

XML Core Interface Documentation

-Pete French.

March 21, 2009

Contents

1	Introduction	4
2	XML core basic calling conventions	5
3	The <i>start_session</i> method	7
3.1	Required inputs	7
3.1.1	Example input	7
3.2	Output data	7
3.2.1	Example output	7
4	The <i>event_search</i> method	8
4.1	Required inputs	8
4.1.1	Example input	9
4.2	Output data	9
4.2.1	Example output	9
4.3	Potential failure codes	11
5	The <i>extra_info</i> method	12
5.1	Required inputs	12
5.1.1	Example input	12
5.2	Output data	12
5.2.1	Example output	13
5.3	Potential failure codes	14
6	The <i>date_time_options</i> method	15
6.1	Required inputs	15
6.1.1	Example input	15
6.2	Output data	16
6.2.1	Example output	17
6.3	Potential failure codes	20
7	The <i>availability_options</i> method	21
7.1	Required inputs	21
7.1.1	Example input	22
7.2	Output data	22
7.2.1	Example output	23
7.3	Potential failure codes	25

8	The <i>discount_options</i> method	27
8.1	Required inputs	27
8.1.1	Example input	27
8.2	Output data	27
8.2.1	Example output	28
8.3	Potential failure codes	30
9	The <i>create_order</i> method	31
9.1	Required inputs	31
9.1.1	Example input	31
9.2	Output data	31
9.2.1	Example output	31
9.3	Potential failure codes	32
10	The <i>trolley_add_order</i> method	33
10.1	Required inputs	33
10.1.1	Example input	33
10.2	Output data	34
10.2.1	Example output	36
10.3	Potential failure codes	36
11	The <i>trolley_describe</i> method	37
11.1	Required inputs	37
11.1.1	Example input	37
11.2	Output data	37
11.2.1	Example output	40
11.3	Potential failure codes	41
12	The <i>trolley_remove</i> method	43
12.1	Required inputs	43
12.1.1	Example input	43
12.2	Output data	43
12.2.1	Example output	44
12.3	Potential failure codes	44
13	The <i>make_reservation</i> method	45
13.1	Required inputs	45
13.1.1	Example input	45
13.2	Output data	46
13.2.1	Example output	47
13.3	Potential failure codes	49
14	The <i>get_reservation_link</i> method	50
14.1	Required inputs	50
14.1.1	Example input	50
14.2	Output data	50
14.2.1	Example output	50
14.3	Potential failure codes	50

15	The <i>release_reservation</i> method	51
15.1	Required inputs	51
15.1.1	Example input	51
15.2	Output data	51
15.2.1	Example output	51
15.3	Potential failure codes	51
16	The <i>purchase_reservation</i> method	52
16.1	Required inputs	52
16.1.1	Example input	54
16.2	Output data	54
16.2.1	Example output	55
16.3	Potential failure codes	57

1 Introduction

The Seatem organisation is a ticket marketing and distribution company with access to a wide range of product worldwide. This product is held on a number of different computer systems, some because they are more appropriate to a particular product, others because they are legacy systems that have been acquired during the companies expansion and integration. For the Seatem e-commerce activities a middleware product has been developed that pulls all the different product together and presents a uniform data model with a single point of access that is then used for onward distribution to the various electronic distribution channels.

As the primary means of distribution is through web sites, the middleware architecture is particularly set up to support browser technology, where the customer communicates directly with the system. This applies to the internet selling business, and also some of the more esoteric channels such as satellite television, mobile phones and cable television. In all these cases the customer is communicating directly with the Seatem software, whether they are aware of this or not. The company runs a large affiliate programme, which allows third parties to redistribute the tickets, but this is generally done by the web sites of those companies being linked back to the Seatem sites, which will carry out the dialog with the consumer on the affiliate's behalf.

As the e-commerce business has developed it has become apparent that there is an increasing need for the system to supply tickets through to other companies software where they conduct the customer dialog (either via their own website or through some kind of person to person contact). For some sites that only wish to generate automatic links to the Seatem product then this can be done via the use of the XML data feed (described in a separate document) which exposes the internal product codes used to describe events. This allows the integration of Seatem product into a larger database, though with the customer still being redirected to the Seatem site in order to make the actual transaction.

In the past affiliates wishing to make the entire dialog with the customer themselves have had to resort to trying to re-interpret the information off the Seatem web pages. Obviously this is not ideal, and a direct interface into the systems would be a much better solution. It is this direct interface which is provided by the XML core. This gives a set of XML based remote procedure calls that step through the process in a clearly defined way, right through to the actual purchase of an order. It is intended that all external users will migrate to using this system to make their transactions. This document describes the various calls to the system, and how they are used to make a purchase.

Familiarity with the general affiliate interface is useful in reading this document, as is an understanding of the current XML feed. The data from the XML feed may be used in conjunction with the XML core as a way of integrating the product into an existing database, as it provides a much broader range of options designed to be used by people wishing to maintain their own local database of events.

2 XML core basic calling conventions

The XML core script is named **xml_core.exe** and should be run from the web server like any other CGI script. It may be run as HTTPS to secure customer data when that is being passed to the script. The script requires all its input to be presented as XML data, and this must be provided in a POST request. Use of the GET method is not supported. XML data in the input should be flagged with the use of a content type header which should state that the type is **text/xml**. No other type is supported. If the contents type header cannot be provided then any POST request where the first byte of the data is a “<” character, and the last is a “>” character will also be treated as XML.

The XML script shares all the characteristics of standard Seatem middleware software and thus requires authentication information to be presented. This is normally in the form of a “user_id” element and a “crypto_block” which has been provided by a previous stage in the process. If the user is working on a channel other than the default then a “chan” element is required, plus the optional “lang” element if an alternative language code is being used. Note that the language codes are normally taken from the accepted language header in the POST request, and anything in the “lang” variable is simply prefixed onto the existing list as a preference. The “user_id”, “chan” and “lang” elements should not change during a particular session where the core is being used, or undefined results may occur. In the following documentation these three variables are not explicitly described for each method.

The XML core is capable of performing many different functions. The function required is specified by the name of the enclosing element of the data POSTed to the script. The reply will take the form of an XML element with the same name, postfixed by the string “_result”. If a request fails then the returned element will contain the elements “fail_code” and “fail_desc”. The fail code is a numeric value indicating the problem, with the fail_desc being a human readable string describing the problem. There are also eight generic errors which indicate that a failure occurred before the script had even got as far as attempting to process the XML request. If a generic error occurs then the returned element will always be “script_error” and will contain the elements “error_code” and “error_desc”. As with failures these contain a numeric value indicating the error, plus a human readable description. The eight error codes and their meanings are as follows:

1. A bad username has been provided in the “user_id” element.
2. A bad channel code has been provided in the “chan” element.
3. User authorisation has failed. Either your password or crypto_block are invalid for the given user_id.
4. A connection could not be established to the backend ticketing system.
5. Your IP address is on a network forbidden from using the core.
6. A connection could not be established to the main database.
7. Membership authentication has failed. This should never happen as membership schemes are not being used with the XML core.

8. Bad data has been supplied to the script. This is a very generic error when it is impossible to determine exactly what is wrong.

The standard way of returning descriptive data, such as event information or discount code names, is as plain text which the user is then free to process as they so wish. The Seatem middleware does, however, have the capacity to store full formatted information pages in HTML, WML or several other markup forms, intended for direct display to the user. It is possible to access this information through the core by supplying the necessary type required in the "mime.text.type" element. This has to be one of "plain", "html", "xml" or "vnd.wap.wml" as these are the only valid types supported. The descriptive information is then guaranteed to be in the requested form. If there is no extended information available in the requested form then the plain text information will be provided, ready encoded into the appropriate format for direct display.

3 The *start_session* method

This method validates a the given username against a password, and generates an initial `crypto_block`. This `crypto_block` should be valid for an indefinite period, and is used as verification input for all subsequent calls which do not require state to be carried over from the previous call. This request may also be used as a simple “ping” to detect whether the Seatem system is alive, and to test that the login credentials are valid.

3.1 Required inputs

The required inputs are the username in a “`user_id`” element, and a password in a “`user_passwd`” element. If the system being implemented is intended for on-credit sales to other businesses, then a third inout field of “`sub_id`” may also be necessary, as some of these organisations are sub-divided into separate clients, each with their own login name in addition to the main username.

3.1.1 Example input

```
<start_session>
<user_id>demo</user_id>
<user_passwd>demopass</user_passwd>
</start_session>
```

3.2 Output data

The output data simply consists of the `crypto_block` to be used for the next stage.

3.2.1 Example output

```
<start_session_result>
<crypto_block>M_--qRfq64CDfM9Vj4JBQdRMd0JPW0xEpX1WsPqJY6Sy_XkrL
_G8x-0i4sQu1cKhPNF-2PkEozP_r0W8KEiQSB10Z005hkKzT10Uz6IdkZ5DnW4D
XoVa_AwIc0--Y</crypto_block>
</start_session_result>
```

4 The *event_search* method

Event search takes a set of search criteria and provides a list of events from the database which match. The criteria may be a genuine search, or may be specific codes intended to find a single event. Such codes may be acquired using the XML feed data which is provided as a standard part of the HTML based website affiliate programme. A search may be combined with starting a session in a single operation, rather than using a separate “start_session” method if so desired. The search results are normally ordered alphabetically, but may also be sorted in order of best sales if required.

4.1 Required inputs

The user is normally validated by the provision of a “crypto_block” generated from a “start_session” method. If this is not required then the user’s password may be provided as “user_passwd” instead, and a valid session crypto block will be created.

The various criteria used to search the database are all passed in as elements prefixed by “s_”. They are listed below. Any combination of these may be passed in as required, all are optional. Providing no search elements will give a list of all available events.

- *s_keys* — this element contains a set of keywords separated by spaces. Returned events must have all keywords present in either the event or venue name description text. Searching is not case sensitive.
- *s_dates* — a date range may be specified as two dates in YYYYMMDD form, separated by a colon. A missing start or end date make that part of the search open ended. The returned events are not guaranteed to be valid over the range given, but events which are definitely *not* valid over the given range are excluded.
- *s_coco* — this restricts events to those taking place in the country specified by the ISO 3166 country code specified by this element.
- *s_geo* — to find events in a certain geographical location this element contains a circular search area. This is specified as the latitude and longitude of the centre of the circle, plus the radius in kilometres. The three values are specified in decimals and separated by colons.
- *s_geo_lat*, *s_geo_long*, *s_geo_rad* — these three variables provide an alternative way of specifying the search area as described in *s_geo*.
- *s_src* — only events held on the backend system specified by this code will be returned.
- *s_area* — only events with this area code will be returned.
- *s_ven* — only events with this venue code will be returned.
- *s_eve* — only events with this event code will be returned.

- *s_class* — this element contains either an event class, or an event class and subclass, separated by a “/” character. Only events in the given class will be returned. Classes are arbitrary, however, and thus this search is only useful in conjunction with other information that cannot be extracted automatically.

When using the XML feed an event can be uniquely identified by the tuple (“s_src”, “s_area”, “s_ven”, “s_eve”). All four parts are required to guarantee that an event is being uniquely specified, as the various codes are only defined to be unique within the namespace corresponding to the code above them in the data model. i.e. an event code is only guaranteed to be unique within a particular venue code, though a backend system may actually allocate unique codes for all events system-wide. An easier method than supplying these codes separately, however, is to use the “crypto_block” which is provided in the XML feed as part of the “full_query” element for each event. This method should only be used if the feed was run with the same user and channel as the core is being run with.

If the returned events are required to be in the order of sales then the empty element “s_top” should be supplied. This will order the events by those which have had the most sales over the last 48 hour period. Best sellers will come out first, followed by events in decreasing order of popularity. Note that this is done after the other filtering, so it is possible to pull the top selling events in a class, or country, or any other combination of search terms.

4.1.1 Example input

```
<event_search>
<user_id>demo</user_id>
<crypto_block>M_--qRfq64CDfM9Vj4JBQdRMd0JPW0xEpX1WsPqJY6Sy_XkrL
_G8x-0i4sQu1cKhPNF-2PkEozP_rOmb02B5HSyBIM05hkKzT10Uz6IdkZ5DnW4D
XoVa_AwIc0--Y</crypto_block>
<s_keys>sad well</s_keys>
</event_search>
```

4.2 Output data

The output data consists of the crypto_block to be used for the next script, plus zero or more “event” elements. Each of these contains the data for a single event returned by the search. For each event three elements are provided giving the textual description of the event, the venue in which it is being held, and the source providing the tickets. A code for the ticket source is also present as “source_code”. Events are identified by tokens, which are provided in the “event_token” elements for each event. If the country in which the event is located is known then this is also present as an ISO 3166 code in “country_code”.

4.2.1 Example output

```
<event_search_result>
<crypto_block>M_--qRfq64CDfM9Vj4JBQdRMd0JPW0xEpX1WsPqJY6Sy_X--q
-u7v0oeQH9kWHcDhhX0y1R6--Q5LYNt4zn0xwsy_tT8k2_EW3NjhoJBWdGhx6TV
_mcqV11vM---Y</crypto_block>
```

```
<event>
<event_desc>Beasts</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>5DZR</event_token>
</event>
```

```
<event>
<event_desc>Blush</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>LH7</event_token>
</event>
```

```
<event>
<event_desc>Farruquito</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>NN19</event_token>
</event>
```

```
<event>
<event_desc>Jose Merce</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>232</event_token>
</event>
```

```
<event>
<event_desc>Maria Pages</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>MQ2JX</event_token>
</event>
```

```
<event>
<event_desc>Nutcracker</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
```

```
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>LH4</event_token>
</event>

<event>
<event_desc>Richard Alston</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_code>
<country_code>uk</country_code>
<event_token>JJ115Z</event_token>
</event>

</event_search_result>
```

4.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “start_session”
- **2** — the requested ”mime_text_type” was not a supported value

5 The *extra_info* method

Extra info takes a crypto block generated by event search, along with an event token, and returns the extra information associated with the event. This consists of the event and venue names and descriptions as previously returned, along with the country the event is taking place in, the event's postcode, geographical location, and also paragraphs of text describing both the event and the venue. By default these paragraphs are formatted plain text using linefeeds and spaces, and are intended to be displayed directly, though HTML ((and other) formatted information can be requested using the "mime.text.type" input as described in the section on basic calling conventions. It is also possible to use this method to request additional information about the supplier of an event, including the terms and conditions of sale, contact numbers for customer services and the name which will appear on the card statement, plus media assets such as images which may be available for the event. For backward compatibility these are not enabled by default, however.

5.1 Required inputs

The "crypto_block" provided for validation must have been generated from a previous call to "event_search". The event is identified by the provision of an "event_token" element. If the extra information about the source is required then a blank element called "source_info" should be supplied as well.

5.1.1 Example input

```
<extra_info>
<user_id>demo</user_id>
<crypto_block>M--qRfq64CDfM9Vj4JBQdRMd0JPW0xEpX1WsPqJY6Sy_X--q
-u7v0oeQH9kWHcDhhX0y1R6--Q5LYtL2zZeBLtUFtT8k2_EW3NjhoJBWdGhx6TV
_mcqV11vM---Y</crypto_block>
<event_token>MQ2JX</event_token>
</extra_info>
```

5.2 Output data

The output data consists of the usual event_desc, venue_desc, source_desc and source_code elements containing the same data which was returned by the original event_search method. This information will always be present. If an event has a country code allocated to it indicating where it is taking place then this will be present in a "country_code" element, and similarly a postcode will be returned in a "postcode" element if one is known. For events which can be precisely geo-located then an element named "geo_data" is included. This contains two child elements, "latitude" and "longitude" which contain the values implied by their names, in decimal degree format. The descriptive text about an event or a venue is returned in the "event_info" and "venue_info" elements if it is available. For events where the information is available in multiple languages, then the most appropriate language for the text will be chosen dependent on the list of preferred languages supplied in the POST request, if any. Any extra text which the customer should be made aware of before confirming that they wish to

opurchase the tickets is to be found in the element “collect_confirmation_text”, in the usual format for such text, as documented in the subsequent sections concerning the collection of ordering information.

If the source information was requested using “source_info” then a number of extra elements may be present in the output. The terms and conditions element is always included, but the others are only present if they have a value. The fields are listed below.

- **source_after_sales_email** — The email address which should be used for after-sales customer services.
- **source_card_statement_desc** — The text which will appear on the customers credit card statement as the debtor for this event.
- **source_enquiries_email** — The email address for general enquiries regarding this supplier.
- **source_local_phone** — Details of how to telephone customer services for customers located in the suppliers country.
- **source_local_fax** — Details of how to send a fax to customer services for customers located in the suppliers country.
- **source_international_phone** — Details of how to telephone customer services for customers located outside the suppliers country.
- **source_international_fax** — Details of how to send a fax to customer services for customers located outside the suppliers country.
- **source_postal_addr** — The postal address for sending physical mail to customer services.
- **source_t_and_c** — The suppliers terms and conditions. This will usually be a long paragraph of legal text which should be made available to the customer in some way.

To request the media assets which may be associated with an event a number of “request_media” elements may be included in the input, each containing the same of a media file to be returned if possible. For each available requested media asset, an “event_media” element will be present in the output Each of these will contain two sub-elements, one being “name” containing the name of the media file, and one being “data” which contains the data in hexadecimal form. At the time of writing the media assets which may be associated with an event include images called “small.jpg”, “large.jpg”, “large-190x100.jpg”, “large-525x200.jpg”, “supplier.jpg”, “seating_plan.jpg”, “banner.gif” and “banner.jpg”. Currently, only the images “small.jpg” and “large-190x100.jpg” are generally available on most events.

5.2.1 Example output

```
<extra_info_result>
<event_desc>Maria Pages</event_desc>
<venue_desc>Sadlers Wells</venue_desc>
```

```
<source_desc>Keith Prowse Ticketing</source_desc>
<source_code>fcg1</source_desc>
<country_code>uk</country_code>
<geo_data>
<latitude>51.52961137</latitude>
<longitude>-0.10601562</longitude>
</geo_data>
<venue_info>Sadlers Wells&#10;Roseberry Avenue&#10;Islington
EC1R&#10;&#10;Nearest Underground: Angel&#10;&#10;Nearest Rail:
Kings Cross&#10;&#10;Wheelchair Access&#10;</venue_info>
</extra_info_result>
```

5.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “event_search”
- **2** — the requested ”mime_text_type” was not a supported value
- **101** — no “event_token” has been supplied
- **102** — the supplied “event_token” is corrupt
- **103** — the event specified could not be found in the database

6 The *date_time_options* method

All events may require some kind of date and time information associated with them. For performance based products such as theatre or concert tickets then this corresponds to the date and time of a particular performance of the event. For products such as theme park entrance tickets then the estimated date of usage of the ticket may be required as not all tickets are valid on all dates. In addition, to aid ticket despatch, it may be necessary to know what date the customer is intending to depart, in order that the tickets may be shipped to them before they leave. For users of the XML feed this method may be combined with starting a session rather than stepping through the “start_session” and “event_search” methods as the event token is produced directly in the feed. Alternatively it may be specified using the “crypto_block” from the “full_query” element of the feed or by using the (“s_src”, “s_area”, “s_ven”, “s_eve”) tuple.

For a given event, specified by an event token from the search method, this method specifies what date and time information should be collected for the event in question, along with any restrictions on that data. A usage date is always mutually exclusive with selection of a date and time from a list, but a departure date may be required for either of them, or even on its own. Certain products may require no date or time information to be collected from the customer. All possibilities should be handled by the user of this method. When a usage date is required then restrictions on the usage date will be specified by this method, and these should be enforced when collecting the information from the customer.

For performance based events a set of optional elements may be provided which will restrict the list of performances returned, this can considerably reduce the amount of data being returned from the script.

6.1 Required inputs

The “crypto_block” from the “event_search” output is required, along with an “event_token” element. If a date bounded search is required then the elements “earliest_date” and “latest_date” may also be provided. Both are optional and have no effect if a performance list would not be returned. The format is “YYYYMMDD” for the dates.

If this method is being used to start a session on a specific event, then the easiest way to do this is to supply the “crypto_block” from within the “full_query” element for the event in the XML feed. Note that this is only valid if the core is being run with the same user and channel as the XML feed from which the data was obtained. Alternatively a “user_passwd” plus all four “s_src”, “s_area”, “s_ven” and “s_eve” may be specified, but all of these need to be supplied in order for this to work.

6.1.1 Example input

```
<date_time_options>
<user_id>demo</user_id>
<crypto_block>M_--qRfq64CDfM9Vj4JBQdRmD0JPW0xEpX1WsPqJY6Sy_X--q
-u7v0oeQH9kWHcDhhX0y1R6--Q5LYtL2zZeBLtUFtT8k2_EW3NjhoJBWdGhx6TV
_mcqV11vM---Y</crypto_block>
```

```
<event_token>MQ2JX</event_token>
</date_time_options>
```

6.2 Output data

The output data consists of the `crypto_block` to be used for the next script, plus one or two other elements, depending on what date and time information needs to be collected. A single element `need_departure_date` is always present, containing a boolean value of “yes” or “no” depending on whether a departure is required. No extra information is given regarding a departure date as it is essentially unrestricted.

If the method is being used to start a session by going directly to an event using the data from the XML feed then there is a possibility that the event may no longer be valid. If this is the case then the only output from the script will be the single element `event_not_found`. The lack of an event is indicated this way rather than using an error code as it is a result which may happen during the correct operation of the system. If data from the XML feed is being cached then this response indicates that the cached data should be refreshed.

For events where a performance needs to be selected from a list then an element `using_perf_list` will be present. This contains a number of “performance” elements, each representing a single selectable performance of the event. A “performance” element contains a `perf_token` to be carried through to the next stage of the process, plus information about the date and time of the performance. Each is represented in machine readable `YYYYMMDD` and `HHMMSS` format in the `date_yyyymmdd` and `time_hhmmss` elements, plus also in human readable form in the `date_desc` and `time_desc` elements. As some events have performances where the time is not relevant, the two time variables will not necessarily always be present.

Some performances will also have a `perf_name` element containing extra human readable description of the performance. This is typically used where performances of an event differ significantly in more than just date and time. One will not always be present, but if it is, it must be shown to the customer.

The element `is_limited` is also present, containing a boolean value of “yes” or “no” to indicate those performances where the supply of tickets is known to be extremely limited. This is usually taken to be fewer than four remaining, though not all ticket sources provide this information. Where the information is not available the flag will always be set to “no”. If no valid performances can be found then the performance list will be present but empty.

For events where a usage date is required then a `using_usage_date` element will be present. As usage dates are restricted then this element will always contain information indicating the periods within which a usage date may be selected. The validity of a ticket is held as the first date on which it is valid, plus the last date on which it is valid. Within this period there may be ranges of dates where the ticket is not valid. These are held as separate ranges, indicated by the first invalid day and the last invalid day. Finally it is also possible that certain products will not be valid on individual days of the week, for example an attraction which is never open on a Sunday.

All of this information is available within the `using_usage_date` element. The overall range within which the usage date must be chosen is given by four elements, these being `first_valid_date_yyyymmdd` and `first_valid_date_desc`

plus “last_valid_date_yyyymmdd” and “last_valid_date_desc”. As with performances all dates are presented with both a machine readable YYYYMMDD format and a descriptive text format suitable for direct display. If there are any periods of invalidity within this range then these are indicated by “invalid_range” elements. Each of these contains a similar set of four elements to the valid range, except that they use “invalid” rather than “valid” in the element name. Any invalid days of the week are represented by “invalid_weekday” elements. These contain a “weekday_number” and “weekday_name”. The days of the week are numbered with zero representing Sunday and six representing Saturday. The full weekday names appear in the “weekday_name” elements.

It is possible that a given event may have extra pieces of text which should be displayed when collecting a departure date, usage date or performance. These pieces of text are used to carry item specific information which is only relevant to the customer at the point where the date in question is being collected. Three optional elements are used to hold such text, being “collect_depart_text”, “collect_perf_text” and “collect_usage_text”, and each corresponding to the obvious piece of information. Within these elements will be one or more “collect_text” elements holding the actual text to be displayed. It is important to note that there may be more than one piece of text associated with each piece of information to be collected.

6.2.1 Example output

The following example data is for a theatre event requiring the selection of one of a list of performances. No departure date is required. The full set of performances here has been trimmed as they all follow the same format.

```
<date_time_options_result>
<crypto_block>k--2mkKzCy6__eBiVuAEjWbPP-6j-t23qyus1WRvXvxPO_-r
K5taZhef6g-L_wPBFpfm0wEtdYhbqK0hQytgJjN1yNrlwZyf3B_zUF6_us7q107
fqqXlxx7L2PNHZukG90JPgLi6jKdqzssmHPOXLu2L---Z</crypto_block>
<need_departure_date>no</need_departure_date>
<using_perf_list>

<performance>
<perf_token>s---DdzV9tDBzGehTwFy16GDZk5YWBvNoQ4GCNVrYpFqkaCHWbR
kPODHytBwE2ZAwzNF_PiwgmZrxa4-Y</perf_token>
<is_limited>no</is_limited>
<date_yyyymmdd>20040210</date_yyyymmdd>
<time_hhmmss>193000</time_hhmmss>
<date_desc>Tue, 10th February 2004</date_desc>
<time_desc>7.30 PM</time_desc>
</performance>

<performance>
<perf_token>s---XU1BHYZkojP5EUX-i20s9Az_mNvX4kGxzqm1M_LZcX5uMju
YS10jZ0yvQVdMQhr1_PiwgmZrxa4-Y</perf_token>
<is_limited>no</is_limited>
<date_yyyymmdd>20040211</date_yyyymmdd>
<time_hhmmss>193000</time_hhmmss>
```

```

<date_desc>Wed, 11th February 2004</date_desc>
<time_desc>7.30 PM</time_desc>
</performance>

<performance>
<perf_token>s---3XXiqHKj_8GxiUfiE8moZmjL9dc_-DjNHg-cDM7L4dszUek
_61Abs0PqWdn95HY-_PiwgmZrxa4-Y</perf_token>
<is_limited>no</is_limited>
<date_yyyymmdd>20040212</date_yyyymmdd>
<time_hhmmss>193000</time_hhmmss>
<date_desc>Thu, 12th February 2004</date_desc>
<time_desc>7.30 PM</time_desc>
</performance>

<performance>
<perf_token>s---XaU-erR3L1SA09dP-pdBmZim32j38IFEssfG9qk0VQBZpa0
-VJJXmZId0C02P_G_PiwgmZrxa4-Y</perf_token>
<is_limited>no</is_limited>
<date_yyyymmdd>20040213</date_yyyymmdd>
<time_hhmmss>193000</time_hhmmss>
<date_desc>Fri, 13th February 2004</date_desc>
<time_desc>7.30 PM</time_desc>
</performance>

<performance>
<perf_token>s---GdtcgF2mq0j64J7XUpdwHpQIroMwnF7vTk-PYjjwk9x04L
-4SHWkeuaKCZmNwoZ_PiwgmZrxa4-Y</perf_token>
<is_limited>no</is_limited>
<date_yyyymmdd>20040214</date_yyyymmdd>
<time_hhmmss>143000</time_hhmmss>
<date_desc>Sat, 14th February 2004</date_desc>
<time_desc>2.30 PM</time_desc>
</performance>

<performance>
<perf_token>s---AbiZwDdZo2IRYmd5syKAobBBYQzAWytxi60fJrtADdu3G1w
2vOVRs73EPHDcefb4_PiwgmZrxa4-Y</perf_token>
<is_limited>no</is_limited>
<date_yyyymmdd>20040214</date_yyyymmdd>
<time_hhmmss>193000</time_hhmmss>
<date_desc>Sat, 14th February 2004</date_desc>
<time_desc>7.30 PM</time_desc>
</performance>

</using_perf_list>
</date_time_options_result>

```

This example is for an event which requires a usage date. The event is valid from the 20th of February 2003 through to the 20th of April, though it cannot be booked between the 3rd and 15th of March and also from the 27th of March

to the 8th of April. In addition the attraction is not open on Tuesdays and Fridays.

```
<date_time_options_result>
<crypto_block>M--3KyQLySr7Wc1RgqbXGBcoZeAPiXU8unG2vFyCP63ds5fRi4
egU56PXoH8-U8HC1a5qGD8kqYCI8ONZfANDAM4ilg_RK1kDLxN0qR5ovB1K7VVkIN
EtehA--Y</crypto_block>
<need_departure_date>no</need_departure_date>
<using_usage_date>
```

```
<first_valid_date_yyyymmddd>
20030220
</first_valid_date_yyyymmddd>
<first_valid_date_desc>
Thu, 20th February 2003
</first_valid_date_desc>
<last_valid_date_yyyymmddd>
20030420
</last_valid_date_yyyymmddd>
<last_valid_date_desc>
Sun, 20th April 2003
</last_valid_date_desc>
```

```
<invalid_range>
<first_invalid_date_yyyymmddd>
20030303
</first_invalid_date_yyyymmddd>
<first_invalid_date_desc>
Mon, 3rd March 2003
</first_invalid_date_desc>
<last_invalid_date_yyyymmddd>
20030315
</last_invalid_date_yyyymmddd>
<last_invalid_date_desc>
Sat, 15th March 2003
</last_invalid_date_desc>
</invalid_range>
```

```
<invalid_range>
<first_invalid_date_yyyymmddd>
20030327
</first_invalid_date_yyyymmddd>
<first_invalid_date_desc>
Thu, 27th March 2003
</first_invalid_date_desc>
<last_invalid_date_yyyymmddd>
20030408
</last_invalid_date_yyyymmddd>
<last_invalid_date_desc>
Tue, 8th April 2003
```

```
</last_invalid_date_desc>
</invalid_range>

<invalid_weekday>
<weekday_number>2</weekday_number>
<weekday_name>Tuesday</weekday_name>
</invalid_weekday>

<invalid_weekday>
<weekday_number>5</weekday_number>
<weekday_name>Friday</weekday_name>
</invalid_weekday>

</using_usage_date>
</date_time_options_result>
```

6.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “event_search”
- **2** — the requested ”mime_text_type” was not a supported value
- **201** — no “event_token” has been supplied
- **202** — the supplied “event_token” is corrupt
- **203** — the event specified could not be found in the database
- **204** — the “earliest_date” element was badly formatted
- **205** — the “latest_date” element was badly formatted

7 The *availability_options* method

With an event and its associated date and time information this method calculates the currently available tickets and returns a list of the types of tickets which may be purchased, along with any restrictions on the number of tickets. Availability is divided into a set of ticket types, and within each ticket type a subdivision into pricing bands. A pricing band will contain the price of a single ticket, plus the value of any per-ticket surcharge which will be added to the booking. The maximum number of tickets that are allowed to be booked for that particular price band is also indicated.

Some events are restricted to certain combinations of tickets, the most common being tickets which may only be purchased in pairs. A list is returned containing the set of valid possibilities for the number of tickets that may be purchased. Only values from the list will be accepted for bookings. A list of possible methods of despatching the tickets to the customer is also provided. Depending on the particular product, a ticket may be held for collection, or posted to the customer (subject to a number of restrictions on timing and postal areas). Each method in the list has an associated cost which will be added to the overall ticket charge.

It should be noted that the availability provided by the system is in real time and represents what was available at the time the request was made. There is thus no guarantee that the tickets will still be available to purchase at some future time. Tickets are only guaranteed to be held for a customer following a successful “reserve_order” method.

7.1 Required inputs

The basic required input is the “crypto_block” produced at the output of the “date_time_options” method. This holds the details of the event required. If the event requires a departure date then this should be specified in a “departure_date” element in YYYYMMDD form. Similarly, if the event requires a usage date then this should be provided in a “usage_date” element, also in YYYYMMDD form. For events which require a performance to be selected from a list then the appropriate token should be provided in a “perf_token” element. The inputs should have been validated before being presented to the method as they will be checked by the method and failure codes reported as appropriate if any problems are found. This method has more potential failure codes than any other in the system.

If the client software is capable of handling “self print vouchers” then this must be indicated to the code via the “self_print_mode” input. Setting this to “html” asks the core to include despatch methods requiring the customer to print out a page of HTML representing their tickets. Setting it to “custom” asks the core to include methods where the client software can generate its own voucher as agreed in advance with the venue. It is possible that no extra despatch methods will be returned by using this element: the product may not support it, or in the case of “custom” there may be no arrangement for the caller to generate customer self print vouchers. If the client software picks a selfprint despatch option, the same value of “self_print_mode” must be passed to subsequent calls.

If a set of orders is being accumulated in a trolley then the current set may be

passed in as “trolley_token”. If this is done then the list of options returned by the method will be restricted to remove any which are known to be incompatible with the current set of orders in the trolley.

7.1.1 Example input

```
<availability_options>
<user_id>demo</user_id>
<crypto_block>k--2mkKzCy6__eBiVuAEjWbPP-6j-t23qyus1WRvXvxPO_-r
K5taZhef6g-L_wPBFpfm0wEtdYhbqK0hQytgJjN1yNrlwZyf3B_zUF6_us7ql07
fqqXIxx7L2PNHZukG90JPgLi6jKdqzssmHPOXLu2L---Z</crypto_block>
<perf_token>s---AbiZwDdZo2IRYmd5syKAobBBYQzAWytxi60fJrtADdu3G1w
2v0VRS73EPHDcefb4_PiwgmZrxa4-Y</perf_token>
</availability_options>
```

7.2 Output data

The output data from the method divides into four sections. These are the event availability, the list of valid ticket quantities, the list of available despatch methods and the currency in use.

The ticket availability is found within the “availability” element. This contains zero or more “ticket_type” elements, each representing a single type of ticket for which availability has been found. If there is no availability for the ticket then the element will be empty, and none of the subsequent elements will be present. Within each “ticket_type” element the description of the ticket type is to be found in the “ticket_type_desc” element. The actual availability is contained within one or more “price_band” elements. Each of these represents a particular pricing option for tickets of that type. A “price_band” element contains the main price as a three decimal place value in the “ticket_price” element, plus any per ticket surcharge which may be applied in the “surcharge” elements. The number of tickets actually available to be booked is present in the “number_available” element for each price band, and a token to identify the band for the purposes of creating an order is provided as “band_token”.

Some tickets are restricted such that only specific quantities may be purchased. This is most commonly used to restrict purchases to even numbers of tickets only, or a single ticket at a time. The allowable quantities are listed in the “quantity_options” element, which contains a list of “valid_quantity” elements, each containing a single integer. The quantity must be chosen from this list.

The options for despatching the tickets which may be selected are listed within the “despatch_options” element. If no options are available then this element will be empty and the tickets cannot be purchased. Each available despatch method is described within a “despatch_method” element. The type of the method is held in “despatch_type”, a text description of the method in “despatch_desc”, the cost of the method in “despatch_cost” and finally a token to specify the actual method in “despatch_token”.

The methods come in four basic types. The most generic is the “dynamic” type. This type is provided when the user cannot select how the tickets will be sent to them. The normal method would be to have the tickets posted, but if this is not possible then they will be held for collection. The choice between

these two is determined at purchase time and the buyer informed at this point. Until then the method is not known. A despatch method of “collect” indicates that the tickets should be collected by the customer and will not be posted. Under normal circumstances the collection point is the box office at the venue. The despatch type of “post” indicates that the tickets will be posted to the customer’s address. There may be several different post methods giving various levels of postal service as options.

The final despatch type is “selfprint”. This indicates that the customer can print out their own ticket and take it to the venue for admission. This despatch type will not be available unless it is specifically requested via the “self_print_mode” input. Despatch methods of type “selfprint” are usually supported by TicketSwitch generating an HTML page representing the voucher. The client software must allow the customer to print this HTML. Alternatively special arrangements can be made between the client and individual venues to accept client-generated vouchers. Which of these two cases is permitted can be determined by the two elements “has_html_page” and “can_generate_self_print”, which will be “yes” or “no”. Both may be “yes” to allow both types of selfprint voucher, but both will never be “no” at once.

Both the “post” and “dynamic” despatch methods may result in tickets being posted, and some systems restrict the areas within which they allow tickets to be sent. Under such circumstances a list of countries will be provided and the customer must have an address located within one of those countries in order to be able to select the method. The list is provided in a “permitted_countries” element, with each country within a “country” element. The country’s name is provided as “country_desc” along with a letter code as “country_code”. The code “UK” is used in preference to “GB”, all other codes are as per ISO 3166.

If the preceding three lists are not empty then a final element “currency” will be present indicating the currency of all listed prices. This element contains the ISO 4217 three letter code and numeric identifier as “currency_code” and “currency_number”. For display purposes the currencies symbol is also present. As some currencies display the symbol before the numeric value, whilst others display it after the value, then two elements are provided to cope with this. These are “currency_pre_symbol” and “currency_post_symbol”. The values should be displayed between these two strings, one of which will normally be of zero length.

As with the date and time options there may be extra pieces of text which should be displayed to the customer when collecting the ticket type, number of seats or despatch method. The three elements used to hold this text are “collect_type_text”, “collect_seats_text” and “collect_send_text”, with the final one corresponding to the despatch method, as it is usually related to how the tickets will be sent. Within these elements will be one or more “collect_text” elements holding the actual text to be displayed, and as with the date and time options there may be more than one piece of text associated with each piece of information. A fourth piece of text “collect_confirmation_text” may be present if there is information that the customer should be made aware of before confirming that they want to purchase the tickets.

7.2.1 Example output

```
<availability_options_result>
```

<crypto_block>M0--JM6i0EQWP6-8XYFS5NtKtyH9n9jxN3okpA68pf0qZmu_3
V_aTkh0s2-VaPIeqyYmSrPflpYcGgI1N4ueM4rtMBAuHU0s-Chiqu58LuRCDGk6
aTfewQEJR0e3Jh3agWR8HxBKzW3bS09mym-lqjodJdM32pH2JMBZboMXX2u16MA
GffiMjj1a0adFFBviIwBwLsdDuQYfvi6-Z</crypto_block>

<availability>
<ticket_type>
<ticket_type_desc>Dress Circle</ticket_type_desc>
<price_band>
<ticket_price>36.000</ticket_price>
<surcharge>0.000</surcharge>
<number_available>4</number_available>
<band_token>U_--ATV757qGkCzGDsDy_gCsu0ZBFFQp1Svzf0RE81wJA6-bic_
3MR4vDntfdzeQuD5Wpi85avRMwrPGz7Pt9q09m50gMThRVv616kYbnEvV9fqatN
Wd02ycuV6DqXqPLBFbY</band_token>
</price_band>
</ticket_type>

<ticket_type>
<ticket_type_desc>Front Stalls</ticket_type_desc>
<price_band>
<ticket_price>33.000</ticket_price>
<surcharge>0.000</surcharge>
<number_available>4</number_available>
<band_token>U_--KZ0dpI-zHbEkkg_2xkORPEG6QYZ10iy85gv_KPpIUAbic_
3MR4vDHobW9p4lyP_pi85avRMwrPGz7Pt9q09mLXPV0Sa5hJVCfjP4V-HgiASYU
uR93hITX6DqXqPLBFbY</band_token>
</price_band>
</ticket_type>
</availability>

<quantity_options>
<valid_quantity>1</valid_quantity>
<valid_quantity>2</valid_quantity>
<valid_quantity>3</valid_quantity>
<valid_quantity>4</valid_quantity>
<valid_quantity>5</valid_quantity>
<valid_quantity>6</valid_quantity>
<valid_quantity>7</valid_quantity>
<valid_quantity>8</valid_quantity>
<valid_quantity>9</valid_quantity>
</quantity_options>

<despatch_options>
<despatch_method>
<despatch_type>collect</despatch_type>
<despatch_desc>Collect</despatch_desc>
<despatch_cost>2.200</despatch_cost>
<despatch_token>-_-HSah6JpCk1lRW6751B2tGMmQ6IWjrA9Re8VC-wenb8W
rAVpWWLH0pmfFRAKPZnaomxlMt2aMjmZMD2e38gPzc_--Y</despatch_token>

```

</despatch_method>

<despatch_method>
<despatch_type>post</despatch_type>
<despatch_desc>Post (uk only)</despatch_desc>
<despatch_cost>2.200</despatch_cost>
<despatch_token>M_--b1G0pkpUhA74yv2l_B3uKkP_TOjnfS4-BGSuUxTPI4E
Bi9_RumDFvcK06CuuYNGSwak0wgh99t4ikSliaoX5SbMBZDMmFBgxr10qe6cs8e
z3SuXC5tui90--Y</despatch_token>
<permitted_countries>
<country>
<country_code>uk</country_code>
<country_desc>The United Kingdom</country_desc>
</country>
</permitted_countries>
</despatch_method>

<despatch_method>
<despatch_type>post</despatch_type>
<despatch_desc>Post</despatch_desc>
<despatch_cost>3.750</despatch_cost>
<despatch_token>-_--ATP0VERai-3xC7y4v5IgyhRdukn1IxoAZII1vDd8Hrb
lFe-RN4DE9GK1j1dMhTcyvdhAGR1UvgEdwMZ3Lz_UB0--Y</despatch_token>
</despatch_method>
</despatch_options>

<currency>
<currency_code>gbp</currency_code>
<currency_number>826</currency_number>
<currency_pre_symbol>&#163;</currency_pre_symbol>
<currency_post_symbol/>
</currency>
</availability_options_result>

```

7.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “date_time_options”
- **2** — the requested ”mime_text_type” was not a supported value
- **301** — no “departure_date” has been supplied when required
- **302** — a “departure_date” has been supplied when not required
- **303** — no “usage_date” has been supplied when required
- **304** — a “usage_date” has been supplied when not required
- **305** — no “perf_token” has been supplied when required
- **306** — a “perf_token” has been supplied when not required
- **307** — the “departure_date” element was badly formatted

- **308** — the “usage_date” element was badly formatted
- **309** — the specified departure date is in the past
- **310** — the specified usage date is in the past
- **311** — the specified usage date is before the departure date
- **312** — the specified usage date is not allowed
- **313** — the supplied “perf_token” is corrupt
- **314** — the specified departure date is after the performance
- **315** — the supplied “trolley_token” is corrupt

8 The *discount_options* method

The tickets prices returned by the “availability_options” method standard tickets at whatever the default price is. In many cases there are discounts which may be applied to an order such as children’s tickets, or student discounts for example. This method takes the specification for a set of available tickets, and lists the discounts which may be applied to the tickets, including the standard price. For some products a single discount may be applied across the whole order only, whereas for others each ticket within the order may have an individual discount applied to it, though there may be a limit to the number of different combinations allowable in total.

Certain backend systems do not support discount codes, and products may only be sold at the face value given in the “availability_options” method. In this case there will be no data returned from this method other than the “crypto_block”, and this should be passed directly to “create_order”. This is very different from the method returning an empty list of discounts, which indicates that there are no valid discounts available for the selected product, and it cannot be purchased.

8.1 Required inputs

The “crypto_block” from the preceding availability method is required, along with a “band_token” specifying the desired price band, a “despatch_token” specifying the desired despatch method, and also an integer as “no_of_tickets”. The number of tickets must be one of the values from the “quantity_options” list and must not be greater than the number of available tickets in the specified price band. If a trolley is in use then this should be passed in as “trolley_token”, as this may affect the outcome of the method.

8.1.1 Example input

```
<discount_options>
<user_id>demo</user_id>
<crypto_block>M0--oX9hZ06WidmBy_GzZH_v3BTAN5QiFzArykxocsSIk2tu4
Z1LKIZOPBHgL5I2diqffFKYNgSrKOL8PuRAX-gL0stZ7Bv10-zLZ_Z8FJnLEK-Z
_dqCESP3hsmf7GyBcH6EVsX4PhjhdppKw1kFV_r71QjGnUIqz1Cg2QZRsvQrd1
ujFk0y8xRrk5LyTq4Q-3FLsdDuQYfvi6-Z</crypto_block>
<band_token>U_--KZ0dpI-zHbEkgg_2xkORPEG6QYZ10iy85gv_KPpIUnAbic_
3MR4vDHobW9p4lyP_pi85avRMwrPGz7Pt9q09mLXPV0Sa5hJVCfjP4V-HgiASYU
uR93hITX6DqXqPLBFbY</band_token>
<no_of_tickets>2</no_of_tickets>
<despatch_token>M_--b1G0pkpUHA74yv2l_B3uKkP_T0jnfS4-BGSuUxTPI4E
Bi9_RumDFvcK06CuuYNGSwak0wgh99t4ikSliaoX5SbMBZDMMFBgxrl0qE6cs8e
z3SuXC5tui90--Y</despatch_token>
</discount_options>
```

8.2 Output data

The output data always includes a “crypto_block” to be passed to the “create_order” method. If the order is for a product which cannot be discounted,

then no other information will be present in the output. For orders which may be discounted, however, there will always be a “blanket_discount_only” element and one or more “discounts” elements following the “crypto_block”.

The “blanket_discount_only” element contains a boolean value of “yes” or “no”. This indicates whether or not discounts may be selected for each individual ticket within the order, or whether a single blanket discount must be applied across the order as a whole. Blanket discounts are unusual, but may occur in such situations as a product which may be purchased at a normal rate, or a special offer rate, but not a mixture of the two. More common is for a limit to be placed on the number of different types of discount which may be applied to an order. If this is the case then an element “discount_limit” will be present, containing an integer indicating the number of different discount types that may be mixed in the order. In order to distinguish discount types there is a “discount_type” element associated with each discount. Discounts with the same “discount_type” are considered to be the same even though they will have different tokens.

For blank discounts there will only be one “discounts” element, else there will be a “discounts” element for each of the seats in the order. One discount should be chosen from each of the lists, taking care not to exceed any limit set by the “discount_limit” element.

A “discounts” element contains a list of “discount” elements. If a list is empty it indicates that there are no valid discounts for the order, and thus the order cannot be purchased. This may happen despite the apparent presence of ticket availability at the previous stage. The system does not guarantee the availability of any product until a reservation has been made. Each “discount” element in the list contains “ticket_price” and “surcharge” elements for that particular discount. These have the same meaning as in the “price.band” returns from the “availability_options” method. A token is provided to identify the discount in a “discount_token” element. Each discount normally has a “discount_desc” element containing the human readable description of the discount. The only discount which does not contain this is the one corresponding to the standard price for the product. There is not necessarily a standard discount available in the list, but if so then it will be indicated by the lack of a description. Each “discounts” list will contain different tokens for the elements of the list so in order to be able to tell how many different types of discount have been selected a “discount_type” element is present for each, which indicates which tokens refer to the same type of discount.

As with all other pieces of information there may be extra text to be displayed when prompting for a discount code. If any such text exists it will be found in the optional “collect_discounts_text” element, with each piece of text being within a “collect_text” element. This is the same as for the extra text associated with the date/time options and the availability options.

8.2.1 Example output

```
<discount_options_result>
<crypto_block>U1--Nq2HPJSjr9Crde1qd1Mo4566Y3uXgV5uorhh00Tj7Uema
MCdc_OdGkbgD7w8NVmr80hEB45iIGe03wfRwn1U-9_kWJcvIXy3Q0oPq5qwP-85
WNAEBo4CfCn0u6dS8czfD1H-0y0tLFeVCb6Aqgn41_5pJry1jlp8oGgcKXVtPtp
owaoG6ieefxBL12spf1RDgjtjpcqfykcyVZgeQrSiMiENVrdsHOVLDMh9TPSJyq
```

```

7qk2KQwEvabA0rs-vW9otaXdPAJmVE8MCUS60TYJ-hDInDdo5roes5i4UpGGzuq
yA-Z</crypto_block>
<blanket_discount_only>no</blanket_discount_only>
<discount_limit>3</discount_limit>

<discounts>
<discount>
<discount_desc>First Call Standard Sale</discount_desc>
<ticket_price>33.000</ticket_price>
<surchage>0.000</surchage>
<discount_token>M_--IbGqi00j3yb0pqyoBkzpk7VaFCjbF7_8CJX4e5KsEP1
3SM94d2GAQWwBs4LIQjijy4IGDFmAA14aDA1A1Cgvba0db5kQwqgIYZYts850V7
xYDjouGoFr30--Y</discount_token>
<discount_type>0</discount_type>
</discount>

<discount>
<discount_desc>Westminster Residents Card.</discount_desc>
<ticket_price>16.500</ticket_price>
<surchage>0.000</surchage>
<discount_token>M_--8hntACS_vMmM5of6SXwEpKI9dhTztSubkH0tXwxPkd
S1BHV5991VT4h_wD5hIJDbMsZqlwT1-7ga9Lbp4NOMdKUf_Pgwja5W10bZwQfk
7_GOEFywVtp---Y</discount_token>
<discount_type>1</discount_type>
</discount>
</discounts>

<discounts>
<discount>
<discount_desc>First Call Standard Sale</discount_desc>
<ticket_price>33.000</ticket_price>
<surchage>0.000</surchage>
<discount_token>Q_--8gf6jbc0j3yb0pqyoBkzpk7VaFCjb7_8CJX4e5KsEP1
3sM94d2GAQWwBs4LIQjijy4IGDemAA14aDA1A1Cgvba0db5kQwqgIYZYts850V7
xYDjouGoFr30--Y</discount_token>
<discount_type>0</discount_type>
</discount>

<discount>
<discount_desc>Westminster Residents Card.</discount_desc>
<ticket_price>16.500</ticket_price>
<surchage>0.000</surchage>
<discount_token>F_hgt8ntACS_vMmM5of6SXwEpKI9dhTztSubkH0tXwxPkd
S1BHV5991VT4h_wD5hIJDbMsZqlwT1-7ga9Lbp4NOMdKUf_Pgwja5W10bZwQfk
7_GOEFywVtp---Y</discount_token>
<discount_type>1</discount_type>
</discount>
</discounts>

</discount_options_result>

```

8.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “availability_options”
- **2** — the requested ”mime_text_type” was not a supported value
- **401** — no “band_token” has been supplied
- **402** — the supplied “band_token” is corrupt
- **403** — no “despatch_token” has been supplied
- **404** — the supplied “despatch_token” is corrupt
- **405** — no “no_of_tickets” element has been supplied
- **406** — the “no_of_tickets” element is not an integer
- **407** — the requested number of tickets is not an allowed value
- **408** — the supplied “trolley_token” is corrupt

9 The *create_order* method

Given a “crypto_block” from the “discount_options” method, along with a set of discount tokens, this method produces an order object incorporating those discounts, which can then be added to a trolley for reservation and purchasing. For products where only a single blanket discount can be applied across the whole order then only a single discount token is required at the input, else a discount for each ticket must be present, one take from each of the lists of discounts returned by the “discount_option” method. The discounts do not need to be present in any particular order, and groups of the same discount do not need to be presented together. If discounts are not supported then only the “crypto_block” should be presented at the input, and no discount tokens at all.

9.1 Required inputs

The “crypto_block” from the “discount_options” method must be present. If discounts are not supported then this is all that is required. If a blanket discount is required then a single “discount_token” should be supplied, which will be applied across the whole order. Else a “discount_token” should be present for each ticket in the order, one from each of the discount lists, such that the total number of tokens equals the total number of tickets in the order.

9.1.1 Example input

```
<create_order>
<user_id>demo</user_id>
<crypto_block>U1--Nq2HPJSjr9Crde1qdlMo4566Y3uXgV5uorhh00Tj7Uema
MCdc_OdGkbgD7w8NVmr80hEB45iIGe03wfrwn1U-9_kWJcvIXy3Q0oPq5qwP-85
WNAEBo4CfCnOu6dS8czfD1H-0y0tLFvVCb6Aqgn41_5pJry1jlp8oGgcKXVtPtp
owaoG6ieefxBL12spf1RDgjtjpcqfykcYVZgeQrSiMiENVrdsHOVLdMh9TPSjyq
7qk2KQwEvabAOrs-vW9otaXdPAJmVE8MCUS60TYJ-hDInDdo5roes5i4UpGGzuq
yA-Z</crypto_block>
<discount_token>M_--IbGqi00j3yb0pqyoBkzpk7VaFCjbf7_8CJX4e5KsEP1
3SM94d2GAQWwBs4LIQjijy4IGDFmAaL4aDA1A1Cgvba0db5kQwqgIYZYts850V7
xYDjouGoFr30--Y</discount_token>
<discount_token>Q_--Ightfvxcj3yb0pqyoBkpk7VaFCjbf7_8CJX4e5KsEP1
httpuhwevtgHXX4LIQjijy4IGDFmAaL4aDA1A1Cgvba0db5kQwqgIYZYts850V7
xYDjouGoFr30--Y</discount_token>
</create_order>
```

9.2 Output data

The output simply consists of the fully discounted order contained in an “order_token” element. The “crypto_block” produced here only contains the authentication information and is equivalent to one generated from “start_session”. It can thus be used as the starting point for a new search, or else as the authentication input for any of the trolley handling methods.

9.2.1 Example output

```
<create_order_result>
```

```
<crypto_block>U_--LfP7rp3MoitCr2ofVaNShLkd0lMurxfADnkYLS0FxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoeiun5eRalXx5Tz-H7Q7UreSD
lXCTGZEvwqddnGIUevnZY</crypto_block>
<order_token>E1--_mjRuXT_ko3XfykAhrLF2pU83_vWRy2eo2GoU9K0eM8D7b
UtQfbYDKQI5wjcxNkQTawn1t0Uky8dHP9v_uT5yWnr3610uYUEh0a3xJVCr7zMH
s58F-3to-nZ7djPUY085aJnq4PTKDRFizaLrId_Sy6H1Iw76yy3KcJEHCAAydx
SMrtV4Mx3uLcWGX5DUzFnXs4MId_FllMAqUksPTnn9npn7LybccDxU0pai2HJ4
IwJ6rJkvmEeoreJIIHKrSeeQzfyUY483T1upBF445xK_--Z</order_token>
</create_order_result>
```

9.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “discount_options”
- **2** — the requested ”mime_text_type” was not a supported value
- **501** — a corrupt “discount_token” has been supplied
- **502** — discounts tokens supplied when the product order not support discounts
- **503** — no discount tokens were supplied
- **504** — multiple discount tokens were supplied for an order which requires a blanket discount
- **505** — the wrong number of discount tokens were supplied for the order in question
- **506** — the number of different discount types provided exceeds the allowed limit
- **507** — more than one discount token has been supplied from the same “discounts” element

10 The *trolley_add_order* method

This method is used to bundle single orders into a single block known as a “trolley”. It is this trolley which is the unit used to reserve and then purchase orders. This method takes an order token and an optional trolley token and adds the order to the trolley. If no trolley token is supplied then a new empty trolley is created for the order to be added to. Not all orders may be mixed with all other orders for a variety of reasons, and thus this method may not always succeed. In the situation where the order cannot be added then a set of elements is returned indicating the reasons why the operation could not be performed. Attempts to add an order for the same event on the same date twice to a trolley will succeed, but will result in the original order being replaced. The number of items in the trolley is returned as part of the output, so it is easy to detect if this has occurred. For ease of developing a client for the code, it is possible for this method to include a description of the new trolley at the output following a successful addition, via the use of a single input element. Orders which have been added to a trolley are assigned a unique item number, which is never re-used during the lifetime of a single trolley. This item number provides a reference through which an order within a trolley can be identified for such purposes as removal.

10.1 Required inputs

A “crypto_block” from “start_session” method must be present. This may come from the original “start_session” or may be one of the crypto blocks produced from any of the other methods which are documented as generating a “start_session” compatible block. In particular the output of “create_order” and all the trolley modifying methods (including this one) produce such a block. An “order_token” must be present. An optional “describe_trolley” element may also be present if the full trolley contents are required in the output result. The contents of this element are irrelevant as it is merely its presence which is taken into account.

10.1.1 Example input

```
<trolley_add_order>
<user_id>demo</user_id>
<crypto_block>U--LfP7rp3MoiCr2ofVaNShLkd01MurxfADnkYLSOFxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoeiun5eRalXx5Tz-H7Q7UreSD
lXCTGZEvwqddnGIUevnZY</crypto_block>
<order_token>E1--_mjRuXT_ko3XfykAhrLF2pU83_vWRy2eo2GoU9K0eM8D7b
UtQfbYDKQI5wjcxNkQTawn1t0Uky8dHP9v_uT5yWnr3610uYUEh0a3xJVCr7zMH
s58F-3to-nZ7djPUY085aJnq4PTKDRFizaLrId_Sy6H1Iw76yy3KcJEHCAAydx
SMrtV4Mx3uLcWGX5DUzFnXs4MIId_FllMAqUksPTnn9npgn7LybccDxU0pai2HJ4
IwJ6rJkvmEeoreJIIHKrSeeQzfUY483T1upBF445xK--Z</order_token>
</trolley_add_order>
```

10.2 Output data

The output data from the method always contains an “add_possible” element with is a either “yes” or “no”. This indicates whether the addition of the order into the trolley was possible or not. Where an addition cannot be performed there are additional elements provided, all beginning with the prefix “trolley_bad_”, which are used to indicate the reason for the failure. There are seven of these elements, and all of them are always present in the output, each one containing either a “yes” or a “no” value. There may be more than one reason why an addition cannot be made, and thus it is not necessarily the case than only a single element will have a “yes” value. For some of the elements there will be additional elements added in the case of a “yes” to indicated more detailed information about the reason for the failure. The seven failure reasons are as follows:

- **trolley_bad_bundle**

A single supplier of tickets may only bundle up to a certain number of orders at any one time (this may be “one” for systems which do not support the concept of trolleys). This element indicates that no more orders for the particular supplier may be added to the trolley. An extra element “trolley_bad_bundle_max_size” will be present if this is true, which contains an integer indicating the maximum number of allowable orders from this supplier which may be placed in the trolley.

- **trolley_bad_combo**

Some users and channels are allowed to mix tickets from different suppliers in the same trolley, and some are not. This element indicates that the mixing of different suppliers is not allowed, and that the order you are attempting to add is from a different supplier to those already in the trolley. If this is true the extra elements “trolley_bad_combo_system” and “trolley_bad_combo_system_desc” will be present describing the existing supplier. The first holds the code for the ticket supplier, and the second the textual description of the supplier. These are the same code and text normally referred to as “source_code” and “source_desc” in other parts of the XML core.

- **trolley_bad_card_types**

Each supplier of tickets has a certain list of card types which are accepted for payment. This element indicates that there are no cards accepted by the supplier of the current order that are also accepted by the suppliers of the orders already in the trolley. As a single payment card is taken for purchase of the entire trolley then it is necessary that there is at least one acceptable card type in common across all the various suppliers in the trolley.

- **trolley_bad_countries**

Some despatch methods that require delivery of the tickets to the customer are restricted to a certain list of countries to which they may be sent. This element indicates that the current order has a list of countries which does not overlap with the list of countries in the current trolley. As only a

single delivery address is collected for despatch, then there needs to be at least one acceptable country in common across all suppliers in the trolley. This is a very similar restriction to the preceding card type restriction.

- **trolley_bad_currency_mix**

A single bundle of tickets from one of the ticket suppliers is purchased using a single debit on the payment card. For this reason all orders from a particular supplier must be priced in the same currency. This element indicates that there are already orders present from this supplier in the trolley, and that these orders are priced in a different currency to the current order. The elements “trolley_bad_currency_system” and “trolley_bad_currency_system_desc” hold the code and description of the ticket supplier, in the same way as for the “trolley_bad_combo” element. In addition the actual currency details for the existing supplier are present in the elements “trolley_bad_currency” and “trolley_bad_currency_name” etc, carrying all the information usually associated with a currency, but with the prefix “trolley_bad.”. The usual codes, places and symbols are all present.

- **trolley_bad_depart**

Some orders require a departure date to be specified for various reasons, usually related to despatch of tickets. A trolley needs to have a single departure date for all tickets from all suppliers, and thus this element indicates that the current order has a different departure date to the orders already in the trolley. This restriction only applies to orders which need a departure date. Orders without may be freely combined with orders that do have a date.

- **trolley_bad_send**

A bundle of orders from a single supplier is purchased as a single transaction, and thus has a single despatch method applied to all of the orders. This element indicates that the current order has a different basic despatch (a.k.a “send”) type from those already in the trolley from this supplier. This restriction can be avoided by passing in the current trolley to the “availability_options” method as this will result in only those despatch methods which are allowable being returned.

Where the “add_possible” element contains a “yes” value then the trolley addition has succeeded. A “start_session” compatible “crypto_block” element is produced which may be used as input for any of the other trolley manipulating methods, plus a token for the new trolley in the element “trolley_token”. Additions does not always result in an increase in the number of items in the trolley, as only a single instance of a particular event on a particular date may be purchased, and thus an addition of another order for the same event on the same date will result in the original being replaced. The number of orders in the new trolley is always available in the element “trolley_order_count” following a successful addition. The item number allocated to the newly added order is available in the “added_item_number” element.

If the “describe_trolley” element was present in the input request then a complete description of the trolley will be found in the element “trolley”. For documentation on this please refer to the “trolley_describe” method.

10.2.1 Example output

```
<trolley_add_order_result>
<crypto_block>U--LfP7rp3MoitCr2ofVaNShLkd0lMurxfADnkYLS0FxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoeerLZptl3ybUpxBT61BnuGMD
lXCTGZEvwqddnGIUevnZY</crypto_block>
<add_possible>yes</add_possible>
<trolley_token>k1---IHQgb4xt2KimGiw-fPma50h1PFVyQJAx1hTrvv60DMz
73WJH8WgUVjBoxB5BLQo9pEilPGEmlk0uVX1Xb5gdejndvVPYmf205VU1PVYwtd
zGfr1vBNznQp80YLwx7kwNbQAFbcF_KlquVRXGLIH9YFGQARVJWGGLUDixXEkyq
fsF7DLEHd6kJYTaNOjzPGMMLdWsMIh1Nnms4jrDs1vzjUBVrv_uNXOCK4ezWxgp
jy1toCmQ-aWcrxk07N-U1-Z9HX44b1CWwXDxVxecEhdjSUPrcKIBOY6ZaHQABER
XTyMSsp3j4SZ36ESZjbOP8PfZ</trolley_token>
<trolley_order_count>1</trolley_order_count>
<added_item_number>0</trolley_order_count>
</trolley_add_order_result>
```

10.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “start_session”
- **2** — the requested ”mime_text_type” was not a supported value
- **601** — no “order_token” has been supplied
- **602** — the supplied “order_token” is corrupt
- **603** — the supplied “trolley_token” is corrupt
- **604** — the trolley has already been purchased
- **605** — the trolley has already been reserved

11 The *trolley_describe* method

This method takes a “trolley_token” as an input and produces a full description of the contents of the trolley. This is broken down in to “bundles” which correspond to a set of orders from a particular supplier, and within each bundle the orders are listed individually in the order in which they were added to the trolley. A trolley may be described at any point, including after it has been purchased. Extra information which is acquired during the lifetime of the trolley, such as reference numbers and seat allocations, will be described in the output if present.

For the sake of efficiency, all methods which manipulate the contents of a trolley have the ability to produce a trolley description in their output. This is to avoid the need to perform a trolley operation and then follow it immediately with a call to this method in order to display the results. The final purchase method always produces a full trolley description, as this contains all the reference codes, seat IDs and final despatch methods produced by the purchase operation.

11.1 Required inputs

A “crypto_block” from “start_session” method must be present. This may come from the original “start_session” or may be one of the crypto blocks produced from any of the other methods which are documented as generating a “start_session” compatible block. A “trolley_token” holding the trolley to be described must also be present.

11.1.1 Example input

```
<trolley_describe>
<user_id>demo</user_id>
<crypto_block>U_--LfP7rp3MoitCr2ofVaNShLkd01MurxfADnkYLSOFxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoeODYcVYViU_lXNIqmcKQa9ID
lXCTGZEvwqddnGIUevnZY</crypto_block>
<trolley_token>c3--ESt4iSq-Oh6ewW5aurWrMlaHazPjAoh6aiaqiyucuEQx
b5d3JTr2KnNS_eNj_LC5-RHyLa40-cKULcj1lwjpSzx49vJ0mniPPBFgDeZRkfK
gz1x-gLCqsvdp1DTkgKyWu2_GM6_TV7DX6G1fLRCM7oX6D8j802Rz81NB2VI7hy
dIva0ixEfHu48UMiYd0REp9XDIwdoYi4R0tF05ZQt4KUzKRp7Rn9i0UNPyIVtMM
2IRP6ob-Dy-054kp0GnFihWBXBkP4kmSM3L0wxCBxVxjq7y2V1RIGg0Q2w1-w8k
-Ui17qiKMn_ZflHByRyv5MBSkiXrJF0ZLKRboPT73uq5oxz3NgrHiCeGULnF345
zp6Jz94huDRbV6w4nSv7CVBgJ_H7iZZJcf08Z5q570Ao09e9hZe44ZDG15H1L61
QD3KDyCzUS3EH118JMEoOkQF1Xb4w2joeUbTP7IPhVPUS4zBNcfQGmoc-IZ
</trolley_token>
</trolley_describe>
```

11.2 Output data

The complete trolley description is entirely held within a “trolley” element. If the trolley has been reserved this begins with the system wide reference code in a “transaction_id” element. A trolley is organised as a set of orders, grouped into “bundles”, with each bundle corresponding to a single ticket supplier. The total number of orders within the trolley is present in the “trolley_order_count”

element, and the total number of bundles as “trolley_bundle_count”. If the trolley has a departure date then this is in the element “departure_date_yyyymmdd” in YYYYMMDD format, together with a human readable form in “departure_date_desc”. This is then followed by the individual bundles, each within a “bundle” element.

For trolleys which have been purchased there will be a “purchase_result” element present after the individual “bundle” elements. This always contains a single element “success” which has the value “yes” or “no”. As a purchase may only be partial there is also an element “is_partial” which contains “yes” if the trolley contains failed bundles, else “no”. In the case where the purchase was not successful and thus this element contains “no” then two boolean elements will also be added into the “purchase_result” elements. These are “failed_cv_two” and “failed_avs”. They contain either “yes” or “no” and are used to indicate the reason for a card failure when it is definitely known. If these flags are “no” then it does not mean that the AVS or CV2 check passed, merely that no information is available from the backend. Finally a “purchase_error” element is added for failed purchases, which holds one of the following strings to indicate the cause of the failure.

- **refusal** — one or all of the backend systems refused the booking
- **auth_failure** — one or all of the payment card debits failed authorisation
- **auth_timeout** — one or all of the payment card debits timed out whilst authorising
- **fraud_blocked** — the booking details were refused as they matched a system fraud trigger
- **already_purchased** — an error caused by attempting to purchase a second time. This should never happen.
- **already_failed** — an error caused by attempting to purchase a second time. This should never happen.
- **unspecified** — some unspecified system error has occurred

The bundle elements group together all the information common to a set of orders from a single supplier. The supplier’s descriptive name is always present in “bundle_source_desc”, with the corresponding code in “bundle_source_code”. The number of orders in the bundle is held in “bundle_order_count”, and the collective costs associated with a bundle are present in “bundle_total_seatprice”, “bundle_total_surcharge” and “bundle_total_despatch”, with a grand total being calculated and placed in “bundle_total_cost”. All costs are to three decimal places and the currency of these costs is denoted by the “currency” element. This element always contains five sub elements; “currency_code” which holds the three letter ISO4217 code for the currency, “currency_number” which holds the ISO4217 numeric code for the currency, “currency_places” which holds the number of decimal places that the currency is subdivided into (normally two), plus “currency_pre_symbol” and “currency_post_symbol”. These latter two hold the symbol necessary to display the currency. One of them is normally blank and using the two values to bracket the currency will give a correct display of

the value. Following the “currency” element, the individual orders in the bundle are then described, each in an “order” element.

Purchased bundles, as with purchased trolleys, have a “purchase_result” element present after the individual orders. This always contains a “success” element which has the value “yes” or “no”. If the value is “no” then the element will also contain an element “failure_reason” which is one of “auth_failure”, “auth_timeout” or “unknown” to give some indication, if known, as to why the purchase failed. Individual CV2 and AVS failures are not reported. For successful bundles there will also be an element “is_semi_credit”. This will normally be “no” except in the case of certain on-credit sales to particular users. When this flag is “yes” it indicates that the full agent cost needs to be paid to the ticket supplier before the tickets will actually be despatched, despite the fact that the sale was made on-credit. For on-credit sales, whether semi-credit or not, there may also be an “agent_cost” element. Certain suppliers provide this when the cost of the tickets to the agent is different to the face value. This element contains the cost in a “total_agent_cost” element, plus another “currency” element as the cost to the agent is not necessarily in the same currency as the face value of the tickets.

The individual “order” elements hold all the details about a single order within the bundle. Each order begins with an “item_number” element, holding the integer which uniquely identifies it within the entire trolley. Item numbers start at zero and increment as more orders are added. They are never re-used, even when an order is removed from a trolley. For orders in a purchased trolley, the reference number for the purchase which was issued by the backend system is to be found in “backend_purchase_reference”. This is normally the same across all orders in a bundle as they are purchased in a single operation, but it is held at the order level to support those systems which provide individual references for each order. The venue at which the event is taking place is described in “venue_desc”, followed by the event itself in “event_desc”.

For orders where the date on which the tickets are to be used is known, there now follows either a “performance” or a “usage” element, depending on whether the date is a time specific performance (such as for theatre events) or merely a day on which the tickets are to be used (such as for theme parks). Both elements will contain the date, in the elements “date.yyymmdd” and “date_desc” for machine and human readable forms respectively. The “performance” element may additionally contain the time, held in “time_hhmmss” and “time_desc”, for performances where a time is relevant.

The text describing how the tickets are to be despatched is held in the “despatch_desc” element. For orders which have not been purchased then this is the only information regarding the despatch method available. After purchase, however, a second element “despatch_final_type” containing either “post” or “collect” depending on whether the tickets are to be sent to the customer, or if they have to collect them at the venue. If any special conditions are associated with the actual despatch method then these will be present in the element “despatch_final_comment” and this text should be displayed to the customer.

The type of ticket for the event is described in the “ticket_type_desc” element. This is then followed by one or more “discount” elements, each holding a set of tickets at a single price. If a discount element is not the default price then a description of the discount is present in the “discount_desc” element. The price of each tickets is in “seatprice”, and the per ticket surcharge in “surcharge”.

The total number of tickets in the discount is present in “no_of_tickets”. If individual seat IDs have been allocated then a “seats” element will be present, containing one string for each ticket in a set of “id” elements. For each of these there is also a corresponding “id_details” element which contains the ID broken down by row and column, together with an separator between them. Seats are allocated by some systems at reserve time, and others at purchase time. For certain systems there is no guarantee that actual IDs will ever be allocated at all.

Finally the total price of all seats in the order is held in “total_seatprice” with the total surcharge in “total_surcharge”. The total number of tickets in the order is in the element “total_no_of_tickets”. If there is any commission to be paid to the affiliate for this order then an element “affiliate_commission” is also added. This contains the amount in an element called “value” and also a full “currency” element. This is because the commission may be in a completely different currency to the sale currency, and may vary across a bundle depending on the source currency of the product.

11.2.1 Example output

```
<trolley_describe_result>
<trolley>
<transaction_id>4A2B-7AC9-F630-2749</transaction_id>
<trolley_order_count>1</trolley_order_count>
<trolley_bundle_count>1</trolley_bundle_count>

<bundle>
<bundle_source_desc>Keith Prowse Ticketing</bundle_source_desc>
<bundle_source_code>fcg3</bundle_source_code>
<bundle_order_count>1</bundle_order_count>
<bundle_total_seatprice>27.500</bundle_total_seatprice>
<bundle_total_surcharge>4.150</bundle_total_surcharge>
<bundle_total_despatch>1.500</bundle_total_despatch>
<bundle_total_cost>33.150</bundle_total_cost>

<currency>
<currency_code>gbp</currency_code>
<currency_number>826</currency_number>
<currency_pre_symbol>&#163;</currency_pre_symbol>
<currency_post_symbol/>
</currency>

<order>
<item_number>1</item_number>
<backend_purchase_reference>AF291</backend_purchase_reference>
<venue_desc>The Dominion Theatre</venue_desc>
<event_desc>We Will Rock U</event_desc>
<performance>
<date_yyyymmdd>20031230</date_yyyymmdd>
<date_desc>Tue, 30th December 2003</date_desc>
<time_hhmmss>143000</time_hhmmss>
```

```

<time_desc>2.30 PM</time_desc>
</performance>
<despatch_desc>Post (uk only)</despatch_desc>
<despatch_final_type>post</despatch_final_type>
<despatch_final_comment>Please bring your credit card with you
when you collect your tickets.</despatch_final_comment>
<ticket_type_desc>Stalls</ticket_type_desc>
<discount>
<discount_desc>First Call Standard Sale</discount_desc>
<seatprice>27.500</seatprice>
<surcharge>4.150</surcharge>
<no_of_tickets>1</no_of_tickets>
<seats>
<id>WW40</id>
<id_details>
<row_id>WW</row_id>
<separator/>
<col_id>40</col_id>
</id_details>
</seats>
</discount>
<total_seatprice>27.500</total_seatprice>
<total_surcharge>4.150</total_surcharge>
<total_no_of_tickets>1</total_no_of_tickets>
<affiliate_commission>
<currency>
<currency_code>gbp</currency_code>
<currency_number>826</currency_number>
<currency_pre_symbol>&#163;</currency_pre_symbol>
<currency_post_symbol/>
</currency>
<value>1.500</value>
</affiliate_commission>
</order>
<purchase_result>
<success>yes</success>
<is_semi_credit>no</is_semi_credit>
</purchase_result>
</bundle>
<purchase_result>
<success>yes</success>
<is_partial>no</is_partial>
</purchase_result>
</trolley>
</trolley_describe_result>

```

11.3 Potential failure codes

- 1 — the supplied crypto block was not generated by “start_session”

- **2** — the requested "mime_text_type" was not a supported value
- **701** — no "trolley_token" has been supplied
- **702** — the supplied "trolley_token" is corrupt

12 The *trolley_remove* method

This method takes a “trolley_token” plus a list of orders to remove, specified by their item numbers, and produces a trolley with those items removed from it. More than one item may be removed at a time, and an item may be removed more than once without error. As item numbers are not re-used within a trolley then it is not necessary to check that item numbers are actually present in the trolley, as the end result is a trolley absent of those items. As with all other trolley manipulation methods, it is possible to ask the method to produce a full description of the altered trolley in the output XML tree.

12.1 Required inputs

A “crypto_block” from “start_session” method must be present. This may come from the original “start_session” or may be one of the crypto blocks produced from any of the other methods which are documented as generating a “start_session” compatible block. A “trolley_token” holding the trolley from which the items are to be removed must also be present. The items are then specified by zero or more “remove_item” elements, each containing an integer specifying the item number of the order to be removed.

An optional “describe_trolley” element may be present to indicate that the output data should contain the full trolley description.

12.1.1 Example input

```
<trolley_remove>
<user_id>demo</user_id>
<crypto_block>U_--LfP7rp3MoitCr2ofVaNShLkd0lMurxfADnkYLSOFxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoeODYcVYViU_lXNIqmcKQa9ID
lXCTGZEvwqddnGIUevnZY</crypto_block>
<trolley_token>k1---IHQgb4xt2KimGiw-fPma50h1PFVYQJAX1hTrvv60DMz
73WJH8WgUVjBoxB5BLQo9pEi1PGEmlk0uVX1Xb5gdejnDvVPYmf205VU1PvYwtd
zGfr1vBNznQp8OYLwx7kwNbQAFbcF_KlquVRXGLIH9YFGQARVJWGGLUDixXEkyq
fsF7DLEHd6kJYTaN0jzPGMLdWsMIh1Nnms4jrDs1vzjUBVrv_uNXOCK4ezWxgp
jy1toCmQ-aWcrxk07N-U1-Z9HX44b1CWwXDxVxecEhDjSUPrcKIBOY6ZaHQABER
XTyMSsp3j4SZ36ESZjbOP8PfZ</trolley_token>
<remove_item>2</remove_item>
<remove_item>0</remove_item>
</trolley_remove>
```

12.2 Output data

The output contains a “crypto_block” marked as coming from “start_session” for use in any subsequent calls, together with the modified trolley in a “trolley_token” element. The number of items in the modified trolley is present as “trolley_order_count”. If the “describe_trolley” element was present at the input then a “trolley” element will be produced in the output containing the description of the new trolley as documented for the “trolley_describe” method.

12.2.1 Example output

```
<trolley_remove_result>
<crypto_block>U_--LfP7rp3MoiCr2ofVaNShLkd01MurxfADnkYLS0FxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoe8EDBCmbWAKI6wq5NjH60XHD
lXCTGZEvwqddnGIUevnZY</crypto_block>
<trolley_token>k---w8x03m6u8WH90wHc5Z9zQAeJBtYfbKE0oi5Xy5DgPJnp
P8i4zXEux9q1V0V0vqDOY</trolley_token>
<trolley_order_count>0</trolley_order_count>
</trolley_remove_result>
```

12.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “start_session”
- **2** — the requested ”mime_text_type” was not a supported value
- **801** — no “trolley_token” has been supplied
- **802** — the supplied “trolley_token” is corrupt
- **803** — the trolley has already been purchased
- **804** — the trolley has already been reserved

13 The *make_reservation* method

This method is used to take a trolley and to make a reservation for all the orders contained within it. Due to the real-time nature of the ticketing systems it is possible that a product which was available when an order was placed in the trolley has since ceased to be available, thus it is not guaranteed that all the items in the trolley will be reserved successfully. This method may therefore have a “partial success” status in which only a subset of the orders were reserved successfully. The failed orders will be listed in the output if this occurs.

Once a set of orders has been reserved, it is guaranteed to be held for a certain length of time exclusively for the user. During this time an appropriate set of information necessary to complete the purchase must be collected and the reservation purchased. At the end of the time period the reservation will be released back into the pool. The output from this method indicates the information which is needed to make a successful purchase. This always consists of the customer information, but may or may not require payment card details. If a payment card is required then the types of cards which are accepted will be indicated in the output. Customer data includes both an email address and also an “agent reference” which can be generated by the user of the core. These are normally optional, but under certain circumstances may be compulsory. If this is the case then this will also be indicated in the return. If any of the address data may be prefiled then this is also passed back, together with a flag indicating whether or not it may be modified.

13.1 Required inputs

A “crypto_block” from “start_session” method must be present. This may come from the original “start_session” or may be one of the crypto blocks produced from any of the other methods which are documented as generating a “start_session” compatible block.

If the “despatch_token” requested corresponds to a method with despatch type of “selfprint” then a “self_print_mode” input is required, with the same value as that passed to the “availability_options” call. For simplicity, is safe always to pass this input even if a non-selfprint method was chosen.

An optional “describe_trolley” element may be present to indicate that the output data should contain the full trolley description.

13.1.1 Example input

```
<make_reservation>
<user_id>demo</user_id>
<crypto_block>U_--LfP7rp3MoitCr2ofVaNShLkd01MurxfADnkYLSOFxUoQo
YUZz_Kq23SaaVvPmF43Ndbq2o1t5Nq60tvNucAoeODYcVYViU_lXNIqmckKQa9ID
lXCTGZEvwqddnGIUevnZY</crypto_block>
<trolley_token>M2--2GJtF2mPyKFtMCwqIIB_-MwqruvxEvZd65EDPSomn7jp
aY8wMlDEfE0aR8n4BVFihQ9qfSrxAlTJWCaScnRZTALRMyVpi4yAIurPnH28Jv
nXcDTP53yUcoi0c0sTpY80I9jzXnWDZrGmyryYP_WSrY31S9yeu5JmXolM9iV1M
d7Y3YED7aJwzWIeqkB7h_saLv2WaHL9j-5f-ORutfVPSsYZKXgQLmCiTTebkq1U
yLE2lSeUdvSjKc8XqZENBc_id0r5hTMQvmxhyCBgdZR5Bjy6gOorFCr1XyCqb08
7hFMxHwXW605tAHVHnvB3iSPgVvvZY0keGuQjSiSDGuLSCsa6FP2Rdb7nZsxd0o
```

```
R6LK1HWSKjGHo5---Z</trolley_token>
</make_reservation>
```

13.2 Output data

It is possible that none of the orders in the trolley are available when the reservation request is made, and in this case a completely empty response will be returned. This does not indicate an error from the cores point of view, however, as this is part of the normal operation of the system.

In most cases there will be one or more orders in the trolley which will be placed on reservation in order that they may be purchased. In this case a “crypto_block” will be produced for use as input to the purchase method, or possibly the release method if the purchase is abandoned. The allocated reference for this transaction is available in the “transaction.id” element, and the number of minutes that the reservation will remain held for is returned as “minutes_left_on_reserve”. This is a decimal value and may change according to the amount of time it takes to actually make the reservation, as the time limit begins when the first successful reservation is made from the orders within the bundle. If any of the orders have failed to be reserved then these will be found within the “failed_orders” element. This is always present, but will be empty if the entire trolley was reserved without problem. In the case where some orders failed, then these are described as individual “order” elements within this element, each having the format it would in a trolley description. The failed orders are presented as a flat list, and are no longer grouped into bundles.

Purchases may be made from the system either on-credit (with the user of the core taking the payment, and being invoiced later for the amount) or directly though the core, with the payment being taken directly from a customer’s card. Which of these systems is in use is indicated by the boolean element “need_payment_card” which is always present in the output. If this has the value “yes” then payment card details must be provided. The cards which will be accepted for the transaction are then listed in the element “acceptable_cards” which will be present if a card is required. This consists of a set of “card” elements, each containing the card type in “card_type” and a human description of the type in “card_desc”. The current list of known types is as follows.

- **access** — Access
- **amex** — American Express
- **diners** — Diners Club
- **discover** — Discover
- **electron** — VISA/Electron
- **fraser** — House Of Fraser
- **jcb** — JCB
- **magasin** — Magasin
- **marks** — Marks and Spencer

- **mastercard** — Mastercard
- **otb** — OTB
- **searsuk** — Duet/Sears
- **solo** — Solo
- **style** — Style
- **switch** — Switch (with issue number)
- **switch_ni** — Switch (without issue number)
- **visa** — VISA/Delta

Some systems are capable of making an address check against a credit card before allowing it to be debited. Under normal circumstances this will be the address provided by the customer. Certain systems may allow a separate billing address to be specified so that the card used for payment need not be registered to the customer’s address. The boolean element “supports_billing_addr” indicates if this is the case. If it is “yes” then a separate address may be provided at purchase time, and this will be used to validate the card address should any such check be made.

For those channels which require an email address the boolean element “needs_email_address” will be present containing the value “yes”. For normal channels this will be set to “no”. Certain users may also require that an agent reference is always supplied (this is normally optional) and if so the boolean element “needs_agent_reference” will contain the value “yes”. If the reference is optional it will be set to “no”.

Some commercial users, particularly those purchasing on-credit, will have an address already held in the system which should be prefilled for the user. If this is the case then an element “prefilled_customer_data” will be present containing all the address fields, the phone numbers and the email address to be prefilled (even if they are blank). This address is sometimes fixed and may not be edited by the user at all. This is indicated by the “can_edit” element within the prefilled data, which is either “yes” or “no”. Note that even if the address cannot be edited, it is still permissible to fill in any prefilled fields which are blank.

The reserved trolley is embedded in the “crypto_block” to be used as input to the next stages of the process, but so that it can also be described if necessary using the “trolley_describe” method then “trolley_token” and “trolley_order_count” elements are also present in the output. As with the trolley modification methods, it is possible for the full description of the reserved trolley to be presented in the output by the addition of a “describe_trolley” element at the input.

13.2.1 Example output

```
<make_reservation_result>
<crypto_block>U4--BSro8S0HG3xBzCMFTcex-Pga_4Lq6ovGfmwjM5zN8vcYU
QEL6iX_nPn29FvBUAyCFf_DFP1JopqaSaH7v4kaGdwLQOD--hMnwmgP1ckwncRj
qSQ2I4Qq3B9JmGSbjY7GEwwQ2lmATJbpHdKBOjL-AQfoS_BBjujD5-2yiJ2J7Sw
```

Z9kSAsELH2JCU0Jddkrb6MZntCcFrdeLJYJCCFV4MK1t3xj80CzdafdJZCnBwr6
d7x2MZWdA85i4sRGpq3_z3oSv5w8FZI-yc6JuiBrWV-asiRbt8qN_vkqatZKuEz
ZdVOCFM_2QFvuPcwp0aME7h2Itl8NCPiWmrAEgOMBZ41pazEMHPdz1xnGnM4LEq
UzUhr0df76N2zYsUvBjhamq5qSK5s5iztdoNT__aHN0ctF8DyuhaEzpxuq082Zl
wBMUTud3ZCnYPxQFP7AoaQTA7Evxtt6rEfgRd3qSnEoWcK4egauQccXS1hKa30x
NLA0x_TDODXYD-F8sIf3Q07XqbIAm7_910C-srJ_kLx_nSvg9hgLdsQ2uztGY-w
T4M9m9-Z</crypto_block>
<transaction_id>AE6D-6F28-D9AE-2877</transaction_id>
<minutes_left_on_reserve>10.000</minutes_left_on_reserve>
<failed_orders/>
<need_payment_card>yes</need_payment_card>

<acceptable_cards>
<card>
<card_type>mastercard</card_type>
<card_desc>Mastercard</card_desc>
</card>

<card>
<card_type>switch</card_type>
<card_desc>Switch</card_desc>
</card>

<card>
<card_type>visa</card_type>
<card_desc>VISA/Delta</card_desc>
</card>
</acceptable_cards>

<supports_billing_address>no</supports_billing_address>
<needs_email_address>yes</needs_email_address>
<needs_agent_reference>no</needs_agent_reference>

<prefilled_customer_data>
<can_edit>no</can_edit>
<address_line_one>Seatem House</address_line_one>
<address_line_two>39 Moreland Street</address_line_two>
<town>Angel, Islington</town>
<county>London</county>
<postcode>EC1V 8BB</postcode>
<country_code>uk</country_code>
<work_phone>020 7014 8466</work_phone>
<home_phone></home_phone>
<email_address></email_address>
</prefilled_customer_data>

<trolley_token>63--8wGlsnLXhW0fmpIifMyoA6rd2b1bMktB0JQ612kwikoz
xBnduwhe-Mf2bqWfGF1aiwhKgu1s1bNIFM6a7304q6MtvVDRPYiEF7xbGSUuAj-
eqowm7Q35mLOGw7FcEMnrBb3zo-6TxrGjnreP46jJ2YIA9TFmgGABE1gz283a4W
uYBuQe6ePh-rEaegcR98UjmuquYeLTnrh6By39GVhhX9K-LblH9T3ZGm3ZyXapm

```
xXkHkMjkveVRZR2cGvs_Rdx1MYAsVJD4npJhtXRzGJTqLan90my6D00PKQVVILF
5mrsv1UAY4bQESCw_JNfDkNvR1fXOmW9ZFhT_3eLJLHVuq0oJm2M9QnIwmeMHeh
hfhtzveFAMCVpY78IJ08fkWjeYi5oCvppsB24VOUj-W7m9bM89rd9SjWL3FdJa
ZAY5JHbSOLE2w7R_--Z</trolley_token>
<trolley_order_count>2</trolley_order_count>
</make_reservation_result>
```

13.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “start_session”
- **2** — the requested ”mime_text_type” was not a supported value
- **901** — no “trolley_token” has been supplied
- **902** — the supplied “trolley_token” is corrupt
- **903** — the trolley has already been purchased
- **904** — the trolley has already been reserved

14 The *get_reservation_link* method

This method returns a link to a web page to reserve the given trolley. It is an alternative to calling “make_reservation” for users who cannot provide their own checkout and payment pages, or who choose not to. The transaction will be completed by the customer interacting directly with TicketSwitch’s servers.

14.1 Required inputs

Like “make_reservation”, this call requires a “start_session” compatible crypto block and a “trolley_token”. It will not respond to a “describe_trolley” element.

14.1.1 Example input

```
<get_reservation_link>
<user_id>demo</user_id>
<crypto_block>M--hK9EA5-2pQPbRp1fLhr5lQCCZ9sPs54qY7MaAk98kZiHR
6hNwNQW6QmEWJYEtXvdZ8CRmfzkw5DvpXrn_GLXI dyvVWTv70s11Nh5fbp5zsU2
JvdsDoYct0--Y</crypto_block>
<trolley_token>c1--qfcq1g9hID4DEj3yf4ZBB6_gbEmAhEYChLQMfJCELmm5
jZtgn1uZs6xIXQ6vqjGon_-Z5ev1fiaRcMwruNrjy9gBXH6ZJ02KWK6tTbvculH
_XUxCSF-f9Ix6uw0eDeLTiVVeWlKSIhqtPWBPTdvFoyQyHnxvJKAX7blzg81ec
pIjTSB1H8VuT0dY8QIOY34165sYULFE_sLyBISywwCm-m5Sf3Ao27PidFr6l0bi
0I1-6LzK_Zoi9--Z</trolley_token>
</get_reservation_link>
```

14.2 Output data

If successful, the output of this method is a single “reservation_link” element containing a valid absolute URL¹ where the customer should be redirected to complete the purchase. The trolley will not be reserved by the xml_core.

The URL should be presented so that it is hard for a ‘double click’ to cause an extra reservation, which could cause genuine reservation attempts to fail.

14.2.1 Example output

```
<get_reservation_link_result>
<reservation_link>https://www.ticketswitch.com/tickets/web_addr
.buy?crypto_block=c2--tNKVifrR8YKaXPYb4H5Zr8Ncu07UAhjhkBky1-e6P
vugwPQR0Jtn837pKjXFFF5QTQ9CoELONvPgVQ9oQIKiEvxtAYEYSRj26MQjZy5_
gmm3Xf9XWW411JWj0zh5tfoaIcsd7e2-sfoEYQfdgQl5vPB1WPA9t9tj1qgZ0vmU
12-1DbigUWap1V1HKEs2p1W66CMUmBFQtSGNVZNXycc6P4GQWqnjDNmirRg7nJM
z9eBfF35DPbQsSURQ7sWnrHYsZk7Dk1I2BKltN2-Z</reservation_link>
</get_reservation_link_result>
```

14.3 Potential failure codes

This call may respond with any failure codes returned by “make_reservation” with the same meanings.

¹Depending on the context, valid URLs may require additional encoding before being output. This applies in particular to URLs containing ampersands if the output is HTML.

15 The *release_reservation* method

If a reservation is no longer required, and will not be purchased, then it may be released using this method. Reservations will be released automatically after a certain time period, but this method can be used to pre-empt this if a transaction is explicitly rejected. The naming of the method is somewhat inaccurate, however, as a successful return merely indicates that the order is no longer reserved. If this method is called on a purchased transaction then no action will be taken and a successful result will be returned. Similarly if it is called on a transaction which has already been released. A failure from this method is highly unlikely and will indicate some kind of problem with the underlying infrastructure.

15.1 Required inputs

The only required input is the “crypto_block” generated by “make_reservation”.

15.1.1 Example input

```
<release_reservation>
<user_id>demo</user_id>
<crypto_block>U4--BSro8S0HG3xBzCMFTcex-Pga_4Lq6ovGfmwjM5zN8vcYU
QEL6iX_nPn29FvBUAyCFf_DFP1JopqaSaH7v4kaGdwLQOD--hMnwmgP1ckwncRj
qSQ2I4Qq3B9JmGSbjY7GEwwQ2lmATJbpHdKBOjL-AQfoS_BBjujD5-2yiJ2J7Sw
Z9kSAsELH2JCU0Jddkrb6MZntCcFrdeLJYJCCFV4MK1t3xj80CzdafdJZCnBwr6
d7x2MZWdA85i4sRGpq3_z3oSv5w8FZI-yc6JuiBrWV-asiRbt8qN_vkqatZKuEz
ZdVOCFM_2QFvuPcwp0aME7h2Itl8NCPiWmrAEgOMBZ41pazEMHPdz1xnGnM4LEq
UzUhr0df76N2zYsUvBjhamq5qSK5s5iztdoNT__aHN0ctF8DyuhaEzpxuq082Zl
wBMUTud3ZCnYPxQFP7AoaQTA7Evxtt6rEfgRd3qSnEoWcK4egauQccXS1hKa30x
NLA0x_TDODXYD-F8sIf3Q07XqbIam7_910C-srJ_kLx_nSv9hgLdsQ2uztGY-w
T4M9m9-Z</crypto_block>
</release_reservation>
```

15.2 Output data

The output consists of a single element named “released_ok” which contains either a “yes” or “no”. A “yes” value indicates that the transaction is guaranteed to no longer be on reserve. A “no” value indicates that the operation could not be performed, though it may be re-attempted at a later time.

15.2.1 Example output

```
<release_reservation_result>
<released_ok>yes</released_ok>
</release_reservation_result>
```

15.3 Potential failure codes

- **1** — the supplied crypto block was not generated by “make_reservation”
- **2** — the requested “mime_text_type” was not a supported value

16 The *purchase_reservation* method

Purchase reservation completes the purchasing process by taking a reserved transaction and confirming it in the backend systems. Customer data is supplied at this point, along with any payment data which may be required. A successful result from this means that the payment has been taken, that the customer data was acceptable and that at least some of the tickets have been purchased. Note that it is possible for a purchase to partially succeed, with some individual transactions with specific suppliers failing. The individual results for each order in the trolley are then output, along with the despatch options available for each and any allocated seat IDs. If the backend system has produced a reference number for each order then these are also included. The purchased trolley is produced as a token in the output, and it may also be fully described, if requested, as usual.

A purchase request may fail for a number of reasons. The customer data supplied must be complete, and the request will be rejected if it is not. The credit card data must also be plausible (passes appropriate checksums and is of an accepted type), and will only be accepted if it is submitted over an encrypted connection. If the purchase fails for any of these three reasons then the purchase may be re-attempted with different data. If a purchase fails due to the credit card authorisation failing, or due to the reservation having timed out, then the purchase must be restarted from the “make_reservation” stage. The output data is structured such that permanent failures are not reported as errors, but are reported as an actual result from a successful operation.

For purchases where a card payment is being made, a separate debit will be made on the card for each supplier within the trolley. This should be made clear to the customer, as they may be expecting a single debit. It is perfectly possible for a transaction to take place in different currencies for the different suppliers.

For purchases where a despatch method with a type of “selfprint” was selected, and where TicketSwitch provides an HTML representataion of the voucher, a URL will be returned from where this HTML may be downloaded. The client software must fetch the HTML and present it to the customer to print.

16.1 Required inputs

A purchase always requires the “crypto_block” generated by “make_reservation” and also a “customer_details” element containing the data for the particular customer. If a payment card is required then an element “card_data” is required containing the payment card details.

If one of the “despatch_token”s originally requested corresponds to a method with despatch type of “selfprint” then a “self_print_mode” input is required, with the same value as that passed to the “availability_options” call. For simplicity, is safe always to pass this input even when no selfprint method was chosen.

The “customer_data” element contains the name and contact details for the customer. The customers name can include a title such as “Mr” or “Dr” as well as a suffix in order to handle such things as professional status or honours. The complete customer name is thus held in five elements called “title”, “first_name”, “initials”, “last_name” and “suffix”. Of these only the first and last names are

required to be present and not whitespace. Address data consists of four lines, nominally corresponding to the first two lines of the address and a town and county, plus a postcode and an ISO 3166 country code. These should be supplied in elements named “address_line_one”, “address_line_two”, “town”, “county”, “postcode” and “country_code”. Of these at least one of the first two lines plus the town must be present and not whitespace. The country code is compulsory due to the need to verify against the chosen despatch method. If the customer has an email address then this can be supplied in the “email_address” element, and for certain channels this is compulsory as indicated by the response from “make_reservation”. If supplied, the address must correspond to RFC822 and the syntax will be checked before a purchase is attempted. Finally two telephone numbers must be supplied in the “work_phone” and “home_phone” elements. To avoid a proliferation of failure codes there is only one code for incomplete customer data, and it is the responsibility of the user to check that the data passes these rules before attempting the purchase.

The “customer_data” element may also contain an “agent_reference” element, which will, if present, be stored against the transaction and passed through to all backend systems. For certain users this is compulsory, and if absent or blank will cause the customer data to be marked as incomplete.

The XML core also allows the collection of simple data protection information relating to the booking which specified who may use the customers data. There are two distinct classes of data user, the first being the user making the booking, and the second being the supplier of all the tickets. If the element “user_can_use_data” is present in the input then the booking will be flagged to indicate that the user may have the customers contact details. If the element “supplier_can_use_data” is present then each backend system will be told that they may contact the customer. There is a third flag “world_can_use_data” which is used to indicate that the data may be passed on to third parties for them to contact the customer. If this flag is true then it implies that both of the other flags are also true.

If a card payment is being made then the “card_data” element should be provided to hold all the necessary information to perform a debit. This consists of “card_number”, “expiry_date” and “cv_two” to hold the three mandatory fields for a card. The format of the expiry date is MMY and for cards with no CV2 value then the “cv_two” element should be present but empty. An “issue_number” element should also be present for those cards which have one, but must not be present for cards which do not. A “start_date” element may also be provided for those cars which have it. Certain types of card require this, and an error will be generated if it is not supplied when one of these cards is used. For transactions which allow an alternate card billing address then this may be specified using the elements “address_line_one”, “address_line_two”, “town”, “county”, “postcode” and “country_code” within the “card_data” element. As with the customer data the billing address requires that at least one of the first two lines is not whitespace, along with at least a town or a county.

Sales made on credit are normally as a result of the user having a credit agreement with Seatem and will be invoiced for the amount. Certain users, however, may not have credit facilities, but are still allowed to make credit sales, as long as full payment is made to Seatem before the tickets are despatched. The booking process will indicate if payment is required before despatch for a successful sale if the user has one of these credit arrangements.

It should be noted that a backend system is at liberty to refuse a booking based on its own internal criteria, which cannot be determined by the core. This may be due to an internal fraud check algorithm being used, or possibly due to unknown restrictions on customer data which may have been violated. An example of this would be a system that does not allow single character customer names, or names containing only vowels.

16.1.1 Example input

```
<purchase_reservation>
<user_id>demo</user_id>
<crypto_block>s2--AgBUIdqCoM8BTmPgrUz8Zep1a9SB4hvttQdoT5qxaN96T
ogehZ-l_SI4rqN2uJJQHH-tH-v20N4yZEA3jWCBzK-3dqQsjfa_MYufnw-bCaZT
b-NaqlraAOBBwCymw2pojKN-8BRagvzW_cf3Gd9s-XiCz8I74B6A1XSMNmkqN6M1
CgAZFw8JHYra2Qp9PXw0h95Y_8_pPVRsJTNLlv4HoBcj-sf4d7C1A-7auwfB52
gC9jfi16cXSkDhfI5dgZmRHVs6H8Y95N9QQYuXfXuD9eDsEqEqZKNBn5-RTfwfu
yzWQGOPfI1xjsXhEaRejKQcKPVLRABORbaIEr437itxWg2LhSzbSd8LfcWrJ9Uw
-Q_EfsKtVcFhF453ewreN1aeqvRkCGJ8x8DWlwzeg7sCCeRAB1NctLs6Z
</crypto_block>

<customer_data>
<title>Mr</title>
<first_name>Peter</first_name>
<initials>C</initials>
<last_name>French</last_name>
<suffix>MIEE, MEng</suffix>
<address_line_one>ECommerce Dept, Seatem UK</address_line_one>
<address_line_two>39 Moreland Street</address_line_two>
<town>Angel, Islington</town>
<county>London</county>
<postcode>EC1V 8BB</postcode>
<country_code>gb</country_code>
<work_phone>020 7014 8466</work_phone>
<home_phone>020 7014 8466</home_phone>
<email_address>petefrench@keithprowse.com</email_address>
<supplier_can_use_data/>
</customer_data>

<card_data>
<card_number>4929123456788</card_number>
<expiry_date>1204</expiry_date>
<cv_two>123</cv_two>
</card_data>
</purchase_reservation>
```

16.2 Output data

Completing a successful purchase of a trolley is the final stage in the process and thus no “crypto_block” is produced as there are no subsequent stages to be undertaken. A purchase results in a large amount of data being associated with

each order, such as seat identifiers, any special options on send methods, and costs that will be incurred by the agent for on-credit bookings. All of this data is held in the trolley, and thus the output from a successful purchase is a complete description of the trolley in the output. This is in the form documented for the “trolley_describe” method, including the “purchase_result” element. So that this information may be reproduced if necessary, the completed trolley is also output in a “trolley_token”, along with the associated “trolley_order_count” for ease of code-reuse.

Another element which may be present is “fraud_warning” which indicates that the purchase matched one or more of the fraud triggers. This is only in the case where the trigger allows the purchase to proceed. If the trigger causes the purchase to be blocked then an error will be produced as described below.

For a purchase which fails for a normal operational reason, there will be elements called “purchase_fail_code” and “purchase_fail_desc” present. These indicate a permanent failure of the purchasing process, and their possible values and meanings are documented below.

Purchases that result in TicketSwitch generating HTML representing a self-print voucher will return a “self_print_html_page” element for each selfprint order. This will contain a “page_url” element containing a relative URL and an “item_no” corresponding to the “item_number” of the order. The latter allows the orders and vouchers to be matched up in case there are several. The “page_url” element can be turned into a fully-qualified URL by concatenating “http://” and the hostname from the URL currently being used for communication with the core with the contents of “page_url”. The client must download the HTML and present it to the customer for printing.

16.2.1 Example output

```
<purchase_reservation_result>
<trolley>
<transaction_id>4A2B-7AC9-F630-2749</transaction_id>
<trolley_order_count>1</trolley_order_count>
<trolley_bundle_count>1</trolley_bundle_count>

<bundle>
<bundle_source_desc>Keith Prowse Ticketing</bundle_source_desc>
<bundle_source_code>fcg3</bundle_source_code>
<bundle_order_count>1</bundle_order_count>
<bundle_total_seatprice>27.500</bundle_total_seatprice>
<bundle_total_surcharge>4.150</bundle_total_surcharge>
<bundle_total_despatch>1.500</bundle_total_despatch>
<bundle_total_cost>33.150</bundle_total_cost>

<currency>
<currency_code>gbp</currency_code>
<currency_number>826</currency_number>
<currency_pre_symbol>&#163;</currency_pre_symbol>
<currency_post_symbol/>
</currency>
```

```

<order>
<item_number>1</item_number>
<backend_purchase_reference>AF291</backend_purchase_reference>
<venue_desc>The Dominion Theatre</venue_desc>
<event_desc>We Will Rock U</event_desc>
<performance>
<date_yyyymmdd>20031230</date_yyyymmdd>
<date_desc>Tue, 30th December 2003</date_desc>
<time_hhmmss>143000</time_hhmmss>
<time_desc>2.30 PM</time_desc>
</performance>
<despatch_desc>Post (uk only)</despatch_desc>
<despatch_final_type>post</despatch_final_type>
<despatch_final_comment>Please bring your credit card with you
when you collect your tickets.</despatch_final_comment>
<ticket_type_desc>Stalls</ticket_type_desc>
<discount>
<discount_desc>First Call Standard Sale</discount_desc>
<seatprice>27.500</seatprice>
<surcharge>4.150</surcharge>
<no_of_tickets>1</no_of_tickets>
<seats>
<id>WW40</id>
<id_details>
<row_id>WW</row_id>
<separator/>
<col_id>40</col_id>
</seats>
</discount>
<total_seatprice>27.500</total_seatprice>
<total_surcharge>4.150</total_surcharge>
<total_no_of_tickets>1</total_no_of_tickets>
<affiliate_commission>
<currency>
<currency_code>gbp</currency_code>
<currency_number>826</currency_number>
<currency_pre_symbol>&#163;</currency_pre_symbol>
<currency_post_symbol/>
</currency>
<value>1.500</value>
</affiliate_commission>
</order>
<purchase_result>
<success>yes</success>
<is_semi_credit>no</is_semi_credit>
</purchase_result>
</bundle>
<purchase_result>
<success>yes</success>
</purchase_result>

```

```

</trolley>

<trolley_token>c3--ESt4iSq-0h6ewW5aurWrMlaHazPjAoh6aidaqiyCuEQx
b5d3JTr2KnNS_eNj_LC5-RHyLa40-cKULcj1lwjpszx49vJ0mniPPBFgDeZRkfK
gz1x-gLCqsvdp1DTkgKyWu2_GM6_TV7DX6G1fLRCM7oX6D8j802Rz81NB2VI7hy
dIva0ixEfHu48UMiYdOREp9XDIwdoYi4R0tF05ZQt4KUzKRp7Rn9i0UNPyIVtMM
2IRP6ob-Dy-054kp0GnFihWBXBkP4kmSM3L0wxCBxVxjq7y2V1RIGg0Q2w1-W8k
-Ui17qiKMn_Zf1HByRyv5MBSkiXrJF0ZLKRboPT73uq5oxz3NgRHiCeGULnF345
zp6Jz94huDRbV6w4nSv7CVBgJ_H7iZZJcf08Z5q570Ao09e9hZe44ZDG15H1L61
QD3KDyCzUS3EH118JME0kQF1Xb4w2joeUbTP7IPhVPUS4zBNcfQGmoc-IZ
</trolley_token>
<trolley_order_count>1</trolley_order_count>
</purchase_reservation_result>

```

16.3 Potential failure codes

The following failure codes are to be treated as any other codes in the system would be, and appear inside “fail_code” and “fail_desc” elements as usual. Apart from code 1101 the reservation will still exist after one of these errors has been generated, and may be re-attempted with different data.

- **1** — the supplied crypto block was not generated by “make_reservation”
- **2** — the requested “mime_text_type” was not a supported value
- **1101** — the reservation has expired
- **1102** — no “customer_data” element has been supplied
- **1103** — no “card_data” element has been supplied when required
- **1104** — a “card_data” element has been supplied when not required
- **1105** — no “country_code” element was present in the customer data
- **1106** — the chosen despatch method does not allow the given country code in the customer data
- **1107** — the supplied email address fails RFC822 syntax checking
- **1108** — the customer details supplied are incomplete
- **1109** — no “card_number” element was present in the payment card data
- **1110** — the payment card type is not known from the supplied card number
- **1111** — the payment card type is not one of those accepted for this transaction
- **1112** — the card number given is not valid for cards of that type
- **1113** — no “expiry_date” element was present in the payment card data
- **1114** — the expiry date given is not valid

- **1115** — no “cv_two” element was present in the payment card data
- **1116** — the CV2 value given is not valid
- **1117** — no “issue_number” element has been supplied when required
- **1118** — an “issue_number” element has been supplied when not required
- **1119** — the issue number given is not valid
- **1120** — alternate card billing address supplied when not supported
- **1121** — the alternate card billing address details supplied are incomplete
- **1122** — no “start_date” element has been supplied when required
- **1123** — the start date given is not valid
- **1124** — the users prefill data is not editable, and does not match the customer data supplied
- **1125** — a “card_data” element has been supplied over a non-encrypted connection

The following codes are not errors in the normal way, but instead indicate a failure to purchase the tickets due to some other cause. Failures on this list are permanent, and the reservations are no longer valid once one of these failures has occurred. They are differentiated from normal errors by the fact that the element names are “purchase_fail_code” and “purchase_fail_desc”.

- **1** — the combination of customer and payment card details have matched a system fraud trigger and the purchase has not be allowed
- **2** — the payment card provided could not be authorised for the value of the transaction
- **3** — the attempt to authorise the payment card timed out with no response from the bank
- **4** — the reservation purchase has already been made successfully
- **5** — the reservation purchase has already been tried and failed
- **6** — the reservation purchase has been refused by the backend
- **7** — an unspecified systems error has occurred during purchase