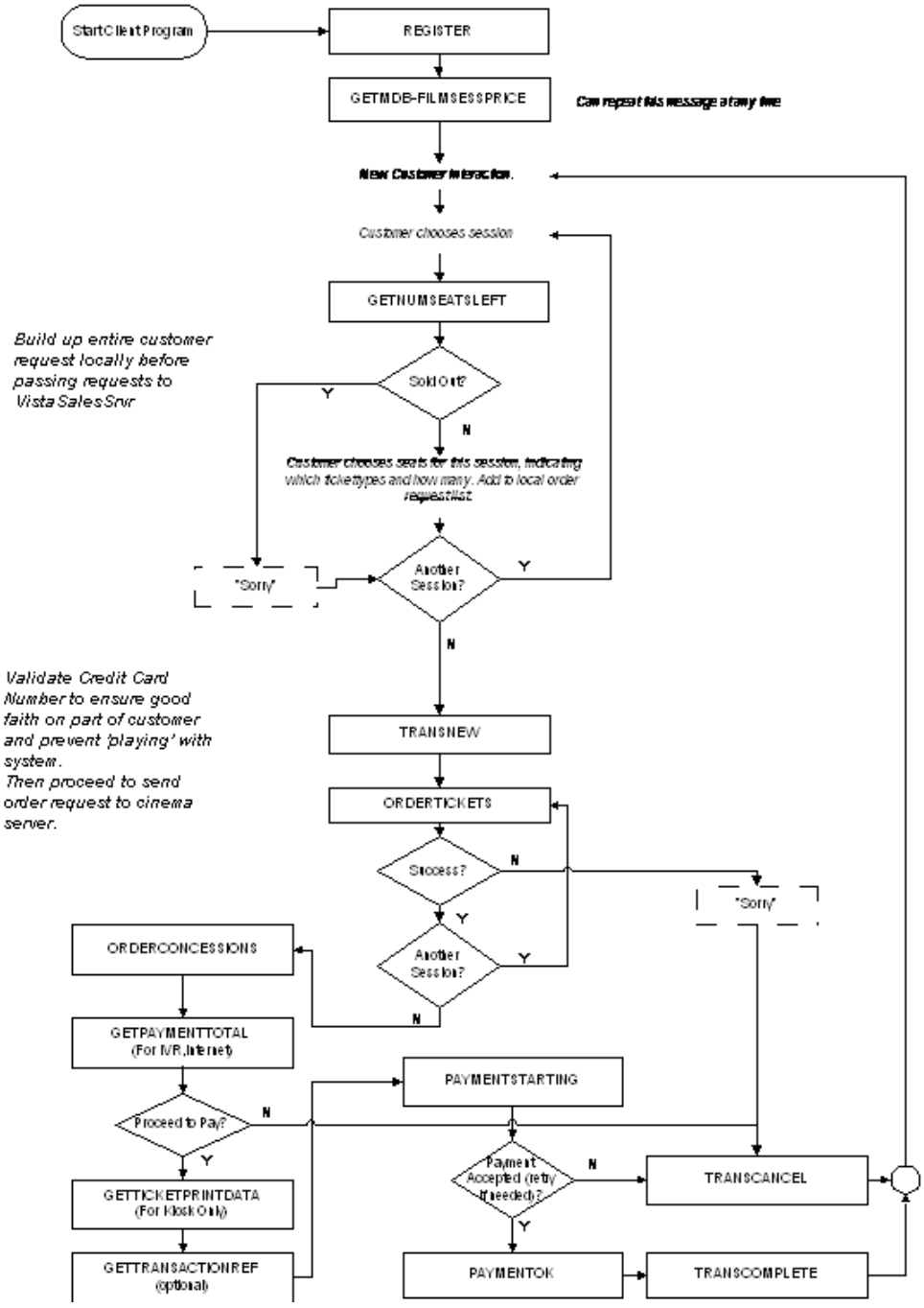
Client software communicates with Vista Sales Server via an interface of pre-defined commands. Messages are passed across the network using the DCOM protocol. The client must create a remote object reference to the Vista Sales Server Component, and utilize its external interface members.

Because a remote server could be shut down at any time, any client system must trap the error which occurs if their remote DCOM object is suddenly non-existent. They should flag the transaction status locally and when they next make contact with that cinema server should issue either a TRANSCANCEL, or a TRANSCOMPLETE if the payment process was completed before the DCOM object was lost.

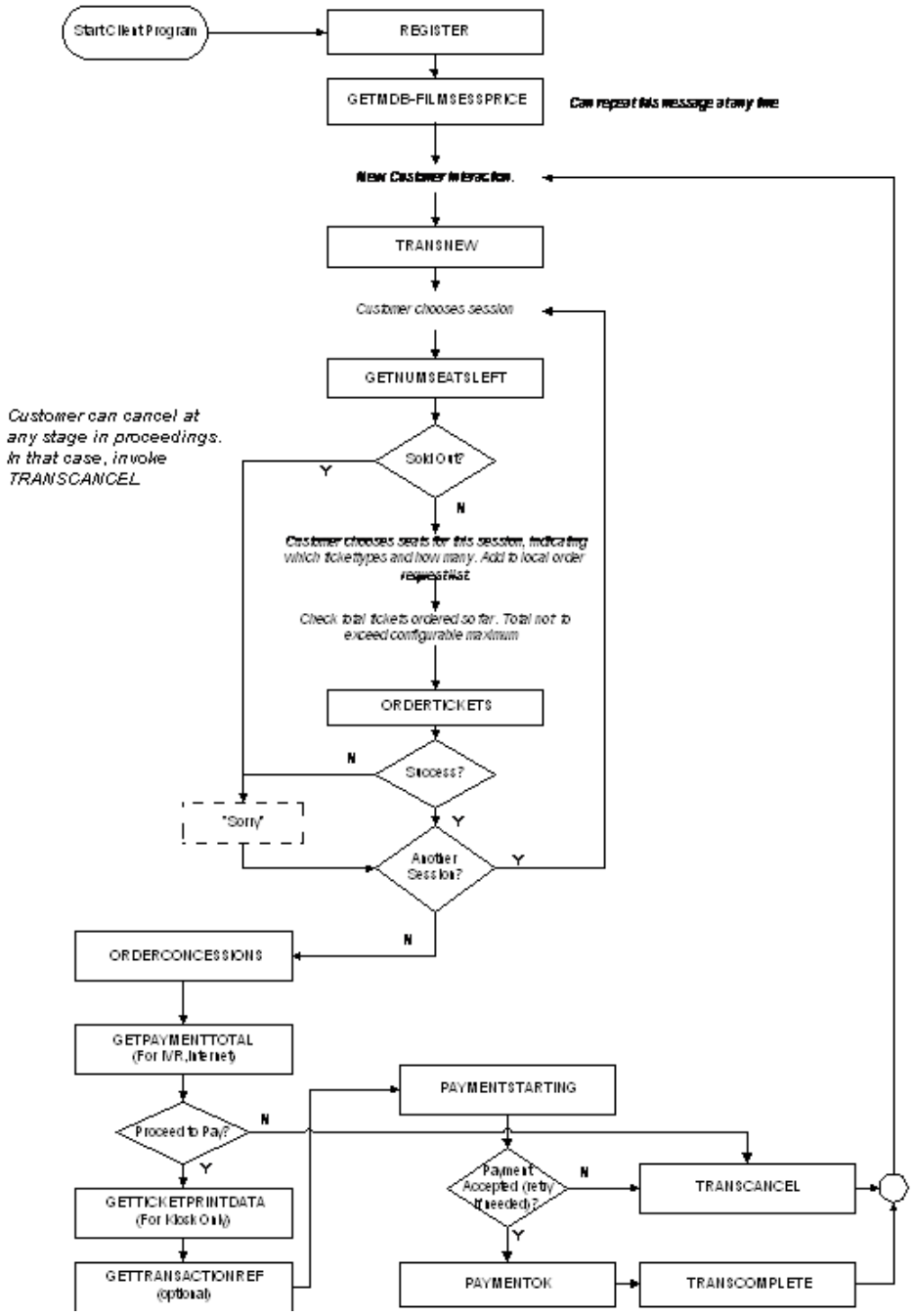
CHAPTER 1

Ticket Purchase Messaging Flow

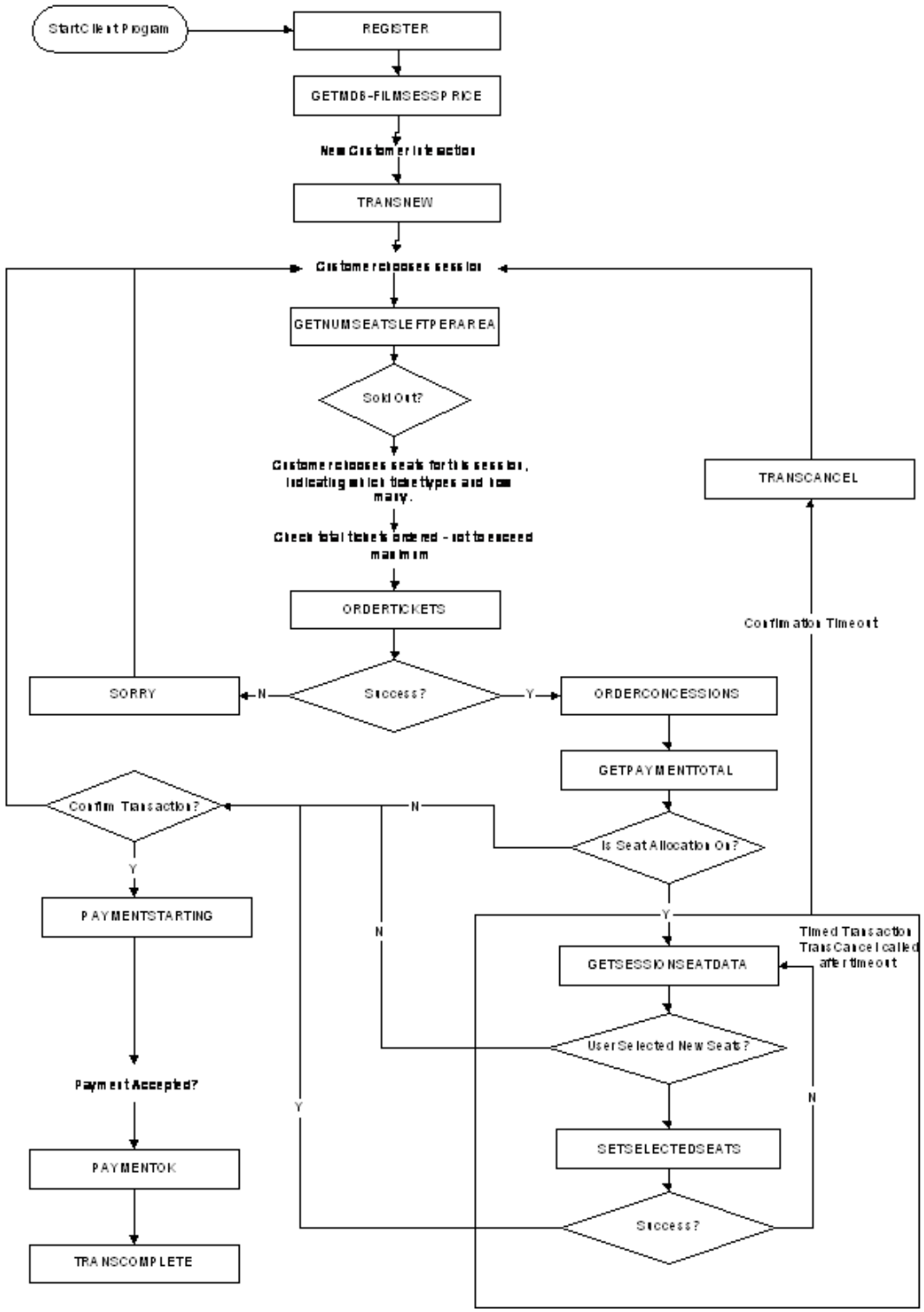
Style 1 - Pass order details after all sessions requested



Style 2 - Pass order details as each session is requested



Style 3 - Pass order details for single session incorporating Customer Selected Seating



Ticket Pickup Messaging Flow

The following are the steps required by VistaSalesSrvr for the ticket collection process:

1 *Customer arrives at kiosk and selects Ticket Pickup option*

2 *Customer swipes card; kiosk reads number*

3 Message: GetBookingsPickupPrintStream

4 *Kiosk prints tickets and vouchers*

5 Message: BookingsCollected

Note: The BookingsCollected message MUST always be sent if printing has been attempted, even if a printing error occurred. If the GetBookingsPickupPrintStream message has a problem then do not attempt to print and do not send message BookingsCollected.

General Process

Different message requests are passed to the Vista Sales Server component, and data is received back, via 3 core methods:

- CMD: Client sends a message - the message is quickly validated. A speedy response here also confirms that the network is operational.
- EXECUTE: Client instructs server to action the message - a reply is given to say whether the action was successful
- GETDATA: Client requests data if the message is one that delivers data

It is recommended that in the client side software, these 3 phases should be wrapped within a single function, with suitable error checking. See Example in the appendix.

These core methods encompass sending and receiving response for a single message or command. These 'delivery' functions have some header structure over and above the structure of the messages themselves.

The real features of the system are contained within the variety of messages that can be sent. Some messages may have 'parameters'; some may return a stream of data.

Core Object Methods

The first stage is the creation of an object in the Vista Sales Server Component.

For example, in Visual Basic:

```
Dim objMsg as Object
Set objMsg = CreateObject("VistaSalesSrvr.KioskInterface")
```

1. Cmd(,,,) : Send Message

```
ConversationId = objMsg.Cmd(<ClientId>, <TransId>,
<MessageName>[|<DatabaseName>], <msgparam1>, …,<msgparam6>)
```

e.g.

```
ConversationId = objMsg.Cmd( "130.201.177.192", "1478599", "REQ_SESSIONS",
"19970621","" ,"",..., "")
```

or (using optional Database Name Identifier)

```
objMsg.Cmd( "130.201.177.192", "1478599", "REQ_SESSIONS|VISTADB",
"19970621","" ,"",...,"")
```

Passes a message request to the server.

Parameters:

All parameters are strings.

ClientId	Globally unique name of client machine calling this function. Must be a valid TCP/IP address
TransactionId	a string, usually of digits, representing a unique number for this order
MessageName	eg "GETMDB_FilmSessPrice" [Optional: "|" *DatabaseName*]

The optional DatabaseName portion of the Message Name allows a cinema to be operating more than one Vista database on a single Server. If used this optional argument must be separated from the MessageName by a pipe delimiter "|"

param1

param2

param3

param4

param5

param6

Up to 6 params can be supplied *for the message*, which have specific interpretation based on the MessageName. If less than 6 parameters are required for a particular message, the remaining parameters must be specified as empty strings ("").

Function Return:

A *string* representing a conversation identifier (though it could be interpretable as a long integer):

>= "0" represents success, a valid identifier eg "02363210789"

An empty string ("") indicates a problem with message name, network or server.

The server will log to a diskfile any messages which are not validated.

The string returned is to be regarded as a unique conversation identifier for *this* message from this client. It will be included in any data stream returned later on, to be absolutely sure that the data response matches the particular message that requested the data.

Other clients could be given the same identifier (unlikely), but the Client name is also included in any return data string to ensure that the return data is uniquely identified overall as being associated with a particular Client and message request.

Returning it as a string keeps it in a form suitable for direct comparison with the conversation identifier included in every return data stream.

Message name is not case sensitive.

Note: If DatabaseName is ever specified as part of the MessageName argument then a database name must always be used to identify a database - during that SalesSrvr session. Once specific named database is connected to - passing a message command without a database name will generate an error as SalesSrvr will attempt to connect to database "". This behaviour is by design.

2. Execute() : Action the Message

Result = objMsg. Execute()

Actions the last message sent using the Cmd() method. This function must wait until the server finishes processing before it returns a value. A timeout value of say 30 secs is typically set at the server, so that if the action has not completed in that time it will abort and this function will exit, with some sort of success/fail code.

Function Return:

The return is an *integer*:

0	cmd executed normally
1	bad message name
2	TicketHold: asked for seats but no seats left
3	AllocateSeats: asked for seat numbers but unable to allocate them in

	a contiguous block
5	General Failure: Unexpected result (hardware or logical). Client should attempt to cancel the Transaction.

Note: for a complete list of function return codes see appendices

3. Getdata() : Get Message Response

```
Result = objMsg.GetData()
```

If the message was executed OK and the Client expects data back, do this function. This is appropriate for only a few messages which deliver data.

Eg GetSessionInfo, GetNumSeatsLeft

Function Return:

string	comprising a header portion and a message reply-data portion, described in next section
empty string ("")	means no data was intended to be returned by the message previously executed.

Message Response Format

A delimited Ascii String, which breaks the stream into logical fields. Fields and records can be of any length via this mechanism.

Default Field Delimiter | (vertical bar)

Usually an escape character is needed to cater for cases where the field delimiter is to be interpreted as data and not a delimiter. The occurrence of the escape character means that the following character is to be treated literally, and not in terms of its special function. Only the field delimiter and escape character itself need to be 'escaped' if they occur as genuine data.

In practice, a "|" is never valid data, and the 'Escaping' mechanism has not been implemented.

Interpretation of fields within a return data message, in order

Header Portion:

Message starts with a field delimiter

ClientId	TCP/IP address assigned to client machine
MessageId	The unique conversation identifier string which was returned from the obj.Cmd(…) function when the message request was first passed. It is passed back with the data to ensure matching of message and reply. eg "2370189"
MessageName	Repeat of the name of the message request eg "GETSESSIONS" uppercase… belt & braces
NoRecs	string representing the number of records (groupings of fields) that will follow eg "23". If "0", it means the final data portion is taken

	literally, and is not considered to have any fields within it.
NoFields	string representing the number of fields per record eg "4". If NoRecs is 0, this value is the length of the string that follows the delimiter "|"

Then the data portion:

if NoRecs > 0	nn delimited fields, where nn = NoRecs * NoFields Any character in these data fields which is identical to the field delimiter will be 'escaped'. Message ends with a field delimiter
OR if NoRecs = 0	a string of length <NoFields>, where everything is interpreted as data, including "|" Message DOES NOT end with a field delimiter

Special Case - Binary string data where NoRecs = 0

In effect, what this is doing is using the GetData() function as a way of transferring files from one machine to another. In the original specification, any occurrences of the field delimiter in the data portion of the GetData() string needed to be 'escaped' so that it would not be interpreted as a delimiter. *In this scenario of a large binary string, this would add processing overhead, both on the GetData() encoding end and at the Client end where the escape character would have to be stripped out. To avoid this, the special interpretation of the GetData() return string is added:*

If the NoRecs field is '0', then the NoFields field will be interpreted as a string length, and all characters following the delimiter after the NoFields field will be considered as data. The escape character will not be used as such. Every character in the data portion will be interpreted as data, and not a delimiter. Furthermore, the normal trailing field delimiter will not be added. The value in NoFields represents the length of this string, allowing a simple cross check that all characters were received.

Examples:

GetData() returns:

"|130.277.123.102|2370189|GETSESSIONS|3|4|Title|Date|Time|scn|Batman|19970624|1800|scn3|Mr Bean|19970624|1800|scn2|"

Then we have the situation of 3 records of 4 fields separated by the delimiter "|". The data portion of the return-data string is in bold

GetData() returns:

"|130.277.123.102|2370189|GET_SOME_INFO|0|12|**abc$#@|x|54x**"

We have the situation where 0 records means the data is a binary string, 12 chars(8-bit) long, and everything following the "12|" is data, in this case **abc$#@|x|54x**

Formatting of Specific Datatypes

All parameters passed with messages, and all data received from the GetData() method is in the form of strings. Where the string is to be associated with a specific datatype, the following applies:

Strings which are Interpreted as Date and/or Time

Date is yyyymmdd	eg "19970624"
Time is hhmm	eg "0500"ÀÀ4 digit, 24hr clock
DateTime is yyyymmddhhmm	eg "199706240500"

Note: No " : - /" separator symbols to be included in these strings ! The client reprocesses these data internally as required, depending on country, ...

eg As Kiosks can be multilingual inside a particular country (eg USA, both English and Spanish), the Kiosk then seems the place to determine what sort of date & time and currency to display. Otherwise the Kiosk must send formatting details to the server, and re-read session/film information each time the Kiosk user selects another language.

Strings which are interpreted as Numbers

Decimal values of any scale and precision are permitted in this message; however they must conform to the Sales Server decimal definition - regardless of the locale or regional settings in place at either the client side - or the environment in which sales server is operating. This definition enforces the following rules:

- - Only the period "." character represents a decimal separator - any other non numeric characters are ignored.
- - Only one decimal separator is permitted in a value.
- - No Thousands separators are permitted.
- - Decimal values will be stored in Vista at the precision and scale defined by the Vista Database

Strings which are interpreted as Currency

Numbers representing currencies will be passed/received as a string representing the smallest currency unit (eg cents) so that it will remain interpretable as an integer at all times. No currency symbol will be attached to such numbers.

Single Character Strings which are interpreted as Numbers

For the purpose of maximizing the data range fixed length character strings can represent a function (GetSingleChrCode) is included in Utils.bas which can convert the numbers 0 through 61 to a single character. Currently the WWW client deciphers this character into a recognizable integer. Furthermore another function (GetdblChr) can modify the resulting value to return two characters to maintain a fixed length of two.

CHAPTER 2

Messages Used in Ticket Purchase Process

The following are the definitions for messages, which may be used to carry out a ticket purchase.

Messages are passed via the obj.Cmd(, , ,...) method.

Parameters

In this context refers to parameters of the *message*, which are supplied by adding them as parameters to the obj.Cmd() method, following the messagename parameter of that function.

Data Returned

Here refers to the data-portion of the string returned from the message via the obj.GetData() function. The data portion is what remains after stripping out the header portion of the return string, up to and including the delimiter following the <NoFields> field. *See 'The Message Process, GetData()'.*

Messages are listed here alphabetically. See 'Ticket Purchase Message Flow Diagram' to see the context in which each message should be used.

GetMDB_FilmSessPrice

Purpose:

Get details of sessions which are available for sale, along with film and price details.

Use:

Whenever the client program wishes to refresh the session, film & price details it currently holds locally. This should be done on a regular basis, say every 30 mins, so that any changes to settings made at the cinema will find their way to the client.

Parameters:

IncludeZeroPrices	Y/N default N If Y zero value ticket types including complimentary and loyalty only ticket types marked as available to remote selling devices are returned with other Ticket Types.

Data Returned:

A single string (binary) representing an MDB access file (Office97 structure). This must be written to local hard disk by the client (with any *.mdb filename decided by the client) before the client can access it.

The MDB represented by the above return string contains a series of tables. Contact Vista for an example access database applicable to the latest version of Sales Srvr.

Note: The list of sessions available for sale is determined solely by what is returned within the MDB, which is determined by Vista Sales Server internally. Vista Sales Server builds this MDB based on looking ahead N days from Today(), where N is a configuration setting for remote sales which is set within Vista at each cinema (ATMDaysAdvancePurchase).

The MDB is populated on the server as follows:

- Find all OPEN sessions (available for selling), whether they have seat allocation or not, for the next N days where N is a cinema configuration parameter.
- Find all movies associated with these sessions
- Get ALL price information, whether or not the above sessions refer to it.

Each *session* has ONE pricegroup code associated with it. Referring to this code in the Price table generates the list of tickettypes available for sale for this session, and their pricings. This list is then further limited by the area categories which are valid for the session - as found in the Session Area Categories table.

Please note that prices output by Sales Server using GetFilmSessPrice are always passed Tax Inclusive - and therefore prices returned to Sales Server are also considered to be Price Inclusive.

Price books are consulted when deriving prices for Inventory items. Promo Code and Price List Code fields are populated where a price book has been used. These fields sould be returned to sales server when concession items are ordered using the optional 5 element format outlined in OrderConcessions. Using this format requires passing a “Y" as argument 2 in Order Concessions. On receipt of an Inventory order the Price books are again consulted to derive the 'best price' if clients request concessions using the -1 price flag.

The PriceBookRefreshRequired table holds a single field (PriceBookRefreshRequired) with a single value indicating the date and time that the next Price Book will become active. Clients can call GetFilmSessPrice immediately following this time trigger to retrieve updated price book data.

GetSellingDataXML[File/Stream]

Purpose:

These commands (GetSellingDataXMLFile, GetSellingDataXMLStream)

Generate an XML document representation of the data available provided in GetMDB_FilmSessPrice

Parameters:

OutputRequest	String indicating which reference data categories are required. Any combination of the following values is acceptable - each must be separated by the pipe delimiter (no leading or trailing delims): ALL | SESSIONS | FILMS | PRICES | PRICESALL | PRICESINCLZERO | CONCESSIONS | CINEMAOPERATORS | CARDDEFINITIONS | PRICEPACKAGES | ITEMALT | BOM | CONCESSIONSINCLZERO | BOMINCLZERO | ITEMALTINCLZERO | CONCESSIONSALL | POSBUTTONS | CONCESSIONBUTTONS | CONCESSIONTABS | I.e.: SESSIONS | PRICES | FILMS will only return film, session and ticket price data. Only one selection from PRICES or PRICESINCLZERO is required. If PRICESINCLZERO is passed zero value ticket types including complimentary and loyalty only ticket types are returned. Note: POSBUTTONS, CONCESSIONBUTTONS, CONCESSIONTABS information will NOT be returned if client is generally requesting “ALL" data. A client that wants this information will need to explicitly request it. i.e. “ | ALL | POSBUTTONS | " will return ALL data AND POS button layout. POSBUTTONS is new to V3R3 and only supported in version 33.0.0.8 and above CONCESSIONBUTTONS, CONCESSIONTABS are new to V3R3 and only supported in version 33.0.0.15 and above.
HoursToSelect	Optional Positive Integer Indicating how many hours into the future session and film data should be selected for. Selection start time is always calculated by Sales Server as the current time subtract the Vista configuration setting ATMTIMESHOWMOVIEAFTERSTART. Example: if the command is called at 1:30 pm, the configuration value is 20 mins and HoursToSelect is requested at 8 hours - the selection of sessions and their related films will be from 1:10pm to 9:10pm.
UseCompare	Y/N - only return data if change has occurred since Clients last request
ForceRefresh	Force a refresh even though using compare mode
CompressXML	This option forces sales server to compress the xml file or stream generated. If using a web service to connect to sales server you must be using the ExecuteXML command to use this option or the binary data will not be transmitted correctly
ClientIdlist	This is an optional field. Pipe line delimited string indicating which clients' workstation setting data sales server will retrieve. When this filed is missing, sales server will retrieve required workstation setting data for current requesting client.

	i.e. | 111.111.111.111 | 111.111.111.112 | ... When ClientIdlist is supplied: POSBUTTONS returns the POS button layout of the clients listed in this field. CONCESSIONBUTTONS returns the concession button layout of the clients listed in this field. CONCESSIONTABS returns the concession tab layout of the clients listed in this field. When ClientIdlist is not supplied: POSBUTTONS returns the POS button layout of the requesting client. CONCESSIONBUTTONS returns the concession button layout of the requesting client. CONCESSIONTABS returns the concession tab layout of the requesting client.

Return Data:

The usual header information followed by:

GetSellingDataXMLFile: The full file name and path on the SalesSrvr server where the file is saved to. As for GetMDB_FilmSessPrice the output file is overwritten with each new call in a sub-directory of SalesSrvr named /XML. The file is named as per the ClientId of the calling application - ie a ClientId of 111.111.111.111 will produce an output file named 111_111_111_111.xml

GetSellingDataXMLStream:

The string returned following the ReturnData header is the complete xml file stream.

Xml structure / format:

Contact Vista for the latest selling data schema available for the version of Sales Srvr being targeted. Vista will continue to support all published schemas but reserves the right to add additional elements and data to the schema as required.

All dates are formatted as YYYYMMDDHHNNSS

All numeric values are converted to strings

All Price values are recorded as whole numbers in cents

In this method pipe characters (“ | ") in the response stream are NOT expected and will generate a parse error.

GetSessionDisplayData

Purpose:

Returns an xml string containing full details of session information based on a set of accepted format styles and limited by a set of defined filters.

Parameters:

Format String indicating output format. ALL (default) = All session data returned; COUNTS = Session Id, and seat counts info only.

Filter Delimited string containing an optional set of filters to apply (see below)

Comments:

The Format parameter indicates the field list, and ordering algorithm used.

Currently recognised Format values are:

All	all session fields, + a limited list of film and screen, session type fields are returned. results are ordered by Session_dtmRealShow, Screen_strName (contact vista for latest schema)
COUNTS	Session_lngSessionId + a list of counts related to seat information are returned. Results are ordered by Session_lngSessionId. (contact vista for latest schema)

The Filter parameter is a delimited string containing an optional list of filter values. Currently recognised filter values are:

DATESTART	datetime - (yyyymmddhhmm), inclusive date from which to start session selection, should always be specified to avoid returning all session records! Date is filtered by the real showing time (Session_dtmRealShow)
DATEEND	datetime - (yyyymmddhhmm), inclusive date to end session selection, should always be specified to avoid returning all session records! Date is filtered by the real showing time (Session_dtmRealShow)
MAXRECORDS	integer - maximum number of records to return, records selected are dependant on the order clause used, which is defined by the Format selected. Zero(default) indicates no filter
ATMONLY	Y/N - only return atm available sessions
OPENONLY	Y/N - only return open sessions

e.g. | DATESTART | 200501010600 | DATEEND | 200501020600 | OPENONLY | Y | MAXRECORDS | 0 | ATMONLY | N |

Return Data:

The usual header information followed by a xml payload of session data. Root node = "VSS_SessionDisplayData", each session element delimited by the <s> tag, child nodes contain a list of relevant fields from the Vista session, film, and screen tables. This field list will continue to grow as new fields are added to the Vista session table.

Xml structure / format:

All dates are formatted as YYYYMMDDHHNNSS

All numeric values are converted to strings

All string values contain the pipe (" | ") character are replaced with the string "^~^"

Contact Vista for the latest selling data schema available for the version of Sales Srvr being targeted. Vista will continue to support all published schemas but reserves the right to add additional elements and data to the schema as required. Date is output to xml using the standard Sales Srvr xml string format, for example:

```
<VSS_SessionDisplayData>
  <s>
    <Session_lngSessionId>10303</Session_lngSessionId>
    <Film_strTitle>The Ferris Wheel Movie</Film_strTitle>
  </s>
</VSS_SessionDisplayData>
```

GetNumSeatsLeftPerArea

Purpose:

This returns number of seats available to a session per area category.

Parameters:

SessionId:	string of digits representing session id e.g. "21785"

Return Data:

The usual header information:	…. | N | 2 (N rows of 2 columns)

The data following the header

Example:

(…header…) | A | 7 | T | 57 | 0000000002 | 11 | 0000000005 | 0 | 0000000001 | 46 |

A	special code indicating that the number which follows represents the number of tickets AVAILABLE for this session that can be sold by the remote client device. This is the number to be used by kiosk, ivr, web etc. It differs from 'T' below because the cinema owner has specified that the last 50 seats in the cinema cannot be sold by remote devices.
7	7 tickets max. can be sold to this session.
T	special code indicating that the number which follows represents the TOTAL number of seats left in the session. (This is intended to help with advanced messaging to the user, especially for the call centre system, such as "there may be tickets available at the venue".)
57	57 seats are left at the cinema
0000000002	The number that follows is the number of seats remaining in the Area Category which has this code
11	11 seats left in Area category 0000000002
0000000005	The number that follows is the number of seats remaining in the Area Category which has this code
0	0 seats left in Area category 0000000005
0000000001	The number that follows is the number of seats remaining in the Area Category which has this code

46	46 seats left in Area category 0000000001

Comments:

The return data string can have any number of area categories, depending on how the cinema operator has set up his system. Note that there is always at least one area category listed.

From this return data you can also infer that the only area categories which can be sold for THIS session are "0000000002", "0000000005", "0000000001".

All devices should use the number held in A (meaning Available) to determine if any tickets can be sold to the session. If it is 0, then there are none available for sale via remote devices, though there could be tickets still available at the venue. Devices should use the number held in A to validate the number of tickets selected by the user and not let them proceed to the next stage if they try to select more tickets than are available

The "T" value is ONLY used to assist with advanced messaging - it is not used to decide if seats are available for sale. The sum of the seats in each area category balances the number held in T. If, say, T has a value say 36, and if the cinema operator has specified that the last 50 seats can only be sold at the venue, then A will be 0 and there are no more tickets available for sale via a remote device.For a complete handling of messaging:

If A = 0 and T = 0, say "There are no more seats available to this session" (or equivalent)

If A=0 and T > 0, say "There are no more seats available for purchase to this session via this system. There may be tickets available at the venue" (or equivalent meaning)

GetPaymentTotal

Purpose:

To calculate the total value of the current order in progress

Use:

Used prior to starting the payment process, so that customer can be informed of total cost for them to choose whether to continue the transaction or not.

Parameters (in order):

BookingFeeScheme	legacy, unused, not supported
IncludeTax	Y/N - default N. Include detailed tax data in output stream
IncludeLoyaltyPoints	Y/N - default N. Include loyalty points totals in output stream

Data Returned:

One record with 4 columns:

- Total value of order in cents currency unit e.g. 2435 = $24.35
- Value of Tickets in the order
- Value of Food in the Order
- Value of BookingFee in the order

If IncludeTax = Y	1 record with 4 columns (above) + 16 Tax columns ▪ BookingFeeTax1 ▪ BookingFeeTax2 ▪ BookingFeeTax3 ▪ BookingFeeTax4 ▪ TicketTax1 ▪ TicketTax2 ▪ TicketTax3 ▪ TicketTax4 ▪ ConcessionTax1 ▪ ConcessionTax2 ▪ ConcessionTax3 ▪ ConcessionTax4 ▪ TotalTax1 ▪ TotalTax2 ▪ TotalTax3 ▪ TotalTax4
If IncludeLoyaltyPoints = Y	1 record with 4 columns (above) + [16 Tax columns (if IncludeTax = 'Y')] + 3 LoyaltyPoints columns (note: these are decimal fields represented as above in cents - i.e 230 = 2.3 points, 100 = 1 point): ▪ TicketPoints

	▪ ConcessionPoints ▪ BookingFeePoints

Comment:

During this process each ticket is individually priced according to pricing held in the ticketing server and is completely 'up to date' with any price changes.

Vista also includes the cost of any food order, plus booking fee which it calculates based on the details of the order. The client device need not know how to calculate it. It should calculate food and tickets and total it internally.

- The IncludeTax parameter is supported from v3 sp03 build f
- The IncludeLoyaltyPoints parameter is supported from v3 sp04

GetTicketPrintData

Purpose:

Produce a print stream ready to send to printer to produce tickets, already formatted to the style of the cinema, and with embedded printer commands suitable for the model of printer installed in the client device.

Use:

KIOSK only.

Should only be called once, and prior to starting eftpos payment. It includes tickets for all seats to all sessions.

Parameters (in order):

PrinterType	eg "KTX" as set in Kiosk's INI file. Valid names are negotiated as required
ReleasePrintFlag	see GetBookingsPickupPrintStream
IncludeReceipt	Optional Y/N, default N. If set to Y and a receipt.txt template file is located in the printer directory the receipt print stream is included. This call should be made AFTER payments have been confirmed via the PaymentOK command if a receipt is required.
CollectionVoucher	Optional Y/N, default N. If set to Y the print stream is assumed to be used for informational purposes or a collection voucher only, not for actually printing physical tickets. This is useful if a data stream printer/print template is used to return order details which are displayed to the end user - or printed to a booking voucher, which will be presented by the customer to produce actual tickets at the cinema. In such cases a 'Booking' barcode is included in the print stream rather than 'Ticket' barcodes.

Data Returned:

N records of one column each, where n is number of tickets, and the column holds:

- A print stream ready for direct printing, containing print instructions for a *single* ticket, including cutter command.

In this way, the kiosk can know from the number of records how many tickets should be printed, and can print them individually, checking each time that the ticket was successfully printed. Thus a kiosk can determine if any of the tickets were not printed.

Comments:

While the kiosk typically asks for the print streams prior to payment (to minimise time between payment approved and finalising transaction within Vista, and to minimise customer waiting time for printing), it only sends them to the printer after payment is successful.

If tickets are printed successfully, the kiosk must wrap up the order with a TransComplete message (passing the number of tickets printed).

If tickets are not ALL produced successfully, the kiosk MUST still follow it with a TransComplete message, passing the number of tickets actually printed, before shutting itself down.

Ticket Numbering is applied to orders just prior to ticket printing. This applies to both Orders being processed by the client - and for Booked Ticket Pickups (eg at kiosk). This minimizes ticket number loss. Any transactions being processed which have ticket numbers generated for them cannot be permanently removed. The ticket numbers are crucial audit items. Therefore calling Trans Cancel for an order which has had ticket numbers generated for it will return success - however the transaction will not be removed from the system.

The PrinterType parameter is used by Sales Server to identify the \PrintTemplates\ <printername> directory to look for print templates. The PrinterType must match to sub directory accessible to Sales Server. Templates can be included for tickets (must exist), a single combined food voucher, and a customer receipt (ticket.txt, voucher.txt and receipt.txt respectively). The receipt.txt template is optional - if it is not found, the related print streams are not included in the return data.

It is possible to create Print Templates which return data fields in a simple formatted string, allowing the client application to retrieve a stream of data about the order which can be used render to the customer display, or feed into another ticket printing system for output. For example you create a PrinterType 'HTMLOutput' - by creating a directory \PrintTemplates\HTMLOutput which contained a ticket.txt and a voucher.txt file you could retrieve an HTML formatted stream of order details using this command. Contact Vista for specific instructions on authoring print templates for Vista.

If you wish to render the information for display only, or for creating booking vouchers used for pickup at the cinema you Must specify the NotForPrint parameter as 'Y' - this ensures the booking can be processed, and later picked up at the cinema.

GetFiscalReceiptPrintStream

[Available from v3r1 - 31.0.0.50 or greater]

Purpose:

Produce a receipt print stream for fiscal devices.

Use:

Used if the client application is connected to a fiscal printer

Parameters (in order):

PrinterType	
IncludeTickets	Y/N - if the print stream should include ticket data
IsCashPayment	Y/N - if the payment tendered by the customer was a cash payment - used by some fiscal systems

Data Returned:

1 record of 1 column which holds:

A print stream ready for direct printing, containing print instructions for a fiscal receipt

Comments:

This call should be made AFTER payments have been confirmed via the PaymentOK command if a fiscal receipt is required.

GetReceiptPrintStream

[Available from v3r1 - 30.5.26 or greater]

Purpose:

Produce a receipt print stream ready to send to printer. This receipt print stream is also included in the general print stream returned from GetTicketPrintData if a valid receipt.txt template file exists in the printer directory.

Use:

Useful as an alternative to GetTicketPrintData if ONLY the receipt is required, i.e. tickets for entering the cinema are not produced.

Parameters (in order):

PrinterType	e.g. "KTX" as set in Kiosk's INI file. Valid names are negotiated as required

Return codes:

43	Receipt.txt template file not found in requested printer directory.

Data Returned:

N records of one column each, where n is number of print items, and the column holds:

- A print stream ready for direct printing, containing print instructions for a receipt, including cutter command.

Comments:

This call should be made AFTER payments have been confirmed via the PaymentOK command if a receipt is required.

GetTransactionRef

Purpose:

Pass a Vista internal reference for this transaction back to client so it can print it on an eftpos receipt or on credit card statement if required. This reference can be cross checked in the Vista system in case of disputes.

Use:

KIOSK only. (IVR for credit card statement ?)

So kiosk could in principle print it at bottom of eftpos receipt to allow easier resolution of customer complaints in the case of printing failures, as it would then be possible to find the sale at the counter and reprint (credit/resell, or exchange) the tickets.

Use after customer commits to going to payment, but just prior to starting the payment process with the payment agency.

Parameters (in order):

UnpaidBooking	Y/N/UC Is this an unpaid booking. If an empty string is passed the Vista system setting VSS_IsBookingPaid value is read and honoured. This system setting is Yes by default (all bookings paid). If the client app explicitly passes this parameter the system setting is ignored. 'UC' indicates the booking is Unpaid, but Confirmed (see comments).

Data Returned:

Single string including a Vista Final Transaction Number (not the temporary one used for the order).

Comments:

Unpaid Bookings must include the UnpaidBooking = Y (or UC) argument - this ensures that Transaction records are not incorrectly generated even though payment has not been made for the order. UC = Unpaid Confirmed, these bookings will NOT be released if the customer does not pickup their tickets prior to session show time. This argument should ONLY be used by client applications which are authorised to create these bookings, as generally a surcharge of some form will be charged at booking time by the client device, independently of Vista software. *This feature available from v3r0 sp04.*

The UpaidBooking parameter is settable via GetTransactionRef, GetTramsactionRefEx, PaymentOK, PaymentOKEx - with each of these calls the UnpaidBooking value overwrites any previous value set, allowing the client application to modify its status as required, but also requiring that the client application set the value explicitly with each/any of these calls.

OrderConcessionsEx

Purpose:

Allow a client POS to pass a food order to the Vista server. This command extends the OrderConcessions command by introducing a new, more flexible and extensible format for passing concession order information to sales server. This command is introduced from version 3.01 of the Sales Server system (file version 31.x.x)

Parameters:

ConcessionDetailXML	see below
CancelExisting	optional Y/N default N: Removes any concessions already in the order first. See CancelConcessions for more details. [Available from v3r2 - 32.0.0.42]
ExplicitDelivery	Optional Y/N (default N). Only set delivery on the item if delivery information has been explicitely set. [Available from Release: v3r3sp01]

Return Data:

None

Parameters Defined:

ConcessionDetailXML:

Note: see the VSS_OrderConcessionsEx.xsd file for the complete schema definition

This parameter holds a XML document node of a defined schema This schema includes a root node named <order> which in turn contains a sequence of concession nodes named <conc>. Each concession node includes a sequence of nodes defining the standard concession ordering parameters, and can also include nested nodes of concession under two group nodes - features and bomoptions. See comments below.

Comments:

Ordering features:

The features group nested within a concessions node includes a sequence of concession items which have been purchased as additional 'features' added to the parent concession. To select features the parent concession MUST have a requested quantity of 1.

Ordering optional items:

Some concessions in the vista system are 'virtual' or 'optional' - the concession cannot be purchased but is a parent item that has a list of concessions that can be optionally be selected in its place. When ordering a concession with such a definition the client application must request the selected item, and include the <optparentitemid> element which defines what the optional parent item was.

Ordering optional recipe items:

Some recipes in the vista system include 'virtual' components as described above within the recipe. Again the 'virtual' component itself cannot be purchased and a valid optional item must be selected in its place. When ordering a concession with such a recipe the client application is responsible for ensuring that the all of the recipes optional components have been selected. The bomoptions group nested within the concession node contains this data represented as a sequence of concession items. Each concession node within the bomoptions group must include an element containing the BOM level 1 item id <bomlevelitemid>, and an element containing the optional parent item id optparentitemid> from which the selected concession was derived from. Combined with the selected itemid these fields make up the unique recipe key that allows Sales Server to map a selected bomoption to its related recipe record. Sales Server will validate that all required optional recipe components have been selected before accepting a concessions order.

Note: refer to the schema and command definition for GetSellingDataXML for information on retrieving concession selling reference data from sales server.

PaymentStarting

Purpose:

Tells the Vista server that customer billing is about to start (whether by Credit card or eftpos)

Replaces older message 'EftposPaid', which is retained for backward compatibility with existing kiosks.

Use:

Call just prior to starting the billing process

Parameters (in order):

Amount of Payment in smallest currency unit (cents)

Parameter 2 not used

If paying a tab the tab code [Available from Release: v3r3sp01]

Data Returned:

None

Comments:

VistaSalesSrvr crosschecks the amount against what it has recorded for the order, just like 'Eftposstarting', after taking booking fee into account. If for some reason the amount to be billed does not match what is stated in the current order, the messaging Execute() method returns a Fail (5).

PaymentRequest

Purpose:

Perform Remote Payment Services via VistaPayment at the Cinema.

Use:

To action Payment Requests at the cinema rather than using a local Payment Service.

Parameters (in order):

Request Type	As defined by Vista Payment component request types
Application ID	Must exist in the target cinemas tblPaymentModuleConfig table
Request String	A delimited string containing all request parameters as required by the Vista Payment component. This string is an exact replica of that which would be passed to the Vista Payment component if the component were being called directly.

Data Returned:

Request Return as defined by Vista Payment module

Comments:

Consult the Vista Payment Module documentation for complete details of Request Input and Output parameters required

PaymentOK

Purpose:

Tell Vista payment was confirmed, and in the case of credit cards tell Vista the details of the card number which made the booking, to assist with collection of tickets later at the cinema. Replaces the older message 'EftposPaid', which is retained for backward compatibility with existing kiosks.

Use:

When the payment process has been completed OK with the payment agency. If Payment was rejected or was otherwise not completed (timed out) then this message should not be used; in that case the TransCancel message should be used.

To register payment actions for one or more tender types (multiple payment actions for a single transaction) - use PaymentOKEx

Parameters (in order):

Card Type	eg: "EFTPOS", "VISA", "MASTERCARD" - see comments
CardNo	Card no of Credit card eg "4453789256789123"
Expiry	yyyymm card expiry eg "200509"
CustomerName	Booking name for pickup at VistaPOS
UnpaidBooking	Y/N/UC Is this an unpaid booking. If an empty string is passed the Vista system setting VSS_IsBookingPaid value is read and honoured. This system setting is Yes by default (all bookings paid). If the client app explicitly passes a Y/N flag the system setting is ignored. 'UC' indicates the booking is Unpaid, but Confirmed (see comments).
CustomerPhone	Phone number for pickup at VistaPOS

Data Returned:

None

Return Codes:

15	Unpaid Bookings Closed For at least one session in the orde

Comments:

As of Sales Server 7.105.x Unpaid Bookings are supported where authorised. It is the client applications responsibility for ensure that unpaid bookings are legitimate. A card number is still required for booking pickup from POS. Unpaid bookings from remote clients are only available to clients other than Kiosk and PDA. GetBookingsPickupPrintStream will only return PAID bookings - thus at time of writing Unpaid Bookings cannot be picked up Kiosk or PDA applications. Cinema Side rules may exist which restrict Unpaid Bookings from being made. Return Codes from PaymentOK will identify when these rules have been violated and Unpaid Bookings have not been accepted.

The CardType parameter has been extended to allow the client application to specify more card information in a predefined format using the " | " delimiter. Parameters can be omitted by leaving an empty dtring in the placeholder. The format for extended card number data follows:

CardType | TenderCategory | PaymentTypeCode

Examples:

tender category known	"VISA | CREDIT"
payment type known, tender category not	"VISA | | R | "
known data	"AMEX | CREDIT | R | "

UnpaidBooking = UC. Unpaid Confirmed, these bookings will NOT be released if the customer does not pickup their tickets prior to session show time. This argument should ONLY be used by client applications which are authorised to create these bookings, as generally a surcharge of some form will be charged at booking time by the client device, independently of Vista software. This feature available from v3r0 sp04.

The UpaidBooking parameter is settable via GetTransactionRef, GetTramsactionRefEx, PaymentOK, PaymentOKEx - with each of these calls the UnpaidBooking value overwrites any previous value set, allowing the client application to modify its status as required, but also requiring that the client application set the value explicitly with each/any of these calls.

PaymentOKEx

Purpose:

Indicates to Vista that payment was confirmed. This version accepts a freeform parameter list and can be used to perform multiple tender transactions for a single order.

Note: Event if payment is being authorised using PaymentRequest - PaymentStarting and PaymentOK are required calls. Also for UnpaidBookings these commands are required - as they indicate Card and customer details used for booking pickup etc. Also internally these commands set flags allowing the order to be finalised using TransComplete.

Use:

When the payment process has been completed OK with the payment agency. If Payment was rejected or was otherwise not completed (timed out) then this message should not be used; in that case the TransCancel message should be used.

Created in v3sp01 - to support multiple tender type transactions. For each tender transaction a separate call should be made to the PaymentOKEx command.

Parameters (in order):

PaymentAmount	standard whole number in cents
PaymentData	see comments
UnpaidBooking	Y/N/UC - see PaymentOK comments
RedemptionsAllowed	Y/N/UC - Redemption tickets allowed in the order (default is no)

Data Returned:

None

Return Codes:

15	Unpaid Bookings Closed For at least one session in the order
74	Debtor tab limit has been exceeded

Comments:

The PaymentData parameter is a name/value pair list of parameters allowing for the inclusion of many parameters now - and further extension in the future. The format is a delimited string. This parameter maps closely to the format and parameter names as the PaymentData parameter used in the ActionRefundTrans command.

Additional parameters are available with this function which may be ignored when calling the ActionRefundTrans command. Element list (names must be adhered to):

- PAYTYPECODE (Vista PayType_strTypeCode to refund to - not required)

- PAYMENTCATEGORY - (Tender Category - used for Payment Type lookup if PayTypeCode is not passed)
- CARDNUMBER (use 'CASH' for card number if refunding to cash)
- CARDTYPE (required if refunding to a card)
- CARDNAME (Used for reference purposes)
- CARDEXPIRY (not required)
- CUSTPHONE
- BOOKINGREF - Y/N. Y (default) indicates that the details of this card (including name, phone number etc - should be used for Booking identification in Vista. As Y is the default value if a client app does not specify this element the last tender type details passed to PaymentOKEx will be used for Booking reference)
- REFNO - stored in the reference number field in tblTrans_Cash
- REFDETAIL - stored in the reference detail field in tblTrans_Cash
- AUTHCODE - payment authorisation code - stored in Payment Trans Ref field in tblTransCash
- BIZPARTNER - indicates the BUSINESS_PARTNER_CODE if returned from Vista Payment. For partial payments this value will override the BIZPARTNER assigned to the whole order (see TransNew COMMAND), but only for this individual payment line.
- TABCODE - the tab code as returned from CreateTab command - should NOT be used for Unpaid Bookings, tab or debtor payments are considered PAID transactions.

Examples of the Refund Payment Data string:

| CARDNUMBER | 4555..1234 | CARDTYPE | VISA | CARDNAME | GSMITH | PAYMENTCATEGORY | CREDIT |

| PAYTYPECODE | C | CARDNUMBER | CASH | CARDNAME | Grant Smith | BOOKINGREF | N |

PaymentVoidEx

[Available from Release: v3r3]

Purpose:

Used to remove a payment from the order (after PaymentOK has been called).

Parameters (in order):

PaymentAmount	standard whole number in cents
PaymentData	see comments

Data Returned:

None

Return Codes:

5	Unexpected Error

Comments:

The PaymentData parameter is a name/value pair list of parameters allowing for the inclusion of many parameters now - and further extension in the future. The format is a delimited string. This parameter maps closely to the format and parameter names as the PaymentData parameter used in the ActionRefundTrans command.

Additional parameters are available with this function which may be ignored when calling the ActionRefundTrans command. Element list (names must be adhered to):

- PAYTYPECODE (Vista PayType_strTypeCode to refund to - not required)
- PAYMENTCATEGORY - (Tender Category - used for Payment Type lookup if PayTypeCode is not passed)
- CARDNUMBER (use 'CASH' for card number if refunding to cash)
- CARDTYPE (required if refunding to a card)
- CARDNAME (Used for reference purposes)
- CARDEXPIRY (not required)
- CUSTPHONE
- BOOKINGREF - Y/N. Y (default) indicates that the details of this card (including name, phone number etc - should be used for Booking identification in Vista. As Y is the default value if a client app does not specify this element the last tender type details passed to PaymentOKEx will be used for Booking reference)
- REFNO - stored in the reference number field in tblTrans_Cash
- REFDETAIL - stored in the reference detail field in tblTrans_Cash
- AUTHCODE - payment authorisation code - stored in Payment Trans Ref field in tblTransCash
- BIZPARTNER - indicates the BUSINESS_PARTNER_CODE if returned from Vista Payment. For partial payments this value will override the BIZPARTNER assigned to the whole order (see TransNew COMMAND), but only for this individual payment line.

- TABCODE - the tab code as returned from CreateTab command - should NOT be used for Unpaid Bookings, tab or debtor payments are considered PAID transactions.

Examples of the Refund Payment Data string:

| CARDNUMBER | 4555..1234 | CARDTYPE | VISA | CARDNAME | GSMITH | PAYMENTCATEGORY | CREDIT |

| PAYTYPECODE | C | CARDNUMBER | CASH | CARDNAME | Grant Smith | BOOKINGREF | N |

GetTransactionRefEx

Comments:

As per GetTransactionRef but extended to include Vista Booking Number and Transaction Number sequentially after string (delimited by pipes)

i.e.

Boston 1456/27 June 2003 | 1456 | 99876 | BookingId |

BookingRef | BookingNumber | TransactionNumber | BookingId |

[BookingId only available in version 33.0.0 or above]

Parameters (in order):

UnpaidBooking	Y/N/UC Is this an unpaid booking. If an empty string is passed the Vista system setting VSS_IsBookingPaid value is read and honoured. This system setting is Yes by default (all bookings paid). If the client app explicitly passes this parameter the system setting is ignored. 'UC' indicates the booking is Unpaid, but Confirmed (see comments).
RedemptionsAllowed	Y/N/UC - Redemption tickets allowed in the order (default is no)

OrderTickets

Purpose:

This message orders tickets AND allocates seats if appropriate. All the ticket requirements for a session are sent in a single message. That means that seat allocation can all be handled internally without the need for explicit client requests.

For Package ticket types Sales Server will divide the package up into constituent Tickets and Inventory items and order each element individually.

Parameters:

SessionId	
SessionDateTime	
TicketDetailsList	see below
UserSelectedSeating,	Y/N/S Optional (default = N - is customer selected seating available from this client (see notes below) In v3r2 (32.0.0.20 or above) it also supports multi-values separated with delimiter. For more detail see notes below
TicketPackageDefinitio n	[Available from v3 sp04]
TicketDetailsListFormat	(string) this parameter allows for ongoing modifications to the construction of the Ticket Details List parameter, an empty string (or no parameter) indicates that the default format described below is used. Other supported formats are discussed below also [Available from v3 sp04]

Return Data:

number of seats left	
number of seats in largest free block	has a value of 0, except if seat allocation fails, in which case it is non-zero
approved order detail	Member provider can override ticket price. In this case the approved price will be different from the price specified in TicketDetailList. Please refer to VSS_ApprovedOrderDetail.xsd for the complete schema definition

Return codes:

2	not enough seats left (existing code)
3	unable to allocate seats in a contiguous block (new)

13	unable to order the concession components of a ticket package ticket type
14	the price passed to Sales Server for a Ticket Package Ticket Type ordered does not conform to the total value of elements in the ticket package as calculated in Order Tickets
15	Unpaid Bookings are no longer permitted for this session
41	The Session is not valid for this client applications Sales Channel
45	The Voucher is not valid.
80	Unable to initialise Vista Member Manager
81	Member Manager preload member data failed
83	Member Manger ticket order validation failed
85	Unknown Member Manager erro

Comments:

This passes everything for a session in one bundle, which means no separate seat alloc logic needed at the client side. Can also be more efficient for network traffic.

This message will take care of seat allocation internally, if seat allocation is required, without any explicit knowledge by the client. Except that one of the error codes from executing this message may contain the fact that it failed due to being unable to get seats together.

Ticket Details List:

All tickets required for the requested session (SessionId parameter) are included in this definition list.

- All definitions begin with the number N of different ticket types in the order list, followed by the appropriate content definition.
- All definitions use the " | " as a field separator.
- The list always starts with a delimiter and ends with a delimiter.
- Prices are always listed in cents (base 100)
- Decimal values MUST adhere to the Sales Server decimal definition
- -1 indicates use cinema price

a If TicketDetailsListFormat parameter is empty (default):

Number of tickets in the list followed by N triples of

TicketCode | Qty | PriceEach |

i.e. " | N | ticketcode | qty | price | ticketcode | qty | price | ..."

e.g. " | 2 | 0001 | 2 | 1100 | 0002 | 1 | 800 | "

- 2 ticket types two of type code 0001 at $11.00 each, one of type code 0002 at $8.00 each , 3 tickets in total

Note: Version 33.0.34 and above supports surcharge on redemption tickets. Client application can specify a surcharge cost in ticket detail list by over writing the price filed of ticket detail list with surcharge value. Surcharge value becomes the selling price and redemption value of the ticket is surcharge plus ticket price.

b If TicketDetailsListFormat parameter = "L"

Indicates loyalty recognition data is included. Number of tickets in the list followed by N chunks of 6. Use empty placeholders where recognition data is not relevant to the ticket type.

i.e. " | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | ..."

e.g." | 2 | 0001 | 1 | 1100 | 1101512 | 7785 | 342.1 | 0002 | 1 | 800 | | | | "

Additional loyalty parameters:

balancetypeId - loyalty specific balance type id: string(10)

recognitionId - loyalty specific recognition id: string(10)

pointscost - loyalty specific points cost of recognition: decimal (see Strings as Numbers definition)

c If TicketDetailsListFormat parameter = "LB"

As above for "L" with the addition of a field for Loyalty points conversion rate for booking fees (bfeeconvrate). If non empty then the value in this field (decimal as pointscost above) is applied to any booking fee value for the ticket(s). The Vista booking fee logic is applied as normal to all tickets in the order. Those tickets which were requested with a bfeeconvrate have their booking fee converted to Loyalty points, using the conversion rate supplied - and the dollar value of the fee is set to zero. Then the booking fee minimum maximum rules are applied to the dollar value booking fees only. This parameter allows the loyalty system to offer Vista booking fees using points, while maintaining support for the complex booking fee system implemented in Vista. If used the client application should re-check that a members points balance has not been exceeded by adding the total booking fee points in the order to the other points used before finalising the transaction. This can be retrieved from the GetPaymentTotal function.

i.e. " | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | bfeeconvrate | bfeerecognitionid | ..."

d If TicketDetailsListFormat parameter = "LBC"

As above for LB with the addition of an additional loyalty member card number in position 9, and loyalty member name in position 10

" | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | bfeeconvrate | bfeerecognitionid | addmembercard | addmembername | ..."

e If TicketDetailsListFormat parameter = "M"

As above for LBC with additional member provider name in position 11, member card number in position 12 and member data of birth in position 13 in YYYYMMDD format

" | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | bfeeconvrate | bfeerecognitionid | addmembercard | addmembername | providername | providermembercard | memberDOB | ..."

f If TicketDetailsListFormat parameter = "MP"

As above for LBC with additional member provider name in position 11, member card number in position 12 and member data of birth in position 13 in YYYYMMDD format and Provider override price in position 14

" | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | bfeeconvrate | bfeerecognitionid | addmembercard | addmembername | providername | providermembercard | memberDOB | ProviderPrice | ..."

g If TicketDetailsListFormat parameter = "V"

As above for MP with additional voucher barcode in position 15 (note: member manager data is not processed, to use both vouchers and member manager, use "VM" below)

" | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | bfeeconvrate | bfeerecognitionid | addmembercard | addmembername | providername | providermembercard | memberDOB | ProviderPrice | VoucherBarcode | ..."

h If TicketDetailsListFormat parameter = "VM"

As above for MP with additional voucher barcode in position 15

" | N | ticketcode | qty | price | recognitionId | balancetypeId | pointscost | bfeeconvrate | bfeerecognitionid | addmembercard | addmembername | providername | providermembercard | memberDOB | ProviderPrice | VoucherBarcode | …"

Ticket Package Definition:

The format of the TicketPackageDefintion parameter follows:

This optional parameter contains a distinct list of each ticket package ticket type include in the TicketDetailsList parameter. If passed Sales Server will honor the prices and sales tax codes defined in this last, rather than consulting the cinema reference tables. Allowing Sales Server to accept the total package price as passed by the calling application, even if that price differs from the total package price, as calculated from the cinema reference tables. Differences between the prices may arise if changes are made to the package definitions at the cinema, and the remote client application has not refreshed its reference data definitions.

- All definitions begin with the number N of different ticket types in the order list, followed by the appropriate content definition.
- All definitions use the " | " as a field separator.
- The list always starts with a delimiter and ends with a delimiter.
- Prices are always listed in cents (base 100)

Format: N = Number of element definitions - there will generally be more than one element per parent package - N describes the total number of elements in all packages in the list. Followed by N chunks of 6 fields

i.e. " | N | ParentTicketCode | ElementType | ElementId | ElementQty | ElementPriceEach | ElementSalesTaxCode | ParentTicketCode | ElementType…"

e.g. " | 2 | 0101 | I | 113 | 2 | 100 | S | 0101 | T | 0001 | 2 | 1100 | X | "

- 2 package elements from a single parent package, one concession item (ElementType ="I") ItemId = "113" at $1.00 with a sales tax code "S". Two tickets (ElementType ="T") of type code 0001 at $11.00 each with a sales tax code "X".

Return value of max number of seats in a single block:

This is the second number in the data return string. This is to assist where enough seats were held OK but they were unable to be allocated in a single block. In that case, return this number so the client POS can in principle tell the customer the maximum they can get, so customer can decide whether to order a smaller number of seats.

Since determining this value is a little intensive on processing time, this number will only be determined when seat Allocation fails. If seats are allocated OK to a session, this number will always be set to "0". If a session does not have seat allocation ON, the number returned will be "0".

Choosing same session twice:

If the user selects a second session which happens to be identical to the first, the Client POS should treat this as if these are 2 different sessions and pass the ticket requests in 2 "chunks". Doing this would make it possible, for someone who knows the system, if they cannot get 8 seats in a block to try to do 2 passes of 4 seats each and succeed. (This is only a factor if allocated seating is ON - and Customer Selected Seating is not.)

Customer Selected Seating:

If the client application supports Customer Selected Seating include the UserSelectedSeating = 'Y' parameter. Where this parameter is set to Y Sales Server will not cancel orders requiring seat allocation - if seats cannot be allocated in a single block (return code 3). This allows the client application to let the customer select their own seats as required. Be aware that if this flag is set to Y the responsibility of cancelling these transactions if user selected seating is not invoked or used for the particular transaction in question lies with the client application.

Alternatively as of v3r1 sp01 (31.1.0.22 or greater) - the client application can specify an 'S' indicating sales server should skip the auto seat allocation process. It is then required for the client application to then specify which seats will be taken using SetSelectedSeats.

In v3r2 (32.00.20 or greater) - client application can also request seats with starting seat number. The seat number will be used as the location where the best seat searching should begins. In this case the seating format is pipelined values and should always start and end with delimiter.

i.e. | SeatingOptions | RequireSeatNumberTranslation | Row | Column |

Field 1	Seat allocation options (Y/N/S) Optional (default = N - is customer selected seating available from this client (see previous comment)
Field 2	specifies whether row and column ids are used or the physical row and column numbers/names are used. When physical names are used, Sales server will translate the row and column IDs before using it.
Field 3	specifies seat ROW i.e. M
Field 4	specifies seat COLUMN i.e 19

Examples:

" | Y | Y | M | 19 | "	Sales server will translate this seat number to corresponding seat id. This is used when client application doesn't have the reference to seat row and column IDs.
" | Y | N | 7 | 12 | "	The given row and column number are the seat row and column IDs

Ticket Packages:

When ordering Ticket Package ticket types the client should only order the Parent Ticket Type items - Sales Server will process the package internally. However this means that the value of the Parent Ticket Package must conform to the total value of all elements in the package - as calculated by Sales Server during the order creation process. Sales Server will return an error code of 14 if this exception is generated. At this point the Client should cancel the order - refresh reference data through GetFilmSessPrice - and redisplay the corrected Ticket Type prices and structure as appropriate.

Alternatively as of v3 sp03 - the client application can specify the package ticket structure that was used at sale time in parameter 5. This allows for changes in Package ticket content without failing the order. See Ticket Package Definition above.

OrderConcessions

Purpose:

Allow a client POS to pass a food order to the Vista server.

Parameters:

ConcessionDetailList	see below
UsePriceBooks	optional Y/N default N
ConcessionDetailListFormat	optional default empty (see below) [Available from v3 sp04]

Return Data:

None

Concession Details List:

All concession items required for the order are included in this definition list.

- All definitions begin with the number N of different concession types in the order list, followed by the appropriate content definition.
- All definitions use the " | " as a field separator.
- The list always starts with a delimiter and ends with a delimiter.
- Prices are always listed in cents (base 100)
- Decimal values MUST adhere to the Sales Server decimal definition
- -1 indicates use cinema price

a If ConcessionDetailListFormat parameter is empty (default) AND UsePriceBooks <> Y:

Number of item types in the list followed by N triples of

ItemCode | Qty | PriceEach |

i.e. " | N | itemcode | qty | price | itemcode | qty | price..."

e.g. " | 2 | 113 | 2 | 100 | 180 | 1 | 350 | "

- 2 item types two of type code 113 at $1.00 each, one of type code 180 at $3.50 each , 3 items in total

b If ConcessionDetailListFormat parameter is empty (default) AND UsePriceBooks = Y:

Number of item types in the list followed by N chunks of 5

ItemCode | Qty | PriceEach |

i.e. " | N | itemcode | qty | price | promocode | pricelistcode | itemcode | qty | price..."

e.g. " | 2 | 113 | 2 | 100 | ABC | ZZ | 180 | 1 | 350 | ABC | ZZ | "

- 2 item types two of type code 113 at $1.00 each, one of type code 180 at $3.50 each , 3 items in total. Both with a promo code of "ABC" and a price list code of "ZZ"

c If ConcessionDetailListFormat parameter = "L" (UsePriceBooks ALWAYS forced to be Y at sales server)

Indicates loyalty recognition data is included. UsePriceBooks is forced to Y here therefore forcing the format string to be N number of tickets in the list followed by N chunks of 8. Use empty placeholders where recognition data is not relevant to the ticket type.

i.e. " | N | itemcode | qty | price | promocode | pricelistcode | recognitionId | balancetypeId | pointscost | "

e.g. " | 2 | 113 | 2 | 100 | ABC | ZZ | 12313 | 1221 | 1.23 | 180 | 1 | 350 | ABC | ZZ | | | | "

- 2 item types two of type code 113 at $1.00 each with a loyalty recognition id of 12313, a balance type of "1221" and a points cost of 1.23, one of type code 180 at $3.50 each , 3 items in total. Both with a promo code of "ABC" and a price list code of "ZZ"

Additional loyalty parameters:

balancetypeId	loyalty specific balance type id: string(10)
recognitionId	loyalty specific recognition id: string(10)
pointscost	loyalty specific points cost of recognition: decimal (see Strings as Numbers definition)

Price Books:

As of Vista 2000 v2 release 8, Concession items are priced using Price Books - to process items priced using a price book correctly Sales Server requires the Promo Code and Price List Code associated with a price (this data is included in the output .mdb file) Backward compatibility is maintained for clients - however for those clients aware of price book pricing the Details List is of the second format shown above. Including this format requires passing a "Y" as parameter 2.

For Clients who pass a -1 as the price per item Price Books are consulted when Sales Server calculates the total value of the order. For clients who pass a price back to Sales Server these prices are always honored. Please note that prices output by Sales Server using GetFilmSessPrice are always passed Tax Inclusive - and therefore prices returned to Sales Server are also considered to be Price Inclusive.

See GetFilmSessPrice to refer to Price Book data output.

Comments:

Modified to include Cinema Operator logic, new Tax calculations, improved BOM recording and stock location data Nov 2001

LoyaltyRequest

Purpose:

Perform Remote Loyalty Services via Vista Loyalty Module at the Cinema.

[Available from v3 sp04]

Use:

To action Loyalty Requests at the cinema rather than using a local service.

Parameters (in order):

Request Type	As defined by Vista Loyalty request types
CardType	Optional (default blank) provides support for multiple loyalty systems operating at a single cinema site. Not currently implemented
LoyaltySettings	Optional (default blank) allows the client application to specify the Loyalty Module Configuration settings to be used.
RequestParams	Required. A delimited string containing all request parameters as required by the Vista Loyalty component.
DataItemNodes	Optional (default blank). A string containing a well-formed xml message containing Loyalty DataItem xml nodes (these must conform to the Vista Loyalty message schema). These nodes are appended to the Loyalty message generated by the Loyalty System. This parameter provides support for complex request message data.

Data Returned:

Two fields are returned in the return data string:

ReturnCode	this is the Vista Loyalty return value (integer) returned directly from the loyalty system. This value does NOT indicate any particular message was validated. The client app must consult the response message to determine if a message was 'accepted' by the loyalty system. This value - if less than zero indicates that the loyalty subsystem encountered a technical issuing processing the request. Consult the 'Vista Loyalty Connector Guide' for a breakdown of error codes and their meaning/location.
ResponseMsg	The Vista loyalty system returns complex data structures and therefore this response is preserved in its original xml format. Consult the 'Vista Loyalty Connector Guide' for a complete description of how to interpret this message. Optionally the visLoyaltyInterpreter can be used to parse these messages. Consult Vista for more detail.

Comments:

LoyaltyRequest is a 'shared' function - it does not require an instance of an order be generated before being able to call the function. Therefore the TransIdTemp header parameter can be empty. However the ClientId parameter is required as normal.

LoyaltySettings parameter:

If a client app includes these settings in their configuration system explicitly specifying them here will ensure the Loyalty Module honours these settings. This allows for remote client apps to connect to a cinema server without the need to configure loyalty settings for each device in the loyalty configuration table at the cinema. The format for this parameter is a semi-colon separated string with name value pairs linked by the equals character ("="). This format allows for the embedded inclusion of pipe (" | ") delimited parameter values within the string as shown below.

i.e. "setting1=abc;setting2=xyz;setting3= | servicesetting1=s1value | sevicesetting2=s2value | "

The parameters recognised within this format map directly to the Field names of the Vista tblLoyaltyModuleConfig table. Apart from the '"LMC_strConnectorProgId ' and LMC_strUseLoyalLink fields (always required) - the loyalty connector determines which fields must be populated. A description of these fields can be found in the 'Vista Loyalty Connector Guide' document which describes the API for interacting with Vista Loyalty.

e.g. "LMC_strConnectorProgId=visLoyalty.LoyaltyRequest; LMC_strUseLoyalLink =Y;LMC_strConnectorServer=myserver;LMC_strServiceSettings1= | SvcVersion=abc.11 | URL=http://theserver.hostsvc.aspx | Timeout=30 | "

Note: If LoyaltySettings are NOT passed, the Loyalty system will consult its configuration table for settings. Sales Server passes the ClientId parameter to the Vista Loyalty Module as the 'ClientId', and the Client Class (Sales Channel) as the 'ApplicationId' - using these values an administrator can configure device specific loyalty settings in the loyalty configuration table if required, or leverage the loyalty configuration 'default' mechanism to tailor settings to sales channels etc. Consult the 'Vista Loyalty Connector Guide' for more detail.

RequestParams parameter:

This is a standard delimited string parameter containing all relevant RequestParams to be passed to the Loyalty module. For a complete description of RequestParams required for any message consult the 'Vista Loyalty Connector Guide' document

e.g. " | CardNumber | 1299874hi9 | MemberName | Gerald Ford | "

DataItemNodes parameter:

For complex request messages the DataItemNodes parameter can be used, however the rules defining this message structure and content are well defined and MUST be adhered to for a message to be processed correctly. For this reason this implementation is currently restricted to use by authorised client applications. For more detail contact Vista.

Register

Purpose:

Tell the Vista Sales Server what type of client it is, what printer, etc

Use:

Whenever the client program is restarted and connects to a cinema for the first time after that. It doesn't matter how many times this is done but once a day or less is sufficient. It is REQUIRED before anything else happens whenever the client connects to a particular cinema VistaSalesServer for the very first time.

Parameters:

ClientLocation	string(30) Physical location of Kiosk eg "Broadway #1"
ClientModel	string(30) freeform as per client requirements
ClientEndOfDayTime	hhmm e.g. 0100
ClientClockDateTime	yyyyMMddhhmm e.g. 199811210950 by which server can determine clock offset
PrinterType	string(30) allows for "NONE"
ClientType	string(10) Valid values: " IVR", "WWW", "KIOSK*", "PDA","CALL"

Data Returned:

None

Comments:

In the message header (the leading parameters in the Cmd(,,,) object method), transaction number is irrelevant. It should be passed as an empty string, but there is no problem if any other transaction number is passed with this message.

None of the above parameters can be blank.

If no printer is attached, as in the case with IVR and Internet, pass "NONE" for the printer type. If a printer is attached, the string must be a valid string, agreed between client developer and Vista Sales Server developer.

Day end time is particularly relevant for kiosks and eftpos. If it is not relevant to a particular client (eg IVR or Internet), this time must be passed as "0000".

ClientLocation and ClientModel must be supplied. These are used in reporting.

KioskClockDateTime and KioskEndOfDayTime *could* have relevance later to assist with producing any reports that will be required from the Vista system which will allow a reconciliation of the EFTPOS bank statements sent to Head Office. Vista reports must break at the same times as the Kiosk end-of-day time.

As at file version 7.1105.x if the client passes the string 'MULTITIMEZONE' in the KioskClockDateTime field the validation check performed on Kiosk datetime matching server datetime is NOT performed. Clients operating across time zones should utilise this approach.

As at release version 3.1 the ClientLocation parameter can be extended to include a unique workstation id. To extend the parameter include a pipe character (" | "). The workstation id is a 10 character string designed to provide a circuit-wide unique code for each remote client device. The client application and/or customer administration is responsible for ensuring this code is unique and correct for each device. No 2 devices should ever use the same code. If supplied the value is used as the Vista Workstation code when creating a workstation entry.

RegisterPDA

Purpose:

An extension to the Register command. Designed to allow PDA clients to Logon Users to the PDA workstations. Also allows/requires PDAs to use a fully configured Workstation record from the Vista Workstation Table. Thus enabling the use of configured Workstation Profile buttons for Concession Sales.

Use:

Whenever a new user logs onto the PDA workstation. Method resets the Workstation and Kiosk table records to use the new User ID

Parameters:

ClientLocation	string(30) Physical location of PDA eg "FrontHouse"
WorkstationId	Actual WorkstationId as configured in Vista Back Office. Each PDA should use a unique ClientID, and a unique WorkstationID to enable correct assignment for sales reporting and POS Session data.
ClientEndOfDayTime	hhmm eg "0100"
ClientClockDateTime	yyyyMMddhhmm eg "199811210950" (by which server can determine clock offset)
UserID	Logon details - User ID
UserPIN	Logon details - User PIN

Data Returned:

None

Return Codes:

50	Logon Details not validated
51	Workstation record not found
52	WorkstationID exists with a different ClientID
53	ClientID exists with a different WorkstationID
54	User Logged on at another Workstation

Comments:

This command can also be called using the command name 'LogOnUser'.

The ClientID, WorkstationID combination must remain unique to avoid errors in order allocation create by Client applications. Therefore if for any reason the ClientID/WorkstationID combination requires reconfiguring amongst PDA units the System administrator will need to manually delete the existing PDA records from tblKiosk to reset its state to allow the allocation of new combinations in the RegisterPDA method.

In the message header (the leading parameters in the Cmd(,,,) object method), transaction number is irrelevant. It should be passed as an empty string, but there is no problem if any other transaction number is passed with this message.

None of the above parameters can be blank.

Printer, ClientModel is derived from the Workstation Table - not passed to this command as is the case for Register. Client Type is always assumed to be "PDA"

Day end can be passed as "0000" .

Client Location is used in reporting.

KioskClockDateTime and KioskEndOfDayTime *could* have relevance later to assist with producing any reports that will be required from the Vista system which will allow a reconciliation of the EFTPOS bank statements sent to Head Office. Vista reports must break at the same times as the Kiosk end-of-day time.

RegisterOfflinePOS

Purpose:

An extension to the Register command. Designed to allow offline processing engines from manned POS clients to register and post transactions using a known Vista User/Workstation.

Use:

When processing offline transactions via Sales Server. Should be called for each new user session

Parameters:

ClientLocation	string(30) Physical location of PDA eg "FrontHouse"
WorkstationId	Actual WorkstationId as configured in Vista Back Office. Each PDA should use a unique ClientID, and a unique WorkstationID to enable correct assignment for sales reporting and POS Session data.
ClientEndOfDayTime	hhmm eg "0100"
ClientClockDateTime	yyyyMMddhhmm eg "199811210950" (by which server can determine clock offset)
UserID	Logon details - User ID

Data Returned:

None

Return Codes:

50	User does not exist
51	Workstation record not found
52	WorkstationID exists with a different ClientID
53	ClientID exists with a different WorkstationID

Comments:

This command can also be called using the command name 'LogOnOfflinePOS'.

The ClientID, WorkstationID combination must remain unique to avoid errors in order allocation create by Client applications.

In the message header (the leading parameters in the Cmd(,,,) object method), transaction number is irrelevant. It should be passed as an empty string, but there is no problem if any other transaction number is passed with this message.

None of the above parameters can be blank.

Printer, ClientModel is derived from the Workstation Table - not passed to this command as is the case for Register. Client Type is always assumed to be "PDA"

Day end can be passed as "0000".

Client Location is used in reporting.

LogOffUser

Purpose:

Allows PDA clients to Logoff Users to the PDA workstations.

Use:

Whenever a new user logs onto the PDA workstation or the Workstation is shutdown the current user should first log off.

Parameters:

UserId	(len <=30) User ID Number
WorkstationID	(len <=30)

Data Returned:

None

Command Specific Return Codes:

None

Comments:

None

LogOffOfflinePOS

Purpose:

Allows Offline POS processing engines to Logoff Users - allowing the clientid to be reused for another user.

Use:

Call whenever a transaction processing for a user session is complete to clear the user state data from the Sales Server state store.

Parameters:

UserId	(len <=30) User ID Number
WorkstationID	(len <=30)

Data Returned:

None

Command Specific Return Codes:

None

Comments:

None

TicketCancel

Purpose:

To cancel *all* reserved seats to a *specified session* for the current transaction(order). Similar to TransCancel, but cancels only a selected few seats rather than the entire order.

Use:

If customer is purchasing seats to more than one session in the same order, and the seats have been held successfully for the first session, but they can only get some of the seats to the second session or seat numbers can't be allocated to the second session in a block, then seats to the second session can be cancelled without losing the earlier part of the order.

Previously TransCancel has been used for order cancellation, requiring an order be rebuilt by the client machine. But that runs the risk of not being able to re-reserve seats to the first session if that session is popular and if there were already few seats remaining. In this case, the customer will lose his seats to the first session when they had previously been held successfully.

TransCancel works successfully on client applications which do not attempt to reserve any seats until the customer has entered the entire order for all sessions. Since 98% of sales are to a single session, very few customers *might* be inconvenienced if the order was restarted and they lost their seats to the first session.

Parameters (in order):

Session_ID	eg "2173"Ą(Primary key from Session table)

Data Returned:

None

Comments:

Remember that the transaction number for the current order is *always* passed in the header portion accompanying the message.

TransCancel

Purpose:

Abort the entire transaction - cancel ALL seats that have been held.

Use:

- If the customer failed to complete the payment transaction (eg eftpos, credit card) successfully
- walked away and Kiosk timed out
- in the case of IVR, hung up the phone or didnt hang up and was considered to have timed out

Parameters (in order):

None

(Transaction Number is included as usual in message header)

Data Returned:

None

Comments:

Modified by GPS 15 Nov - If the transaction Id passed to the function does not exist in the database the function returns as execution successful. This has been modified to account for the fact that the VSSTransSweeper.dll is being run as a scheduled job and may run TransCancel on transactions in the background. A remote client could conceivably attempt to cancel the same transaction later if a lag period has existed on the remote UI.

Any transactions being processed which have ticket numbers generated for them cannot be permanently removed. The ticket numbers are crucial audit items. Therefore calling Trans Cancel for an order which has had ticket numbers generated for it will return success - however the transaction will not be removed from the Order tables - booked tickets are also not reversed.

TransComplete

Purpose:

Indicate that the current transaction has been finished with the sale confirmed (payment has been made).

Use:

Although the order is technically considered complete once payment has been acknowledged, this message confirms the success of printing and the overall transaction. Use after tickets have been printed.

Parameters (in order):

Number of Tickets	Number of tickets plus food vouchers printed. eg "3" For KIOSK: Pass number of tickets and vouchers actually counted as being printed succ For IVR and Internet: Tickets are not printed, pass number of tickets as "0"
Don't Commit Vouchers	Set "Y" to not commit any vouchers in the order. VoucherCommit must be called separat

Data Returned:

None

Comments:

When this message is executed it will return a 'fail' if the number of tickets is not correct, or if any of the final order processing steps could not be completed. If possible it is advisable that the client application keep a record of any failed transactions that occur after the customer has been charged to allow for the identification of genuine orders requiring a manual refund.

Message Specific Return Codes:

55	Loyalty updates failed, transaction commit aborted. Note: Transaction is NOT automatically cancelled by Sales Server, to retain an Audit trail of failed transactions.

TransContinue

Purpose:

Reset any timeout counters associated with the current transaction.

Use:

Some installations may include a background cleanup agent which deletes any orders not completed within a specific time period. If a customer is genuinely processing an order the client device can utilize this message to restart the timeout counter.

Parameters (in order):

None (Transaction Number is always passed in message header portion)

Data Returned:

None

TransNew

Purpose:

Initiate a new order. The Vista server generates a temporary transaction number for the client device to reference the order. This transaction number must be supplied in the header portion of every subsequent message relating to this transaction

Use:

At the start of every new order

Parameters (in order):

AdditionalData	(optional default = "") Pipe delimited string containing additional transaction fields (see comments)
ExistingBookingNo	(optional default = "") used internally by Vista applications
ExistingTransNo	(optional default = "") used internally by Vista applications
ActivatePOSS	Y/N - (optional default N) used internally by Vista applications. If set to Y POS Sessions created or logged on when performing SalesSrvr transaction are marked as 'Active' - not 'Closed' (default)

Data Returned:

Temporary Transaction number	(string interpretable as a number) eg "200000000231"

An error is indicated if the string returned is empty, in addition to the normal error traps in the Execute() method.

Comments:

The header portion of this message should pass an empty transaction number ("").

Parameter1 'AdditionalData' is a pipe delimited string which can optionally contain additional transaction information.

Currently accepted fields [as at V3 r 0 sp04/30.5.27 or greater]

CustName	(string 50) - overrideable in the PaymentOK/ PaymentOKEx commands
CustPhone	(string 50) - overrideable in the PaymentOK/ PaymentOKEx commands
BizPartner	(string 20)

e.g. | CustName | John H. Smith | BizPartner | CityMall |

This method includes some parameters reserved by Vista applications which are not useful to third party applications. Pass empty string values to accept defaults.

CancelUnpaidBooking

Purpose:

Releases all data for an 'Unpaid' booking, including seat allocation information and session counts.

Use:

Release an Unpaid Booking.

Parameters (in order):

Booking Number	(ClientId, and TransIdTemp are contained in the standard header)
BookingBarcode	(Optional field. If this field is supplied, booking number field can be omitted)

Data Returned:

None

Comments:

None

Return codes:

Execute() may return the following errors:

17	Booking Does Not Exist
18	Booking Not Unpaid
19	Booking Cancelled
20	Session Unavailable

GetRefundTransValue

Purpose:

Returns:

a The total refundable value of a paid order (or part of an order).

b a list of all vista transaction sequence numbers viable for Refund.

Use:

This method allows the client application to verify that an order can be refunded and which elements are refundable. The total refundable value is returned enabling the client application to action a refund payment transaction to the customer. If a list of requested sequence numbers is passed this may be altered to include any elements which must be refunded together (i.e. ticket package items) and/or exclude any elements which cannot be refunded (i.e. they may already be refunded or are excluded due to the other argument flags passed)

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Vista Transaction Number	Long Integer
Transaction Sequence List	Delimited string in normal format (requested individual transaction elements to refund)
RefundBookingFee	Y/N
RefundInventory	Y/N
PriorDayRefund	Y/N - Default is N. Indicates a refund is "prior day" - ie ignore usual refund time limit. [Available from Release: v3r3sp01]

Data Returned:

TotalOrderValue	in cents
Return Sequence List	updated to reflect actual sequence numbers that will be refunded due to cinema or client rules

Comments:

An example of a Sequence list:

| 1 | 3 | 5 | 6 | 10 |

Use the Refund Inventory flag when refunding orders which have yet to be picked up.

Or where the concession items are returned.

Note: however that concession items which are part of package tickets must always be refunded when the package is refunded

Message specific return codes:

Execute() may return the following errors:

20	No Transaction Lines exist which are refundable
22	If a Booking Exists it is Cancelled
23	If a Booking Exists it is Unpaid
24	If a Booking Exists it is already picked up

ActionRefundTrans

Purpose:

Refund a paid Transaction or part of a transaction

Returns:

a Total Refunded value

b A list of all vista transaction sequence numbers actually Refunded.

Use:

This command performs a Refund. Again the sequence list may be adjusted and will be returned by Sales Server according to cinema or client requirements for refunds.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Vista Transaction Number	Long Integer
Transaction Sequence List	Delimited string in normal format (requested individual transaction elements to refund)
RefundBookingFee	Y/N
RefundInventory	Y/N
Total Refund Value	cents
PaymentData	see comments

Data Returned:

TotalOrderValue	in cents
Return Sequence List	updated to reflect actual sequence numbers that will be refunded due to cinema or client rules

Comments:

The total Refund Value passed MUST match the total refund value Sales Server calculates prior to performing the refund. This check ensures that the actual value the client application has refunded to the customer is correct.

The PaymentData parameter is a delimited string containing a combination of the following named elements (element names must be adhered to):

- PAYTYPECODE (Vista PayType_strTypeCode to refund to - not required)
- CARDNUMBER (use 'CASH' for card number if refunding to cash)

- CARDTYPE (required if refunding to a card)
- CARDNAME (Used for reference purposes)
- CARDEXPIRY (not required)

Examples of the Refund Payment Data string:

| CARDNUMBER | 455512456781234 | CARDTYPE | VISA | CARDNAME | GSMITH | - credit card refund, default credit card payment type will be used.

| PAYTYPECODE | C | CARDNUMBER | CASH | CARDNAME | Grant Smith | - cash refund (eg PDA).

For refunds to credit or debit card the card number and card type data is required.

If a pay type code is passed it must be a valid active payment type in the cinema database

If pay type code is not passed Sales Server will derive the default code from other information passed - e.g. if a CardNumber of 'CASH' is passed sales server will locate the cash payment type in the cinema database, otherwise the default credit card payment type will be used.

Use the Refund Inventory flag when refunding orders which have yet to be picked up.

Or where the concession items are returned. Note however that concession items which

are part of package tickets must always be refunded when the package is refunded

The SetMemberInfo command can be used prior to refunding a transaction if loyalty member details are known (see SetMemberInfo).

Message specific return codes:

Execute() may return the following errors:

20	No Transaction Lines exist which are refundable
22	If a Booking Exists it is Cancelled
23	If a Booking Exists it is Unpaid
24	If a Booking Exists it is already picked up

ActionPriorDayRefundTrans

[Available from Release: v3r3sp01]

Purpose:

Refund a paid prior day Transaction or part of a transaction

Returns:

a Total Refunded value

b A list of all vista transaction sequence numbers actually Refunded.

Use:

This command performs a Refund. Again the sequence list may be adjusted and will be returned by Sales Server according to cinema or client requirements for refunds.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Vista Transaction Number	Long Integer
Transaction Sequence List	Delimited string in normal format (requested individual transaction elements to refund)
RefundBookingFee	Y/N
RefundInventory	Y/N
Total Refund Value	cents
PaymentData	see comments

Data Returned:

TotalOrderValue	in cents
Return Sequence List	updated to reflect actual sequence numbers that will be refunded due to cinema or client rules

Comments:

The total Refund Value passed MUST match the total refund value Sales Server calculates prior to performing the refund. This check ensures that the actual value the client application has refunded to the customer is correct.

The PaymentData parameter is a delimited string containing a combination of the following named elements (element names must be adhered to):

- PAYTYPECODE (Vista PayType_strTypeCode to refund to - not required)

- CARDNUMBER (use 'CASH' for card number if refunding to cash)
- CARDTYPE (required if refunding to a card)
- CARDNAME (Used for reference purposes)
- CARDEXPIRY (not required)

Examples of the Refund Payment Data string:

| CARDNUMBER | 455512456781234 | CARDTYPE | VISA | CARDNAME | GSMITH | - credit card refund, default credit card payment type will be used.

| PAYTYPECODE | C | CARDNUMBER | CASH | CARDNAME | Grant Smith | - cash refund (eg PDA).

For refunds to credit or debit card the card number and card type data is required.

If a pay type code is passed it must be a valid active payment type in the cinema database

If pay type code is not passed Sales Server will derive the default code from other information passed - e.g. if a CardNumber of 'CASH' is passed sales server will locate the cash payment type in the cinema database, otherwise the default credit card payment type will be used.

Use the Refund Inventory flag when refunding orders which have yet to be picked up.

Or where the concession items are returned. Note however that concession items which

are part of package tickets must always be refunded when the package is refunded

The SetMemberInfo command can be used prior to refunding a transaction if loyalty member details are known (see SetMemberInfo).

Message specific return codes:

Execute() may return the following errors:

20	No Transaction Lines exist which are refundable
22	If a Booking Exists it is Cancelled
23	If a Booking Exists it is Unpaid

AddSurcharge

Purpose:

Add client defined booking fees (surcharge) to an order. Calling this method overrides the Vista cinema booking fee system - and cinema booking fees are NOT applied to the order. This allows the client app to specify a series of surcharge values and quantities oto tickets in the order. If required the client can assign specific values to specific ticket types in the order - but NOT specific tickets.

Use:

Call this command after all tickets have been ordered for a transaction, and before calling GetPaymentTotal. If tickets are removed or new tickets are added after this command has been called re-calling it will drop all existing booking fees and recreate them based on the parameter contents. Thus the parameter list must always include ALL surcharges for ALL tickets in the order.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Surcharge Details List	Delimited string (see below)
NoFieldsPerItem	Number of fields in each Surcharge Detail List item in parameter 1

Data Returned:

None

Comments:

In this release 2 different Surcharge Detail List (param1) formats are supported, which one is used is determined by the NoFieldsPerItem parameter (param2). For each surcharge item with different details the field list is repeated. The first value in the field list indicates the total number of different surcharge items in the detail list. If only one surcharge item is to be used only one set of fields need be included.

Format 1: | NoDetails | Value | Qty | Value | Qty | … NoFieldsPerItem = 2

Format 2: | NoDetails | Value | Qty | TicketType | Value | Qty | TicketType | … NoFieldsPerItem = 3

Examples:

a client application has ordered 2 adult tickets and one child, and wishes to apply a surcharge of $1.00 to each ticket: param1 = " | 1 | 100 | 3 | " param2 = "2"

b client application has ordered 2 adult tickets and wishes to apply a single surcharge of $2.50 to one of the tickets: param1 = " | 1 | 250 | 1 | " param2 = "2"

Note: SalesSrvr decides which ticket to add the surcharge to.

c client application has order 2 adult tickets (tickettype = 0001) and one child (ticket type = 0004) and wishes to apply a $1.00 surcharge to each adult ticket and a $0.50 surcharge to the child ticket: param1 = " | 2 | 100 | 2 | 0001 | 50 | 1 | 0004 | " param2 = "3".

Message specific return codes:

Execute() may return the following errors:

4	Invalid input data - the NoFields parameter does not match the number of fields calculated in the Surcharge Detail List
36	No 'Surcharge' booking fee item has been defined in the vista system - contact cinema admin staff to rectify

SetMemberInfo

Purpose:

Assigns member information to the current order. For example may be used to assign loyalty member info to the order in process so that member info can be recorded against the transaction records created in the Vista database. This command can also optionally be used before initiating a refund for loyalty member. Setting the AutoTransUpdate parameter to Y will ensure the loyalty system is informed of the refund of tickets and concessions to for the member.

Use:

Call this command after calling TransNew and before finalising the order with TransComplete.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

MemberInfo	Delimited string (see below)
AutoTransUpdate	(Y/N default N) - if Y the loyalty system will be automatically updated during TransComplete with the details of the transaction. [Available from v3r0 sp03 hot fix 01]
LoyaltySettings	Optional string containing a Loyalty system settings string as defined in LoyaltyRequest [Available from v3r0 sp03 hot fix 01]

Data Returned:

None

Comments:

None

MemberInfo parameter:

The MemberInfo parameter allows for the client app to pass an extensible list of member info in a single delimited parameter. The following elements are currently recognised:

MEMBERNO	string (50) - the membership id
MEMBERCARDNO	string(50)

i.e. | MEMBERNO | 112as1212 | MEMBERCARDNO | 999888777666 |

LoyaltySettings parameter:

The LoyaltySettings parameter passed to Sales Server here is used when processing the transaction update to Loyalty during TransComplete - if the AutoTransUpdate parameter is set to Y. The settings passed here are NOT used automatically by Sales Server when calling the LoyaltyRequest command. Settings must be passed with EACH call to LoyaltyRequest because this is a 'shared' function, which does not require an instance of an order be generated before using the command.

See Loyalty Request for a complete description of the LoyatySettings parameter

Message specific return codes:

None

SetBookingRequired

Available in Release: v3 r0 sp03

Purpose:

Allows client applications that were previously assumed to make 'direct sales' (at the time of writing this includes KIOSK and PDA client class apps) to create booking records for each transaction. This enables a modification to call flow so that bookings can be created and completed, without printing. Printing can then be performed as a separate independent action. Thus if print failure occurs on the Kiosk the customers bookings can still be collected and printed elsewhere.

Use:

Useful for KIOSK or PDA client class apps (or any future client class apps added to the 'direct sale' group) to modify Sales Servers default behaviour for these client classes.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Data Returned:

None

Comments:

None

Message specific return codes:

None

SetTransStartDateTime

Available in Release: v3 r0 sp03

Purpose:

Allows client applications processing historical offline transactions to specify the start datetime for the transaction. Call AFTER calling TransNew. Not required by real-time client applications.

Use:

Useful for offline transaction processing clients only.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

StartDateTime	YYYYMMDDHHNNSS formatted date string

Data Returned:

None

Comments:

None

Message specific return codes:

None

SetBookingAutoPay

[Available in Release: v3 r0 sp04]

Purpose:

Allows client applications to specify that a when an Unpaid booking is picked up at Vista POS the payment process should be initiated automatically using the card used to place the order. This functionality created for use by specific client applications only. Contact Vista for implementation details if you intend to use this command.

Use:

Useful for offline transaction processing clients only.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

AutoPay	Y/N default is N

Data Returned:

None

Comments:

None

Message specific return codes:

None

ValidatePaymentComplete

[Available in Release: v3 r0 sp04]

Purpose:

Allows a client application to validate that Sales Server has been notified of successful payments which total the value of the order.

Use:

This function is primarily used by offline transaction 'clean-up' applications which may need to validate that PaymentOK messages were received and processed.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Data Returned:

None

Comments:

None

Message specific return codes:

56	Payments have not been made, or do not total the value of the order.

CancelConcessions

[Available in Release: v3r2 - 32.0.0.42]

Purpose:

Remove concessions from an order leaving all other order details remaining. Concessions that are part of a ticket package, and concession items like booking fees are not removed.

Use:

Allows concessions to be removed from an order after OrderConcessions has been called.

Parameters (in order):

(Transaction Number is included as usual in message header)

Data Returned:

None

Comments:

Before this function was introduced, after calling OrderConcessions the only way to remove concessions from an order would be to cancel the order. Clients like Kiosk hold the concessions internally and call OrderConcessions at the end, but for clients like the web where the order needs to be reloaded for each page load, this allows more flexibility and also doesn't put too much processing burden on the order completion process because the concessions are already in the order.

This function can also be called by OrderConcessionEx by setting the appropriate parameter, and this allows any concessions already in the order to be removed and replaced with the new ones.

GetSessWithNumSeatsAvail

Purpose:

This returns a list of sessions with number of requested seats available for sale from the calling client.

Parameters:

Inputdata	Pipe delimited string containing search criteria

Example:

Input Data: "|ScreenNum|AreaNum|NumSeats|HoursToSelect|
InclSpecialSeat|FilmCode|CinemaOperatorCode|"

ScreenNum	Long integer indicating which screen the session is scheduled on
Code	Integer indicating which area category of the screen this search is performed on.
NumSeats	Integer indicating how many seats should be available in a session. For example, if NumSeats is 5, Sales server will only return the sessions that have at least 5 seats still available for sale at the time of searching. Note that the seats are NOT reserved at this point of time. Number of seats customer can booking still subject to the availability of seats at the time VSS receives the booking order.
HoursToSelect	Optional Positive Integer Indicating how many hours into the future session should be selected for. Selection start time is always calculated by Sales Server as the current time subtract the Vista configuration setting ATMTIMESHOWMOVIEAFTERSTART. Example: if the command is called at 1:30 pm, the configuration value is 20 mins and HoursToSelect is requested at 8 hours - the selection of sessions and their related films will be from 1:10pm to 9:10pm.
InclSpecialSeat	Optional single character indicating if special seat will be counted as available seats. Default value is “N” if omitted. “Y” indicates special seats will be included in searching.
FilmCode	Optional field
CinemaOperatorCode	Optional field

Return Data:

The usual header information: ….|1|1 (1 rows of 1 columns)

The data following the header

Example:

(…header…)|1|1|XMLData|

```
<?xml version="1.0"?>
<VSS_SessionAreaSeatNum_Data>
  <SessionSeatData>
    <SessionId>33140</SessionId>
    <ScreenNum>1</ScreenNum>
    <AreaNum>1</AreaNum>
    <AreaCatCode>0000000002</AreaNum>
    <CinOperator>BN</CinOperator>
    <FilmCode>HO00000059</FilmCode>
    <FilmTitle>Curious George</FilmTitle>
    <Showtime>20080125191500</Showtime>
    <NumSeatsAvailable>141</NumSeatsAvailable>
    <IncludSpecialSeats>Y</IncludSpecialSeats>
  </SessionSeatData>
</VSS_SessionAreaSeatNum_Data>
```

Comments:

NumSeatsAvailable	This is the number of seats available for sale from the requesting client.
IncludSpecialSeats	Indicates whether NumSeatsAvailable includes unallocated special seats as available seats or not

VoucherValidate

[Available in Release: v3r3 - 33.0.0.35]

Purpose:

Allows a voucher ticket to be validated through OVV (Online Voucher Validation).

Use:

Validates a voucher is valid (eg not expired, not already redeemed, has been sold etc) before it is added to an order or committed (redeemed).

Returns the ticket type code for this voucher to be used in OrderTickets.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

Barcode	String. The full voucher barcode to validate.
SessionId	String. Optional. The session id to check if the voucher ticket type is valid for this session.
ValidateIndividually	Y/N. Default N. If Y the voucher will not be compared with any other vouchers already in the order (to check for things like duplicate vouchers or rules that may limit the number of vouchers in an order).

Data Returned:

The usual header information: ....|VOUCHERVALIDATE|1|1| (1 rows of 1 columns)

The data following the header: Will be either the Ticket Type code or a message saying why validation failed.

Comments:

None

Message specific return codes:

45	Voucher not valid.

ValidateVouchersInOrder

[Available from Release: v3r3sp01]

Purpose:

Validates all vouchers already in an order through OVV (Online Voucher Validation).

Use:

Validates all vouchers are valid (eg not expired, not already redeemed, has been sold etc) and not committed (redeemed).

Data Returned:

The usual header information: ....|VALIDATEVOUCHERSINORDER|1|1| (1 rows of 1 columns)

The data following the header: If failed a message saying why validation failed.

Comments:

None

Message specific return codes:

45	Voucher not valid.

VoucherCommit

[Available in Release: v3r3 - 33.0.0.35]

Purpose:

Commits (redeems) one or all vouchers in an order through OVV (Online Voucher Validation).

Use:

Commits vouchers in a booking after a booking has been made with TransComplete.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

BookingNumber	String. The order booking number.
Barcode	String. Optional. The full voucher barcode to commit. If this is empty, all vouchers in the booking will be committed.

Data Returned:

The usual header information: ....|VOUCHERCOMMIT|1|1| (1 rows of 1 columns)

The data following the header: This will be empty if the voucher(s) committed ok, otherwise a message saying why commit failed.

Comments:

This function must be called to commit vouchers in an order, otherwise a paid booking will exist and the vouchers will not be redeemed (so they could be used again).

Vouchers are validated again before committing to make sure they are still valid.

Message specific return codes:

45	Voucher not valid.
46	Voucher commit has failed.
47	Voucher has already been committed.
48	No vouchers to commit (the client application must decide whether this is an error or not, ie was it expecting vouchers to be in the order).

ValidateMemberTickets

Purpose:

Returns an XML string containing the validation result of member tickets prior to ticket ordering

Parameters:

Session Id	
TicketDetailXml	<?xml version="1.0"?> <MemberTickets> <MemberTicket> <TicketCode></TicketCode> <ProviderName></ProviderName> <ProviderMemberCard></ProviderMemberCard> <MemberDOB></MemberDOB> <TicketPrice></TicketPrice> <TicketQty></TicketQty> </MemberTicket> </MemberTickets>
ForceRevalidate	Y/N , force to void all previously validated tickets and re-validate the new order list. Use this flag when ticket has been removed from order or has been changed after being validated by member providers.

Data Returned:

ValidationResultXml :

<?xml version="1.0"?>

<ApprovedOrder>

<OrderDetail>

<TicketCode></TicketCode>

<ProviderName> </ProviderName>

<ProviderMemberCard></ProviderMemberCard>

<ApprovedPrice></ApprovedPrice>

```
            <ApprovedQty></ApprovedQty>
        </OrderDetail>
</ApprovedOrder>
```

Message specific return codes:

80	Unable to initialise Vista Member Manager
81	Member Manager preload member data failed
83	Member Manger ticket order validation failed
85	Unknown Member Manager error
86	Ticket type doesn't belong to member provider
87	Requires membership subscription
88	Member ticket order request is not approved by provider

CHAPTER 3

Messages Used in Unpaid Booking Processing/Collection

GetUnpaidBookingTotal

Purpose:

Return the total payable value of an Unpaid Booking.

Use:

Used when preparing to process an unpaid booking as for payment and ticket collection.

Parameters (in order):

Booking Number	
BookingBarcode	String Optional, if supplied, Booking Number field can be omitted

Data Returned:

Total Order Value

Booking Fee Value

Number of tickets

Number of Concessions (excluding booking fees)

Message specific return codes:

17	Booking Does Not Exist
18	Booking is Not Unpaid (unexpected status)
19	Booking Cancelled (auto or manually cancelled at cinema)
33	Booking is already Paid and or Collected
44	booking has no sessions which have not already started

Comments:

This command ONLY supported in version 7.1105.11 or greater.

ReloadUnpaidBooking

Purpose:

Reloads an Unpaid Booking allowing the marking an Unpaid Booking as paid ('UnpaidBookingPaymentOK' command). This method is ONLY required if 'GetUnpaidBookingPrintStream' is NOT called.

Use:

Used when preparing to process an unpaid booking for payment and ticket collection without printing.

Parameters (in order):

Booking Number	
BookingBarcode	String optional. If supplied, Booking Number field can be omitted

Data Returned:

None

Message specific return codes:

17	Booking Does Not Exist
18	Booking is Not Unpaid (may be paid or cancelled etc)
42	booking has broken seats
44	booking has no sessions which have not already started

Comments:

This command ONLY supported in version 3 sp03 (build e)

After calling this method TransCancel MUST be used to release any resources/locks created by this command on the booking being processed if errors occur.

GetUnpaidBookingPrintStream

Purpose:

Return a print stream suitable for printing concession vouchers and tickets when an unpaid booking is to be collected. This command allows the client app to retrieve the print stream and save it BEFORE calling UnpaidBookingPaymentOK. This minimises the impact of communication failure AFTER accepting payment and marking the booking as paid.

If any errors occur in this method TransCancel must be called allowing the client app to attempt to restart the collection process for the Unpaid Booking.

This method internally calls ReloadUnpaidBooking, and therefore that command need not be called if bookings are to be printed and marked collected.

Use:

Call after calling TransNew

Call BEFORE calling UnpaidBookingPaymentOK (passing new TransIdTemp)

Parameters (in order):

(ClientId and TransIdTemp are required header arguments)

PrinterType	eg "KTX" as set in Kiosk's INI file.
CreditCard	not currently supported use booking number instead
BookingNumber	required
ReleasePrintFlag	see GetBookingsPickupPrintStream
CollectionVoucher	see GetBookingsPickupPrintStream
BookingBarcode	optional, if supplied, bookingNumber can be omitted

Data Returned:

N records of one column each, one for a food voucher, if any, and one for each ticket.

The column holds a print stream ready for direct printing, containing print instructions for a *single* ticket or food voucher, including cutter command.

In this way, the kiosk can derive from the number of records how many tickets/food vouchers should be printed, and can print them individually, checking each time that the ticket was successfully printed.

Note: It is critical that the calling application parse and record the number of printable items returned and then successfully printed, because this value must be accurate when calling UnpaidBookingPaymentOK to mark the bookings as paid and collected.

Message specific return codes:

Execute() may return the following errors:

9	a previous print attempt failed
17	booking does not exist
18	booking is not unpaid (may be cancelled)
42	booking has broken seats

Comments:

This command ONLY supported in version 3 sp03 (build e)

UnpaidBookingPaymentOK

Purpose:

Allows an unpaid booking to marked as paid (and optionally collected) when the client application has received payment (and optionally the print stream is stored and ready to print). Payment transactions are recorded and the booking is marked as paid.

If the Number of Items Printed parameter is not -1 - this method internally sets the Booking Collected flag and therefore assumes all tickets and vouchers are printed or are printable by the client application. Once successfully called the Paid booking is irreversibly marked as paid - calling TransCancel after this command returns successfully - (or a return value of 34 is received) will NOT reverse the action.

This method does NOT support multiple calls for multiple tender transactions.

Use:

Call after calling TransNew.

Call after ReloadUnpaidBooking OR GetUnpaidBookingPrintStream

Call after payment is received for an Unpaid booking.

Parameters (in order):

(ClientId and TransIdTemp are required header arguments)

BookingNumber	integer
Amount Paid	(integer type - in cents)
CardType	(If 'CASH' the card data parameters can be omitted)
CardNumber	(a valid numeric card number)
CardExpiry	(in standard format i.e. mmyy)
NumItemsPrinted	integer (use -1 to indicate no data printed, and therefore bookings not collected)
BookingId	not supported in the function, use UnpaidBookingPaymentOkEx instead

Data Returned:

None

Message specific return codes:

Execute() may return the following errors:

4	Incorrect input data types

17	booking does not exist
18	booking is not unpaid (may be cancelled)
33	booking is already paid
34	the booking was successfully marked as paid but the collected process failed - any tickets and vouchers can still be printed and provided to the customer
35	Unpaid booking was not loaded correctly using ReloadUnpaidBooking or GetUnpaidBookingPrintStream

Comments:

This command ONLY supported in version 3 sp03 (build e)

TransCancel MUST be called if any processing errors occur

UnpaidBookingPaymentOKEx

Purpose:

See UnpaidBookingPaymentOK - extends this command to allow the passing of a complex Payment Data parameter to mimic the passing of payment information available to PaymentOKEx.

This method does NOT support multiple calls for multiple tender transactions.

Use:

See UnpaidBookingPaymentOK

Parameters (in order):

(ClientId and TransIdTemp are required header arguments)

BookingNumber	integer
Amount Paid	(integer type - in cents)
CardData	see PaymentOKEx
NumItemsPrinted	integer (use -1 to indicate no data printed, and therefore bookings not collected)
BookingBarcode	String pptional, if this filed is supplied, Booking Number field can be omitted

Data Returned:

None

Message specific return codes:

See ***UnpaidBookingPaymentOK*** (on page 94)

Comments:

See ***UnpaidBookingPaymentOK*** (on page 94)

BookingsCollected

After Processing an Unpaid Booking as Paid, Bookings Collected can finally be called for the now Paid Booking if it is also being collected. See Bookings Collected command for details.

CHAPTER 4

Messages Used in Paid Booking Collection Process

GetBookingsForCard

Purpose:

Returns booking header content for all uncollected bookings for a Card.

Use:

May be used when preparing to process a booking pickup for a swiped card where specific booking numbers are NOT known. Cancelled bookings

Parameters (in order):

Card Number	
BookingBarcode	optional - new feature available in v3.1 (31.0.0.52 or greater). The barcode format must conform to one of the standard Vista booking barcode formats. If the Card Number parameter is empty the barcode parameter will be used.
AdvancedSearch	See comments below for examples. [From Release: v3r3sp01]
PartialPickup	Y/N (default N) - Partial booking pickup allowed. [Release: v3r3sp01]

Data Returned:

Delimited pairs of data containing | BookingNumber | BookingDataXML | BookingNumber | ...

Message specific return codes:

75	invalid barcode for booking pickup
57	incorrect branch for booking
76	contains sessions marked unavailable

Comments:

Examples of advanced search (parameter 3):

|NAME|John Smith|

|PHONE|021 123 4567|

SetBookingDeliveryEx

[Available from Release: v3r3sp01]

Purpose:

Sets delivery on an item(s) in an already created booking.

Use:

Can add delivery to items that have been booked on web and are now being picked up by Mobile POS.

Parameters (in order):

OrderDeliveryXML	Same delivery XML as used in OrderConcessionsEx

Data Returned:

None.

Message specific return codes:

89	Order already started or cancelled.

GetBookingsPickupPrintStream

Purpose:

Return a print stream suitable for printing concession vouchers and tickets when a prepaid booking is collected from a kiosk. GetBookingsPickupPrintStream will only return PAID bookings - Unpaid Bookings cannot be picked up via this command. However functionality is available to collect Unpaid Booking print streams, sign off paymentreceived and mark the unpaid booking as collected (see Unpaid Booking Pickup section)

This method has been enhanced in versions 7.1105.x to allow client applications to specify a particular booking number.

Use:

Call after customer has swiped their card as part of ticket pickup process.

Call after payment is received for an Unpaid booking, and a print stream is required.

Parameters (in order):

PrinterType	e.g. "KTX" as set in Kiosk's INI file.
CardNumber	excluding expiry, no spaces or hyphens eg 9889455623458819
BookingNumber	optional - new feature available in 7.1105 - if booking number is included the Credit Card Number parameter can remain empty.
ReleasePrintFlag	Y/N - default N Forces VSS to release the KioskIsPrinting flag immediately after retrieving the print stream data. This allows newer client apps to override legacy behaviour in Sales Server which locked the retrieval of a booking to a kisk - if print failed that booking was not collectable via another kiosk - only via Vista POS.
CollectionVoucher	Optional Y/N, default N. If set to Y the print stream is assumed to be used for informational purposes or a collection voucher only, not for actually printing physical tickets. This is useful if a data stream printer/print template is used to return order details which are displayed to the end user - or printed to a booking voucher, which will be presented by the customer to produce actual tickets at the cinema. In such cases a 'Booking' barcode is included in the print stream rather than 'Ticket' barcodes.
BookingBarcode	optional - new feature available in v3.1 (31.0.0.52 or greater). The barcode format must conform to one of the standard Vista booking barcode formats. If the BookingNumber and CardNumber parameters are empty the barcode parameter will be used. BookingBarCode can also be the BookingId barcode

Data Returned:

N records of one column each, one for a food voucher, if any, and one for each ticket.

The column holds a print stream ready for direct printing, containing print instructions for a *single* ticket or food voucher, including cutter command.

In this way, the kiosk can know from the number of records how many tickets/food vouchers should be printed, and can print them individually, checking each time that the ticket was successfully printed. Thus a kiosk can determine if any of the tickets were not printed.

It is critical that the calling application parse and record the number of printable items returned and then successfully printed, because this value must be accurate when calling BookingsCollected to mark the bookings as collected.

Message specific return codes:

Execute() may return the following errors:

9	problem getting tickets because a previous attempt had a problem… go to counter for assistance
10	no bookings found
11	bookings are found but are all marked 'collected' (none remaining to collect)
42	booking has broken seats
75	invalid barcode for booking pickup
57	incorrect branch for booking
76	contains sessions marked unavailable

Comments:

The food voucher is the first item in the print stream.

There can be no more than 1 food voucher in a print stream.

Valid printer names are agreed by designers of kiosk and Vista. The names need only be determined at installation time. (although any printer formatting needs to be 'programmed' in advance).

GetBookingsPickupPrintStreamEx

[Available from Release: v3r3sp01]

Purpose:

Print only the specified tickets/concessions.

Use:

Used for partial booking pickup on Mobile POS.

Parameters (in order):

PrinterType	e.g. "KTX" as set in Kiosk's INI file.
TicketSequences	list of ticket sequences to print
ConcessionSequences	list of concession sequences to print
ReleasePrintFlag	Y/N - default N Forces VSS to release the KioskIsPrinting flag immediately after retrieving the print stream data. This allows newer client apps to override legacy behaviour in Sales Server which locked the retrieval of a booking to a kisk - if print failed that booking was not collectable via another kiosk - only via Vista POS.
CollectionVoucher	Optional Y/N, default N. If set to Y the print stream is assumed to be used for informational purposes or a collection voucher only, not for actually printing physical tickets. This is useful if a data stream printer/print template is used to return order details which are displayed to the end user - or printed to a booking voucher, which will be presented by the customer to produce actual tickets at the cinema. In such cases a 'Booking' barcode is included in the print stream rather than 'Ticket' barcodes.
BookingNumber	The booking number.

Data Returned:

N records of one column each, one for a food voucher, if any, and one for each ticket.

The column holds a print stream ready for direct printing, containing print instructions for a *single* ticket or food voucher, including cutter command.

In this way, the kiosk can know from the number of records how many tickets/food vouchers should be printed, and can print them individually, checking each time that the ticket was successfully printed. Thus a kiosk can determine if any of the tickets were not printed.

It is critical that the calling application parse and record the number of printable items returned and then successfully printed, because this value must be accurate when calling BookingsCollected to mark the bookings as collected.

Message specific return codes:

Execute() may return the following errors:

9	problem getting tickets because a previous attempt had a problem… go to counter for assistance
10	no bookings found
11	bookings are found but are all marked 'collected' (none remaining to collect)
42	booking has broken seats
75	invalid barcode for booking pickup
57	incorrect branch for booking
76	contains sessions marked unavailable

Comments:

The food voucher is the first item in the print stream.

There can be no more than 1 food voucher in a print stream.

Valid printer names are agreed by designers of kiosk and Vista. The names need only be determined at installation time. (although any printer formatting needs to be 'programmed' in advance).

BookingsCollected

Purpose:

Notify Vista that all items from a Paid Booking have been printed successfully and collected. The Order is flagged as collected and cannot be collected again. If the kiosk detects a printing problem it must call this message, passing the number of items it believes were printed, before shutting down. If the number of items printed does not match the order held in Vista then Vista can flag that there was a problem during collection.

This method has been enhanced in versions 7.1105.x to allow client applications to specify a particular booking number.

Use:

Call after tickets have been printed.

Parameters (in order):

Credit Card	
PrintItemCount	Total number of separate items that were printed (with cutter), where an item is a food voucher or a ticket.
BookingNumber	optional - new feature available in 7.1105.x - if booking number is included the Credit Card Number parameter can remain empty.
BookingBarcode	optional - new feature available in v3.1 (31.0.0.56 or greater). The barcode format must conform to one of the standard Vista booking barcode formats. If the Card Number parameter is empty the barcode parameter will be used. BookingBarcode can also be BookingID barcode
TicketSequences	For partial booking pickup the list of ticket sequences [Available from Release: v3r3sp01]
ConcessionSequences	For partial booking pickup the list of ticket sequences [Available from Release: v3r3sp01]

Data Returned:

None

Message specific return codes:

Execute() may return the following errors:

12	the number of tickets + vouchers printed passed does not equal the number recorded by Vista. The booking will be marked as collected regardless. The kiosk can continue operation- but should probably check that printing is still OK.

75	invalid barcode for booking pickup
57	incorrect branch for booking
76	contains sessions marked unavailable

Comments:

This message is not relevant for direct sales and is only used as part of automated ticket pickup software.

If Vista is unable to record that the booking has been picked up, Execute returns a code 5. The kiosk should retry, and if no success it should shut down (otherwise the booking could be collected and reprinted all over again... free tickets)

Ticket Printing

Tickets will be printed on the standard ticket roll in the style as used in most cinemas, though the printer in the Kiosk may not be the same model as used in-cinema. In the first instance, send full ticket printing information, including control codes, and implement it as a template style later if speed issues make it necessary. Ultimately, if it is cost effective, a printer independent ticket formatting language is the ideal solution.

Each Kiosk will include its printer type in with the print message so that the server can determine what print stream to send.

Print Data

Printing can be done using the messaging system already proposed. It is merely a different message.

```
ObjMsg.Cmd( "GETTICKETPRINTDATA")
ObjMsg.Execute ()
PrintData = ObjMsg.GetData()
```

The print data will consist of N records, each of 1 field (each field corresponding to the print stream for a complete single ticket), and as such will fit into the messaging scheme already described.

The Kiosk can count the number of records received back for printing (ie the number of tickets) as a check that the order was registered properly (in terms of number of tickets anyway). For example:

```
 | KioskId | 217402 | GETPRINTDATA | 2 | 1 | ADLT$^%Batman*&^3*(&000714/001/3
| CHLD$%^Batman&^%3$#@000714/002/3 |
( 2 is no. of tickets, 1 field for each, Vista transaction number embedded
in the above printdata stream)
```

CHAPTER 5

Messages Used in Customer Selected Seating

GetSessionSeatData

Purpose:

Return a stream of data representing all assigned seats for a session, including the status of each seat.

Use:

For allocated seating sessions only. After an Order has been created, and seats have been system allocated. System allocated seats are identified using seat status: "1" - booked, "3" - house or special, "5" - system allocated for this order, "0" - available for user selection.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header).

SessionID	Integer
IncudeBrokenSeats	Optional Y/N - default N If Y broken seats are represented with a status of 6, broken seats sold to the user are represented with a status of 7 - these should not be available for selection - so the customer should be forced to modify these.
IncludeHouseSpecial	Optional Y/N - default N If Y house seats are represented with a status of 3, special seats 4.
IncludeGreySofa	Optional Y/N - default N If Y grey seats are represented with a status of 8 and seat priority is included in the seat data chunk. The seat priority can be used to determine sofa seat lengths
SoldCatOnly	Optional Y/N - default N If Y only areas that tickets have been ordered for will be returned.
IncludeSeatNumbers	Optional Y/N - default N If Y the seat data chunk will include seat numbers as the first 5 characters in the seat data.

Data Returned:

The structure of the SessionSeatData stream follows. Each element is delimited using system delimiter (|).

OutputStringLength	(excludes trailing area category code summary)
Scale Factor	(integer)
ScreenHeight	(integer) 'Y
ScreenWidth	(integer) 'X
TotalNumberOfAreas	(integer)

(repeated for each area)

AreaNum	(integer)
AreaCategoryCode	(string)
Area Y Start Position	(integer)
Area X Start Position	(integer)
IsSelectable	(ie Seat Allocation on) (string = Y/N)
Area Desc	
Area Alt Desc	(single space represents null - due to Java handling)
TotalNumberOfRows	(integer)
Sofa Mode	(ie Sofa Mode on) (string = Y/N)

(repeated for each row)

SeatGridRowID	(integer) (maps to tblSession_RowAllocation etc)
RowPhysID	(ie A)

(repeated for each seat - this section has a fixed width for each seat -these elements are NOT pipe delimited, all data for all seats in a row occur in a single chunk of string)

Seat Number	(5 chr padded with spaces) only returned if IncludeSeatNumbers is set to Y
GridSeatNum	(2 chr coded - represents the column number from Session tables)
Seat Y Pos	(2 chr coded)
Seat X Pos	(2 chr coded)
Seat Status	(single chr)
Seat Priority	(single chr) only returned if IncludeGreySofa is set to Y

The following data trails the seat data string - it represents a summary of the number of seats booked in each area category (which have NOT been allocated by the system) - utilised when system allocation has failed to allocate seats in a contiguous block - allowing the client application

to the limit how many seats the user selects in each area category appropriately. The OutputStringLength does not include this data - therefore the client application can split the two strings using the OutputStringLength value.

AreaCategoryCode

QtySeats

Comments:

Seats must have been booked for the TransId. The session must have allocated seating on. The number of tickets a user selects for each Area Category Code in a session must match the number of tickets the system booked in each Area Category Code. This prevents the user from selecting seats from different price bands after booking seats. The actual physical areas selected can change - as long as the area category codes are equivalent.

The OutputStringLength value does not include the trailing summary data. It provides a simple way for client applications to split the return string and summary data string.

Seat Status Values:

The Seat Status is a single character value indicating the current state of the seat.

0	available
1	booked (by another customer/order) (or house or special if IncludeHouseSpecial is not Y)
5	system allocated for this order

The following are only used if IncludeBrokenSeats is Y

6	broken
7	booked by user and broken (force user to change these)

The following are only used if IncludeHouseSpecial is Y

3	House
4	Special
2	Chr Coding is used for each seat coordinate and GridSeatNum value. This allows for numbers greater than 99 to be represented with 2 characters. See APPENDIX III for a java script example of interpreting these values.

The following are only used if IncludeGreySofa is Y

8	Grey Indicates a seat from another area shown for seat layout display

SetSelectedSeats

Purpose:

Return a stream of data representing all assigned seats for a session, including the status of each seat.

Use:

Allows for the re-allocation of seats for a transaction and session to user defined seat selection

Parameters (in order):

SessionID	
Selected Seats	(string containing the number of seats being set followed by N Selected Seats - in quads of AreaCatCode, AreaNum, GridSeatRowID,GridSeatNum). Example: | 2 | 00000001 | 1 | 2 | 3 | 00000001 | 1 | 24 | (ClientId, and TransIdTemp are contained in the standard header)

Data Returned:

None

Comments:

This function checks that the new list of selected seats conforms to the system allocated area category code blocks (note AreaCategory blocks are assigned in CreateOrder regardless of Seat Allocation on/off/successful). For example if 2 tickets were assigned to Area Category 1, and one to Area Category 2, the list passed to this function must conform to these allocation quantities by Area Category. Physical areas within each Area Category can be moved freely between.

GetTransStartTime

Purpose:

Returns a delimited list of date data for the transaction start time. The format of this return data has been tailored specifically for use by the WWW java applet as per spec from Sonjoy Banik. Each delimited element represents a piece of the date.

Use:

Allows for the re-allocation of seats for a transaction and session to user defined seat selection

Parameters (in order):

None

(ClientId, and TransIdTemp are contained in the standard header)

Data Returned:

String containing six delimited elements - one for each of the date parts of the transaction start time.

i.e. element(1) Year, element(2) Month, element(3) Day, element(4) Hour (0-23), element(5) Minute, element(6) Seconds

Comments:

GetReservedSeatInformation

Purpose:

Returns a delimited list of seat data for the selected transaction and session

Use:

Informational, currently used internally by GetSessionSeatData to identify system allocated seats for a transaction and session.

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

SessionID	

Data Returned:

Delimited string of N Tickets in the form: i.e. AreaNum, GridSeatRowId, GridSeatNum, SeatRowId, SeatNum, AreaTicketDesc

Comments:

None

GetReservedSeatDataEx

Purpose:

Returns a delimited list of seat data for the selected transaction and session. An extension to GetReservedSeatInformation

Use:

Informational, outputs the reserved seat data for a session within an order

Parameters (in order):

(ClientId, and TransIdTemp are contained in the standard header)

SessionID	

Data Returned:

Delimited string of N Tickets in the form: i.e. TicketId, SequenceNum, PackageGroupNo, ParentTTypeCode, AreaCatCode, AreaNum, TicketTypeCode, GridSeatRowId, GridSeatNum, SeatRowId, SeatNum, AreaTicketDesc

Comments:

None

CHAPTER 6

Messages Used in Inventory Processing

ProcessStockReceipt

Purpose:

Validate and apply a batch of Stock Receipt or Matched Stock Receipt Reversal entries.

Use:

For external software systems which must interface to and update the Vista cinema inventory system. The ProcessStockReceipt message enables a client application to enter both Stock Receipts and Matched Stock Receipt Reversals via the Sales Server interface. Processing is performed in batches - each batch is considered a single stock receipt file and is reported this way in the Vista Inventory reporting system. However each detail line within a batch is processed in a self contained way - so that if network or database connectivity errors are generated for any reason mid-processing a stock receipt or stock receipt reversal may be partially processed. In such cases the client application is notified of the last successful detail item which was processed. To bind the client application and Vista more closely in this respect the client application is required to include a unique sequence number with each detail item - Sales Server will only process a receipt if these numbers are sequential whole numbers beginning with one for the first detail line. This sequence provides the client application with a permanent reference to the number used to identify each detail line of a stock receipt in the vista system. Thus if a stock receipt is partially processed the client application is returned the sequence number of the last successful line processed. A successful or partially successful response also includes the return of a Vista Transaction Number to the client application. This Transaction Number reference is also required when performing matched reversals so should be stored by the client application.

Reversals are supported for matched stock receipts processed previously from the client application. Reversals can be performed for all or part of a previous receipt. However no modifications to the quantity or price of the processed line can be altered during a reversal. To process a reversal the client application must pass the Vista Transaction Number and a list of sequence numbers which should be reversed. These references are described above.

Parameters (in order):

(ClientId is contained in the standard header. For the purposes of this message TransIdTemp can be left empty).

VendorCode	string(8)
ReceiptCode	string(255) - freeform receipt reference tag for non-reversals, Vista Transaction Number to reverse for reversals
TaxInclPricing	Y/N are the input data prices tax inclusive
InputData	Delimited string - see below
IsReversal	Y/N is this a reversal or a normal stock receipt

InputDataFormat	From v3r3sp02 - the format of the input data delimited string

Parameters Described:

None

Detail Line Structure (non-reversal):

SequenceNo	long integer
ItemId	string(15)
StockLocationCode	string(4)
QtyUOMReceived	Sales Server decimal format
ReceivedUOM	string(4)
Value	decimal Note: In version 33.0.28 and above this is an optional field. Default to use item cost price as set up in Vista if the value is omitted.
HOPK	Only applies if InputDataFormat parameter = "H" from v3r3sp02 Optional Head Office Primary Key string(50) If included this will be used to find the Item and not the ItemId

eg:

If InputDataFormat parameter not "H" (or before v3r3sp02)

| 1 | 110 | 2 | 5.34 | KILO | 12.6543 | 2 | 337 | 2 | 20 | CNT | 110.75 |

If InputDataFormat parameter = "H"

| 1 | 110 | 2 | 5.34 | KILO | 12.6543 | HO1234 | 2 | 337 | 2 | 20 | CNT | 110.75 | HO5678 |

Detail Line Structure (reversal):

SequenceNoToReverse From a previously processed Stock Receipt.

eg: | 1 | 3 | 4 | 5 | 10 |

Vendor Code, Stock Location Code, Received UOM and ItemId are all reference values found in the Vista Inventory system - these can be queried from vista directly on a read-only basis. These values must all validate before processing begins - any invalid values will result in an execute() return code of 32. The received UOM must be valid for the Item in the detail line.

In version 33.0.28 and above:

Vendor Code is an optional field. When this value is omitted, each receipt detail line will be given a default Vendor Code, which is the Vendor Code of the Item in the detail line as setup in Vista Inventory System.

Data Returned:

All values returned in a pipe delimited string as per standard Sales Server return data.

successful (return code 0):

VistaTransactionNumber	
LastSucessfulSequenceNumber	should match last sequence number sent from client app.

partially processed (return code 32):

VistaTransactionNumber	
LastSucessfulSequenceNumber	will identify the last sequence number sent which was successfully processed - all sequence numbers prior to this number will ALSO have been applied. Any sequence numbers following this number will have been abandoned.

Return codes:

Execute() may return the following errors:

31	input data was invalid - check sales server error log for more detail
32	reversal partially processed - check last successful sequence number in return data string

Comments:

TransNew should NOT be called before initiating this message - therefore TransCancel is also not required on failure.

ProcessStockTransfer

Purpose:

Validate and apply a batch of Stock Transfers IN or OUT of the cinema Inventory system.

Use:

For external software systems which must interface to and update the Vista cinema inventory system. The ProcessStockTransfer message enables a client application to transfer in and out of the Vista cinema system via the Sales Server interface. Processing is performed in batches - each batch is considered a single stock transfer file and is reported this way in the Vista Inventory reporting system. However each detail line within a batch is processed in a self contained way - so that if network or database connectivity errors are generated for any reason mid-processing a stock transfer may be partially processed. In such cases the client application is notified of the last successful detail item which was processed. To bind the client application and Vista more closely in this respect the client application is required to include a unique sequence number with each detail item - Sales Server will only process a receipt if these numbers are sequential whole numbers beginning with one for the first detail line. This sequence provides the client application with a permanent reference to the number used to identify each detail line of a stock transfer in the vista system. Thus if a transfer is partially processed the client application is returned the sequence number of the last successful line processed. A successful or partially successful response also includes the return of a Vista Transaction Number to the client application.

Transfers OUT of the cinema are expected with a positive (not negative) quantity for each line. Negative quantities are automatically reversed to positive and processed by sales server if encountered.

Parameters (in order):

(ClientId, is contained in the standard header. For the purposes of this message TransIdTemp can be left empty).

IN/OUT	string(3) indicates a Transfer In or Transfer Out operation
RelatedCinema	string(255) freeform field which can be used to Identify the related cinema in the transfer - i.e. for a Transfer IN the field will refer to the cinema or location the stock came from, for a Transfer OUT it will refer to the destination cinema or location. Alternatively the field can be used to store any required reference or comments the client application wishes to store in the Vista system.
TaxInclPricing	Y/N- are the input data prices tax inclusive
InputData	Delimited string - see below
InputDataFormat	From v3r3sp02 - the format of the input data delimited string

Parameters Described:

Detail Line Structure (non-reversal):

SequenceNo	long integer

ItemId	string(15)
StockLocationCode	string(4)
QtyUOMReceived	Sales Server decimal format
ReceivedUOM	string(4)
Value	decimal (only accepted from the calling application for transfer IN, Transfers OUT use the Vista system cost price for stock). For Transfer OUT a 0 can serve as a placeholder for the cost price parameter field.
HOPK	Only applies if InputDataFormat parameter = "H" from v3r3sp02 Optional Head Office Primary Key string(50) If included this will be used to find the Item and not the ItemId

eg:

If InputDataFormat parameter not "H" (or before v3r3sp02)

| 1 | 110 | 2 | 5.34 | KILO | 12.6543 | 2 | 337 | 2 | 20 | CNT | 110.75 |

If InputDataFormat parameter = "H"

| 1 | 110 | 2 | 5.34 | KILO | 12.6543 | HO1234 | 2 | 337 | 2 | 20 | CNT | 110.75 | HO5678 |

Stock Location Code, Received UOM and ItemId are all reference values found in the Vista Inventory system - these can be queried from vista directly on a read-only basis. These values must all validate before processing begins - any invalid values will result in an execute() return code of 32. The received UOM must be valid for the Item in the detail line.

Data Returned:

All values returned in a pipe delimited string as per standard Sales Server return data.

successful (return code 0):

VistaTransactionNumber	
LastSucessfulSequenceNumber	should match last sequence number sent from client app.

partially processed (return code 32):

VistaTransactionNumber	
LastSucessfulSequenceNumber	will identify the last sequence number sent which was successfully processed - all sequence numbers prior to this number will ALSO have been applied. Any sequence numbers following this number will have been abandoned.

Return codes:

Execute() may return the following errors:

31	input data was invalid - check sales server error log for more detail
32	reversal partially processed - check last successful sequence number in return data string

Comments:

TransNew should NOT be called before initiating this message - therefore TransCancel is also not required on failure.

CHAPTER 7

Miscellaneous Messages

GetLogEntryList

Available in Release: v3 r0 sp03

Purpose:

Returns a pipe delimited list (standard VSS format) for log entries generated during the last command.

Use:

For stateful clients which maintain a reference to an instance of sales server this command can be called IMMEDIATELY following any call to Sales Server. It will return a list of log entries (description attribute) generated by the last command. This list is cleared with each new command. Useful when developing applications to verify the cause of invalid return codes to the client application

Parameters:

none

Data Returned:

Pipe delimited list of log entry descriptions.

LogVistaMsg

Purpose:

Create a log entry in the Vista system log table.

Use:

Used sparingly for system specific issues, errors and information as authorised by Vista Entertainment Solutions only. Please contact Vista for authorisation.

Parameters (in order):

MessageHeader	string(50)
MessageDetails	string(500)
MessageSource	string(50)
State	string(50) acceptable values: 'ERROR ' all others logged as 'INFO'

Data Returned:

None

Return codes:

Execute() may return the following errors:

5	unexpected error

Comments:

None

CreateTab

Purpose:

Creates a running Tab or Debtor record to which payments can be made using the PaymentOKEx command (TabCode parameter must be supplied)

Use:

Use restricted to applications authorised by Vista.

Parameters (in order):

TabXML	an XML document fragment containing details used to identify the tab (see comments below)

Data Returned:

TabCode	The internal 10 character string which identifies the tab. This value MUST be stored by the client application and passed to the TabCode parameter when making payments using PaymentOKEx.

Return Codes:

73	The details supplied have already been used by another active Tab in the system. The validation rules applied to keep Tab details unique are identical to those used by VistaPOS.

Parameters Defined:

TabDetailXML	This parameter holds a XML document node fragment of a defined schema. This schema includes a root node named <tab> which contains a sequence of detail nodes. Available nodes include: ▪ <id> string(30) identifier for the tab, may be a barcode, tab card number, screen/seat combination etc ▪ <name> string(255) name of the customer ▪ <codeword> string(20) customer code word for the tab ▪ <location> string(30) the customers table, seat or location

These elements are effectively optional - however at least one of these elements must be supplied to create a Tab. It is suggested that the client application force the use of the <id> property, this ensures each active, unpaid tab in the system uses a unique identifier.

Example:

```
<tab>
      <id>ABC01</id>
```

```
      <name>Joe Bloggs</name>
      <codeword></codeword>
      <location>Table01</location>
</tab>
```

Comments:

None

GetTabList

Purpose:

Returns an XML string containing a list of tab header information matching the tab properties passed by the client application.

Use:

Used by applications authorised by Vista Entertainment Solutions only. Please contact Vista for authorisation.

Parameters (in order):

TabXML	an XML document fragment containing details used to identify the tab (see CreateTab command which uses the same XML schema)

Data Returned:

TabListXML	a list of <tab> XML nodes - the schema for each node matches the schema for the <tab> element detailed in the VSS_TabList.xsd file.

Return codes:

None

Comments:

A return code of zero with an empty string in the GetData return indicates that no Tabs entries with matching data were located.

GetTabDetail

Purpose:

Returns an XML string containing tab details, and transactions.

Use:

Used by applications authorised by Vista Entertainment Solutions only. Please contact Vista for authorisation.

Parameters (in order):

TabCode	string(10) - The Tab Code for the tab as returned from GetTabList and CreateTab commands

Data Returned:

TabDetailXML - see the VSS_TabDetail.xsd file for the schema of data returned.

Return codes:

None

Comments:

A return code of zero with an empty string in the GetData return indicates that Tabs entries with matching data were retrieved.

GetTabSlipPrintStream

[Available from Release: v3r3sp01]

Purpose:

Gets a tab slip print stream.

Use:

Used to print out a tab from remote sales devices (eg Mobile POS)

Parameters (in order):

PrinterType	Printer Type (eg "Zebra")
TabCode	string(10) - The Tab Code for the tab as returned from GetTabList and CreateTab commands

Data Returned:

Print stream containing the tab slip.

Return codes:

None

UpdateTabDetail

Purpose:

Update an existing Tab with modified details. All details passed are applied to the tab so ensure all details are passed with each Update statement - not just those fields which were modified.

Use:

Used by applications authorised by Vista Entertainment Solutions only. Please contact Vista for authorisation.

Parameters (in order):

TabCode	string(10) - The Tab Code for the tab as returned from GetTabList and CreateTab commands
TabXML	an XML document fragment containing details used to update the tab (see CreateTab command which uses the same XML schema)

Data Returned:

TabListXML	a list of <tab> XML nodes - the schema for each node matches the schema for the <tab> element detailed in the VSS_TabList.xsd file.

Return Codes:

73	The details supplied have already been used by another active Tab in the system. The validation rules applied to keep Tab details unique are identical to those used by VistaPOS.

Comments:

A return code of zero with an empty string in the GetData return indicates that no Tabs entries with matching data were located.

GetPrepOrderList

Purpose:

Returns an XML string containing a list of prep order header information matching the order properties passed by the client application.

Use:

Used by applications authorised by Vista Entertainment Solutions only. Please contact Vista for authorisation.

Parameters (in order):

PrepOrderRequestXML	an XML document fragment containing details used to identify the tab (see CreateTab command which uses the same XML schema)

Data Returned:

PrepOrderListXML	a list of <preporder> XML nodes - the schema for each node matches the schema for the <tab> element detailed in the VSS_PrepOrderList.xsd file.

Return codes:

None

Comments:

A return code of zero with an empty string in the GetData return indicates that no Prep Order entries with matching data were located.

GetPrepOrderDetail

Purpose:

Returns an XML string containing all prep order item details including content and delivery information.

Use:

Used by applications authorised by Vista Entertainment Solutions only. Please contact Vista for authorisation.

Parameters (in order):

OrderId(List)	Pipe delimited string containing a single, or list of PrepOrder_lngNumber values for the order as returned from the GetPrepOrderList command in the xml return data

Data Returned:

PrepOrderDetailXMLList - see the VSS_PrepOrderDetail.xsd file for the schema of data returned.

Return codes:

None

Comments:

A return code of zero with an empty string in the GetData return indicates that Tabs entries with matching data were retrieved.

CreateExpectedReceipts

Purpose:

Creates Expected Receipts using xml data supplied by client applications

Use:

Use restricted to applications authorised by Vista.

Parameters (in order):

<table>
<tr>
<td>ReceiptData</td>
<td>an XML document containing details used to generate Expected

Receipts (see VSS_DistributionOrder for schema definition). The important xml elements are:

Header Elements:
<table>
<tr><td>ItemItem OrderNumber</td><td>HO item codeHO item codeOrder number</td></tr>
<tr><td>ExpectedDelivery ExpectedDelivery Vendor</td><td>date of the delivery in yyyymmdd formatdate of the delivery in yyyymmdd formatvendor id</td></tr>
<tr><td>QtyQtyOrderDat e</td><td>Quantity of ordered itemQuantity of ordered itemdate of the order in yyyymmdd format</td></tr>
<tr><td>StatusStatusCom ment</td><td>Status of order. Must be Ostatus of order. Must be Ocomment</td></tr>
</table>
(leave this comment here - DT)

Orders:</td>
</tr>
</table>

Data Returned:

One field is returned in the return data string:

ResponseMsg	The Vista BackOffice reorder worksheet program returns complex data structures and therefore this response is preserved in its original xml format. Consult VSS_DistributionOrder schema as to how to interpret this message. The import xml elements are: Header Elements: Item

HO item code

ExpectedDelivery

date of the delivery in yyyymmdd format

Qty

Quantity of ordered item

Status

Status of order. Must be SErrors

This is optional. When order has been processed successfully, Errors element will be omitted.

MsgCode	Error message code (see comment for more information)
Msg	Error message detail (see comment for more information)

Orders:

This is optional. When orders are successful, each order is confirmed here with each order status changed to S.

Item	Code of the item to be delivered
ExpectedDelivery	date of the delivery in yyyymmdd format
Qty	Quantity of ordered item
Status	status of order. Successful order must returns status S

Return Code:

0	Reorder worksheet program is executed successfully. This value does NOT indicate any particular message was validated. The client app must consult the response message to determine if a message was accepted by the Cinema
5	Unable to make call to Reorder Worksheet program to perform the task. i.e. unable to visReorderWorksheet.dll or visReorderWorksheet failed unexpectedly

Comments:

MsgCode element in reorder worksheet response message are :

75600:	Vendor validation error
75610:	Item validation error

(Any other value means unknown Reorder Worksheet error)

Msg element in reorder worksheet response message describes the error detail. If certain

item has caused the failure, error description will be followed by the item code i.e.

"The vendor does not exist at this cinema"

Or

"The item does not exist at this cinema, 55"

ItemCreate

Available in Release: v3 r3 sp01

Purpose:

Allows a new item to be created or edited.

Use:

Use restricted to applications authorised by Vista.

Parameters (in order):

ItemXML	an XML document fragment containing details used to create or edit the item. (see comments below)

Data Returned:

ErrorMessage	If the create or edit is not successful a message will be returned with the reason for the fail.

Return Codes:

90	Creating or editing the item has failed.

Parameters Defined:

ItemXML	This parameter holds a XML document node fragment of a defined schema. This schema includes a root node named <Item> which contains a sequence of detail nodes. Available nodes include: ▪ <Item_strItemId> The item code, if it already exists the item will be updated, otherwise a new item will be created. ▪ <CinOperator_strCode> The cinema operator code. ▪ <Item_strItemDescription> Item description. ▪ <Item_strBaseUOMCode> The base unit of measure. ▪ <Class_strCode> Item class. ▪ <Item_strItemType> Item type. ▪ <Item_strStatus> Item status. ▪ <Vendor_strCode> Vendor code. ▪ <Item_strItemShortDesc> Item short description. ▪ <STax_strCode> Tax code. ▪ <Item_intCostPrice> Cost price. ▪ <Item_strBarcode> (optional) Item barcode.

Example:

```
<?xml version="1.0"?>
<Item>
   <Item_strItemId>20000013</Item_strItemId>
   <CinOperator_strCode>RI</CinOperator_strCode>
   <Item_strItemDescription>Material Text</Item_strItemDescription>
   <Item_strBaseUOMCode>EACH</Item_strBaseUOMCode>
   <Class_strCode>MISC</Class_strCode>
   <Item_strItemType></Item_strItemType>
   <Item_strStatus>A</Item_strStatus>
   <Vendor_strCode>MRP001</Vendor_strCode>
   <Item_strItemShortDesc>MText</Item_strItemShortDesc>
   <STax_strCode>2</STax_strCode>
   <Item_intCostPrice>450</Item_intCostPrice>
   <Item_strBarcode>B123456789</Item_strBarcode>
</Item>
```

Comments:

None

CHAPTER 8

Messages Used in Group Booking Process

In general the process is as follows:

1 Create a normal unpaid booking with no booking fees.

2 After creating the unpaid booking call TransNew then ReloadGroupBookings with the unpaid booking number and the number of seats for the initial purchase.

3 Pay for the seats and call CompleteGroupBooking.

4 Subsequent requests for bookings call TransNew then ReloadGroupBookings, pay and call CompleteGroupBookings.

The first call to ReloadGroupBookings should use the booking number from the unpaid booking. This will return the Group Booking Reference, and this should be used for subsequent calls to ReloadGroupBooking.

The SeatStatusGroupBooking function returns the number of seats available for a group booking.

Initial Booking:

1. Make unpaid booking as normal
2. TransNew as normal
3. ReloadGroupBooking(unpaid booking no, number of seats)
4. Payment as normal
5. CompleteGroupBooking(Group Booking Ref, Number of Tickets Printed)

Subsequent requests for tickets:

1 SeatStatusGroupBooking(Group Booking Ref)

2 TransNew as normal

3 ReloadGroupBooking(Group Booking Ref, number of seats)

4 Payment as normal

5 CompleteGroupBooking(Group Booking Ref, Number of Tickets Printed)

SeatStatusGroupBooking

Purpose:

Gets the total number of seats, used seats, in progress seats and available seats for a group booking.

Use:

Can be called anytime to find the seats available for a group booking.

Should be called before calling ReloadGroupBookings to check seats are actually available.

Parameters (in order):

Group Booking Ref	The Group Booking Reference Number as returned by the first call to ReloadGroupBooking after the initial unpaid booking is made.

Data Returned:

The usual header information: ....|SEATSTATUSGROUPBOOKING|4|1| (4 rows of 1 columns)

The data following the header: | Total Number of Seats in the Group Booking | Used Number of Seats | In Progress Number of Seats | Available Number of Seats |

eg. |10|2|4|4| means 10 seats in total, 2 used, 4 in progress, 4 available.

Comments:

Does not check the status of the unpaid booking (eg the unpaid booking may be cancelled) - however ReloadGroupBookings will fail if the number of seats requested is not available.

In progress seats mean an order has started for these seats, but not complete yet. These seats are automatically released back to the group booking after the number of minutes specified in this system setting: GroupBookingSeatsHeldTimeout (default 10 minutes).

ReloadGroupBooking

Purpose:

Reloads the requested number of seats from the unpaid group booking into the temporary sales server tables so they can be paid and the order completed.

Use:

Call this to start a purchase for the requested number of seats in a group booking.

Parameters (in order):

Group Booking Ref/Unpaid Booking Number	The Group Booking Reference Number as returned by the first call to ReloadGroupBooking after the initial unpaid booking is made. The first time this function is called after the initial unpaid booking is made the unpaid booking number should be passed instead.
Number of Seats	The requested number of seats from the group booking.

Data Returned:

The usual header information: ....|RELOADGROUPBOOKING|7|1| (7 rows of 1 columns)

The data following the header: | Group Booking Ref | Order Total | Ticket Total | Concession Total | Booking Fee Total | Number of Seats | Seat Information |

eg. |FB605034|2400|2200|0|200|2|1 K 13 6 7;1 K 12 6 8|

The Seat Information is a ; delimited list of 5 elements for each seat in the order.

The elements are:

Area_bytNum (area number)

ScreenD_strPhyRowId (row id)

ScreenD_strSeatId (seat id)

ScreenD_bytGridRowId (grid row id)

BookingD_bytGridColId (grid column id)

Comments:

Once seats have been reloaded from a group booking, payment should be made as for a normal order (eg PaymentStarting, PaymentRequest, PaymentOKEx).

These seats will be "held" until either the GroupBookingSeatsHeldTimeout (default 10 minutes) time is reached or the order is cancelled with TransCancel.

TransCancel will release the seats back to the unpaid booking.

Message specific return codes:

92	The requested number of seats is not available.

CompleteGroupBooking

Purpose:

Completes the order for seats from a group booking and permanently removes the seats from the original unpaid booking.

Use:

Behaves just like a TransComplete for a regular booking except it also removes the seats from the original unpaid booking.

Parameters (in order):

Group Booking Ref	The Group Booking Reference Number as returned by the first call to ReloadGroupBooking after the initial unpaid booking is made.
Number of Tickets	Number of tickets printed (for a booking this is usually 0).

Data Returned:

The usual header information: ....|COMPLETEGROUPBOOKING|3|1| (3 rows of 1 columns)

The data following the header: | Group Booking Ref | Booking Number | Booking Id |

eg |FB605034|605035|I5SQOK2|

Comments:

The booking created with this function is exactly like a normal booking.

APPENDIX I

SAMPLE CODE

Disclaimer: This example visual basic 6.0 code snippet is intended merely to clarify the principles being used. It is not intended for use as-is.

```
Simple Ticket Order:

Sub OrderMyTickets()
    sClientId = "111.111.111.222"
    sServerName = "MyRemoteServer" ' (192.167.32.1)
    sSessionId = "10031"
    sDateTime = Format(Now, "YYYYMMDDHHNN")
    sSessionDateTime = Format(Me.txtSessionDate, "YYYYMMDDHHNN")

    'Create a list of trequested tickets formatted as : 'code | quantity | price
    '(-1 price indicates use cinema defined price)
    For i = 0 To Me.txtTTCode.UBound
        If Len(Me.txtTTCode(i).Text & "") <> 0 And Len(Me.txtQty(i).Text & "")
<> 0 Then
            nTickets = nTickets + 1
            sTicketList = Me.txtTTCode(i).Text & DELIM & Me.txtQty(i).Text &
DELIM & "-1" & DELIM
        End If
    Next i
    'Get SalesSrvr Object via DCOM on remote server
    Set oVSS = CreateObject("VistaSalesSrvr.KioskInterface", sServerName)
    If oVSS Is Nothing Then
        LogOutput ("Could Not Create Sales Srvr component @ " & sServerName)
        GoTo Out
    End If
    With oVSS
        sResult = .Cmd(sClientId, sTransId, "Register", _
                  "TestWorkstation", "VSSTest", "0000", _
                  sDateTime, "NONE", "WWW")

        If Len(sResult) <> 0 Then
            nReturn = .Execute
            If nReturn <> 0 Then
                LogOutput ("Error: Failed Register Execute")
                GoTo Out
            End If
        Else
            LogOutput ("Error: Failed Register Command")
            GoTo Out
        End If
        'Start new order
        sResult = .Cmd(sClientId, sTransId, "TransNew", _
                       "", "", "", "", "", "")
        nReturn = .Execute
```

```
        If nReturn = 0 Then
            sResult = .GetData()
        Else
            LogOutput ("Error: Failed Trans New Execute")
            GoTo Out
        End If
        'parse return for TransIdTemp used for this order
        sResult = Right(sResult, 12)
        sTransId = Left(sResult, 11)
        LogOutput ("TransIdTemp being used: " & sTransId)
        'Order Tickets using the ticket list we constructed earlier
        sResult = .Cmd(sClientId, sTransId, "OrderTickets", Me.txtSessionId,
sSessionDateTime, _
                  sTicketList, "N", "", "")
        nReturn = .Execute
        If nReturn <> 0 Then
            If nReturn = 3 Then
                LogOutput "Ordered Tickets OK but seats could not be allocated
contiguously - Cancelling Order"
                Call TransCancel(sTransId, oVSS)
            Else
                LogOutput ("OrderTickets Failed with return: " & nReturn & ".
Check VSS Log for more detail")
                Call TransCancel(sTransId, oVSS)
            End If
        Else
            LogOutput ("Order Tickets OK")
        End If
        sResult = .Cmd(sClientId, sTransId, "GetPaymentTotal", "", "", "", "",
"", "")
        nReturn = .Execute
        If nReturn <> 0 Then
            LogOutput ("GetPaymentTotal Failed with return: " & nReturn)
            Call TransCancel(sTransId, oVSS)
        Else
            sResult = .GetData
        End If

        'get payment value from return data - shld match client calc
        'remove standard header elements from return data
        For i = 1 To 6
            nPos = InStr(1, sResult, DELIM)
            sResult = Right(sResult, Len(sResult) - nPos)
        Next i
        sResult = Left(sResult, Len(sResult) - 1)
        'Total Value is first item
        nPos = InStr(1, sResult, DELIM)
        sValue = Left(sResult, nPos - 1)
        LogOutput ("Payment Value Being Used: " & sValue)
        'Get Trans Ref is a required call - returns booking ref etc
        sResult = .Cmd(sClientId, sTransId, "GetTransactionRef", "N", "", "",
"", "", "")
        nReturn = .Execute
        If nReturn <> 0 Then
            LogOutput ("GetTransactionRef Failed with return: " & nReturn)
            Call TransCancel(sTransId, oVSS)
        End If

        sResult = .Cmd(sClientId, sTransId, "PaymentStarting", sValue, "", "",
```

```
"", "", "")
        nReturn = .Execute
        If nReturn <> 0 Then
            LogOutput ("PaymentStarting Failed with return: " & nReturn)
            Call TransCancel(sTransId, oVSS)
        End If

        sResult = .Cmd(sClientId, sTransId, "PaymentOK", "VISA",
"41111111111111", "10/05", _
                    "GRANTSMITH", "N", "0213333333")
        nReturn = .Execute
        If nReturn <> 0 Then
            LogOutput ("PaymentOK Failed with return: " & nReturn)
            Call TransCancel(sTransId, oVSS)
        End If
        'Finally complete the order
        'zero tickets printed via this client
        sResult = .Cmd(sClientId, sTransId, "TransComplete", "0", "", "", "",
"", "")
        nReturn = .Execute
        If nReturn <> 0 Then
            LogOutput ("TransComplete Failed with return: " & nReturn)
            Call TransCancel(sTransId, oVSS)
        End If
    End With
End Sub
```

Process Stock Receipt:

```
Sub ProcessStockReceipt()
    'Get SalesSrvr Object via DCOM on remote server
    Set oVSS = _
    CreateObject("VistaSalesSrvr.KioskInterface", sServerName)
    If oVSS Is Nothing Then
        LogOutput ("Could Not Create Sales Srvr component @ " & sServerName)
        GoTo Out
    End If
    'Create a list of Items to process - include a sequence number with     'each
one which must be sequential and begin at one.
    nSeq = 0
    For i = 0 To MyStockReceipts.Count - 1
        nSeq = nSeq + 1
        sStockItems = sStockItems & " | " & nSeq
        sStockItems = sStockItems & " | " & MyStockReceipts(i).ItemCode
        sStockItems = sStockItems & " | " & MyStockReceipts(i).Quantity
        sStockItems = sStockItems & " | " & MyStockReceipts(i).Location
        sStockItems = sStockItems & " | " & MyStockReceipts(i).UnitMeas
        sStockItems = sStockItems & " | " & MyStockReceipts(i).UOMPrice
    Next i
    'Add last delimiter
    sStockItems = sStockItems & " | """
    With oVSS
        sResult = mobjKioskSrvr.Cmd(sClientId, "", "PROCESSSTOCKRECEIPT", _
                sVendorCode, sReceiptReference, "Y", sStockItems, "N", "")
```

```
        nReturn = mobjKioskSrvr.Execute
        Select Case nReturn
        Case 0, 32
            ' Parse return data to get transaction number and LastSeqNumber
processed
            ' remove standard header elements from return data
            For x = 1 To 6
                nPos = InStr(1, sResult, DELIM)
                sResult = Right(sResult, Len(sResult) - nPos)
            Next x
            sResult = Left(sResult, Len(sResult) - 1)
            'Trans Number is first item
            nPos = InStr(1, sResult, DELIM)
            sVistaTransNo = Left(sResult, nPos - 1)
            'Last Seq Number is second item
            nPos2 = InStr(nPos + 1, sResult, DELIM)
            sSeqNo = Mid(sResult, nPos + 1, _
            nPos2 - (nPos + 1))
            If nReturn = 0 Then
                LogOutput "Process Stock Receipt OK"
                'Store Transaction Number and sequence numbers in internal data
store
                SaveReceipt sVistaTransNo, sSeqNo
            Else
                'Partially committed to Vista - get last successful sequence
                'Save Successful transactions
                SaveReceipt sVistaTransNo, sSeqNo
                'and attempt to resubmit remaining items later or inform user
                LogOutput "Process Stock Receipt Partially Committed - items
" & nSeqNo + 1 & " onwards could NOT be processed"
            End If
        Case 31
            'Invalid Data input
            LogOutput ("Error: Input Data Not Valid")
            GoTo Out
        Case Else
            LogOutput "Error: Unexpected Error " & nReturn
            GoTo Out
        End Select
    End With
End Sub
```

APPENDIX II

EXAMPLE XML OUTPUT

Example Output from GetSellingDataXMLStream/File:

NOTE: This is an example of original structure which will be supplemented as Vista sees appropriate, check with Vista to ensure you are aware of the complete data set. Your parsing engine should accommodate additional fields and table elements without error. For this reason you should not use a validating schema document model when parsing output data, unless it can accommodate additional unexpected elements.

```
<?xml version="1.0"?>
<Vista_Data xmlns="Vista.co.nz\XMLOutput\SalesServer">
      <!--Vista Entertainment Solutions - All Rights Reserved-->
      <Sessions xmlns="">
<Session>
<Cinema_ID>SiteCode</Cinema_ID>
<Movie_ID>BN00000031</Movie_ID>
<Session_ID>3020</Session_ID>
<Date_time>20030106120000</Date_time>
<Day_of_week>2</Day_of_week>
<Seats_Available>233</Seats_Available>
<Seat_allocation_on>N</Seat_allocation_on>
<Price_group_code>1</Price_group_code>
<ChildrenRestricted>Y</ChildrenRestricted>
<Session_strHOSessionID></Session_strHOSessionID>
<CinOperator_strCode>BN</CinOperator_strCode>
<Session_strNoFreeList>Y</Session_strNoFreeList>
<Screen_bytNum>1</Screen_bytNum>
<Screen_strName>Screen1</Screen_strName>
<SType_strSessionTypeCode>N</SType_strSessionTypeCode>
<SType_strDescription>Normal Session</SType_strDescription>
</Session>
      </Sessions>
      <SessionAreaCategories xmlns="">
<SessionAreaCategory>
<Cinema_ID>SiteCode</Cinema_ID>
<Session_lngSessionId>3020</Session_lngSessionId>
<AreaCat_strCode>0000000002</AreaCat_strCode>
</SessionAreaCategory>
            <SessionAreaCategory>
                        <Cinema_ID>SiteCode</Cinema_ID>
<Session_lngSessionId>3021</Session_lngSessionId>
<AreaCat_strCode>0000000001</AreaCat_strCode>
            </SessionAreaCategory>
      </SessionAreaCategories>
      <PriceBookRefreshRequired>20020803000000 </PriceBookRefreshRequired>
      <ConcessionItems xmlns="">
```

```
<Concession>
<Cinema_ID>SiteCode</Cinema_ID>
<ItemCode>156</ItemCode>
<ItemDescription>Combo Small Drink &amp;amp; Popcorn
</ItemDescription>
<Item_price>80</Item_price>
<PromoCode></PromoCode>
<PriceListCode></PriceListCode>
<Class_strCode>BAR</Class_strCode>
<SalesTaxCode>2</SalesTaxCode>
</Concession>
      </ConcessionItems>
      <Prices xmlns="">
<Price>
<Cinema_ID>SiteCode</Cinema_ID>
<Ticket_type_code>0001</Ticket_type_code>
<Ticket_type_description>Adult Standard
</Ticket_type_description>
<Ticket_type_description_2></Ticket_type_description_2>
<Price_group_code>1</Price_group_code>
<Ticket_Price>1500</Ticket_Price>
<Child_ticket>Y</Child_ticket>
<AreaCat_strCode>0000000002</AreaCat_strCode>
<AreaCat_intSeq>1</AreaCat_intSeq>
<SalesTaxCode>1</SalesTaxCode>
<Price_intSequence>1</Price_intSequence>
</Price>
      </Prices>
      <Movies xmlns="">
<Movie>
<Cinema_ID>SiteCode</Cinema_ID>
<Movie_ID>BN00000031</Movie_ID>
<Movie_Name>A BRIDGE TOO FAR</Movie_Name>
<Movie_shortname>A bridge too far</Movie_shortname>
<Rating>U</Rating>
<Rating_description>Contains violence</Rating_description>
<Movie_list_pos>50</Movie_list_pos>
<Movie_FCode>10031</Movie_FCode>
<ChildrenRestricted>Y</ChildrenRestricted>
<Movie_Name_2></Movie_Name_2>
<Movie_shortname_2></Movie_shortname_2>
<Movie_strRating_2></Movie_strRating_2>
<Movie_strRating_Description_2>
</Movie_strRating_Description_2>
<Movie_HOCode></Movie_HOCode>
</Movie>
      </Movies>
</Vista_Data>
```

APPENDIX III

EXAMPLE 2 CHARACTER CODE INTERPRETATION

The following is a section of java script detailing how to interpret 2 chr encoding used for seat coordinates returned from GetSessionSeatData.

```
public void loadCodeString() {
      codeString = new java.util.Vector(62);
      codeString.addElement(new String("0"));
      codeString.addElement(new String("1"));
      codeString.addElement(new String("2"));
      codeString.addElement(new String("3"));
      codeString.addElement(new String("4"));
      codeString.addElement(new String("5"));
      codeString.addElement(new String("6"));
      codeString.addElement(new String("7"));
      codeString.addElement(new String("8"));
      codeString.addElement(new String("9"));
      codeString.addElement(new String("A"));
      codeString.addElement(new String("B"));
      codeString.addElement(new String("C"));
      codeString.addElement(new String("D"));
      codeString.addElement(new String("E"));
      codeString.addElement(new String("F"));
      codeString.addElement(new String("G"));
      codeString.addElement(new String("H"));
      codeString.addElement(new String("I"));
      codeString.addElement(new String("J"));
      codeString.addElement(new String("K"));
      codeString.addElement(new String("L"));
      codeString.addElement(new String("M"));
      codeString.addElement(new String("N"));
      codeString.addElement(new String("O"));
      codeString.addElement(new String("P"));
      codeString.addElement(new String("Q"));
      codeString.addElement(new String("R"));
      codeString.addElement(new String("S"));
      codeString.addElement(new String("T"));
      codeString.addElement(new String("U"));
      codeString.addElement(new String("V"));
      codeString.addElement(new String("W"));
      codeString.addElement(new String("X"));
      codeString.addElement(new String("Y"));
      codeString.addElement(new String("Z"));
      codeString.addElement(new String("a"));
      codeString.addElement(new String("b"));
      codeString.addElement(new String("c"));
```

```
      codeString.addElement(new String("d"));
      codeString.addElement(new String("e"));
      codeString.addElement(new String("f"));
      codeString.addElement(new String("g"));
      codeString.addElement(new String("h"));
      codeString.addElement(new String("i"));
      codeString.addElement(new String("j"));
      codeString.addElement(new String("k"));
      codeString.addElement(new String("l"));
      codeString.addElement(new String("m"));
      codeString.addElement(new String("n"));
      codeString.addElement(new String("o"));
      codeString.addElement(new String("p"));
      codeString.addElement(new String("q"));
      codeString.addElement(new String("r"));
      codeString.addElement(new String("s"));
      codeString.addElement(new String("t"));
      codeString.addElement(new String("u"));
      codeString.addElement(new String("v"));
      codeString.addElement(new String("w"));
      codeString.addElement(new String("x"));
      codeString.addElement(new String("y"));
      codeString.addElement(new String("z"));

  }
public int getIntValue(String s) {
      return codeString.indexOf(s);
}

public int getRealValue(String s) {
int value = 0;
      if (s==null  |  |  s.equals("")) return -1;
      else {
            for (int i=0; i<s.length(); i++) {
                  System.out.println("value at " + i + " is " + value);
                  value += getIntValue(s.substring(s.length() - i - 1,
s.length() - i)) * ((int) java.lang.Math.pow(codeString.size(), i));
            }
            System.out.println("value :" + value);
            return value;
      }
}
```

APPENDIX IV

EXECUTE COMMAND RETURN CODES

ExecuteOK = 0	Execution completed successfully
ExecuteInvalidCmdName = 1	Invalid command request
ExecuteNoSeatsAvailable = 2	Not enough seats are available for sale to fulfil request
ExecuteAllocSeatsFail = 3	Could not allocate all seats in a contiguous block
ExecuteInvalidData = 4	Parameters passed are not correct
ExecuteUnexpectedResult = 5	An unexpected error/database error/component error has occured
ExecuteBookingPickupPrevErr = 9	There was a previous printing problem when trying to pickup tickets from kiosk.
ExecuteBookingPickupNotFound = 10	The selected booking could not be found
ExecuteBookingPickupDone = 11	The selected booking has already been picked up
ExecuteBookingPrintError = 12	The number of tickets printed does not match the number the client has recorded as printed
ExecutePackageItemFail = 13	Could not order concession elements of the ticket package
ExecutePackagePriceError = 14	The value of the ticket package requested does not match the total value of the package as calculated by sale server
ExecuteUnpaidBookingClosed = 15	Unpaid bookings are no longer permitted due to cinema session time/unpaid booking cutoff restrictions
ExecutePaymentRequestFalse = 16	Payment was declined OR an error occurred within the VistaPayment component - client must check the Payment Module return strings for details.
ExecuteUnpaidBookingDoesNotExist = 17	The selected booking does not exist
ExecuteBookingNotUnpaid = 18	The selected booking is not unpaid and cannot be released
ExecuteBookingAlreadyReleased = 19	The selected booking has already been released

ExecuteNoTransLineToCancel = 20	There were no elements in the selected transaction liable for refund
ExecuteNoPaidWithVoucher = 21	Paid orders are NOT permitted when order includes voucher ticket types
ExecuteRefundBookingCancelled = 22	Requested Booking is cancelled
ExecuteRefundBookingUnpaid = 23	Requested Booking is Unpaid
ExecuteRefundBookingPickedUp = 24	Requested Booking is already picked up
ExecuteEditBKPickedUp = 25	Requested Edit Booking already picked up
ExecuteEditBKCancelled = 26	Requested Edit Booking cancelled
ExecuteEditBKInProgress = 27	Requested Edit Booking already in edit mode
ExecuteExistBkStatusChange = 28	Requested Edit Booking status changed by another process
ExecuteNoUnpaidClient = 29	Client App is not authorised to create unpaid bookings
ExecuteOutputNotRequired = 30	Selling Data Output is not required - no reference data changes since last output to client
ExecuteStkRecInvalidInput = 31	Stock Receipt Request included invalid input data
ExecuteStkRecPartialComplete = 32	Stock Receipt requested partially completed - check last successful sequence return data to identify failure point.
ExecuteBookingPaid = 33	Unpaid Booking Payment OK - Booking already paid
ExecuteBookingPaid = 34	Unpaid Booking Processed as paid, however Collected flag could not be set - customer should still receive tickets/vouchers
ExecuteBookingNotLoaded = 35	Unpaid Booking Processing requested but the Unpaid Booking has not been initialised by calling GetUnpaidBookingPrintStream
ExecuteBadSurchargeCount = 36	When calling Add Surcharge the count of elements and element definitions did not equate to a whole number
ExecuteBadSQL = 37	When calling GetSQLXMLStream the SQL statement passed was invalid according to Vista restrictions
ExecuteSQLFail = 38	When calling GetSQLXMLStream the SQL statement passed failed to execute
ExecuteNoRows = 39	When calling GetSQLXMLStream the SQL statement passed returned zero rows
ExecuteWGroupRefundCutOffExpired =	The Refund Cut Off time defined for the client apps

40	Workstation Group has expired for at least one session in the list of booked seats to be removed
ExecuteErrSalesChannel = 41	The client applications Sales Channel is not permitted to Order Tickets for the selected session
ExecuteBookingHasBrokenSeats = 42	A booking (unpaid or paid) belonging to the card, or booking number, includes seats which have been marked as broken. Customer should be instructed to visit a POS to pickup the booking.
ExecuteReceiptTemplateMissing = 43	The client application has requested a receipt only print stream, and the Receipt.txt printer template file is not found for the requested Printer Type.
ExecuteExecuteBookingExpired = 44	The requested booking has no sessions that have not started.
ExecuteLoginError = 50	Invalid Logon Details
ExecuteInvalidWSTN = 51	Invalid WorkstationId
ExecuteClientForWSTN = 52	WorkstationId represented with different clientID
ExecuteWSTNForClient = 53	ClientID represented with different ClientID
ExecuteUserLoggedIn = 54	User logged on elsewhere
ExecuteLoyaltyUpdateFailed = 55	A Loyalty transaction update was required, but could not be completed. NOTE: Transaction is NOT automatically cancelled by Sales Server, to retain an Audit trail of failed transactions.
ExecutePaymentsIncomplete = 56	Payments have not been made which total the value of the order
ExecuteBadBranch = 57	Usher point specific return code
ExecuteBookingNotPaid = 58	Usher point specific return code
ExecuteAlreadyEntered = 59	Usher point specific return code
ExecuteAlreadyExited = 60	Usher point specific return code
ExecuteEventMultiSess As Integer = 61	Usher point specific return code
ExecuteEventMultiSessConc As Integer = 62	Usher point specific return code
ExecuteEventConcExist As Integer = 63	Usher point specific return code
ExecuteEventCollectFail = 64	Usher point specific return code
ExecuteEventMultiTix = 65	Usher point specific return code
ExecuteEventRedemptions = 66	Usher point specific return code
ExecuteEventInvTime = 67	Usher point specific return code

ExecuteEventMULTI_TIX_INVTIME = 68	Usher point specific return code
ExecuteEventMULTI_SESSION_INVTIME = 69	Usher point specific return code
ExecuteEventCONC_EXIST_INVTIME = 70	Usher point specific return code
ExecuteEventMULTI_SESSION_CONC_INVTIME = 71	Usher point specific return code
ExecutePrepPickupFail = 72	The order included concessions which required preparation/pickup, but an error was committing this data to the system.
ExecuteTabDetailExists = 73	The requested Tab details are already in use by another active, unpaid Tab.
ExecuteTabLimitExceeded = 74	The requested Tab order limit has been exceeded.
ExecuteInvalidBarcode = 75	The barcode string supplied does not contain valid content for the requested context.
ExecuteSessionUnavailable = 76	At least one session requested, or in a booking or order being accessed has been marked as unavailable. This usually indicates critical maintenance is occurring on the session - such as being swapped to another screen. No booking collection is possible during such maintenance, but the situation should be resolved within a short period of time.
ExecuteProviderInitError = 80	Failed to initialize Vista Member Manger. visMemberMgr.dll and visMemberMgr.reg should be installed into the same folder where Sales Server runs. Member Manger should also be registered by manually executing visMemberMgr.reg file
ExecuteProviderPreloadError = 81	Member data preload failed by Member Manger. Member Provider can't find the member information with given criteria
ExecuteProviderDataError = 82	Member Day of Birth is in invalid format
ExecuteProviderValidateError = 83	Member Manger failed to validate
ExecuteProviderDBError = 84	The transaction is not held in Member Manger stateless call cache
ExecuteProviderUnknowError = 85	Unknow Error Generated by Member Manger
ExecuteProviderUnknowError = 86	Ticket type doesn't belong to member provider
ExecuteProviderRequireSubError = 87	Requires membership subscription
ExecuteProviderNotApprovedError = 88	Member ticket order request is not approved by provider

ExecuteProviderCommittError = 90	The transaction contains member provider tickets, error occurred during committing the transaction in member manager system

Index

R

S

T

U

V