xBRC Interface Control Document

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **Author** | **Description** |
| 1.0 | 5/1/2012 | Manny Vellon | Release Version |
| 1.0 | 10/1/2012 | Stephen Beecroft | Edit |

**Document Approvers & Sign-Off**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Approver** | **Role** | **Document Accept/Reject** |
| 5/1/2012 | Ric Merrifield | Release Manager | Accept |
| 5/11/2012 | John Stiehl | Release Manager | Accept |

**Table of Contents**

[1 Introduction 6](#_Toc337128922)

[1.1 Purpose 6](#_Toc337128923)

[2 Referenced Documents 8](#_Toc337128924)

[3 xBRC Interfaces 9](#_Toc337128925)

[3.1 Reader Interface 9](#_Toc337128926)

[3.2 Control Interface 10](#_Toc337128927)

[3.3 Messaging Interface 10](#_Toc337128928)

[4 Control Interface 11](#_Toc337128929)

[4.1 GET status 12](#_Toc337128930)

[4.2 GET messages 14](#_Toc337128931)

[4.3 PUT updatestream 15](#_Toc337128932)

[4.4 DELETE updatestream 16](#_Toc337128933)

[4.5 GET gueststatus 16](#_Toc337128934)

[4.6 DELETE gueststatus 17](#_Toc337128935)

[4.7 GET readerlocationinfo 17](#_Toc337128936)

[4.8 GET readerstats/reader/<reader name> 23](#_Toc337128937)

[4.9 DELETE readerstats 24](#_Toc337128938)

[4.10 GET bandevents/band/<long-range bandid> 24](#_Toc337128939)

[4.11 PUT storeconfiguration 25](#_Toc337128940)

[4.12 GET configurations (Note: returns JSON!) 26](#_Toc337128941)

[4.13 GET currentconfiguration 27](#_Toc337128942)

[4.14 POSTconfiguration?name=*name*&description 29](#_Toc337128943)

[=*description* 29](#_Toc337128944)

[4.15 DELETE configuration/name/<name> 29](#_Toc337128945)

[4.16 PUT selectconfiguration/name/<name> 29](#_Toc337128946)

[4.17 GET ekgposition 29](#_Toc337128947)

[4.18 GET ekg 30](#_Toc337128948)

[4.19 PUT logcomment?text=<string> 30](#_Toc337128949)

[4.20 PUT updateconfig 30](#_Toc337128950)

[4.21 DELETE cache 30](#_Toc337128951)

[4.22 GET refreshpackages 30](#_Toc337128952)

[4.23 PUT avmsevent 30](#_Toc337128953)

[4.24 GET perfmetricsmetadata 31](#_Toc337128954)

[4.25 PUT mediapackage 31](#_Toc337128955)

[4.26 DELETE mediapackage 32](#_Toc337128956)

[4.27 PUT playsequence/<readername>?sequence=<sequencename>&timeout=<timeoutlength> 32](#_Toc337128957)

[4.28 GET sequences 32](#_Toc337128958)

[5 Messaging Interface 33](#_Toc337128959)

[5.1 xBRC Models 33](#_Toc337128960)

[5.2 High Level Description of xBRC Messages 33](#_Toc337128961)

[Whether these messages are communicated using JMS or RESTful HTTP, data payloads describing these messages will be formatted in XML. 34](#_Toc337128962)

[5.3 General XML Payload Format 34](#_Toc337128963)

[5.4 Specific Message Payloads 35](#_Toc337128964)

[5.4.1 ENTRY Message 35](#_Toc337128965)

[5.4.2 MERGE Message 36](#_Toc337128966)

[5.4.3 LOAD Message 36](#_Toc337128967)

[5.4.4 INVEHICLE Message 37](#_Toc337128968)

[5.4.5 EXIT Message 38](#_Toc337128969)

[5.4.6 ABANDON Message 38](#_Toc337128970)

[5.4.7 METRICS Message 39](#_Toc337128971)

[5.4.8 ERROR Message 40](#_Toc337128972)

[5.4.9 READEREVENT Message 41](#_Toc337128973)

[5.5 JMS Transport Details 42](#_Toc337128974)

[6 Cruise Ship Considerations 43](#_Toc337128975)

[6.1 Differences in Message Formats 43](#_Toc337128976)

[6.1.1 READEREVENT Message 43](#_Toc337128977)

[6.1.2 ABANDON Message 44](#_Toc337128978)

[6.2 Sequence Diagrams 46](#_Toc337128979)

[7 Park Entry 48](#_Toc337128980)

[7.1 Cast App Transactions 52](#_Toc337128981)

[7.1.1 Logon/Logoff 52](#_Toc337128982)

[7.1.2 Green Light Blue Light Operations 53](#_Toc337128983)

[7.2 OmniTicket Transaction 55](#_Toc337128984)

[7.2.1 Header 55](#_Toc337128985)

[7.2.2 Request 56](#_Toc337128986)

[7.2.3 Answer 58](#_Toc337128987)

[7.3 GXP Interaction 62](#_Toc337128988)

[7.4 xBRC-to-GXP xPass Redemption Interaction 64](#_Toc337128989)

[7.5 GXP-to-xBRC Light Control 65](#_Toc337128990)

[7.6 Reconfiguring xTP Readers 66](#_Toc337128991)

[7.6.1 Get configuration 66](#_Toc337128992)

[7.6.2 PUT selectconfiguration/name/<name> 67](#_Toc337128993)

[8 Reader Registration 67](#_Toc337128994)

[8.1 Automatic Reader Software Updates 69](#_Toc337128995)

[8.1.1 How does the xBRC choose which version to install 70](#_Toc337128996)

[8.1.2 Minimum xBRC version 70](#_Toc337128997)

[8.1.3 How does the xBRC choose which install method to use 70](#_Toc337128998)

[8.1.4 Uploading new reader packages to the xBRC 70](#_Toc337128999)

# Introduction

Please note that throughout this document page names are highlighted in **bold** while page element names are in *italic*.

## Purpose

The xBRC Interface Control Document describes how the xBRC communicates with other component of the Disney Next Generation Experience (NGE). Three use cases are explored: “Attraction”, “Park Entry”, and “Cruise Ship” .

From the perspective of other components that interact with the xBRC, this document serves as a documentation of the xBRC application programming interface (API).

The xBRC is the main conduit for xBand *events* to be communicated to downstream applications that need to act on those events. An xBand event begins with a park guest nearing an xBR long-range reader or touching an xTP reader. The xBRC processes events from multiple readers, then generates messages to downstream components listening for these events. Two message transport mechanisms are supported:

* The SonicMQ Java Messaging System (JMS) communicates using a publish/subscribe model. Applications interested in messages from an xBRC can communicate with Sonic MQ and *subscribe* to the xBRCs message stream (the JMS *topic*).
* Alternatively, the xBRC can communicate messages using HTTP.

Each of the xBRC use cases has an xBRC “model” that is associated with it. In the Attraction and Park Entry cases, the models are correspondingly named:

*com.disney.xband.xbrc.attractionmodel.CEP*   
and

*com.disney.xband.xbrc.parkentrymodel.CEP*

The model for the cruise ship use case is also applicable to the park restaurant model and thus has a more generic name:

*com.disney.xband.xbrc.spacemodel.CEP*

A model is a plug-in “driver” that customizes the behavior of the xBRC for a particular use-case. For example, the Attraction model knows how to communicate with the GXP FastPass+ system and how to calculate various queue wait times as a guest proceeds from entry to exit, while the Park Entry model knows how to interact with the Omni ticketing system and how to handle the acquisition and verification of biometric data. The Space model is much simpler - for the most part, it simply reports the location of a guest every time the location changes.

The model in use by an xBRC is determined by the software package (RPM) that was used to install it. The package contains scripts to initialize the xBRC with the appropriate model-specific configuration parameters.

This document will present information as generically as possible. Where necessary, however, it will identify how the xBRC’s operation differs depending on the selected model.

# Referenced Documents

| Document Name & Version | Issuance Date | Relationship |
| --- | --- | --- |
| Reader Interface Control Document (ICD) | July 29, 2011 | Documents HTTP RESTful protocol between readers and XBRC |
| xBRC High-Level Technical Design | August 8, 2011 | Describes overall xBRC architecture |
| xBand Reader Management System System Architecture Document | July 21, 2011 | Describes overall xBRMS architecture |
| xBand\_RF\_ICD.doc | Feb 22, 2012 | Describes the wireless communications protocol to the xBand intended for use between the xBand and the long range reader or xBR. |

# xBRC Interfaces

The xBRC interfaces with various components, as shown in Figure 1.

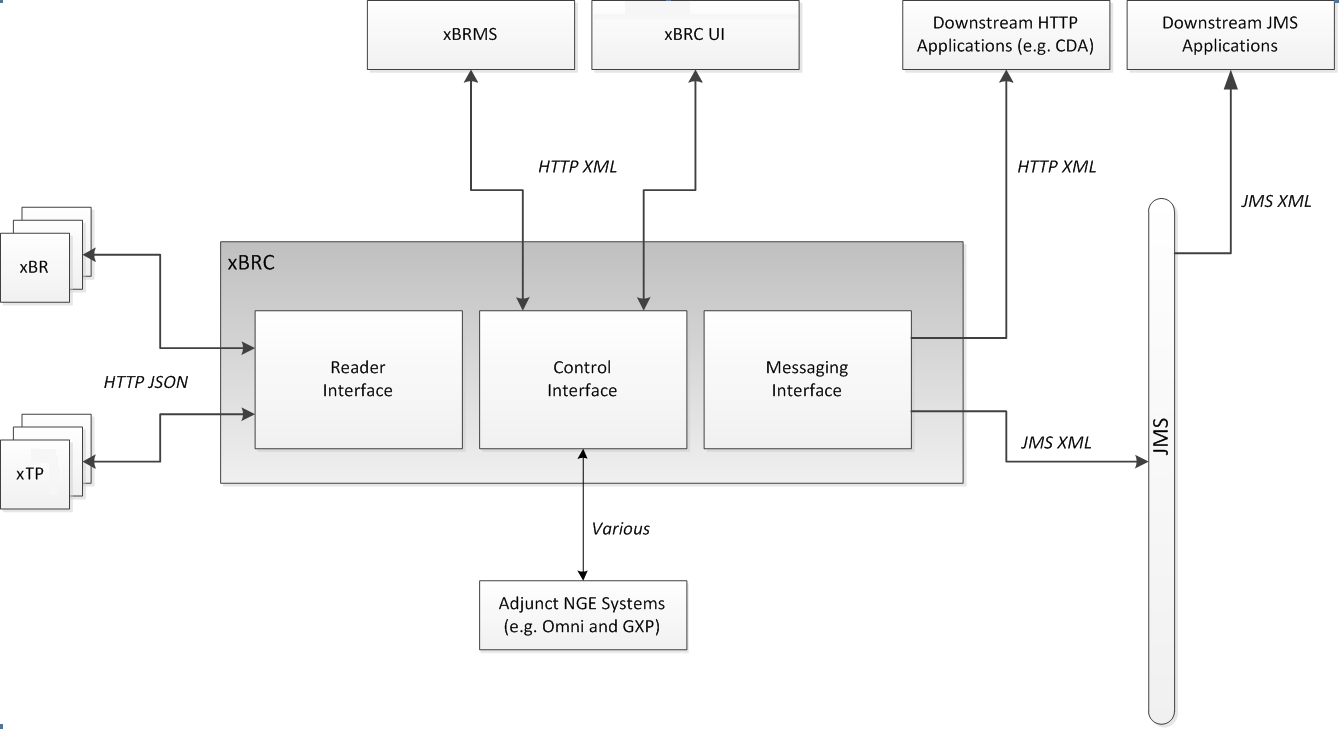


Figure . xBRC interfaces

## Reader Interface

The xBRC reader interface supports communication with long-range and touch readers. These readers are configured to initiate communication with a specific xBRC when they start up. The initiation message consists of an HTTP PUT hello message; in other words, the readers perform an HTTP POST request to an xBRC’s web service port (8080) using the request path “hello”. The data payload of the hello message is a JSON object describing the reader. The details of this message are documented in the xBR and xTP Interface Control Documents.

Once a reader has initiated a dialog with an xBRC, the xBRC will typically perform various HTTP operations to properly configure the reader. These operations typically consist of HTTP POST operations to set the reader’s clock, to set its operating mode, and to rename it, if necessary. The xBRC may also inform the reader that it has new software available for it. Finally, the xBRC puts the reader in “push mode”, telling it to send it events via a particular URL. When a reader is in push mode, it will periodically (as specified by parameters) send any band events to the xBRC. The details of these messages are contained in the respective reader interface control documents.

## Control Interface

The xBRC control interface consists of a set of HTTP requests (GET, PUT and POST operations) that are used to query, configure, and control the operation of an xBRC. These requests are detailed later in this document (section 4).

The xBRC control interface also includes communication with other related NGE systems. For example, in the Attraction use case, the xBRC needs to communicate with GXP in order to validate and redeem Fastpass+ entitlements. In the Park Entry use case, the xBRC needs to communicate with the Omni ticketing system to validate entry entitlements and biometric data. The details of these interfaces are not described in this document. For more information, refer to documentation for these adjunct systems.

## Messaging Interface

The primary role of the xBRC is to interpret high-traffic, low-level reader events and to translate them into low-traffic, more useful messages. Much of this document consists of descriptions of the types of messages generated by the xBRC.

# Control Interface

The xBRC implements the following HTTP requests (listed alphabetically, but described in a more logical order):

|  |  |
| --- | --- |
| Path | Operation |
| Avmsevent | POST |
| bandevents/band | GET |
| Cache | DELETE |
| Configuration | GET, POST, PUT, DELETE |
| Configurations | GET |
| Currentconfiguration | GET |
| Ekg | GET |
| Ekgposition | GET |
| Gueststatus | GET, DELETE |
| Logcomment | PUT |
| Messages | GET, DELETE |
| Mediapackage | PUT, DELETE |
| Playsequence | PUT |
| Readerlocationinfo | GET |
| Readerstats | GET, DELETE |
| Refreshpackages | GET |
| Selectconfiguration | PUT |
| Sequences | GET |
| Status | GET |
| Storeconfiguration | PUT |
| Updateconfig | PUT |
| Updatestream | PUT, DELETE |

## GET status

Returns information describing the xBRC’s status:

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*”>  
 <version>*string*</version>  
 <model>*model class*</model>  
 <status>*string*</status>  
 <statusMessage>*string*</statusMessage>  
 <JMSBroker>*server:port*</JMSBroker> <updateStreamUrl>*url*</updateStreamUrl>  
 <readerLocationsCount>*count*</readerLocationsCount>  
 <messageCount>*count*</messageCount>  
 <lastMessageSeq>*number*</lastMessageSeq> <lastMessageToJMS>*number*</lastMessageToJMS>  
 <lastMessageToUpdateStream>*number*</lastMessageToUpdateStream> <perfMetricsPeriod>*number*</perfMetricsPeriod>  
 <startPerfTime>*date*</startPerfTime>  
 <readerTestMode>*true/false*</readerTestMode>  
 <perfMETRIC>  
 <min>*double value*</min>  
 <max>*double value*</max>  
 <mean>*double value*</mean>  
 </perfMETRIC>  
 … more <perfMETRIC> tags …  
</venue>

| Status Element | Purpose |
| --- | --- |
| version | The xBRC software version. In the form: x.x.x.x.tag |
| model | The name of the Java class implementing the xBRC model. This is typically of the form *com.disney.xband.xbrc.MODEL.CEP* where MODEL is “attractionmodel”, “parkentry” or “space”. |
| status | “Red”, “Green” or “Yellow” |
| statusMessage | Short explanation if status is not “Green” |
| JMSBroker | The server and port number being used to communicate over JMS. Blank or a string prefixed with “#” if JMS communication is disabled. |
| updateStreamURL | If the xBRC is transporting messages over HTTP, this tag identifies the URL where it is PUT’ing its messages. |
| readerLocationsCount | The number of reader locations in the current xBRC reader configuration. |
| messageCount | The number of messages being queued by the xBRC that have yet to be sent to JMS or to an HTTP update stream URL. |
| lastMessageSeq | The last message sequence number used in the HTTP update stream. |
| lastMessageToJMS | The last message sequence number sent to the JMS. If this is not the same as *lastMessageSeq* there are messages awaiting to be sent via JMS. |
| lastMessageToUpdateStream | The last message sequence number sent via HTTP. If this is not the same as *lastMessageSeq* there are messages awaiting to be sent via HTTP. |
| perfMetricsPeriod | The sample size (in seconds) over which performance metrics are collected. Metrics are reset to zero at the end of each sample. |
| startPerfTime | The ISO 8601 date/time when the current performance metrics sample was started. |
| readerTestMode | Set to *true* if the readers connected to the xBRC are operating in “test mode”. |
| perf*METRIC* | min, max and mean values for *METRIC*. The currently available *METRIC*s are: *Events, EventAgeMsec, IDMSQueryMsec, EKGWriteMsec, SingulationMsec, PreModelingMsec, ModelingMsec, PostModelingMsec, ExternalMsec, WriteToReaderMsec, SaveGSTMsec, UpstreamMsec, MainLoopUtilization, Model1, Model2* and *Model3.* |

## GET messages

**GET messages/guestid/<guestid>**

This operation returns an XML structure.

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*” >  
 <message type=”*message type”* time=”*timestamp*”>  
 <sequence>*number*</sequence>  
… additional elements depending on message type …  
 </message>  
… additional messages …  
</venue>

Note the addition of the <sequence> element. This element contains a monotonically increasing number for each xBRC message. It is used in the *after* parameter described below.

In the first form of the path, “GET messages”, the operation returns messages for all guests. In the second form, the operation returns messages only for the identified <guestid>.

Note that a timestamp is included with the venue element as well as with each message element. The first timestamp reflects the time when the batch of events is retrieved while the timestamps in each message identify when the message was originally generated. The operation will support two optional parameters:

| Parameter | Purpose |
| --- | --- |
| after=*number* | Requests only messages whose sequence numbers are greater than the indicated value. If missing, the xBRC will return all of the messages that it has cached (subject to the *max* parameter). |
| max=# | Requests that a maximum number of messages be included in the reply |

Generally, downstream applications will not use GET messages to retrieve information from an xBRC. Instead, they will read messages from the JMS bus or will use “PUT updatestream” to place the xBRC in “push mode”. When in push mode, the xBRC will periodically send queued messages to a configured URL using HTTP PUT. This request is described in the following section.

## PUT updatestream

This operation requests that the xBRC operate in “push” mode. In this mode, the xBRC will use HTTP PUT to communicate messages with a given URL on a designated periodic basis. Push mode can only operate with a single destination. If this message is repeated, the latest updatestream message overrides the settings of any earlier message.

The payload for the operation will can provide values for four elements:

<updatestream>  
 <url>url for update stream</url>  
 <after>number</after>  
 <interval>milliseconds</interval>  
 <max>count</max> <preferredGuestIdType>type</preferredGuestIdType>  
 <messageTypes>message types</messageTypes>  
</updatestream>

|  |  |
| --- | --- |
| Element | Purpose |
| url | Identifies the URL to which the xBRC should post messages |
| after | Requests only messages whose sequence numbers are greater than the indicated value |
| interval | Requests that messages be posted using the designated period (in milliseconds). It is suggested that this value not be smaller than 100. |
| max | Requests that a maximum number of messages be included in each posted |
| preferredGuestIdType | Requests that a particular *type*  of guest id (e.g. “Fidelio”) be provided for guests, if present |
| message types | Space separated list of message types to receive. Use \* to receive all types. Example: “HASENTERED EXITED” |

When messages are posted to the designated URL, the message body is identical to that described in section 7.6.1.

Note that an xBRC that is operating in push mode will still respond to pull requests and will still publish JMS messages if configured to do so.

## DELETE updatestream

Disables “push mode”.

## GET gueststatus

**GET gueststatus/guestid/<guestid>**

This operation returns information about the Guests known to the xBRC. It is primarily intended when using the Space model but can also be used with other models.

In the first form, the operation returns information about all of the Guests known to the xBRC. In the second form, the operation returns only information about the specified <guestid>.

The response from this message will be XML data of the form:

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*”>  
 <guest>  
 <id>*guestid*</id>  
 <xpass>*true or false*</xpass>  
 <state>*string*</state>  
 <location>  
 <name>*location name*</name>  
 <id>*location id*</id>  
 <arrived>*timestamp*</arrived>  
 <latest>*timestamp*</latest>  
 </location>  
 </guest>  
… additional guest information messages …  
</venue>

For each guest, the xBRC reports his/her current location within the venue. The “arrived” element identifies when the Guest first appeared at the location while the “latest” element identifies the time of the most recent band transmission received at the location. Note that Guests may be considered to be at an old location for a few seconds after their departure (unless they’re picked up at another reader). This latency in the xBRCs operation is to account for occasional band transmissions that might be missed by a long range reader.

Note that GET gueststatus also returns information about the guest (whether he or she is an xPass guest and in what model “state” the guest is). State strings are model dependent, but here are some typical values: HASENTERED, HASMERGED, LOADING, RIDING, EXITED.

## DELETE gueststatus

This clears all guest status from the xBRC. This request can be used to “reset” an xBRC by telling it to clear its internal guest status table.

## GET readerlocationinfo

**GET readerlocationinfo/index/<location index>**

**GET readerlocationinfo/name/<location name>**

This operation returns information about the reader locations known to the xBRC. In the first form, the xBRC returns information about all known locations. In the second form, it returns information about the location designated by <location index>. In the last form, the location is identified by name. The data returned looks like:

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*”>  
 <readerlocationinfo>  
 <readerlocation>  
 <name>*location name*</name>  
 <section>*location section*</section>  
 <id>*location id*</id>  
 <type>*type code*</type>  
 <x>*value*</x>  
 <y>*value*</y>

<useSecureId>true/false</useSecureId>

<successSequence>sequence name</successSequence>

<successTimeout>milliseconds value</successTimeout>

<failureSequence>sequence name</failureSequence>

<failureTimeout>milliseconds value</failureTimeout>

<errorSequence>sequence name</errorSequence>

<errorTimeout>milliseconds value</errorTimeout>

<idleSequence>sequence name</idleSequence>  
 <readers>  
 <reader>  
 <name>*reader name*</name>  
 <id>*reader id*</id>  
 <deviceid>*number*</deviceid>  
 <type>*reader type*</type>  
 <lane>*number*</lane> <macaddress>*address*</macaddress>  
 <ipaddress>*address*</ipaddress>  
 <port>*port number*</port>  
 <gain>*double from 0-63.0*</gain>  
 <threshold>*integer from 0-63*</threshold>  
 <timelasthello>*date*</timelasthello>  
 <status>*Red/Yellow/Green*</status>  
 <statusMessage>*string*</statusMessage>  
 <version>*string*</version>  
 <minXbrcVersion>*string*</minXbrcVersion>  
 <xbioSerialNumber>*number*</xbioSerialNumber>

<istransmitter>*true/false*</istransmitter>

<transmitPayload>*use only by simulators*</transmitPayload>

<useSecureId>*true/false*</useSecureId>

<transmitCommands>

<transmitCommand>

<command>*FAST\_RX\_ONLY/SLOW\_PING/FAST\_PING*</command>

<interval>*integer*</interval>

<mode>*REPLY/BROADCAST*</mode>

<recipients>

<recipientLocationId>*integer*</recipientLocationId>

… -- other recipient location IDs

</recipients>

<timeout>*integer*</timeout>

</transmitCommand>

… -- other transmit commands

</transmitCommands>

<hardwareType>*xTP1/xTP2/…*</hardwareType>

<signalStrengthTransitThreshold>*integer from -127 to -40</*signalStrengthTransitThreshold>

</reader>  
 </readers>  
 </readerlocation> … other reader locations …  
 </readerlocationinfo>  
</venue>

|  |  |  |
| --- | --- | --- |
| Element | Purpose | |
| name | Name of the reader location which appears in the <readerlocation> and <readerlocationid> elements in other get requests and messages. | |
| id | Numeric id of the reader location which appears in the <readerlocation> and <readerlocationid> elements in other get requests and messages. | |
| type | Integer identifying the operational type of the location. These values are defined as: | |
|  | 1 (ENTRY) | A reader location used to indicate entry to a venue (in the Attraction and Controller Space model). |
|  | 2 (WAYPOINT) | An intermediate reader location used to trigger an external action or simply to keep track of a guest’s location. |
|  | 3 (EXIT) | A reader location used to detect when a Guest has exited an Attraction or controlled Space. |
|  | 4 (LOAD) | A reader location placed immediately before a Guest boards an attraction “car”. The readers at the location will detect bands attached to attraction cars in addition to those worn by Guests. |
|  | 5 (INCAR) | A reader location placed so that it detects only a particular attraction car and the guests within it. |
|  | 6 (MERGE) | A reader location placed where the xPass and standby queues merge. |
|  | 7 (XPASSENTRY) | A reader location placed at the beginning of the xPass queue. Readers at this location are involved with xPass entitlement checking. |
|  | 8 (COMBO) | A reader location that serves as both XPASSENTRY and MERGE. |
| x | Used by simulators only. Please disregard. | |
| Y | Used by simulators only. Please disregard. | |
| useSecureId | If not provided a global configuration setting is used. If *true*, xBRC instructs xTPs to read Secure ID off of the rfid media used to touch and uses the Secure ID read to verify that media against the IDMS database. Otherwise, Public ID is used. | |
| successSequence | Name of a file containing a combination of lights and sounds sequences to be played by a xTP reader in the event of a successful touch. Definition of a successful touch is model and location dependent. When not explicitly provided on a location, a global configuration setting is used. | |
| successTimeout | Stop playing the success sequence after this many milliseconds. When not explicitly provided on a location a global configuration setting is used. | |
| failureSequence | Name of a file containing a combination of lights and sounds sequences to be played by a xTP reader in the event of a rejected touch. Definition of a rejected touch is model and location dependent. When not explicitly provided on a location, a global configuration setting is used. | |
| failureTimeout | Stop playing the failure sequence after this many milliseconds. When not explicitly provided on a location, a global configuration setting is used. | |
| errorSequence | Name of a file containing a combination of lights and sounds sequences to be played by an xTP reader in the event of an error on touch. Definition of an error on touch is model and location dependent. When not explicitly provided on a location, a global configuration setting is used. | |
| errorTimeout | Stop playing the error sequence after this many milliseconds. When not explicitly provided on a location, a global configuration setting is used. | |
| idleSequence | Name of a file containing a combination of lights and sounds sequences to be played by a xTP reader continuously. When not explicitly provided on a location, a global configuration setting is used. | |

The <readers> element enumerates all of the readers (xBR or xTP) present at the reader location. Long range xBR readers are typically deployed in “gangs” of 4 while touch readers are typically deployed 2-8 per location (two in the Attraction model, eight at park entry). For each reader, the following information is provided:

| Reader Element | Purpose |
| --- | --- |
| name | The name of the reader. Note that reader names are generally chosen to reflect the location. For example, the Entry location may consist of four xBRs named “entry-1”, “entry-2”, “entry-3” and “entry-4”. |
| id | A numeric id associated with the reader. This id remains constant even if the reader is renamed. |
| deviceid | Similar to *id*, but assigned by the OneSource system. |
| lane | In the Attraction model, this is typically 0 or 1 for the left and right-side reader at a location. In Park Entry, however, this is typically 0-7. The value is N/A for long-range readers. |
| type | “lrr” (an xBR), “xTP” (touch) or “xTP+xbio” (touch with biometrics) |
| macaddress | The network MAC address of the reader. |
| ipaddress | The ip address of the reader. This is expected to be an address on a physical or virtual private network of the form 192.168.0.x. Note that the xBRC operates as a DHCP server for devices in the private network. |
| port | The TCP port on which the reader listens for configuration and monitoring requests (typically, 8080). |
| gain | A floating point (double) value from 0-63.0 that is used to attenuate signal strength data received by the reader. An incoming signal strength value from the reader will be multiplied by its gain value. |
| threshold | An integer value from -90 to -40 used to filter out low signal strengths. A band message whose signal strength (after being multiplied by the reader’s *gain*) is less than the threshold value will be ignored. |
| timelasthello | Long integer - milliseconds since the start of the epoch. Identifies when the reader has sent a PUT hello to the xBRC. |
| status | “Red”, “Yellow” or “Green” |
| statusMessage | Explanatory text if status is not “Green” |
| version | The installed version of the reader software |
| minXbrcVersion | The minimal version of the xBRC software with which the reader is designed to work. |
| xbioSerialNumber | The serial number of the biometric hardware, if *type* is *xfp+xbio* |
| x | The x coordinate of the reader. The reader may be assigned an (x,y) location for user interface visualization purposes. This value serves no other functional role in the xBRC. |
| y | They coordinate of the reader. The reader may be assigned an (x,y) location for user interface visualization purposes. This value serves no other functional role in the xBRC. |
| istransmitter | Is this reader capable and configured to transmit commands to xbands. Applies only to xBand readers. |
| transmitPayload | Used by simulators only. Please disregard. |
| useSecureId | Inherited from the location. |
| hardwareType | *xTP1* represents the first version of xTP without the RGB capabilities. *xTP2* represents xTPs with RGB LED lights. |
| signalStrengthTransitThreshold | Transmitting xBR transmits applicable commands in response to a ping from bands transmitting with a signal strength specified in this element. Valid range is -127 to -40. |

The <transmitCommands> element enumerates band commands that a transmitting long range reader is configured to either broadcast or send on reply. For transmit command, the following information is provided.

|  |  |
| --- | --- |
| Element | Purpose |
| command | Command to transmit. Possible choices are *FAST\_PING*, *SLOW\_PING*, and *FAST\_RX\_ONLY*. |
| interval | Band commands will be transmitted as often as specified by the interval. Specified in milliseconds. |
| mode | Either *REPLY* or *BROADCAST*.  *REPLY* mode means that a transmitting xBR will reply to a ping from a band by sending it a command. This means that a command will be sent on a second ping; the first ping is used for detection, and the second ping invokes response.  *BROADCAST* mode translates to a transmitting xBR sending a band command continuously to be picked up by a band as soon as its first ping is detected.  A single transmitting xBR can be configured to send either a single *BROADCAST* command or a list of *REPLY* commands, but not a combination of the two. |
| recipients | One or more location IDs. The list of bands that are to receive this command is comprised of bands seen at these locations since the last time the GST table has been cleared. Applicable to *REPLY* mode commands only. |
| timeout | Number of milliseconds after which a band will revert to the next most energy efficient mode. FAST\_PING reverts to SLOW\_PING, which reverts to FAST\_RX\_ONLY, which in turn reverts to SLOW\_RX\_ONLY. A band may not be set to the SLOW\_RX\_ONLY mode, it can only revert to it. |

Refer to xBand\_RF\_ICD.doc for details on communication protocol between the xBand and the xBR.

## GET readerstats/reader/<reader name>

Returns statistical information regarding the performance of a particular reader (identified by <reader name>:

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*”>  
 <readerstats>  
 <reader>  
 <name>*name*</name>  
 <stats>  
 <stat>  
 <channel>*number*</channel>  
 <frequency>*number*</frequency>  
 <ss>*number*</ss>  
 <count>*number*</count>  
 </stat>  
 … more stat elements …  
 </stats>  
 </reader>  
 … more reader elements …>  
 </readerstats>  
</venue>

The name element value should match the name in the URL. The stats elements provide information about the eight radios present in the reader. These radios are organized into two different channels (0 or 1). Each radio is programmed to one of four frequencies. This request returns a signal strength histogram for the data received by each of the eight radios. Each stat element identifies a radio (channel, frequency) and identifies how many events (count) were received with a particular signal strength (ss). The statistical data for the reader is cleared after the response is generated.

The elements in *stat* are defined as per the following table.

|  |  |
| --- | --- |
| Stat Element | Purpose |
| channel | The channel number (0 or 1) of the radio |
| frequency | The frequency of the radio (2401, 2424, 2450 or 2476) |
| ss | A particular signal strength (-90 to -40). |
| count | The number of events received with the indicated signal strength. Note that if *count* is zero, the entire *stat* element is omitted. |

## DELETE readerstats

**DELETE readerstats/reader/<reader name>**

This clears reader performance statistics for a particular reader (second form) or for all readers.

## GET bandevents/band/<long-range bandid>

This URL is used to retrieve data to support the “power level” UI functionality in the xBRC. The operation retrieves the latest (most recent packet number) low-level xBR (long-range) events for the band with the indicated long-range id:

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*”>  
 <events>  
 <event>  
 <type>*LRR*</type>  
 <xlrid>*id*</xlrid>  
 <time>*timestamp*</time>  
 <readerName>*name*</readerName>  
 <eno>*number*</eno>  
 <pno>*number*</pno>  
 <freq>*number*</freq>  
 <chan>*number*</chan>  
 <ss>*number*</ss>  
 </event>  
 … more event elements …  
 </events>  
</venue>

|  |  |
| --- | --- |
| Event Element | Purpose |
| type | Identifies the type of event. Always “LRR”. |
| xlrid | The long-range id that originated this message. Should match the id specified in the URL. |
| time | The timestamp associated with the event (in ISO 8601 form). |
| readerName | The name of xBR reader reporting the event. |
| eno | The event number of the event. |
| pno | The packet number (0-255) of the event. All the events in the xml message will have the same packet number value. |
| freq | The frequency (2401, 2424, 2450 or 2476) of the radio that received the event. |
| chan | The channel (0 or 1) of the radio bank that received the event. |
| ss | The signal strength (0-64) associated with the event. |

## PUT storeconfiguration

This request stores current configuration information in the “stored configurations” table of the xBRC. An xBRC can store multiple configurations and can switch between them on request. A configuration consists of:

- All the configuration parameter values

- The reader location and reader tables

- The “subway diagram” for the configuration

- Additional, model-specific information

PUT storeconfiguration supports two parameters: *name* and *description*. As parameters are specified as part of the URL, escape these as necessary to conform to URL encoding rules.

|  |  |
| --- | --- |
| Parameter | Purpose |
| name=*string* | Specifies the name to be given to the stored configuration. Note that names must be unique. If a configuration already exists with the given name, a 500 error will be returned. |
| description=*string* | A description of the stored configuration. Note that if this parameter is omitted, the xBRC will assign a description based on the time when the configuration was stored. |

PUT storeconfiguration returns a text-encoded number (mime type text/plain) that is numeric id assigned to the newly stored configuration. This number may be used later in the PUT selectconfiguration request.

## GET configurations (Note: returns JSON!)

This request returns information about the configurations stored in an xBRC. The result is a JSON encoded object of the form:

{  
 “configurations”:  
 [  
 {  
 “configurationId” : *number*,  
 “name” : *string*,  
 “description” : *string*  
 },  
 … other configurations …  
 ]  
}

|  |  |
| --- | --- |
| Element | Description |
| “configurationId” | Identifies the number assigned to the stored configuration. |
| “name” | Identifies the name provided when the configuration was stored |
| “description” | Either the description provided when the configuration was stored or an automatically generated description in none was provided |

This request returns JSON because the primary customer for this request is a GXP application that can more easily parse JSON than XML

## GET currentconfiguration

**GET configuration/name/<name>**

**GET configuration/id/<id>**

This request returns an XML document that describes a configuration. The configuration can be requested by name (including “current” for the current configuration) or by id. The “currentconfiguration” path yields the same results as “configuration/name/current”. The general format of the returned document (of mime type application/xml) is:

<?xml version=”1.0” encoding=”UTF-8” standalone=”no” ?>  
<venue name=”*venue name*” time=”*timestamp*”>  
 <configuration name=”current” type=”full”>  
 <description/>   
 <properties>  
 <property class=”*class”* name=”*name”*>*value*</property>  
 … more properties …  
 </properties>  
 <readerlocationinfo>  
 ……  
 </readerlocationinfo>  
 <griditems>  
 <griditem id=”*number”* type=”*type”* xgrid=”*number”* ygrid=”*number”*>  
 <state>*string*</state> <label>*string*</label><description>*string*</description>  
 <image>*string*</image>  
 <sequence>*number*</sequence>  
 <gueststoshow>*0,1 or 2*</gueststoshow>  
 <locationid>*number*</locationid>  
 </griditem>  
 … more grid item entries…  
 </griditems>  
 <images>  
 … image information …  
 </images>  
 <model>*model specific tags*</model>  
 </configuration>  
</venue>

The name attribute in the configuration tag will always be “current” and the type attribute will always be “full”. The description tag will always be empty. If the retrieved XML is used as a template for uploaded configurations (see section 4.13), these values should be changed as necessary. Note that a “full” configuration completely replaces an existing one (when selected) whereas a “partial” configuration is “additive” (it only replaces existing data with data provided in the new configuration).

The properties section contains all of the name-value pairs that are used to control and fine-tune the operation of the xBRC. xBRCs use dozens of properties to identify adjunct systems (for example, to locate a GXP or Omni system), to provide credentials and to tune various operations (for example, abandonment timeouts or xTP light durations.

The readerlocationinfo section is exactly as described earlier in section 4.6 - it describes all of the reader locations configured in the xBRC and the readers assigned to those locations.

The griditems section describes the “subway diagram” for the xBRC. It consists of a series of griditem tags each of which defines an entry in a subway map square. Each grid item is identified by an id number and is of a provided type and occupies the x, y location in the grid identified by xgrid and ygrid. These and other tags/attributes are described in the following table.

|  |  |
| --- | --- |
| <griditem> tag or attribute | Description |
| id | A numeric id assigned to the item. |
| type | A string identifying the type of grid item: *Gate, HPath, VPath, TNorth, TSouth, TWest, TEast, Cross, ESTurn, WSTurn, ENTurn, WNTurn.* |
| xgrid, ygrid | The x and y coordinate of the item. |
| label | A string to display in a *Gate* type item. |
| image | A URL (local to the xBRC) identifying the bitmap to display for a *Gate.* |
| description | A description (used for popup text) to associate with the item. |
| state | The model state for which the grid item should display guest counts. This tag is mutually exclusive with <locationid>. |
| locationid | The location id for which the grid item should display guest counts. This tag is mutually exclusive with <state>. |
| sequence | A number assigning the priority of this item relative to other items displaying similar guest counts |
| gueststoshow | 0 if all guests, 1 if xPass only, 2 if standby only |

The images section describes any images stored in the xBRC (typically, for use by the subway diagram) while the model section can contain additional model-specific information.

## POSTconfiguration?name=*name*&description

## =*description*

**PUT configuration?name=*name*&description**

**=*description***

This request allows a configuration to be uploaded. If the POST verb is used, a new configuration is created. If the PUT verb is used, the configuration with the given name is replaced by the uploaded version. The information for the uploaded configuration is provided as they payload of the request. The form of this payload is identical to the form described in GET configuration (section 4.13).

The name and description for the configuration can be provided by URL parameters or in the supplied XML.

Either “full” or “partial” type configurations may be uploaded. Partial configurations need not have all sections of the configuration specified - only the sections where they specify changes.

## DELETE configuration/name/<name>

**DELETE configuration/id/<id**

These requests can be used to delete stored configurations by specifying their name or id.

## PUT selectconfiguration/name/<name>

**PUT selectconfiguration/id/<id>**

These requests replace the current configuration with a stored configuration identified by its name or by its assigned id. If the stored configuration is a full configuration, the operation begins by removing all current properties, reader locations, readers and grid items. The configuration must fully specify all the information needed to repopulate these items. If the stored configuration is a partial configuration, however, it need only contains the items that it wants to change. In this case, PUT selectconfiguration does not delete items *en masse*; it deletes an item only when it detects that a value is being provided in the selected configuration. As a result of this algorithm, note that partial configurations are always “additive”. A partial configuration can never “delete” a current setting; it can only overwrite it with a new value.

## GET ekgposition

This request returns the current “cursor” position of the low-level “ekg” event log file. The purpose of this call is to help provide a value for the position parameter in the GET ekg request (section 4.17).

The ekg file is used mostly for diagnosing the low-level operation of the xBRC; it is of little interest once an xBRC has been deployed and configured for a particular facility.

GET ekgposition returns a “long” text encoded number (mime type text/plain).

## GET ekg

This request returns the contents of the low-level ekg event log file. The request supports two parameters:

|  |  |
| --- | --- |
| Parameter | Purpose |
| position=*long int* | Specifies the initial position within the ekg file from which the xBRC should read. |
| max=*int* | Specifies the maximum number of lines that the request should return. |

The response to the request is of mime type text/plain. It consists of the requested contents of the ekg file appended with a text encoded number that identifies the next value that can be used as the position value if another chunk of data is to be read.In general, the algorithm for reading the ekg file through the HTTP interface is:

1. Use GET ekgposition to determine the current end of file

2. Loop

a. Call GET EKG?position=value where value is the number returned in step 1.

b. Read the response but treat the last line as a number providing a new value for position.

## PUT logcomment?text=<string>

Inserts a string into the ekg file. This call can be used by diagnostic tools to insert a recognizable marker into the low-level event file.

## PUT updateconfig

This request is used by external programs (for example, the xBRC UI or xbrcconfig) that modify the xBRCs database directly then need to inform the xBRC that it should re-read its configuration from the database.

## DELETE cache

Removes cached guest and band information from the xBRC. This call should be used if changes have been made to the xBMS or other band service and there is the possibility that the xBRC is holding out-of-date information.

## GET refreshpackages

Request the xBRC to refresh the reader packages from the disk

## PUT avmsevent

Notify the xBRC that a vehicle just passed a detection point. The payload of the request is shown below.

<message type="VEHICLE" time="2012-02-02T22:07:52.5000000-05:00">

<vehicleid>123456</vehicleid>

<attractionid>WMKHAMA</attractionid>

<sceneid>13B</sceneid>

<locationid>01</locationid>

<confidence>99</confidence>

</message>

All the fields in the message are echoed back in the INVEHICLE event published when a guest is associated to a vehicle.

## GET perfmetricsmetadata

This request returns the metadata describing the performance metrics that xBRC collects. The following information about each metric is returned:

|  |  |
| --- | --- |
| Parameter | Purpose |
| name | Name under which the xBRC stores this metric’s data structure on its status object. |
| displayName | Human friendly name to use as a user facing title for this metric. |
| description | Description of performance data collected by this metric. |
| version | Version of this meta data. This is used to preserve historical meaning of this metric. |
| type | Units of performance data collected by this metric. |

## PUT mediapackage

This request allows a file containing media content (color sequences, sound sequences, WAV files) to be uploaded to the xBRC. In turn, the media package is pushed to all of the readers that the xBRC is associated with. The xBRC can be configured through the xBRC user interface to play individual media files or media sequences in response to various xBRC events. In addition, media files and sequences can be queued to a reader using the playsequence API.

A call to mediapackage should include a binary encoded file as the message body of an HTTP POST request. This media package file should consist of a collection of directories and files the are compressed into a single file using TAR and GZIP. The naming convention of the resulting file is not important (an example might be “media.tar.gz”).

The media package folder and file structure is flexible but does require two directories for successful operation. The first is LEDSCRIPTS, which contain scripts to drive a combination of light and sound affects. These scripts can refer to any other file within any arbitrary directory structure within the file. The other required directory is SOUNDS, which contains any number of WAV files that could be referenced by LED scripts or played individually.

## DELETE mediapackage

This allows any existing media package to be deleted from the xBRC. In turn, any associated readers will have their media packages deleted. Given that an xBRC can have only one media package installed at a time, a package name does not need to be specified and any existing package will be deleted in response to the request.

## PUT playsequence/<readername>?sequence=<sequencename>&timeout=<timeoutlength>

This request allows a media sequence (or media resource) to be queued through a specific reader. The specific reader to receive the request must be associated with the xBRC and is identified by the reader’s name through <readername>. The sequence to queue, as identified by <sequencename>, can be a script name, color sequence, or sound file.

The timeout is an optional parameter, as specified by <timeoutlength> in milliseconds, that allows the request to override a sequence’s default sequence length. If no timeout is specified, a sequence script will run according to its own default values and LED sequences and WAV files will play in its entirety. If 0 is specified as the timeout, the specified media sequence will loop indefinitely. To terminate a looping sequence, a playsequence request can be made to the same reader with a reserved sequence name of “off”.

## GET sequences

This request allows the retrieval of all supported sequences within the xBRC’s current media package. The retrieved names can be used as a the <sequencename> parameter to the playsequence API.

The request returns the following example payload (and includes supported default sequence names):

<sequences time="2012-02-02T22:07:52.5000000-05:00">

<sequence>entry\_success</sequence>

<sequence>entry\_exception</sequence>

<sequence>entry\_retry</sequence>

<sequence>gxp\_success</sequence>

<sequence>gxp\_exception</sequence>

<sequence>thinking</sequence>

<sequence>blue</sequence>

<sequence>off</sequence>

</sequences>

# Messaging Interface

This section describes the format of the XML messages sent via JMS or HTTP to downstream applications.

## xBRC Models

Not all of the messages described in this document are generated in all of the xBRC operating models. The following table identifies which messages are used by each model.

|  |  |
| --- | --- |
| Model | Message Types Used |
| Attraction | ENTRY, MERGE, LOAD, EXIT, ABANDON\*, METRICS, READEREVENT\* |
| Space | READEREVENT\*, ABANDON\* |
| Park Entry | ENTRY |

\*Note that the READEREVENT and ABANDON are used in different ways, depending on the model in use.

## High Level Description of xBRC Messages

Before describing how xBRC messages may be read, let us consider what data xBRC messages might contain. In general, xBRC datua reflect an “event” that has occurred due to an xBand transmitter communicating with a reader (touch or long-range). Many xBRC messages correspond one-to-one with xBR events as documented in the Reader Interface Control Document. The following table describes the different types of messages/events that an xBRC can report. The table identifies a message type with a string. These same strings appear, later, in message payloads identifying the nature of the message.

|  |  |
| --- | --- |
| Message Type | Purpose |
| ENTRY | A Guest has entered a venue |
| MERGE | A Guest has touched a merge-point reader (in an xPass queue) |
| LOAD | A Guest has been “loaded” onto a venue vehicle (typically, on some type of “car”) |
| EXIT | A Guest has exited a venue |
| ABANDON | A Guest has abandoned a venue |
| METRICS | A venue is reporting aggregate metrics |
| ERROR | An error condition has been detected (for example, an entitlement check has failed). |
| READEREVENT | A reader has sensed an xBand |
| INVEHICLE | Similar to LOAD event, but generated in response to the VEHICLE event received from the AGC system. |

Table 1 - Message Types

## Whether these messages are communicated using JMS or RESTful HTTP, data payloads describing these messages will be formatted in XML.

## General XML Payload Format

The general format for an xBRC message payload will be:

<venue name=”*venue name*” time=”*timestamp*”>  
 <message type=”*message type”* time=”*timestamp*”>  
… additional elements depending on message type …  
 </message>  
… additional messages …  
</venue>

The venue name identifies the attraction, dining experience or other application in which the xBRC is installed. Message type is a string taken from Table 1 while time is an ISO 8601 formatted time stamp, for example: “2011-06-20T13:41:00:891” (note the inclusion of milliseconds). Times are always stored in UTC form although they may be displayed as local time for usability purposes. Note that the timestamp in the venue element represents the time when the “batch” of messages are delivered whereas the timestamp in the message element represents the time when the message was generated.

## Specific Message Payloads

### ENTRY Message

<message type=”ENTRY*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <xpass>[*true | false*]</xpass>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the entry message. |
| xpass\* | “true” if guest entered the venue through an xPass queue. “false” if through standby queue |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |

\*Note that if an xBRC is operating with the Controlled Space model (as opposed to the Attraction model) that the *xpass* value will always be set to *true*, indicating that the guest entered the controlled space through an xTP touch reader.

### MERGE Message

<message type=”MERGE*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <xpass>*true*</xpass>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
</message>

| Element | | Purpose |
| --- | --- | --- |
| guestid | | Identifies the guest id that triggered the merge message. |
| xpass | | Currently, only xPass guests generate MERGE messages so this value is always *true* |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |

### LOAD Message

<message type=”LOAD*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <xpass>[*true | false*]</xpass>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <carid>*car id*</carid>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the LOAD message. |
| xpass | “true” if guest entered the venue through an xPass queue. “false” if through standby queue |
| Readersection | Identifies the “section” (a configured string) associated with the reader location. |
| Readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| Carid | Identifies the “car” associated with the guestid at the load location |

### INVEHICLE Message

<message type="INVEHICLE" time=”timestamp”>

<guestid>guest id</guestid>

<xpass>true/false</xpass>

<readersection> *reader section* </readersection>

<readerlocation>reader location</readerlocation>

<vehicleid>vehicle id</vehicleid>

<attractionid>attraction id</attractionid>

<sceneid>scene id</sceneid>

<locationid>location id</locationid>

<confidence>confidence</confidence>

<sequence>sequence</sequence>

</message>

|  |  |
| --- | --- |
| Element | Purpose |
| Guestid | Identifies the guest id that triggered the LOAD message. |
| xpass | “true” if guest entered the venue through an xPass queue. “false” if through standby queue |
| Readersection | Identifies the “section” (a configured string) associated with the reader location. |
| Readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| Vehicle Id | Identifies the “vehicle” associated with the guestid. This id is received in the AGC VEHICLE event. |
| Attraction Id | Attraction id received in the AGC VEHICLE event. |
| Scene Id | Scene id received in the AGC VEHICLE event. |
| Location Id | Location Id received in the AGC VEHICLE event. |
| Confidence | Confidence received in the AGC VEHICLE event. |
| Sequence | Sequence received in the AGC VEHICLE event. |

### EXIT Message

<message type=”EXIT*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <xpass>[*true* | *false*]</xpass>  
 <carid>*car id*</carid>  
 <statistics>  
 <waittime>*wait time*</waittime>  
 <mergetime>*merge time*</mergetime>  
 <totaltime>*total time*</totaltime>  
 </statistics>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the EXIT message. |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| xpass\* | “true” if guest entered the venue through an xPass queue. “false” if through standby queue |
| carid\* | Identifies the “car” associated with the guestid at the load location |
| waittime\* | Specifies, in seconds, the time the Guest spent waiting to load the venue’s “ride” (from “entry”) |
| mergetime\* | Specifies, in seconds, the time the Guest spent waiting to reach the merge point. This element is only present if the Guest entered through the xPass queue. |
| totaltime | Specifies, in seconds, the total time the Guest spent in the venue |

\*Note that if an xBRC is operating with the Controlled Space model (as opposed to the Attraction model) that the *xpass* value will always be set to *true* indicating that the guest exited the controlled space through an xTP touch reader. Additionally, *waittime* and *mergetime* values will not be set. The *carid* value may or may not be set, depending on the actual venue. Restaurants, for example, will populate the *carid* value with some reference to Guests’ table locations.

### ABANDON Message

The ABANDON message is used in two ways. In the Attraction and Controlled Space models, the ABANDON message is used to identify guests that have not been reported at any reader in a venue for a configurable amount of time. Typically, this occurs when a guest has left a venue prematurely, perhaps ducking under a queue rope. In the Space model, the ABANDON message indicates that a guest what was being sensed at a reader location is no longer being sensed at that location.

<message type=”ABANDON*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <xpass>[*true* | *false*]</xpass>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <lastxmit>*time*</lastxmit>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the ABANDON message. |
| xpass\* | “true” if guest entered the venue through an xPass queue. “false” if through standby queue |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| lastxmit | Specifies the time (in ISO 8601 format) when the last activity was noted from the band |

\*Note that if an xBRC is operating with the Controlled Space or Space models (as opposed to the Attraction model) that the *xpass* value will always be set to *true*.

### METRICS Message

<message type=”METRICS*”* time=”*timestamp*”>  
 <starttime>*time*</starttime>  
 <endtime>*time*</endtime>  
 <standby>  
 <guests>*guest count*</guests>  
 <abandonments>*abandonment count*</abandonments>  
 <waittime>*wait time*</waittime>  
 <totaltime>*total time*</totaltime>  
 </standby>  
 <xpass>  
 <guests>*guest count*</guests>  
 <abandonments>*abandonment count*</abandonments>  
 <mergetime>*merge time*</mergetime>  
 <waittime>*wait time*</waittime>  
 <totaltime>*total time*</totaltime>  
 </xpass>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| starttime | Time (ISO 8061) format when the statistics started to be collected. |
| endtime | Time when statistics stopped being collected |
| guests | Count (integer) of Guests that entered the venue during the period. |
| abandonments | Count (integer) of Guests that abandoned the venue during the period. |
| waittime\* | Specifies, in seconds, the time the Guest spent waiting to load the venue’s “ride” (from “entry”) |
| mergetime\* | Specifies, in seconds, the average time Guests spent waiting to reach the merge point. Note that this data is only relevant (and present) for xPass data. |
| totaltime | Specifies, in seconds, the average total time Guests spent in the venue |

\*The format of the METRICS message varies depending on whether the xBRC is configured to use the Attraction or Controlled Space model. In the latter case, only the *xpass* section is populated and only a subset of the data is present in that section. Specifically, neither *waittime* nor *mergetime* is present.

The METRICS message is sent out on a periodic basis. By default, this period will be every 10 minutes but a mechanism will be provided to change this time value. The METRICS message provides xBRC-calculated statistics summarizing the operational performance of an attraction. The *startime* and *endtime* elements identify the time period summarized by the message. The *standby* and *xpass* elements contain elements summarizing the operational performance of the standby and xPass queues. Data are provided for guest counts and abandonments as well as for wait times, total times and merge times (xPass queue only).

### ERROR Message

<message type=”ERROR*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <errorcode>*code*</errorcode>  
 <errormessage>*message*</errormessage>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the ERROR message. |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the error. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| errorcode | An application-specific alphanumeric error code. |
| errormessage | A descriptive message indicating the nature of the error. The message will be written to be understandable by a non-technical Cast member. |

The ERROR message is generated when a condition has been detected that needs to be communicated to a cast member. Note that this mechanism is *not* used to indicate entitlement failures (at xPass or park entry).

### READEREVENT Message

<message type=”READEREVENT*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <preferredGuestId>*guest id*</preferredGuestId>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <readerlocationid>*reader location id*</readerlocationid>  
 <readername>*reader name*</readername>  
 <readerid>*reader id*</readerid>  
 <rfid>*rfid*</rfid>  
 <iswearingprimaryband>*true/false*</readername>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the entry message. |
| preferredGuestId\* | Identifies the guest id, using the registered preferred guest id type if available for the guest. Tag is omitted if no preferred type has been registered. |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| readerlocationid\* | Identifies the abstract location of the event by its id rather than name. |
| readername\*\* | Identifies the individual touch reader (by name) originating the event. |
| readerid\*\* | Identifies the individual touch reader (by id) originating the event. |
| rfid\*\* | Identifies the band’s RFID touched to originate the event. |
| iswearingprimaryband\* | True if the guest is wearing a band identified as a “primary” band. |

\*These items are only present in the Space model.  
\*\*These items are only present in the Space model when a touch reader is involved.

This message is used in different ways, depending on which model an xBRC is using.

With the Attraction model, READEREVENT messages are only generated to trigger external activity (for example, personalized displays). It is assumed that external applications will subscribe to xBRC messages and will filter for messages from particular *readerlocations* of interest and will initiate activities as needed.

With the Space model READEREVENT messages will be sent whenever a Guest appears at a new xBR location or when a Guest touches an xTP. A Guest that remains near an xBR for a long period of time will *not* generate multiple READEREVENT messages. Note too that in the Space model this message contains additional information documented in section “Cruise Ship Considerations”.

## JMS Transport Details

The xBRC will use a “publish and subscribe” endpoint called *com.disney.xbrc* on a configured Sonic MQ message broker. It will send *text* messages to this endpoint containing the XML described in this document. Three JMS string properties will be set on all messages.

|  |  |
| --- | --- |
| String property | Description |
| *xbrc\_facility* | Identifies the facility that is sending the message. Typically, this value is a One Source originated entertainment id.. |
| *xbrc\_facility\_type* | Identifies the model in use by the xBRC. This is a “short” form, not a full Java type. The current values are “attractionmodel”, “parkentrymodel” and “spacemodel”. |
| *xbrc\_message\_type* | Identifies the message type (Table 1) of the message. |

These properties allow downstream applications to use JMS filtering services to request only the messages that pertain to them.

# Cruise Ship Considerations

Much of the initial work on the xBRC focused on the Attraction use case. In this model, the xBRC understands an attraction’s *model* and can calculate various wait times and determine things like attraction “abandonment”.

The Cruise Ship scenario is much simpler from the perspective of the xBRC. The logic that determines the state of a guest (for example, whether a guest has checked into a day care area) will be maintained by an external application. The xBRC will use the Space model. As a result, in the Cruise ship use case, the xBRC primarily serves to perform simple “singulation” of events (determining to which long-range reader a Guest is closest) and to send a corresponding READEREVENT message. The Cruise ship applications will listen to these READEREVENT (and ABANDON) messages as well as to its own cast user interface input to implement the requirements of the broader use case.

The Cruise Ship model also supports an ability to delay READEREVENT messages to increase the probability that a Guest is “singulated” to a single location. This reduces the number of READEREVENTS that are sent from Guests that are wandering through different locations. To achieve this, Guests signal strength is monitored over a period of time until a guest settles at a single location. Based on a combination of signal strengths and time at a given location, a “confidence” factor is calculated and included in a READEREVENT message. The length of time, frequency at a single location, and band roles are all use to calculate confidence and can be customized through the xBRC configuration parameters.

The Cruise ship use case also differs from the Park Attraction in that the former will not use JMS for its communications. The Cruise ship application will communicate with the xBRC using RESTful messages. Likely, it will use the HTTP POST /updatestream message to initiate push mode and will then process messages sent to it by the xBRC.

## Differences in Message Formats

The Space Model generates slightly different messages documented in the following sections

### READEREVENT Message

<message type=”READEREVENT*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <preferredGuestId>*guest id*</preferredGuestId>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <readerlocationid>*location id*</readerlocationid>  
 <iswearingprimaryband>*True*</iswearingprimaryband>  
 <readername>*reader name*</readername>  
 <readerid>*reader name*</readerid>  
 <rfid>*band RFID*</rfid>  
 <confidence>*confidence*</confidence>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the entry message. |
| preferredGuestId\* | Identifies the guest id, using the registered preferred guest id type if available for the guest. Tag is omitted if no preferred type has been registered. |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| readerlocationid | A numeric id associated with the location. This number remains constant even if the location’s name is changed. |
| iswearingprimaryband | *True* if the guest is wearing a band designated as a primary band. *False* if not or unknown. For touch readers, this value is *True* if the touched band is a primary or *False* if not or the band is unassigned. |
| readername | *(Present only for touch* *readers)* The name of the reader where the touch event occurred. |
| readerid | *(Present only for touch* *readers)* A numeric id associated with the reader. This number remains constant even if the reader’s name is changed. |
| rfid | *(Present only for touch* *readers)* The RFID of the band that generated the touch event. Note that if the band is not assigned, that the value of the guestid will be empty while this element will still report the band’s RFID. |
| confidence | *(Present only for long range readers)* A percentage value that indicates the calculated probability that a user is at the given reader. |

### ABANDON Message

<message type=”ABANDON*”* time=”*timestamp*”>  
 <guestid>*guest id*</guestid>  
 <preferredGuestId>*guest id*</preferredGuestId>  
 <iswearingprimaryband>*True*</iswearingprimaryband>  
 <readersection>*reader section*</readersection>  
 <readerlocation>*reader location*</readerlocation>  
 <readerlocationid>*location id*</readerlocationid>  
 <lastxmit>*time*</lastxmit>  
</message>

|  |  |
| --- | --- |
| Element | Purpose |
| guestid | Identifies the guest id that triggered the ABANDON message. |
| preferredGuestId\* | Identifies the guest id, using the registered preferred guest id type if available for the guest. Tag is omitted if no preferred type has been registered. |
| iswearingprimaryband | *True* if the guest is wearing a band designated as a primary band. *False* if not or unknown. |
| readersection | Identifies the “section” (a configured string) associated with the reader location. |
| readerlocation | Identifies the reader(s) reporting the event. Typically, there will be multiple readers “ganged” in each reader location. The *readerlocation* element identifies the abstract location of the readers (e.g. “entry”, “load”, etc.). |
| readerlocationid | A numeric id associated with the location. This number remains constant even if the location’s name is changed. |
| lastxmit | Specifies the time (in ISO 8601 format) when the last activity was noted from the band |

## Sequence Diagrams

The following sequence diagram describes the general operation of the xBRC in the DCL scenario:

Annotations for Sequence diagram:

1. The xBRC asks the xBRMS for information to pre-seed its band id cache. This cache allows the xBRC to associate a touch (RF) or long-range ID with a particular guest ID.

2. The DCL’s Band Location Service retrieves location information from the xBRC. The xBRC responds with a description of each of installed xBR or xTP readers.

3. The Band Location Service pre-seeds its table of guest locations by querying the xBRC. The xBRC responds with a list of Guests and their locations.

4. The Band Location Service asks the xBRC to send Guest location changes to a named URL.

5. As Guests move inside a monitored space, the xBRC informs the Band Location Service by invoking HTTP PUT operations to the URL identified in step 4.

# Park Entry

The Park Entry scenario represents a specialized implementation of the xBRC with a unique workflow and validation partners for touch events that are found nowhere else in the park. Park Entry is a three-step process:

Validate the entitlement.

Validate a biometric (fingerprint) parameter associated with the guest id.

Increment the entitlement counter.

In addition to the transactions validating the entitlement, the entire process is monitored by cast members with a hand held device.

The following sequence diagrams demonstrate the various interactions.







## Cast App Transactions

### Logon/Logoff

Park Entry is a financial transaction and therefore must be attended by a cast member. The Logon / Logoff messages ensure that a cast member is associated with every transaction. The Logon transaction signals the xBRC that the next “touch” on one of the xTPs belongs to the cast member to be associated with subsequent transactions.

The following fields apply to all message types.

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| type | This defines which message is being sent. In other message dialects, this is a separate element, but in the cast app language, we use type to separate the message definition. |
| lane | This corresponds to the actual xTP the guest used. It is an offset value 1-8, that corresponds to the actual reader. |
| reader | Identifies the reader(s) reporting the event. Multiple readers make up a Park Entry location. In other dialects, this is referred to as a reader location. |
| errorCode | In the event of an error, this field will contain a string error code such as READER\_OCCUPIED. |
| Error | In the event of an error this field will contain an error description. |

#### Greeter Logon Mode

Cast app starts a logon mode to the reader location.

<message type="95" RFID=”cast member RFID” reader="reader location" force="true/false"/>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| force | Set this flag to “true” to force a logout of currently logged in cast member. |
| RFID | Optional. Used when recovering from a cast app crash. The RFID is the ID of the cast member that was logged into this reader location when the cast app crashed. |

#### Greeter Logon Mode Response

Response to the logon mode message of type 95.

<message tap="true/false" reader="reader location" type="96" errorCode=”error code” error=”error desc”>status</message>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| tap | If “true” then the cast member needs to touch to proceed with the logon. If “false” or not set then no touch is necessary. For example, if the RFID of currently logged on cast member matches that of the RFID attribute of message type=”95” then no touch is necessary. |
| status | The result of the requested operation 1 denotes that the operation was successful, 0 denotes it was not successful. |

#### Greeter Logon Response

<message RFID="cast member RFID" reader="reader location" type="97" errorCode=”error code” error=”error desc”>status</message>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| RFID | The RFID of the cast member that touched a reader. |

#### Greeter Logout

<message type=”98” RFID="cast member RFID" reader=”location name”/>

<message type=”99” reader=”location name” errorCode=”error code” error=”error desc”>status</message>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| RFID | The RFID of the cast member currently logged into the reader. |

### Green Light Blue Light Operations

#### Entitlement Status

The Entitlement Status is sent following a reader green or blue light event.

<message type=”2”>

<RFId reader=”location name” lane=”lane”>BandID</RFId>

<guest id=”guested” fname=”first” lname=”last”>

<celebrations>

<celebration name=”celebration name” date=”celebration date”/>

</celebrations>

</guest>

<status valid=”status”>status text</status>

<Entitlement/>

</message>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| Guest | This element is NOT used in this implementation |
| celebrations | This element is NOT used in this implementation. |
| celebration | This element is NOT used in this implementation. |
| Status | Contains any error code returned from OmniTicket |
| Valid | Tells the cast app which light is lit (0=blue, 1=green) |
| Entitlement | XML document passed directly from OmniTicket containing information on the entitlement to be used for “Blue Lane” processing. |

#### Get Entitlement Status

At any time the cast app may ask for the current entitlement status by sending message type=”1”.

<message type="1" reader="reader location"/>

#### Blue Light Reset

<message type=”4”>

<reset reader=”location name” lane=”lane” />

</message>

<message type=”4”>

<reset reader=”reader location” />  
</message>

#### Blue Light Clear

The Blue Light Clear message is sent to the cast app whenever the reader blue light is turned off.

<message lane="lane" reader=”reader location" type="5"/>

## OmniTicket Transaction

### Header

<Header>

<ReferenceNumber>reference number</ReferenceNumber>

<TransactionId>transaction id</TransactionId>

<RetryCounter>retries</RetryCounter>

<RequestType>request type</RequestType>

<RequestSubType/>

<InterfaceVersionID>version number</InterfaceVersionID>

<DeviceID/>

</Header>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| ReferenceNumber | This is an xBRC generated number indicating the overall transaction. It should stay constant throughout the Guest Gate interaction. |
| TransactionNumber | This is the incremental transaction number. It should start at 1 with each new Guest Gate interaction and increment with each OmniTicket communication. This is used to match up requests and responses/ |
| RetryCounter | This starts with 0 and increments each time a transaction is repeated (i.e. when moving to the second TOR). |
| RequestType | Enumerated value since the Request Answer message set is coarse grained, this is used to distinguish transactions. The values are:  *Login*: Used to login the cast member  *Connect*: Used to introduce each xTP to the OmniTicket TOR. An xTP that is not Connected will not be allowed to admit guests.  *Watchdog*: Once connected, this message must be transmitted from each xTP periodically (once per minute, by default, set in the Connect Answer).  *Entitlement*: Used to decrement entitlements as bands are touched. This is the main transaction for the xBRC / OmniTicket interface. |
| RequestSubType | Used only with the Entitlement RequestType it indicates the type of interaction. The values are:  *Update*: This is the primary subtype, used whenever the xBRC is validating an entitlement and the OmniTicket system is matching Biometrics.  *NoUpdate*: This is an inquiry transaction and is not used by the xBRC.  *NoMatch*: Replaces the Update when the xBio is doing the Biometric matching. |
| InterfaceVersion | Set to V1.0 In future releases, this can be incremented to guarantee that the OmniTicket system can successfully process the messages being sent. The OmniTicket will reject any message version it cannot process. |
| DeviceID | 3 digit (000-254) identifying the xTP the transaction is coming from. |

### Request

#### Login

<Login>

<UserInfo>

<User>

<NumericId>BandID</NumericId>

</UserInfo>

</Login>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| NumericId | This is the band id of the cast member logging in. It can come from any of the xTP in the location once the xBRC has been put into “login” mode. |

#### Connect

<WorkRules>

<Tags>All</Tags>

</WorkRules>

<Connect>

<DeviceType>30</DeviceType>

<Status>1</Status>

</Connect>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| Tags | The level of granularity in the response. Must be set to All |
| DeviceType | The type of turnstile. The xTP is type 30. |
| Status | The operational status of the device. 1 is operational. |

#### Watchdog

<Watchdog/>

There is no value, this tag with the Device ID in the header is enough.

#### Entitlement

<WorkRules>

<Tags>All</Tags>

</WorkRules>

<Entitlement>

<Redeem>1</Redeem>

<BioValidation>1</BioValidation>

<MediaInfo>

<MediaSearchMode>

<xBandID>BandID</xBandID>

</MediaSearchMode>

<BioDeviceFilter>3</BioDeviceFilter>

<BiometricInfo>

<Item>1</Item>

<BioDeviceType>3</BioDeviceType>

<BiometricTemplate>template</BiometricTemplate>

</BiometricInfo>

</MediaInfo>

</Entitlement>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| Tags | The level of granularity in the response. Must be set to All |
| Redeem | Request to redeem the entitlement if it exists, should always be 1 |
| BioValidation | Request to have OmniTicket perform Bio validation. Should be 1 when OmniTicket is doing validation, 0 when the xBio is doing the validation. |
| xBandID | The RFId that was read. This should NOT be the Guest ID used elsewhere in this document. |
| Item | The sequential number of the template being sent. This is only used on the second request when a biometric read is required, should always start with 1. There will only ever be 1 template, |
| BioDeviceFilter | This will always be set to 3 for the xBRC. |
| BiometricTemplate | The template data collected from the Bio device. This is only used on the second request when a biometric read is required |

### Answer

<Error>

<ErrorCode>error code</ErrorCode>

<ErrorShortDescription>Short Desc.</ErrorShortDescription>

<ErrorDescription>Description</ErrorDescription>

</Error>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| ErrorCode | This is the error code generated from the OmniTicket system. This will be 0 if there is no error. |
| ErrorShortDescription | This is the 100 character description of the error. |
| ErrorDescription | This is the full text of the error generated by OmniTicket. This tag will only be sent if there is an error. |

#### Login

The request data is echoed back in the Answer.

#### Connect

<Connect>

<DeviceIP>IP address</DeviceIP>

<TorId>tor id</TorId>

<AccessAreaId>1</AccessAreaId>

<GroupID>1</GroupID>

<WatchDogTimeout>watchdog</WatchDogTimeout>

</Connect>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| DeviceIP | The IP address of the TOR. |
| TorId | The identifier of the currently connected TOR |
| AccessArea | Always set to 1 for xBRC |
| GroupID | This is always set to 1 for xBRC. |
| WatchdogTimer | The amount of time in milliseconds the TOR is expecting to receive WatchDog messages. |

#### Watchdog

The Watchdog tag is simply echoed.

#### Entitlement

<Entitlement>

<MediaInfo>

<xBandID>BandID</xBandID>

<VisualxBandID>printed band id</VisualxBandID>

</MediaInfo>

<EntitlementInfo>

<Decremented>0/1</Decremented>

<BioRequired>0/1</BioRequired>

<SimulateBio>0/1</SimulateBio>

<NewEnrollment>0/1</NewEnrollment>

<EntitlementConfiguration>

<InternalID/>

<ExternalID/>

<Name/>

<ExtendedDescriptions/>

<Text/>

<Biometric>

<BiometricLevel>level</BiometricLevel>

<IdCheck/>

</Biometric>

<FlashAtTurnstile/>

<CodeAtTurnstile/>

</EntitlementConfiguration>

<RemainingValue/>

<Validity/>

<Usages/>

<BiometricInfo>

<Item>item number</Item>

<DeviceType>3</DeviceType>

<MatchThreshold>resolution0</MatchThreshold>

<BiometricTemplate>template</BiometricTemplate>

</BiometricInfo>

<SeasonPassInfo/>

<ShowData/>

<TicketNote/>

<TicketAttribute/>

<AccessInfoList/>

</EntitlementInfo>

</Entitlement>

|  |  |
| --- | --- |
| Element / Attribute | Purpose |
| Decremented | Indicates whether the entitlement has been decremented. When this value is 1, the “green” light should be lit. |
| BioRequired | Indicates that a biometric template must be collected. When this value is 1 and Decremented is 0, the xBRC should ask the xTP for a biometric template. |
| SimulatedBio | Indicates whether a full biometric scan (0) or a beam break (1) is required. This data is passed to the xTP as part of the biometric request. |
| NewEnrollment | Indicates whether a template exists or not. When this value is 1, a new enrollment is required. This information should be passed to the xTP. |
| InternalID | This value is not used. |
| ExternalID | This value is not used. |
| Name | This value is not used. |
| ExtendedDescriptions | This value is not used. |
| Text | This value is not used. |
| BiometricLevel | The threshold for passing a matched template. This is only used when the xBio is doing the match. |
| IdCheck | This value is not used. |
| FlashAtTurnstile | This value is not used. |
| CodeAtTurmstile | This value is not used. |
| RemainingValue | This value is not used. |
| Validity | This value is not used. |
| Usages | This value is not used. |
| Item | This is the index to the list of Biometric templates. This value is only used when the xBio is performing the match. |
| DeviceType | This value is always 3 on the xBRC |
| MatchThreshold | The minimum match value. This is only used when the xBRC is performing the match. |
| BiometricTemplate | The template used for the match. This value is only used when the xBio is performing the match. |
| ShowData | This value is not used. |
| TicketNote | This value is not used. |
| AccessInfoList | This value is not used. |

## GXP Interaction

This section details how the xBRC interacts with GXP in order to validate and consume entitlements. This process is initiated by touching an xTP reader and is concluded by the light on the reader displaying green (successful status) or blue (further action required). In addition to xBRC and GXP, the process involves a separate cast application that allows the cast member staffing the reader to resolve the blue light status by either denying access or by requesting an “override” to the xPass entitlement system.

The diagrams on the next page (borrowed from Disney documents) depict the interactions between the various components. The key interactions explored in this document are those identified as step ➁ in the first diagram and step ➇ in the second. Step ➁ describes how the xBRC interacts with the GXP in order to validate/redeem entitlements while step ➇ describes how the cast application/GXP can reset the status of an xTP after a “blue light” scenario.

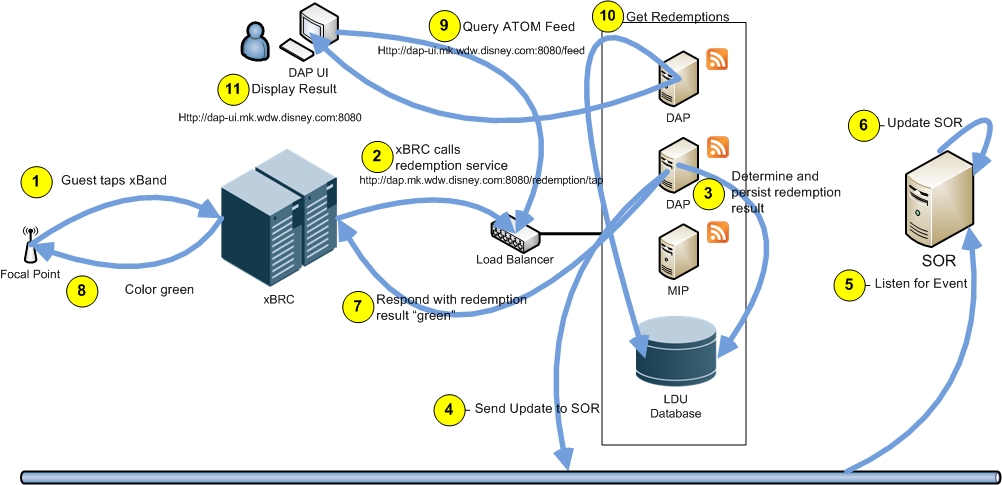


Figure - Successful xPass Redemption

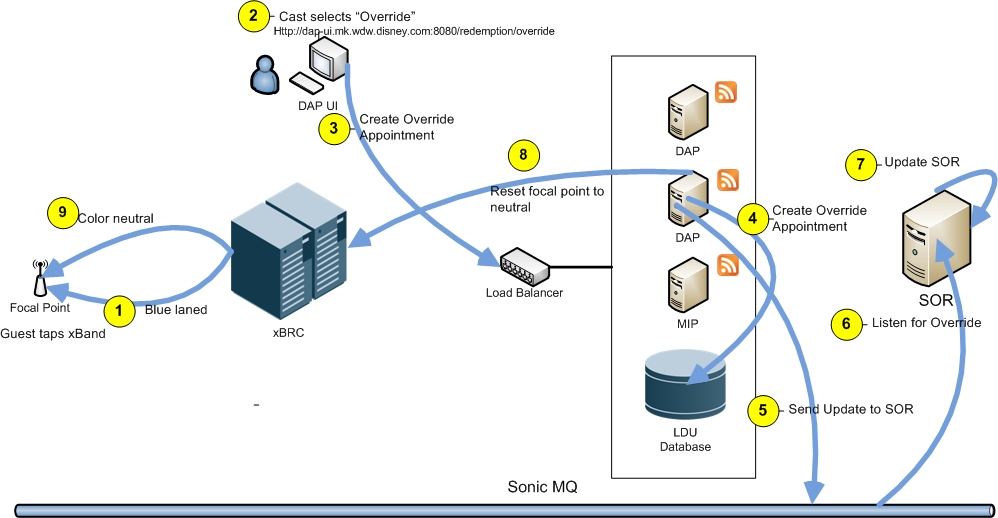


Figure - Unsuccessful Redemption with Override

## xBRC-to-GXP xPass Redemption Interaction

The xBRC will be configured with an HTTP URL identifying the GXP DAP (s) to use for xPass interactions. When a guest touches an xTP designated as requiring xPass interaction (for example, an “xpassentry” or “merge” reader in an attraction), it performs an HTTP POST to the configured URL. The path for the POST will be “/tap” and the payload will be of the form:

{  
 “tapRequest”:

{  
 “tapDate”: *timestamp*,  
 “xBandId”: *band id*,  
 “location”: *location id*,  
 “unitType”: *unit type*,  
 “entertainmentId”: *facility code*,  
 “side”: *side*,  
 “callback”: *url*   
 }  
}

|  |  |
| --- | --- |
| Element | Purpose |
| tapDate | Timestamp of touch in ISO 8601 UTC format. |
| bandId | The band id used to touch the reader. |
| location | The location where the touch occurred. This is a number assigned by One Source |
| unitType | “Entry”, “Merge” or “Combo”. The “Combo” type is used when there is a single station in a location. Note that station types may be reconfigured dynamically as mentioned below. |
| entertainmentId | The facility code where the touch occurred. A number assigned by One Source. |
| side | “Left” or “Right”. For use by the associated cast application |
| callback | The URL that should be used by the cast application in order to change the state of the light on the originating reader |

Note that *callback* might refer to a specific xBRC or might address the venue’s xBRMS. In the case of the latter, the example might look more like “xbrms.wdw.disney.com/ /kimp/light/norway-entrance-left”.

The interpretation of the “/tap” POST is up to the GXP/DAP. In general, however, the GXP will interpret the message according to the *stationtype* field. If the value of this field is “Entry”, the GXP will interpret the request as an entitlement verification. If the field is set to “Merge” then the request will be interpreted as an entitlement redemption. Finally, if the value is “Combo” then the request will perform entitlement redemption while still allowing a cast member override as in the verification step. Note that the details of these interactions are more fully described in Disney GXP documents.

The GXP’s response to the POST /tap request will look like:

{  
 “tapResponse”:  
 {  
 “green” : *boolean,* “reason” : *string*,  
 … other fields …  
 }  
}

|  |  |
| --- | --- |
| Element | Purpose |
| green | *True* if the light should be turned green, *False* if blue. |
| reason | Readable cause of an “invalid” *state* |
| … other fields … | Ignored by the xBRC. |

## GXP-to-xBRC Light Control

In the request payload described above, the xBRC provides a URL that allows GXP cast applications (indirectly through a GXP DAP) to communicate back with an xBRC (perhaps through the xBRMS) in order to control the light on an xTP. The GXP DAP performs an HTTP POST to this URL with the following payload:

{  
 “stateChange”:  
 {  
 “timestamp”: 183641341322,  
 “status”: “Reset”  
 “timeout”: 0  
 }  
}

|  |  |
| --- | --- |
| Element | Purpose |
| timestamp | Milliseconds since the start of the epoch (midnight, January 1st, 1970) |
| state | “Reset” to turn off all lights. “Green” to turn the light green. “Blue” to turn the light blue. |
| timeout | Number of milliseconds that the light should remain in the specified state (only relevant for “Green” and “Blue”). 0 indicates that the light should remain in that state indefinitely. |

## Reconfiguring xTP Readers

Not all venues (facilities) will have both xPass entry and merge readers. Some may have a single “combo” reader location. Additionally, some venues may have multiple xPass entry locations (with a single merge location). Finally, in some venues, the configuration of reader locations may be changed “on-the-fly”. For example, a venue may start out with a single “combo” reader location but may then switch to two locations, “entry” and “merge”.

Attraction cast members will be able to change the venue’s (facility’s) reader configuration by an application that will enumerate the available configurations (by name) and will allow them to select the desired one. In order to support these operations, the xbrc will implement two HTTP RESTful paths:

### Get configuration

This path will return the available configurations

{  
 “configurations”:  
 [  
 {   
 “configurationId” : *number*,  
 “name”: *string,*  
 “description” : *string*  
 },  
 {   
 “configurationId” : *number*,  
 “name”: *string,*  
 “description” : *string*  
 },  
 …

]  
}

|  |  |
| --- | --- |
| Element | Purpose |
| configurationId | Numeric identifier used to refer to the configuration |
| name | Name of the configuration. |
| description | Human readable description of the configuration. |

### PUT selectconfiguration/name/<name>

**PUT selectconfiguration/id/<id>**

These requests replace the current configuration with a stored configuration identified by its name or by its assigned id.

# Reader Registration

This section describes how readers (long range, focal point and others) register themselves with an xBRC. Typically, this process starts out by registering a reader with a DHCP server. The DHCP server is configured with the network MAC address of the reader and with the IP address of the xBRC with which that reader is supposed to communicate. When the reader is powered on, it looks for a DHCP server, obtains an IP address lease but also gets configuration information from the DHCP server. The configuration information includes the URL of the xBRC with which the reader is supposed to communicate.

Once the reader knows its xBRC, it then performs an HTTP “PUT hello” request to that xBRC. The payload of the operation is a JSON object that looks like this:

{  
 “mac” : *string,*  
 “port” : *number*,  
 “next eno”: *number*,  
 “reader name”: *string*,  
 “reader type”: *string*,  
 “reader version”: *string,* “min xbrc version”: *string,* “linux version” : *string,*  
 “location id” : *number*  
}

The meaning of these elements is as follows:

|  |  |
| --- | --- |
| Element | Purpose |
| mac | The mac address of the reader (in “XX:XX:XX:XX:XX:XX” form” |
| port | The port number that the reader listens to for incoming HTTP requests. |
| next eno | The next “event number” that will be used by the reader when sending events to the xBRC |
| reader name | The name of the reader |
| reader type | The type of the reader. Currently, the known types are “Long Range”, “xTP”, “xFP+xBIO” and “Mobile Gxp” |
| reader version | (optional) A software version string of the form “a.b.c.d” |
| min xbrc version | (optional) A string of the form “a.b.c.d” identifying the minimum version of the xBRC software expected by the reader. |
| linux version | (optional) A string identifying the version of operating system used by the reader. |
| location id | (optional) The numeric id of the location where the reader is operating. |

When a reader registers itself with an xBRC by sending the PUT hello request, if the xBRC does not “know” the reader (in other words, it has not been configured to anticipate the reader registration), the xBRC will associate the reader with the “UNKNOWN” location id. Note that the reader can override this behavior by explicitly stating a “location id” element.

In addition to letting the xBRC know that a reader is attaching itself, the PUT hello request also serves other purposes. First, the xBRC will perform a software consistency check by analyzing the reported “reader version” and “min xbrc version” strings. If the xBRC has newer software available for the reader (and the reader supports automatic software updates), the xBRC will send an HTTP request to the reader informing it of the available software and allowing it to request the software for automatic update. Second, the xBRC will also respond to the PUT hello request by updating the reader’s clock (again, by sending it an HTTP request, but only if the reader supports such a request). Finally, the xBRC may also perform other reader type-specific operations to initialize the reader.

Long range and focal point readers (with or without biometric readers) listen to the designated *port* for incoming HTTP requests from the xBRC. Other reader types (name, Mobile Gxp) do not. In the former case, the xBRC further responds to the PUT hello request by sending an HTTP POST update\_stream request back to the reader. This request tells the reader to send subsequent event information to a designated URL in the xBRC. In the case of Mobile Gxp readers, these readers send events by performing HTTP PUT stream requests to the xBRC.

Regardless of how readers are initialized, they ultimately send events to the xBRC by encoding them in JSON format and PUTing them to the “stream” path. The JSON formatted events look like:

{  
 “reader name”: *string*,  
 “events”:  
 [  
 {  
 “type”: *string*,  
 “eno”: *number*,  
 “time”: *ISO 8661 date/time string*,  
 … type specific data…   
 },  
 … more events…  
 ]  
}

The meaning of these elements is as follows:

|  |  |
| --- | --- |
| Element | Purpose |
| reader name | The name of the reader |
| type | The type of event. These include “LRR”, “RFID”, and several xBIO related types. |
| eno | A monotonically increasing event number |
| time | A GMT time/date encoded in ISO 8661 string format |

Type-specific data varies depending on the *type* of the event. For RFID data, for example, an event looks like:

{  
 “type”: *string*,  
 “eno”: *number*,  
 “time”: *ISO 8661 date/time string*,  
 “uid”: *string*,  
 “pid”: *string*,  
 “sid”: *string*,  
 “iin”: *string*  
}

Where the last four elements are:

|  |  |
| --- | --- |
| Element | Purpose |
| uid | The universal id of the band/card being touched. |
| pid | (optional) The public id of the band (will be set to the long range id of the band). |
| sid | (optional) The secure id of the band. |
| iin | (optional) The “issuer identification number” associated with the secure id. Disney is not presently using this field. |

## Automatic Reader Software Updates

The xBRC is capable of pushing new reader software to dap and long range readers. The reader software is one or more software package files with the \*.ipk file extension. The xBRC maintains a repository of these files in its web directory, typically in /usr/share/xbrc/www. Currently there are two methods of pushing new software to the readers, one using the /install restful reader call and second using the /upgrade restful reader call. The /install method pushes a single \*.ipk file to the reader, while the /upgrade method pushes a manifest file to the reader causing the reader to download multiple files from the xBRC and then install them.

### How does the xBRC choose which version to install

The xBRC will automatically locate all the \*.ipk, or manifest files in its repository and choose the latest version to push to the reader. A new version will be pushed if the reader version received in the hello message from the reader does not match the latest package version in the xBRC repository. If the reader version is greater than the latest version in the xBRC repository then the reader will be downgraded to the latest version in the xBRC repository.

### Minimum xBRC version

If the “min xbrc version” sent to the xBRC from the reader in the hello message is greater than the currently installed version of the xBRC, the xBRC will not process events from the reader and will report in the xBRC status that there is a problem with the reader version.

### How does the xBRC choose which install method to use

As described earlier, the xBRC can push new software to the readers either using the /install or /upgrade restful calls. The choice is based on the presence of the /usr/share/xbrc/www/packages directory or /usr/share/xbrc/www/repos directory or both. If the /usr/share/xbrc/www/packages directory exists then /install will be called. If the /usr/share/xbrc/www/repos directory exists then /upgrade will be called. If both directories exist then priority is given the /upgrade method.

### Uploading new reader packages to the xBRC

There are four RPM packages that contain the reader software.

xfp-0.0.0-0.rpm

xfpxbio-0.0.0-0.rpm

xbr-0.0.0-0.rpm

xreader-packages-0.0.0-0.rpm

The version will vary depending on the actual version of the software being installed. The first three packages above install individual \*.ipk files used with the /install method while the last package xreader-packages installs multiple \*.ipk files including manifest files using with the /upgrade method.

If the xBRC is running while the packages are installed, then the restful call http://<host>:8080/refreshpackages must be called to cause the xBRC to refresh the list of available packages.