Agile Ticketing Sales API Guidelines

This document is meant to be a guideline for organizations making use of the Agile Ticketing API for event ticketing services. It will provide details about some of the more difficult concepts in the Agile Ticketing system. Also, it will provide a step-by-step flow through the API calls and how they should be interpreted. API method names in this document link to the online document for the specific method.

1. Required Data For API Calls

The Agile Ticketing API requires certain authentication tokens and data to be able to successfully call the API methods. These data items will be provided by Agile Ticketing for use with the API. Every method in the API set will require the following values.

API URL – The URL for making API calls. This could be different depending on what server the organizations data resides on. This is usually http://prod1.agileticketing.net/api/sales.svc/xml/{MethodName}?{parameter1= value}

appKey – A key identifying the application making the call. This is given to you as a globally unique identifier. (e.g. CF369DF2-C67B-4848-AE23-EC34253C1BD2)

userKey – A key identifying the user making the call. The user has a permission set defined that control what areas of the system they have access to. This is given to you as a globally unique identifier. (e.g. 21016F86-77DA-4442-BE17-16397D8F2F9A)

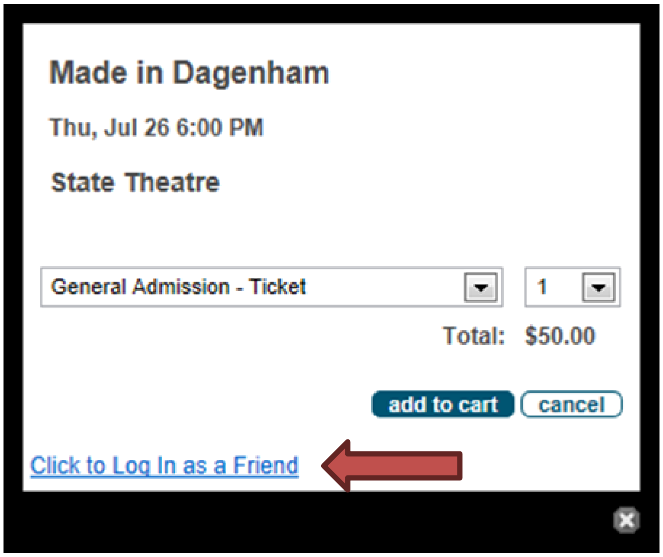
corpOrgID – The numeric identifier for the ticketing organization (e.g. Film Festival or University).

Other data items are also provided by Agile Ticketing that may be needed for certain API calls.

eventOrgID – This is the numeric identifier for the event level organization. This is the area where events are grouped together in the system. Some organizations may actually have more than one event level organization so they can separate out the different types of events. (e.g. Regular Programs, Special Events, and Shorts Programs)

buyerTypeID – The BuyerType in the Agile Ticketing system is our method of segmenting sales to different types of buyers. Organizations normally deal with the standard public, members, and groups. Agile Ticketing will provide the buyerTypeID value for the standard BuyerType. This value will not change once the organization has been established. This will be the buyerTypeID used to call methods like **EventListPrices** when first getting prices for an event.

Member Login – A true/false indicator that the organization will have members that need to log in for special pricing. This can be used to control the "Click to Log In as a member" link on the ticket selection pop-up. If this organization is not a membership organization then this link should be hidden.



Made in Dagenham

Thu, Jul 26 6:00 PM

State Theatre

General Admission - Ticket

1

Total: $50.00

2. Listing Events

The **EventList** API method is used to list events stored in the Agile Ticketing system. The Agile Ticketing system keeps a full set of information about events including the name, start date, end date, venue information, description information, image locations, sales messages, and sales state.

API example URL for the Event List method
https://prod1.agileticketing.net/api/sales.svc/xml/EventList?appkey={APPKEY}&userkey={USERKEY}&corporgid={CORPORGID}&eventorgid={EVENTORGID}&buyertypeid={BUYERTYPEID}&startdate={STARTDATE}&enddate={ENDDATE}

Method Specific Parameters

Parameter Name	Type	Optional	Description
eventOrgID	int	X	The event level organization for listing events
buyerTypeID	int	X	The BuyerType ID to list for listing events
startDateTime	datetime		The start date for listing events
endDateTime	datetime		The end date for listing events

Parameter Details

eventOrgID – This is the numeric identifier for the event level organization. This is the area where the events are grouped together in the system. This optional parameter can be sent in to get only the events under the specific organization. A ticketing administrator may choose to separate events out in the Agile Ticketing system. By using this parameter the events can also be separated out during the sales process. If this parameter is not defined then all event under the corp organization will be returned.

buyerTypeID – This is the numeric identifier for the pricing profile. The BuyerType in our system is our method of segmenting sales to different types of buyers. This optional parameter can be sent in to get only events available on the specific BuyerType. This is often used to separate public events versus member events. To provide a list of member only events the member BuyerType ID should be sent to this method. If this parameter is not defined then events available on all web BuyerTypes will be returned.

startDateTime – This date/time is used as the start of the range for listing events. Events that occur starting after this date will be listed. This parameter is required.

endDateTime – This date/time is used as the end of the range for listing events. Events that occur starting before this date will be listed. This parameter is required.

Response Details

Most of the data items are fairly self-explanatory. The SalesMessage, SalesState, DateTBD, ShowEndDate, and Showtime values may require some explanation. The SalesState and SalesMessage values will only be used if a specific buyerTypeID parameter is defined. If the buyerTypeID is omitted the default SalesState of 2 and SalesMessage will be returned for all events.

SalesState – This is a numeric indicator of the sales status of the event. It will be returned as a value between 1 and 5. The only state where the event should be purchased is if the SalesState is 2.

Value	Description
1	This value indicates that the event is not sellable yet. Each event has a specific start sale date for each BuyerType. In this case the start date has not been reached yet.
2	This value indicates that the event is ready to be sold.
3	This value indicates that sales have ended for the event, but the event start date has not passed.
4	This value indicates that the event date has passed.
5	This value indicates that a custom message must be displayed to the user and that the event should not be sold.

SalesMessage – This should be the text displayed on the link or button for the customer to click on to purchase the event. This message will change depending on what the current SalesState is.

SalesState Value	Example Message
1	On Sale: 10/20/2013 9:00 AM
2	Buy Tickets, Add To Cart, Purchase
3	Check with the box office
4	Event is over
5	Sold Out, Rush Only, Check Back Soon

DateTBD – This true/false value indicates that actual date of the event has yet to be determined. Instead of displaying the StartDate value for the event something like "Date TBD" should be displayed. Normally the event StartDate is displayed as a numeric date including the time of the event or formatted out by spelling out the day, month, year, and time values. (e.g. 10/13/2013 8:00 AM or Sunday, October 13, 2013 8:00 AM)

ShowEndDate – This true/false value indicates that the EndDate of the event should be shown with the StartDate. The dates should be displayed something like "10/13/2013 8:00 AM – 10/15/2013 9:00 PM". This value is normally used for the display of multi-day events.

ShowTime – This true/false value indicates that the time value should be displayed when showing the StartDate and EndDate values. When set to false the dates should be displayed something like "10/13/2013 – 10/15/2013" or just "10/13/2013" when the ShowEndDate value is also false.

3. Displaying Event Details

The **EventGet** API method is used to get details about an event stored in the Agile Ticketing system. The Agile Ticketing system keeps a full set of information about events including the name, start date, end date, venue information, description information, and image locations.

The **EventGetIDFromExternalID** API method may also be needed depending on how events were set up in the Agile Ticketing System. Each event in the Agile Ticketing system has a system defined numeric EventID value defined. We also support the ability to define a string based ExternalEventID value that can be used to link the event in Agile with an event in an external system. If events are listed from an external source, then this method can be used to get the Agile EventID based on the ExternalEventID. The returned EventID can then be used to get details about the event from the Agile Ticketing API. For more information about this method please see the full API documentation on our website.

The **EventGetDescription** API method may also be needed to display a full description of the event. The Agile Ticketing system provides a short description of the event in the **EventList** and **EventGet** methods. If the full description of the event needs to be displayed then call this method with the EventID to get the full description. This description data is separated from the event so it does not have to pulled down with every list or get API call. The full description data can be pulled back when the customer clicks on a "(More Info)" link to see all the description information. For more information about this method please see the full API documentation on our website.

API example URL for the Event Get method
https://prod1.agileticketing.net/api/sales.svc/xml/EventGetDescription?appkey={APPKEY}&userkey={USERKEY}&corporgid={CORPORGID}&eventid={EVENTID}

Method Specific Parameters

Parameter Name	Type	Optional	Description
eventID	long		The numeric Identifier for the event.

Response Details

Most of the data items are fairly self-explanatory. The SalesMessage, SalesState, DateTBD, ShowEndDate, and Showtime values may require some explanation.

SalesState – Will always be 0 for this method. It is not to be evaluated in this return set.

SalesMessage – Will always be empty for this method. It is not to be evaluated in this return set.

DateTBD – This true/false value indicates that actual date of the event has yet to be determined. Instead of displaying the StartDate value for the event something like "Date TBD" should be displayed. Normally the event StartDate is displayed as a numeric date including the time of the event or formatted out by spelling out the day, month, year, and time values. (e.g. 10/13/2013 8:00 AM or Sunday, October 13, 2013 8:00 AM)

ShowEndDate – This true/false value indicates that the EndDate of the event should be shown with the StartDate. The dates should be displayed something like "10/13/2013 8:00 AM – 10/15/2013 9:00 PM". This value is normally used for the display of multi-day events.

ShowTime – This true/false value indicates that the time value should be displayed when showing the StartDate and EndDate values. When set to false the dates should be displayed something like “10/13/2013 – 10/15/2013” or just “10/13/2013” when the ShowEndDate value is also false.

4. Checking Sales Status For An Event

The **EventSalesStatus** API method is used to get information about whether an event is sellable. When first pulling up an event with an unauthenticated customer this method should be called with the standard buyerTypeID and the eventID for the selected event. This method has items that will control the flow of the next steps in the process.

API example URL for the Event Sales Status method
https://prod1.agileticketing.net/api/sales.svc/xml/EventSalesStatus?appkey={APPKEY}&userkey={USERKEY}&corporgid={CORPORGID}&eventid={EVENTID}&buyertypeid={BUYERTYPEID}

Method Specific Parameters

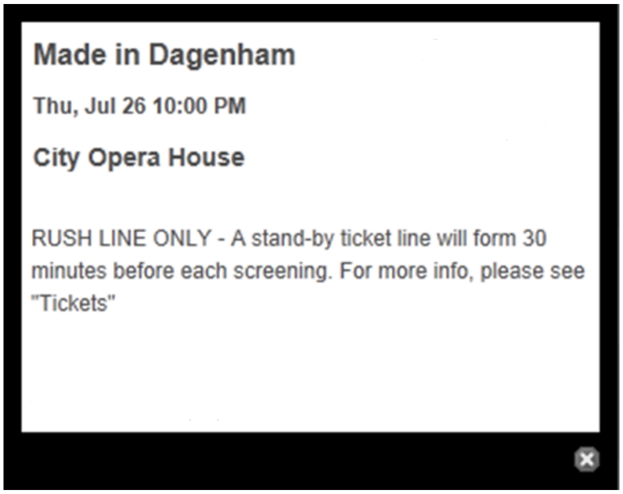
Parameter Name	Type	Optional	Description
eventID	long		The numeric Identifier for the event.
buyerTypeID	int		The numeric identifier for the standard or member BuyerType.

Response Details

The true/false values for OnSale and DisplayMessage are important to whether the event is shown to be sellable or not.

OnSale – This true/false value indicates whether the event is on sale or not. This value should be overridden by the DisplayMessage value if it is true.

DisplayMessage – This true/false value indicates that a custom message should be displayed to the customer instead of showing any prices. The event should not be sold if this value is true. The MessageText string value and AvailabilityText string value should be shown on the lightbox as shown below.



Made in Dagenham

Thu, Jul 26 10:00 PM

City Opera House

RUSH LINE ONLY - A stand-by ticket line will form 30 minutes before each screening. For more info, please see "Tickets"

The MessageText value should be shown where the "RUSH LINE ONLY" text is. The AvailabilityText is a longer string value and should be shown where the "A stand-by ticket line. . ." is. Allowing this text to be different per event allows the ticketing administrator to customize messages to the end customer. Certain events may need to display some custom message to the customer once it is no longer available. This is also how our system takes care of saying the event is sold out. We have an internal process that sets the DisplayMessage value and MessageText value once the event inventory has reduced to 0.

EnforceQuantities – This true/false value indicates that the min/max number of tickets that the customer should be allowed to purchase for this event. This is a per order setting. The MinQtyPerOrder and MaxQtyPerOrder will define the range of tickets allowed to be purchased for this event. These values should be paid attention to otherwise the **TicketsAdd** API method could return error #1033 - InvalidTicketQtyRequest.

StartSales and **EndSales** – These date/time values define the date period in which this event is on sale for the specified BuyerType. If the OnSale and DisplayMessage values are both false you should probably evaluate the dates to display a friendly message to the customer. (e.g. If the current date is before StartSales then display “On Sale: 10/13/2013 8:00 AM”. Or if the current date is after EndSales then display “Sales Ended: 10/13/2013 9:00 PM”.)

5. Listing Prices For An Event

If the data from the **EventSalesStatus** method has been evaluated and the event is sellable then prices should be retrieved for the event by calling the **EventListPrices** API method. This method will list the tiers and prices available for an event.

API example URL for the Event List Prices method
https://prod1.agileticketing.net/api/sales.svc/xml/EventListPrices?appkey={APPKEY}&userkey={USERKEY}&corporgid={CORPORGID}&eventorgid={EVENTORGID}&eventID={EVENTID}&buyertypeid={BUYERTYPEID}&memberid={MEMBERID}&orderid={ORDERID}&transactionID={TRANSACTIONID}&promocode={PROMOCODE}

Method Specific Parameters

Parameter Name	Type	Optional	Description
eventID	long		The numeric Identifier for the event.
buyerTypeID	int		The numeric identifier for the standard or member BuyerType.
memberID	long	X	The numeric identifier for the member so benefits can be retrieved. If this is a member purchase this value will be required.
orderID	long	X	The numeric identifier for the order so that already added ticket quantities can be evaluated. This value should be sent in if an order has been started and this is a member purchase.
transactionID	long	X	The numeric identifier for this transaction so that already added ticket quantities can be evaluated. This value should be sent in if an order has been started and this is a member purchase.
promoCode	string	X	The promo code string for listing restricted promo prices. This value must be provided to retrieve the level 1 – 3 restricted promo prices from this method.

Response Details

The response from this method will control quite a few details on the ticket selection page. The Name, AvailableQty, ShowAvailableQty, SoldOutText, and TierTime should control the display of the tier area of the ticket selection. The TicketType, BasePrice, MinPerOrder, MaxPerOrder, and PromoCodeRequired should control the display of the ticket type area.

Name – The name of the tier.

TierTime – The time related to this tier. If it is provided it should be displayed next to the tier name. (e.g. Tier Name – 9:00 AM)

ShowAvailableQty – This true/false value indicates that the customer should see the currently available quantity of ticket in the AvailableQty value.

SoldOutText – The text that should be shown to the user once the tier AvailableQty value has reached 0.

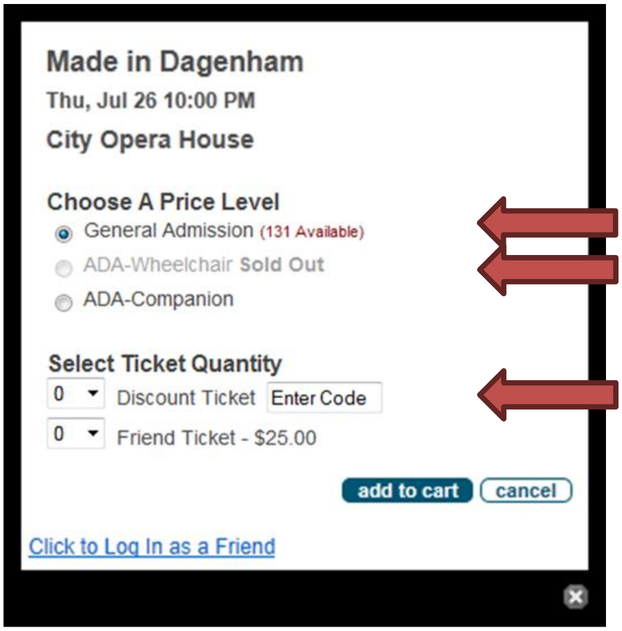
TicketType – The name of the ticket being purchased.

BasePrice – The base price of the ticket. This value does not include the fees and taxes related to the ticket.

MinPerOrder and **MaxPerOrder** – These numeric values define the minimum and maximum quantities required if this ticket type is being purchased. This is especially important when dealing with member tickets that are usually restricted to a certain number during the festival. The Agile API will reduce the number of available benefit tickets available to the member as tickets are added to the cart. This is why it is important to send the orderID and transactionID parameter to the **EventListPrices** method once an order has started. This mechanism gives the member feedback on how many benefit tickets they have left as they are adding tickets through the API interface. (e.g. If MinPerOrder = 2 and MaxPerOrder = 4 then the dropdown for this ticket should contain 0, 2, 3, 4 as the possible ticket quantities.)

PromoCodeRequired – This true/false value indicates that a promotion code is required for adding this ticket. Restriction Level 0 promo required tickets will be returned by default with this method. Agile Ticketing normally provides a code next to this ticket type to enter the code. A code can also be asked before listing prices and then use this to specify the promoCode parameter on this API method to bring back the level 1 – 3 restricted promo prices.

Example Ticket Selection Area With Multiple Tiers As A Separate Selection



Made in Dagenham

Thu, Jul 26 10:00 PM

City Opera House

Choose A Price Level

● General Admission (131 Available)

○ ADA-Wheelchair Sold Out

○ ADA-Companion

Select Ticket Quantity

0 ▾ Discount Ticket Enter Code

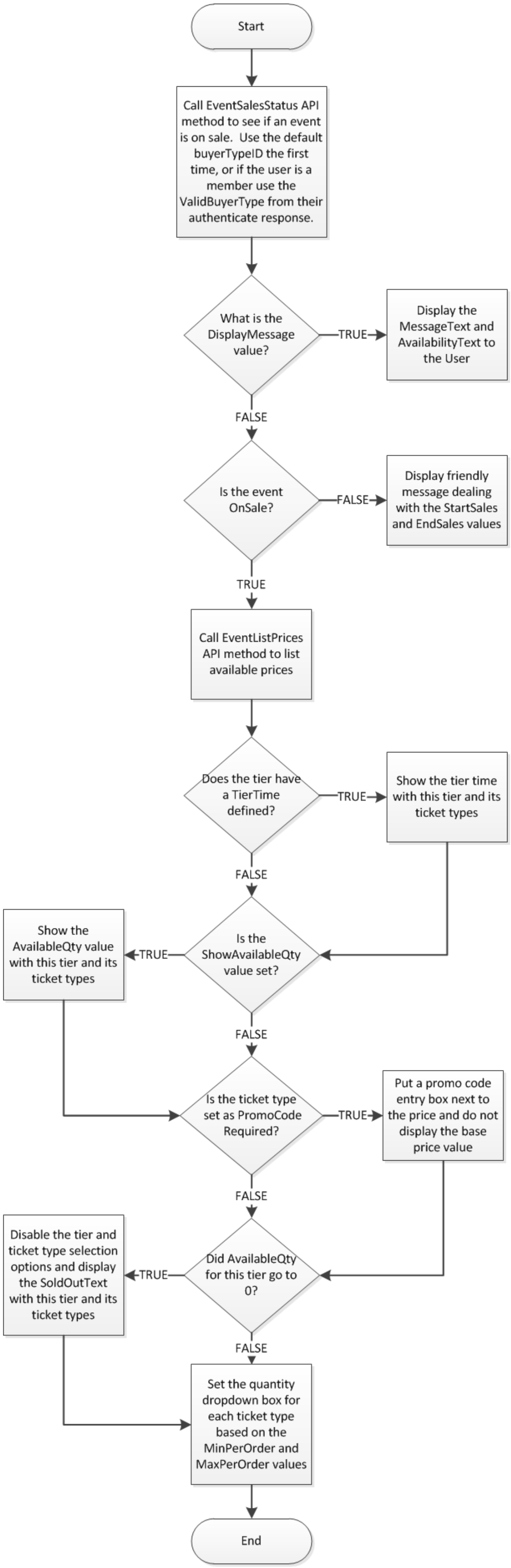
0 ▾ Friend Ticket - $25.00

ShowAvailableQty = true

AvailableQty went to 0, show SoldOutText

PromoCodeRequired

See the diagram below for the run-through of the listing prices process


Start
Call EventSalesStatus API method to see if an event is on sale. Use the default buyerTypeID the first time, or if the user is a member use the ValidBuyerType from their authenticate response.
What is the DisplayMessage value?
TRUE
Display the MessageText and AvailabilityText to the User
FALSE
Is the event OnSale?
FALSE
Display friendly message dealing with the StartSales and EndSales values
TRUE
Call EventListPrices API method to list available prices
Does the tier have a TierTime defined?
TRUE
Show the tier time with this tier and its ticket types
FALSE
Is the ShowAvailableQty value set?
TRUE
Show the AvailableQty value with this tier and its ticket types
FALSE
Is the ticket type set as PromoCode Required?
TRUE
Put a promo code entry box next to the price and do not display the base price value
FALSE
Did AvailableQty for this tier go to 0?
TRUE
Disable the tier and ticket type selection options and display the SoldOutText with this tier and its ticket types
FALSE
Set the quantity dropdown box for each ticket type based on the MinPerOrder and MaxPerOrder values
End

6. Starting an Order and Adding Tickets

The **OrderStart** API method should be called to start an order and tickets should be added right when the customer clicks the "add to cart" button. If any errors occur during the ticket add process they should be shown a friendly error message right here. The integrated application will need to keep up with the OrderID and TransactionID once the order has started. This can be done via a cookie or some other session token mechanism. When another event needs to be added via the Agile Ticketing API the **OrderStatus** API method should be called to make sure the order is still in process and has not expired.

API example URL for the Order Start method
https://prod1.agileticketing.net/sales.svc/xml/OrderStart?appkey={APPKEY}&userkey={USERKEY}&corporgid={CORPORGID}&buyertypeid={BUYERTYPEID}&customerid={CUSTOMERID}&memberid={MEMBERID}

Method Specific Parameters

Parameter Name	Type	Optional	Description
buyerTypeID	int		The numeric identifier for the standard or member BuyerType. If the member has logged in the ValidBuyerType should be used from the authentication response data, otherwise the standard buyerTypeID should be used.
customerID	long	X	The numeric identifier for the customer making the purchase. If this is a member purchase this value will be required.
memberID	long	X	The numeric identifier for the member so we can verify they can purchase on the restricted BuyerType. If this is a member purchase this value will be required.

API example URL for the Order Status method
https://prod1.agileticketing.net/sales.svc/xml/OrderStatus?appkey={APPKEY}&userkey={USERKEY}&corporgid={CORPORGID}&orderid={ORDERID}&transactionid={TRANSACTIONID}

Method Specific Parameters

Parameter Name	Type	Optional	Description
orderID	long		The numeric identifier for the current order.
transactionID	long		The numeric identifier for the current transaction.

Response Details

The **OrderStart**, **OrderStatus**, **OrderUpdate**, and **OrderCancel** methods all return the same type of response data. When using the **OrderStatus** method to check on the status of the order the InProcess and Expired values are very important. Tickets should only be added to an order where InProcess = true and Expired = false. If InProcess ever gets set to false then the customer has completed the order through the Agile Ticketing cart process. This order should no longer be kept up with in the integrated application, and a new order should be started. If the Expired value ever gets set to true then the transaction is no longer valid and the customer should get an error message that their time limit for keeping their tickets in the cart has expired. The Subtotal, OrderTotal, and ItemCount values can also be helpful to keep a very high level summary of the cart on the integrated site.

OrderID – The numeric order identifier.

OrderNumber – A string value representing the order. This is not necessarily the same as the OrderID.

TransactionID – The numeric identifier for the current ticketing transaction.

OpenDatetime – The date/time the transaction was started.

CloseDateTime – The date/time the transaction was finalized.

ExpirationDatetime – The date/time the transaction is set to expire.

Expired – An indicator that the transaction has expired. If this is ever true then the customer should get an error message that their cart has expired.

InProcess – An indicator that the transaction is in process. If this is set to false then the order has been finalized and no further ticket adding activity should occur on this order.

BuyerTypeID – A numeric identifier for the BuyerType on this order.

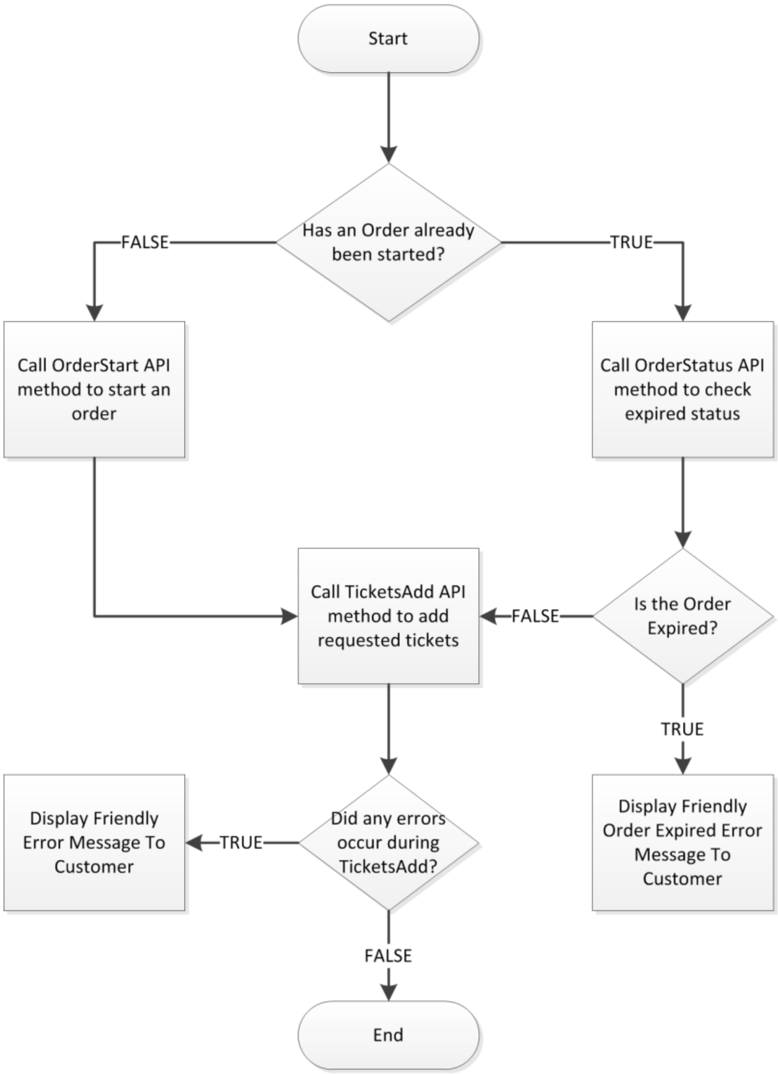
CustomerID and **ContactCustomerID** – Numeric identifiers for the customers tied to this order.

Subtotal – The current order total excluding any fees.

OrderTotal – The current order total including all fees.

ItemCount – The current number of tickets or items in the order.

See the diagram below for the run-through of the order start and tickets add process.


Start
Has an Order already been started?
FALSE
TRUE
Call OrderStart API method to start an order
Call OrderStatus API method to check expired status
Call TicketsAdd API method to add requested tickets
FALSE
Is the Order Expired?
TRUE
Display Friendly Error Message To Customer
TRUE
Did any errors occur during TicketsAdd?
Display Friendly Order Expired Error Message To Customer
FALSE
End

7. Customer/Member Login

Depending on the website the Agile Ticketing API is being used on the integration for customer login can be a somewhat tricky situation. The customer may or may not have a website account, and they may or may not have an Agile Ticketing account. Agile also provides the idea of membership programs that give ticket buying benefits to its members.

In the Agile Ticketing system a Customer and a Member are two distinct data records. The Customer record defines the basic customer information such as name, address, phone numbers, and email. The member record defines the membership program, joined date, expiration date, access to restricted BuyerTypes, and benefits to restricted ticket types. The two are related in that the member record has a reference to the customer record.

The Agile Ticketing API currently provides three authentication methods to authenticate the customer/member.

- **AuthenticateCustomer –** Authenticates the customer based on the Agile Ticketing unique username and password credentials.
- **AuthenticateEmail –** Authenticates the customer based on email address and password. Email address is not a unique customer field in the Agile Ticketing system. The Agile Ticketing API does a best choice approximation to get the customer based on email if there is more than one.
- **AuthenticateMember –** Authenticates the customer based on the Agile Ticketing unique member number and password.

The Agile Ticketing API also provides a mechanism for adding a new customer if needed.

- CustomerAdd – Adds a customer and returns the information needed to start a sale with the new customer.

Response Details

All three authenticate methods and the add method return the same type of response values that include the customer Identifier, customer name, and list of membership information for the customer. It is based on this information that we determine whether this customer is a member and which BuyerType ID should be used for listing prices and starting the order. It should be noted that a customer may have multiple memberships to choose from, and for each of these memberships they may have access to multiple restricted BuyerTypes to purchase under.

Authenticated – A true/false value indicating that this customer has been authenticated.

CustomerID – The numeric identifier for the authenticated customer. This value should be sent to the **OrderStart** or **OrderUpdate** API methods to set the customer on the order.

CustomerName – A string value containing the first and last name of the customer.

Members – A list of membership records containing the following fields.

> **MemberID –** A numeric identifier for the member. This value should be sent to the **EventListPrice** and **OrderStart** methods to be able to correctly utilize membership benefits
>
> **MemberNumber –** The string value for the member's number. This value is unique within a membership program.
>
> **MembershipID –** the numeric identifier for the membership program.

MembershipExpiration – The date/time value for when the membership expires.

ValidBuyerTypes – A list of restricted BuyerTypes this member has access to. This BuyerType value should be sent to the **EventListPrice** and **OrderStart** methods to correctly associate the member with the member restricted BuyerType they have access to.

For most integrations the **AuthenticateMember**, **AuthenticateEmail**, and **CustomerAdd** methods will need to be utilized. The **OrderUpdate** API method will also be used to allow adding a customer to an already started order. This will be helpful for those situations where a customer logs into their website account at some point during the ticket buying process.

The login process will be addressed at two points. The first is when the customer clicks on the "Log in as a member" link on the ticket purchase box. The second will be before they get transferred over to the Agile Ticketing cart page.

Logging in as a Member

The **AuthenticateCustomer** and **AuthenticateEmail** API methods are not ideal for this scenario since they can return multiple memberships that have to be evaluated for expiration and validity. The **AuthenticateMember** API method takes care of this automatically. This method will only return one membership record once authenticated. This method will also allow members who do not have online accounts to set them up during the authentication process. This will fix the need to create usernames and passwords for newly imported members.

The process of logging in as a member is a multi-step process. We must first ask for the customer's member number. The **AuthenticateMember** API method will be called without sending the membershipID and password parameters in this case. Several errors have to be checked for in order to proceed correctly.

Error 1028 – Required Parameter membershipID – This error may be rare, but it should be taken into account. If the festival has multiple membership programs they may have member numbers defined in multiple programs. In this case we need to ask which of the membership programs they have.

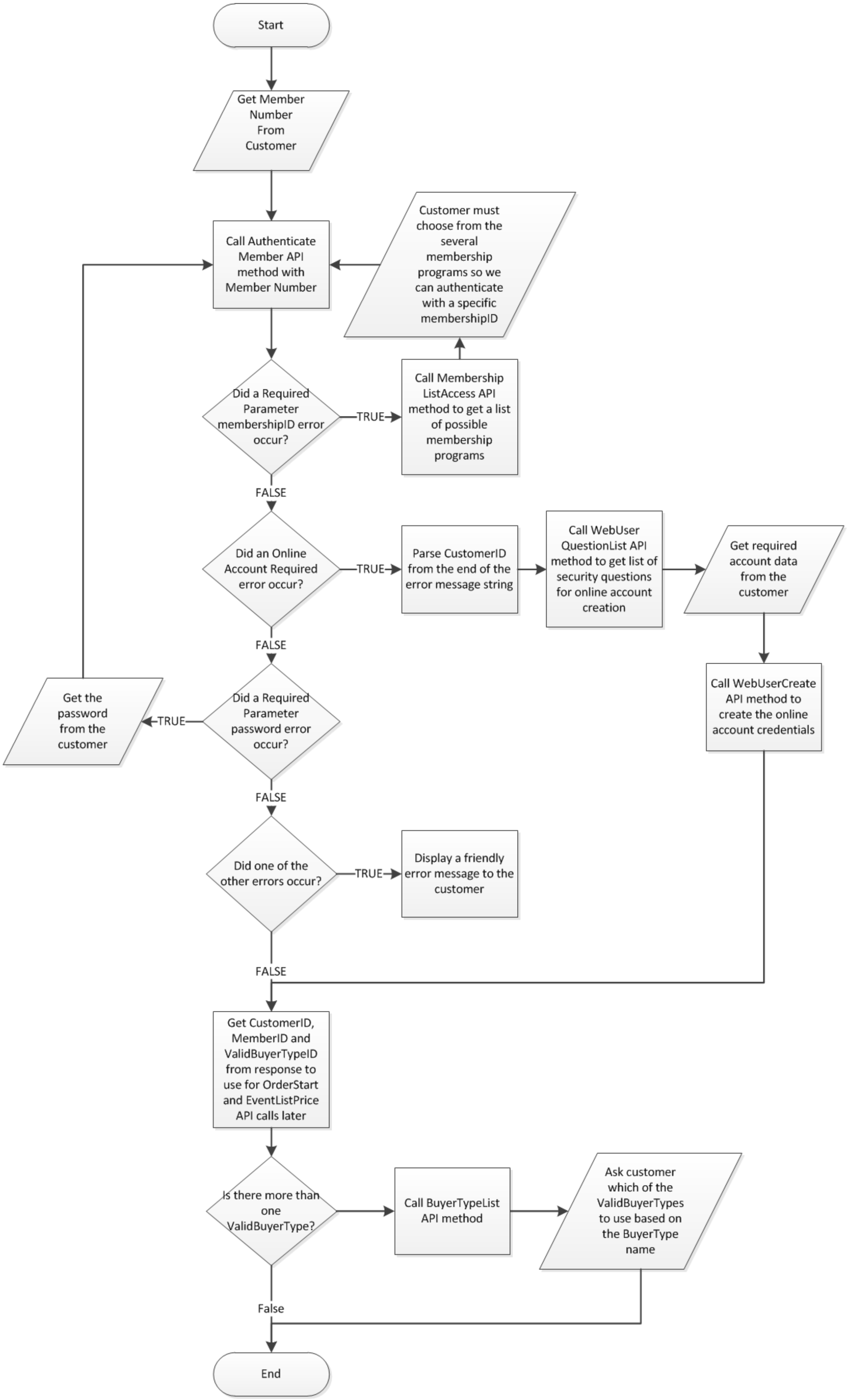
Error 1026 – Online Account Required – This is a common occurrence during the member login process. The festival will regularly import members from an external data source. These members will not have an online account with a password set up yet. The members online account should be created so they can complete the order and are able to be authenticated correctly in the future.

Error 1028 – Required Parameter Password – This is the second step of the authentication process. The customer needs to be asked to type in there password so we can authenticate them.

Other Errors – A friendly error message should be shown for the other errors. (e.g. "Your membership has expired" or "Your membership needs to be renewed")

Once authenticated the customerID, memberID, and ValidBuyerType will be used for the **EventSalesStatus**, **EventListPrices**, and **OrderStart** API methods. The authentication response data returns which member BuyerType should be used for sales. There may be more than one and the customer will need to be asked which they want to log in under. (e.g. Login as a "Member" or Login as a "Sponsor")

See the diagram below for the full run-through of the member login process.


Start
Get Member Number From Customer
Call Authenticate Member API method with Member Number
Customer must choose from the several membership programs so we can authenticate with a specific membershipID
Did a Required Parameter membershipID error occur?
TRUE
Call Membership ListAccess API method to get a list of possible membership programs
FALSE
Did an Online Account Required error occur?
TRUE
Parse CustomerID from the end of the error message string
Call WebUser QuestionList API method to get list of security questions for online account creation
Get required account data from the customer
Call WebUserCreate API method to create the online account credentials
FALSE
Did a Required Parameter password error occur?
TRUE
Get the password from the customer
FALSE
Did one of the other errors occur?
TRUE
Display a friendly error message to the customer
FALSE
Get CustomerID, MemberID and ValidBuyerTypeID from response to use for OrderStart and EventListPrice API calls later
Is there more than one ValidBuyerType?
Call BuyerTypeList API method
Ask customer which of the ValidBuyerTypes to use based on the BuyerType name
False
End

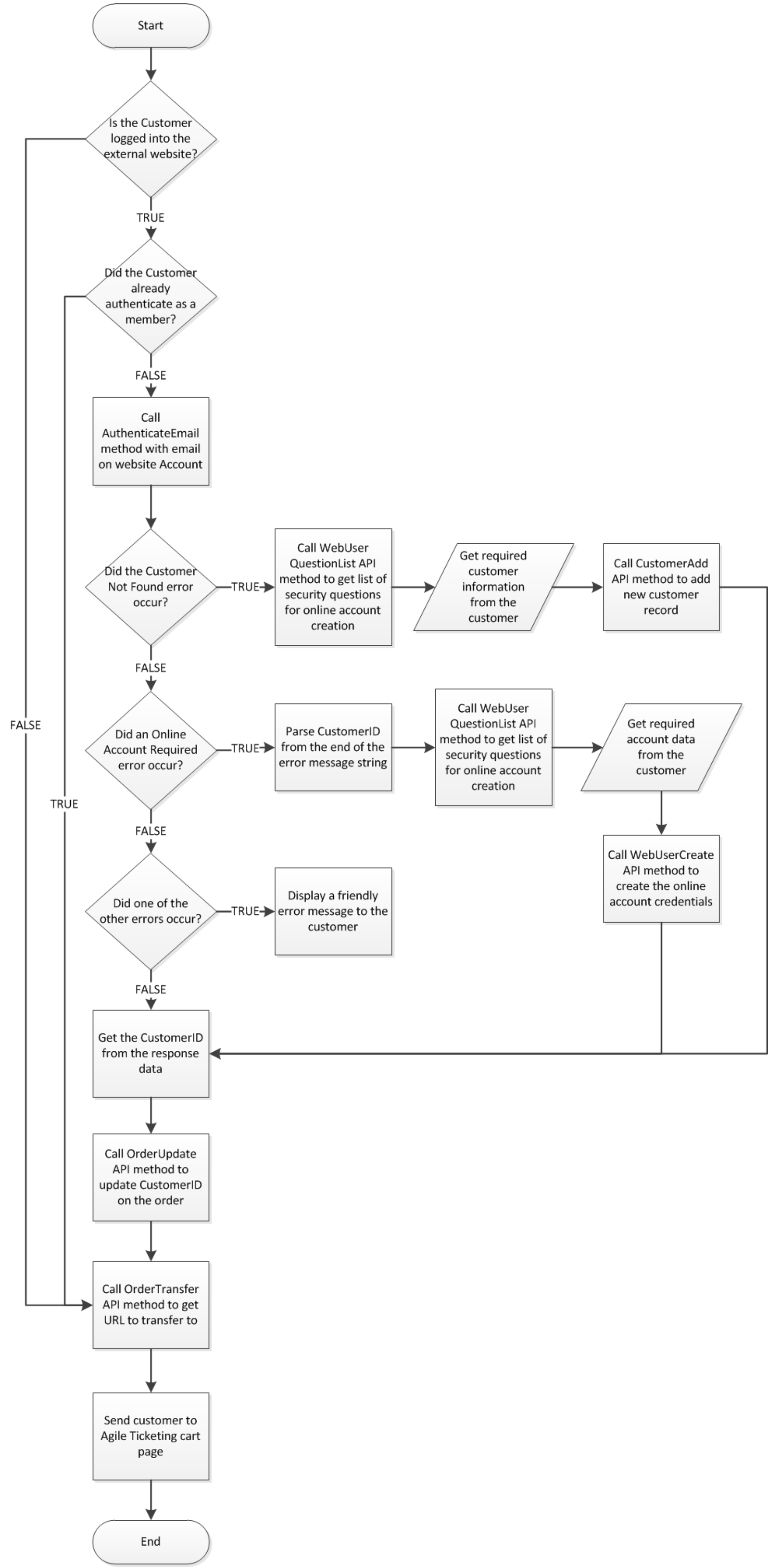
Logging in before they are transferred to the cart page

If the customer has logged into the external website we want to try to authenticate this customer before they get transferred to the Agile Ticketing cart page. This will keep the customer from getting confused by logging into their website account and then seeing a login screen later in the process to login to the Agile Ticketing account. The customer will need to be authenticated via email or a new customer record should be added for them.

The **AuthenticateEmail** method will need to be called with the email address on the website account. The secret authenticationKey parameter will be sent to this request so that a password is not required. If the customer is not found then the **CustomerAdd** method should be called to add the customer information. It is recommended that the email address from the website account be used instead of asking them to type in an email in a customer information form. This will make sure that the next time they visit the website to add tickets the **AuthenticateEmail** method will find a valid match. Once the customer has been authenticated or added the new **OrderUpdate** API method should be called to update the customerID on the order record. Then the customer can be transferred to the Agile Ticketing cart page.

The **OrderTransfer** API method will be used to get the URL needed to transfer the customer to the Agile Ticketing payment cart and payment process. The Agile Ticketing cart page will have a “Continue Shopping” button on it which can send the customer back to any one of the website’s pages.

See the diagram below for the full run-through of this process.


Start
Is the Customer logged into the external website?
TRUE
FALSE
Did the Customer already authenticate as a member?
FALSE
TRUE
Call AuthenticateEmail method with email on website Account
Did the Customer Not Found error occur?
TRUE
Call WebUser QuestionList API method to get list of security questions for online account creation
Get required customer information from the customer
Call CustomerAdd API method to add new customer record
FALSE
Did an Online Account Required error occur?
TRUE
Parse CustomerID from the end of the error message string
Call WebUser QuestionList API method to get list of security questions for online account creation
Get required account data from the customer
Call WebUserCreate API method to create the online account credentials
FALSE
Did one of the other errors occur?
TRUE
Display a friendly error message to the customer
FALSE
Get the CustomerID from the response data
Call OrderUpdate API method to update CustomerID on the order
Call OrderTransfer API method to get URL to transfer to
Send customer to Agile Ticketing cart page
End