xBRMS Architecture

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **Author** | **Description** |
| 1.0 | 5/1/12 | Arkady Glabek | Release Version |
| 1.6 | 4/22/13 | Iwona Glabek | Updates for 1.6.x |
| 1.6 | 6/17/13 | Slava Minyailov | Added “Access Control” section |
| 1.7 | 8/19/13 | Slava Minyailov | Added “Events and Auditing” section |
| 1.7 | 8/20/13 | Francisco Gonzalez | Updated screen shots for ‘xBRMS Topology’, ‘System Health’, ‘Stored xBRC Configurations’, and ‘xBRC Properties’ |

**Document Approvers & Sign-Off**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Approver** | **Role** | **Document Accept/Reject** |
| 5/12/12 | Glenn Curtis | Development Manager | Accept |
| 4/22/13 | Glenn Curtis | Development Manager |  |

**Table of Contents**

Contents

[1 Introduction 5](#_Toc364762687)

[1.1 Purpose 5](#_Toc364762688)

[1.2 Referenced Documents 5](#_Toc364762689)

[1.3 Definitions 5](#_Toc364762690)

[2 System Overview 6](#_Toc364762691)

[3 Deployment 7](#_Toc364762692)

[3.1 DISC Infrastructure 8](#_Toc364762693)

[3.2 LDU Infrastructure 9](#_Toc364762694)

[3.3 Configuration Properties 9](#_Toc364762695)

[3.4 xBRMS Topolgy 13](#_Toc364762696)

[3.5 BigIP Configuration 14](#_Toc364762697)

[4 High Availability Support 15](#_Toc364762698)

[5 Messaging Protocols 15](#_Toc364762699)

[5.1 Monitoring 16](#_Toc364762700)

[5.1 REST Query Interface 16](#_Toc364762701)

[6 Database Structure 17](#_Toc364762702)

[6.1 Table Structure 18](#_Toc364762703)

[6.2 Stored Procedures 19](#_Toc364762704)

[7 Access Control 19](#_Toc364762705)

[7.1 Authentication and Authorization 20](#_Toc364762706)

[7.2 BiTKOO Keystone Authorization 22](#_Toc364762707)

[7.3 SSO 23](#_Toc364762708)

[8 Functionality 23](#_Toc364762709)

[8.1 Web Based User Interface 23](#_Toc364762710)

[8.1.1 xBRMS Topolgy 23](#_Toc364762711)

[8.2 Global Service 23](#_Toc364762712)

[8.2.1 Assign Readers 23](#_Toc364762713)

[8.2.2 Replace Reader 23](#_Toc364762714)

[8.2.3 System Health 24](#_Toc364762715)

[8.3 Park Specific Service 24](#_Toc364762716)

[8.3.1 System and Reader Health 24](#_Toc364762717)

[8.3.2 Stored xBRC Configurations 24](#_Toc364762718)

[8.3.3 xBRC Properties 24](#_Toc364762719)

[8.3.4 REST Query Interfaces 24](#_Toc364762720)

[8.4 Auto Discovery Mechanism 24](#_Toc364762721)

[9 System Health Design 25](#_Toc364762722)

[10 Stored xBRC Configurations 27](#_Toc364762723)

[11 xBRC Properties 29](#_Toc364762724)

[12 REST Query Interface 31](#_Toc364762725)

[13 Events and Auditing 31](#_Toc364762726)

[13.1 Components 31](#_Toc364762727)

[13.1.1 xBRMS 31](#_Toc364762728)

[13.1.2 xBRMS UI 32](#_Toc364762729)

[13.1.3 XGreeter App, IDMS, JmsListener 32](#_Toc364762730)

[13.1.4 xBRC 32](#_Toc364762731)

[13.1.5 xBRC UI 32](#_Toc364762732)

[13.2 Authentication and Authorization 32](#_Toc364762733)

[13.3 Event Categories 33](#_Toc364762734)

[13.4 Events Message Format 34](#_Toc364762735)

[13.5 Description Column 35](#_Toc364762736)

[13.6 Events Collection and Aggregation 36](#_Toc364762737)

[13.7 Cleanup Procedure 37](#_Toc364762738)

[13.8 Event Interceptors 37](#_Toc364762739)

[13.9 Configuration properties 38](#_Toc364762740)

[14 UI Web Interface 39](#_Toc364762741)

# Introduction

xBRMS stands for the xBand Reader Management System. The xBRMS application provides monitoring of as well as inventory and configuration management for the xConnect system. xBRMS makes its data and functionality available both via a Web based UI (User Interface) and REST (Representational State Transfer) Service interfaces.

## Purpose

This document describes [system overview](#_System_Overview), [deployment](#_Deployment), [high availability support](#_High_Availability_Support_1), [messaging protocols](#_Messaging_Protocols), [database structure](#_Database_Structure), [access control](#_Access_Control_1), [functionality](#_Functionality), and Web service interfaces of the xConnect xBRMS system.

## Referenced Documents

|  |  |
| --- | --- |
| **Document Name** | **Purpose** |
| xBRMS Interface Control Document (900-0059) | Describes all available REST endpoints. |
| xBRMS User Guide (900-0061) | User guide to xBRMS User Interface. |
| xBRC Interface Control Document (900-0059) | xBRC ICD |
| xConnect BigIP HA Configuration (900-0174) | Describes BigIP configuration of xConnect system. Includes appropriate iApp templates. |

## Definitions

|  |  |
| --- | --- |
| **Term** | **Definition** |
| xBR | xBand Reader |
| xBRC | xBand Reader Controller |
| xBRMS | xBand Reader Management System |
| xTP | xBand Touch Point |

# System Overview

The xBRMS application is a web service written in Java programming language that runs on TCServer web server. All components of the xBRMS application are [highly available](#_High_Availability_Support) and access to these components is [protected](#_Access_Control). Each REST Service requires a dedicated instance of [SQL Server database](#_Database_Structure) and DNS support. Communication between the Web UI and the REST Service components is accomplished over HTTP REST. Communication between the xBRMS REST Services and external applications is accomplished over HTTP Rest and Java Message Service (JMS).

The Web UI is composed of dynamic web pages generated using the Struts2 application framework and JavaServer Pages (JSP) technology. The xBRMS Web UI is HTML5 enabled and supports the following Web browsers: Mozilla Firefox, Chrome, and IE7 and up.

The xBRMS REST Service communicates over HTTP REST with xBRC, IDMS, JMS Listener, xGreeter, GxP applications, and the xTP and xBR reader devices. JMS messaging is used by the xBRMS [auto discovery mechanism](#_Auto_Discovery_Mechanism). Access to both the xBRMS Web UI and its REST endpoints is protected. Every request is routed via the xConnect xAC ([Access Control](#_Access_Control_1)) component. The xAC component authenticates requests against the KeyStone Service, when available. When an instance of KeyStone is not available a local XML file is used for authentication.

The following diagram shows how xBRMS application interacts with other systems, both xConnect and external.



Figure - xConnect System Architecture

# Deployment

xBRMS application is composed of one Web UI Service, one Global REST Service, and multiple Environment Specific REST Services. Each of these services is [highly available](#_High_Availability_Support_1) and access to them is [protected](#_Access_Control). Both the WEB UI and REST Services run on TCServer. Each of the REST Services requires its own [SQL Server database](#_Database_Structure) and DNS support. The Web UI and the Global REST Services are [deployed to the DISC](#_DISC_Infrastructure) while the environment specific REST Services are [deployed to their respective environment’s LDUs](#_LDU_Infrastructure).

xBRMS REST Services use multiple configuration properties. All the access sensitive properties that must be password protected are specified in the NGE /etc/nge/config/environment.properties file. Properties specific to [park setup](#_Parks_Setup) are specified in the /usr/share/xbrms/parks.xml file generated by the xBRMS REST Services. All the other configuration properties are housed in the [dbo.config database](#_Database_Structure) table and can be updated via the xBRMS Web UI.

The following two subsections showcase the DISC and LDU infrastructure using the Release Lab environment as an example.

## DISC Infrastructure



Figure – Overall xBRMS System Architecture

The following elements are required for a typical DISC deployment:

* [SQL Server database](#_Database_Structure). The database could be created in the SQL Server cluster the IDMS uses.
* Two load balanced xBRMS REST Service VMs (VIP: nge-prod-disc-xbrms) acting as a Global xBRMS.
* Two load balanced xBRMS Web UI VMs (VIP: nge-prod-xbrms-ui). Only one xBRMS Web UI Service is needed to service all xBRMS REST Services.
* The \_rest.\_tcp.xbrms DNS SRV record pointing to the xBRMS REST Service’s VIP address (nge-prod-disc-xbrms). Readers use this record to find the global xBRMS to which to report to for [xBRC assignment](#_Assign_Readers).
* DNS Aliases (CNAME records) corresponding to the Global xBRMS REST Service and the environment specific xBRMS REST Services existing in the LDUs: nge-prod-disk-xbrms-ui , nge-prod-mk-xbrms-ui, nge-prod-hs-xbrms-ui, nge-prod-ep-xbrms-ui, nge-prod-ak-xbrms-ui, etc. These make it possible for users to open multiple copies of the xBRMS UI in a single browser window with each tab opened to the xBRMS UI working with a different environment specific xBRMS REST Service.

## LDU Infrastructure



Figure - xBRMS UI Plumbing

The following elements are required for a typical LDU deployment:

* [SQL Server database](#_Database_Structure).
* Two load balanced xBRMS REST Service VMs (VIP: nge-prod-XX-xbrms). These are environment specific. Typically, there are four xBRMS REST Services deployed (one per LDU) representing the four parks: nge-prod-mk-xbrms, nge-prod-ep-xbrms, nge-prod-ak-xbrms, and nge-prod-hs-xbrms, but there could be more. For example, Thphoon Lagoon could be a separate environment.
* Environment specific DNS SRV records: \_rest.\_tcp.mk-xbrms, \_rest.\_tcp.hs-xbrms, \_rest.\_tcp.ep-xbrms, \_rest.\_tcp.ak-xbrms, pointing to the appropriate VIP address of the environment specific xBRMS REST Service. These records are currently used by the xGreeter application to discover a per-park list of /facilities.

## Configuration Properties

The below table describes all environment.properties configuration properties required by the xBRMS REST Service. These properties must be available on startup. Changes to any of the below properties require an application restart.

|  |  |  |
| --- | --- | --- |
| **Property** | **Value** | **Description** |
| **JMS Related Properties** | | |
| nge.xconnect.parkid | Alphanumeric ID uniquely identifying an environment. | Used by the xBRMS DISCOVERY mechanism to filter JMS messages. Messages from other environments with different park IDs or those that do not include that property will be ignored. |
| nge.eventserver.brokerDomain |  | Properties used by the xBRMS JMS agent to connect to the JMS Bus. JMS messages are used by the xBRMS DISCOVERY mechanism. |
| nge.eventserver.xbrc.connectionfactory.jndi.name | XBRCConnectionFactory |
| nge.eventserver.mgmtBrokerUrl | <management broker hostname>:<port> |
| nge.eventserver.xbrc.uid | User ID |
| nge.eventserver.xbrc.pwd | Password |
| **Database Related Properties** | | |
| nge.xconnect.xbrms.dbserver.url | jdbc:jtds:sqlserver://<DB\_HOSTNAME>:1433/<DB\_NAME> | Used to discover which database to connect to at startup. |
| nge.xconnect.xbrms.dbserver.uid | Database user | Used by the xBRMS REST Service to connect to its database. Must have write, read, and execute privileges. |
| nge.xconnect.xbrms.dbserver.pwd | Database user password | Used by the xBRMS REST Service to connect to its database. |
| nge.xconnect.xbrms.c3p0.maxPoolSize | An integer (e.g., 20) | Initialization properties of the c3p0 JDBC database connection pool the xBRMS REST Service uses to query its database. |
| nge.xconnect.xbrms.c3p0.maxStatementsPerConnection | An integer (e.g., 50) |
| nge.xconnect.xbrms.c3p0.maxStatements | An integer (e.g., 400) |
| **Access Control Related Properties** | | |
| nge.xconnect.ac.logonServer | hostname | Address of the xConnect webapp that serves authentication requests. Default: the same host your browser points to. |
| nge.xconnect.ac.logonPort | port number | Default: 8080 |
| nge.xconnect.ac.logonAppName |  | Name of the webapp that serves authentication requests for the xConnect components. Default: IDMS |
| nge.xconnect.ac.ksDirs | Comma separated list of Keystone directories. | This list is only used when the KeyStone service is not available at the time xBRMS starts up. |
| nge.xconnect.ac.ksApplicationId | 1af99062-f279-4579-94b3-0b1a144566b5 | ID uniquely identifying xConnect applications. |
| nge.xconnect.ac.ksPrimaryAuthURL | URL | URL of the primary. |
| nge.xconnect.ac.ksSecondaryAuthURL | URL | URL of the secondary. |

The table below describes all the required configuration properties housed in the dbo.config database table. The xBRMS application will start even when these properties are missing, but it might not work as expected. Changes to any of these properties do not require a restart.

|  |  |  |
| --- | --- | --- |
| **Class** | **Property** | **Description** |
| MSConfig | xbrcuiport | xBRC Web User Interface application port number. |
| xviewurl | IDMS URL. |
| XBRMSConfig | assignedreadercacherefresh\_sec | Assigned readers cache refresh frequency. |
| httpconnectiontimeout\_msec | HTTP connections which don't return within this number of milliseconds are abandoned. |
| id | xBRMS ID |
| isglobalserver | Is this xBRMS server a global server. |
| jmsmessageexpiration\_sec | Life span of JMS messages . |
| jmsxbrcdiscoverytopic | JMS topic xBRMS listens to for DISCOVERY messages from applications it monitors. |
| ksconnectiontosecs | Number of seconds xBRMS will wait for Keystone response before switching to offline mode. |
| ksexpirelogondataafterdays | Days users' offline logon data is considered valid. |
| lastmodified | Last time this xBRMS updates its heartbeat. |
| masterpronounceddeadafter\_sec | Seconds to determine if the current master is unresponsive and another xBRMS instance should take over. |
| name | xBRMS server’s application human readable name to be displayed in the header in the xBRMS UI. |
| ownipprefix | xBRMS uses this configuration parameter to determine which network card to use when resolving its own host name. Example of a valid network prefix: 10., 192.168., etc…. |
| statusthreadpoolcoresize | Number of threads to keep in the pool even if they are idle. Applied to the thread pool used to check for monitored applications system health. |
| statusthreadpoolkeepalivetime | When the number of threads is greater than the core, this is the maximum time that excess idle threads will wait for new tasks before terminating. Applied to the thread pool used to check for monitored applications' system health. |
| statusthreadpoolmaximumsize | Thread pool maximum size. |
| unassignedrredercachecleannup\_sec | Unassigned readers cache cleanup period. |
| appid | Application instance ID, e.g. venue ID for xBRC, park ID for xBRMS, etc. |
| AuditConfig | collectorurl | Events collector connection URL used by the AuditNetImpl provider to 'push' events to an event collector. |
| enableddips | If isOverride=true, a space-separated list of managed systems (DIPS), on which audit is enabled. |
| isenabled | Is Audit enabled? |
| isoverride | Does this configuration override the global one? |
| keepincacheeventsmax | Maximum number of records to keep in the events cache. |
| keepinglobaldbdaysmax | Maximum number of days to keep audit events in the enterprise (global) database. |
| level | Audit Level: AUDIT\_FAILURE, AUDIT\_SUCCESS, FATAL, ERROR, WARN, INFO |
| pullintervalsecs | How often event collectors must pull events from event providers. |
|  |  |

## xBRMS Topolgy

The Park Selector page in xBRMS Web UI allows users to configure the xBRMS Web UI Service to work with multiple xBRMS REST Service installations. **IMPORTANT**: multiple instances of xBRMS Web UI application accessible via the same VIP must be configured individually and identically.

The topology reflected in the below screenshot is described in detail in the [DISC Infrastructure](#_DISC_Infrastructure) section of this document.

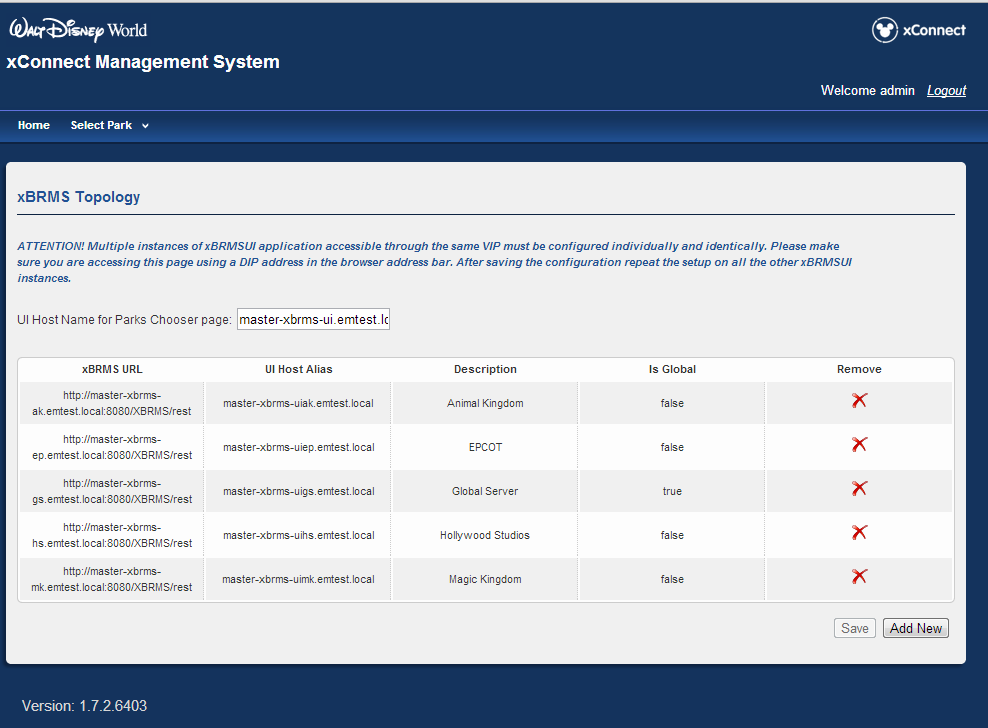


Figure – xBRMS Topology Page

## BigIP Configuration

xBRMS HA (High Availability) architecture supports one-to-many servers per each HA group.

Servers belonging to the same xBRMS UI HA group should be load balanced in a round robin fashion.

The load balancing of the servers belonging to the same xBRMS REST Service HA group is based on the response from the HTTP GET /rest/status request.

For details on how to configure xBRMS load balancing refer to the *xConnect BigIP HA Configuration* document.

# High Availability Support



Figure - xBRMS HA Architecture

# Messaging Protocols

The following diagram illustrates the various types of interfaces implemented by the xBRMS Web Service. Refer to *xBRMS Interface Control Document* for details on the available REST endpoints.

Figure 6 - xBRMS Interfaces

## Monitoring

The xBRMS Monitoring Interface allows xBRMS to monitor the health of various system components. Two communication channels are used, HTTP and JMS (Java Message Queue). The JMS channel is used by the xBRMS to subscribe to the discovery message sent out by the xBRC. This allows for automatic discovery of all xBRC installations on the network. The HTTP channel is used to periodically query the status of any monitored applications.

The xBRMS is capable of monitoring diverse applications. Each new monitored application is added to the xBRMS by subclassing a HealthItem Java class. The HealthItem implementation communicates via HTTP or other protocol with the monitored application to obtain the status of the application. The XBRC, IDMS, and JMS Listener all use the HTTP /status call for this purpose.

Only the master xBRMS pings the monitored applications for status and persists that status into the SQL Server database. The slave xBRMS servers then read that information from the database.

Information on reader health comes directly from the xBRC.

## REST Query Interface

The REST Query Interface is used by the xBRMS UI to retrieve and manipulate data collected by the xBRMS servers. It also allows other applications to query the xBRMS system for information about applications it monitors. Most of the REST endpoints the xBRMS servers make available are protected by [Access Control](#_Access_Control). However, some of the endpoints, e.g., the /facilities endpoint, are left unprotected for backwards compatibility.

# Database Structure

A separate instance of the SQL Server database is needed in each LDU and one in the DISC. Each instance of XBRMS database in the DLU is used by the xBRMS REST Service HA group, and XI and JMS Listener xConnect applications. xBRMS application shares the FacilityID and FacilityTypeID tables shown in the blow diagram with the xConnect XI application.

The database user used by the xBRMS application requires read, write and execute permissions.

## Table Structure



Figure - xBRMS Database Tables

|  |  |
| --- | --- |
| **Table** | **Description** |
| config | Stores non-sensitive application configuration parameters |
| XbrmsHaGroup | Keeps track of all xBRMS REST servers in an HA group. All members of the HA group write their heartbeat to this table. Also, as soon as one of the members assumes master’s responsibilities it indicates that fact in this table. |
| HealthItem | Collection of xConnect application servers currently being monitored by this xBRMS REST Service. |
| HealthItemField | Each health item, regardless of type, has a set of mandatory properties such as IP, hostname or port. xBRMS also collects health item type specific properties which are stored in this table. An example of such property is the ‘Model’ which only makes sense for health items of type xBRC. |
| PerformanceMetric | Collection of performance metric readings collected over time. |
| PerformanceMetricDesc | PerformanceMetric table metadata. |
| Audit | Collection of audit events from the xBRMS itself and all the systems it monitors. |
| PwHash | Logon user name and password hash for Keystone off line mode. |
| schema\_version | Current and historical data on database schema version. |

## Stored Procedures

xBRMS application uses two stored procedures:

* dbo.usp\_PerformanceMetric\_create
* dbo.usp\_PerformanceMetricDesc\_create

# Access Control

Access Control module is implemented as a generic java library. The library is integrated into each servlet-based xConnect application and consists of two parts: servlet filter and Authentication Gateway (xAG).

The servlet filter intercepts HTTP requests for secured resources and makes authorization decisions.

The xAG serves client requests for authorization tokens and collects user logon information via web-forms.

The Access Control module provides the following functionality:

* Authentication of HTTP requests to protected resources (both dynamic and static)
* Authorization of HTTP requests based on security roles
* Integration with BiTKOO Keystone Authorization Server
* SSO between to xConnect components

## Authentication and Authorization

The authentication filter intercepts all HTTP requests to XBRMS and XBRMSUI resources. If HTTP session is not established it creates a new one and tries to authenticate a user by redirecting the request to xAG. In the case of interactive logon xAG collects user credentials via a login form and authenticates them against either an internal identity database or a BiTKOO Authorization Server. On successful authentication the request is redirected back to XBRMS and the filter marks the HTTP session as “authenticated”. After that it makes an authorization decision based on the list of roles contained in the secure token generated by xAG. If BiTKOO Authorizaton Server integration is enabled the list gets copied form the authorization assertion object returned by the server.

The following environment properties control the authentication process:

# IP address or host name of the web application hosting xAG

nge.xconnect.ac.logonServer=192.168.0.149

# Port number of the xAG webapp

nge.xconnect.ac.logonPort=8080

# Name of the web application hosting xAG

nge.xconnect.ac.logonAppName=XBRMS

# How often the secure token must be re-validated

nge.xconnect.ac.reauthSecs=600

# Full name of the authentication class

nge.xconnect.ac.authServiceClass=com.disney.xband.ac.lib.auth.KeystoneAuthService

The access control rules for XBRMS/XBRMSUI application are described in ac-model.xml file, which is packaged into the application WARs. It uses the following format:

<AcModel>

<SchemaVersion>1.0.0</SchemaVersion>

<LastUpdated>2012-07-24T14:44:45.533-07:00</LastUpdated>

<Application>

<AppId>XBRMSUI</AppId>

<!-- Unprotected Resources -->

<UnprotectedUrlPattern>/css/.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/fragments/.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/images/.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/script/.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/.\*ExceptionAction.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/rest/.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/struts/.\*</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/.\*\.css</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/.\*\.js</UnprotectedUrlPattern>

<UnprotectedUrlPattern>/login.\*</UnprotectedUrlPattern>

<!-- Protected Resources -->

<Asset>

<AssetId>All Content</AssetId>

<UrlPattern>/.\*</UrlPattern>

<AllowRole>x-admin</AllowRole>

<AllowRole>xbrms-admin</AllowRole>

<AllowRole>x-operator</AllowRole>

<AllowRole>xbrms-operator</AllowRole>

<AllowRole>xbrms-maintenance</AllowRole>

</Asset>

<Asset>

<AssetId>Editable Content</AssetId>

<DenyRole>x-operator</DenyRole>

<DenyRole>xbrms-operator</DenyRole>

<!-- Struts Actions and Views -->

<UrlPattern>/update.\*</UrlPattern>

<UrlPattern>/deploy.\*</UrlPattern>

<UrlPattern method="POST">/parkssetup.\*</UrlPattern>

<UrlPattern method="DELETE">/parkssetup.\*</UrlPattern>

<!-- Addional Struts Views not covered by the above patterns -->

<UrlPattern>/.\*Edit.\*\.jsp</UrlPattern>

<UrlPattern>/assign.\*</UrlPattern>

<UrlPattern>/storedconfig.\*uploadFromFile.\*</UrlPattern>

<UrlPattern method="POST">/storedconfig.\*</UrlPattern>

<UrlPattern method="PUT">/storedconfig.\*</UrlPattern>

<UrlPattern method="DELETE">/storedconfig.\*</UrlPattern>

</Asset>

...

</Application>

...

</AcModel>

In this file each application entry encloses a list of resource collections - assets. Each asset includes a list of URL patterns and a list of roles that are allowed or denied access to the asset. Note, that the authorization module checks the deny list first. So if a role appears on both deny and allow lists for a resource that is matched by different resource patterns, the deny entry wins.

## BiTKOO Keystone Authorization

Integration with the Keystone server requires the following environment properties be defined:

# Full name of the authentication class

nge.xconnect.ac.authServiceClass=com.disney.xband.ac.lib.auth.KeystoneAuthService

# List of identity directories to be used when xAG fails to automatically

# obtain this data from the Keystone server.

nge.xconnect.ac.ksDirs=ED,WDW,APAC,BiTKOO Directory,DCL-DREAM,DCL-FANTASY,DCL-MAGIC,DCL-WDR,EMEA,LTAM,NENA,SITEMINDER,SWNA

# Keystone application ID of all xConnect components

nge.xconnect.ac.ksApplicationId=1af99062-f279-4579-94b3-0b1a144566b5

# URL of the primary Keystone server

nge.xconnect.ac.ksPrimaryAuthURL=https://keystone-ems.disney.pvt/AuthorizationService/ws.asmx

# URL of the secondary Keystone server

nge.xconnect.ac.ksSecondaryAuthURL=https://keystone-ems-ohco.disney.pvt/AuthorizationService/ws.asmx

## SSO

Any xConnect components that point to the same xAG via the environment.properties (see. *Authentication and Authorization* section) will automatically participate in SSO, which means that once a user logs in to one component he/she will not be prompted for credentials when navigating to another.

# Functionality

The xBRMS Web Service provides the following functionality.

## Web Based User Interface

The UI Web Interface is a collection of web pages written in JSP allowing the user to manage stored configurations, configure the xBRC, monitor the health status of the system, assign readers to xBRCs, replace faulty readers and more. Functionality made available in the UI when the UI is connected to the Global server is different from when it is connected to one of the Environment Specific servers.

For details on all the xBRMS functionality available via the Web UI refer to the *xBRMS User Guide* document.

### xBRMS Topolgy

The **xBRMS Topology** page allows the user to configure the xBRMS Web UI Services to present a list of the xBRMS Web UI Services that should be served.

## Global Service

### Assign Readers

Readers that haven’t yet been configured to report to a specific xBRC phone home to the Global xBRMS and wait on the **Found Readers** page until a user assigns them to a specific xBRC. The readers find the xBRMS to use for this purposes via a DNS SRV record.

### Replace Reader

The **Replace Reader** page allows users to replace faulty readers with fully functional readers. First, the user selects the park and the xBRC the faulty reader resides in and the reader that needs replacing. xBRMS then displays a list of appropriate replacement candidates and, upon user confirming the replacement action, delegates the responsibility of replacing the faulty reader with the new reader to the xBRC.

### System Health

**System health** page in the Global Service monitors state of xConnect component in DISC. Currently there are two such components: IDMS and JMS listener.

## Park Specific Service

### System and Reader Health

The **System Health** monitors the state of the various software and hardware components that make up the xConnect system in each LDU. This currently includes the xBRC, xTP and xBR readers. The status of the monitored components is presented on a web page allowing quick assessment of the health of the entire system. Systems that should be monitored by a given xBRMS are either added to that xBRMS manually or are discovered over JMS. Reader health is reported to the xBRMS indirectly by the xBRCs to which these readers are assigned.

### Stored xBRC Configurations

The xBRMS provides a repository of stored **xBRC Configurations** as well as the ability to deploy a stored configuration to a selected xBRC. A stored xBRC configuration contains a complete set of settings to configure an xBRC, including a list of readers. It is also possible to store and deploy partial configurations.

### xBRC Properties

The **xBRC Properties** mechanism allows for modification of configuration parameters for a single xBRC or a selected group of xBRCs. The main intent of this functionality is to simplify making similar changes to multiple xBRCs.

### REST Query Interfaces

The REST Query Interface allows other systems to query the xBRMS for the information about monitored xBRC applications. This service is helpful to any application that needs to dynamically build an inventory of xBRC installations. For each xBRC a contact URL is provided to make further REST calls to the xBRC. The REST Query Interface is described in detail in a separate document titled. Refer to *xBRMS Interface Control Document* for details on the available REST endpoints.

## Auto Discovery Mechanism

Provided the xBRMS discovery mechanism is enabled, the environment specific xBRMS REST Services consume JMS discovery messages from xBRCs with the [nge.xconnect.parkid](#_Configuration_Properties) property matching its own.

Multiple message properties are set on the discovery messages but xBRMS only cares about the following two properties:

xbrc\_message\_type=DISCOVERY

xconnect\_parkid=<given environment’s unique identifier>

The payload of the xBRC discovery message looks like this:

{"name":"123456787","port":8080,"haStatus":"unknown","venue":"123456787","model":"com.disney.xband.xbrc.spacemodel.CEP","configurationChangedTime":"2013-05-02T21:16:53","ip":"10.110.1.78","hostname":"SIT-XBRC-GFF-B","remoteSVUID":1,"discoveryInterval":60}

For more information on messages sent by the xBRC refer to the xBRC ICD.

To enable the discovery mechanism, configure the [jmsxbrcdiscoverytopic](#_Configuration_Properties) xBRMS property to the JMS topic used by the xBRCs. To disable this service, set the [jmsxbrcdiscoverytopic](#_Configuration_Properties) to #.

# System Health Design

The System Health monitoring is only performed by the environment specific xBRMS REST Services. Its implementation consists of the following components.



Figure - xBRMS System Health Components

1. A Java class for each type of monitored application that inherits from com.disney.xband.xbrms.server.model.IHealthSystem interface. The main functionality of this class is to provide a refreshStatus() method which queries monitored applications for their health status and persists that information to the database.
2. A registration mechanism where each type of monitored application is registered with the xBRMS Web Service. Currently, this registration mechanism includes 1) automatic registration using JMS DISCOVERY messages sent by the xBRC and 2) manual registration initiated by the user by supplying DIP hostname and port of the monitored application using a web page.
3. A background thread that periodically checks the status of each monitored application by calling the IHealthSystem.refreshStatus() method.
4. A UI web page that shows the status of all monitored applications on a single web page that is automatically refreshed. The UI web page offers drill-down functionality to investigate in more detail the status of any monitored application.

The following figure shows the System Health UI web page.

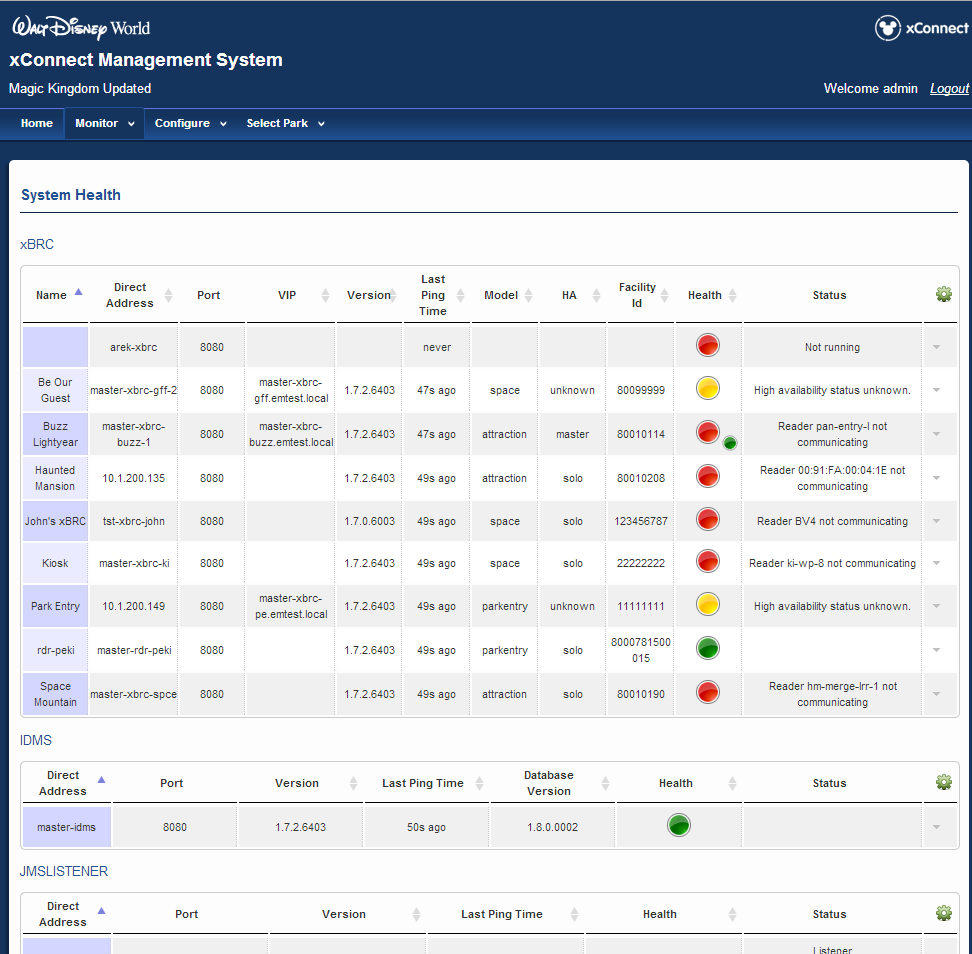


Figure - System Health Page

# Stored xBRC Configurations

The following diagram illustrates the components of the Stored xBRC Configurations functionality.



Figure - Stored xBRC Configurations

The following figure shows the Stored xBRC Configurations web page.

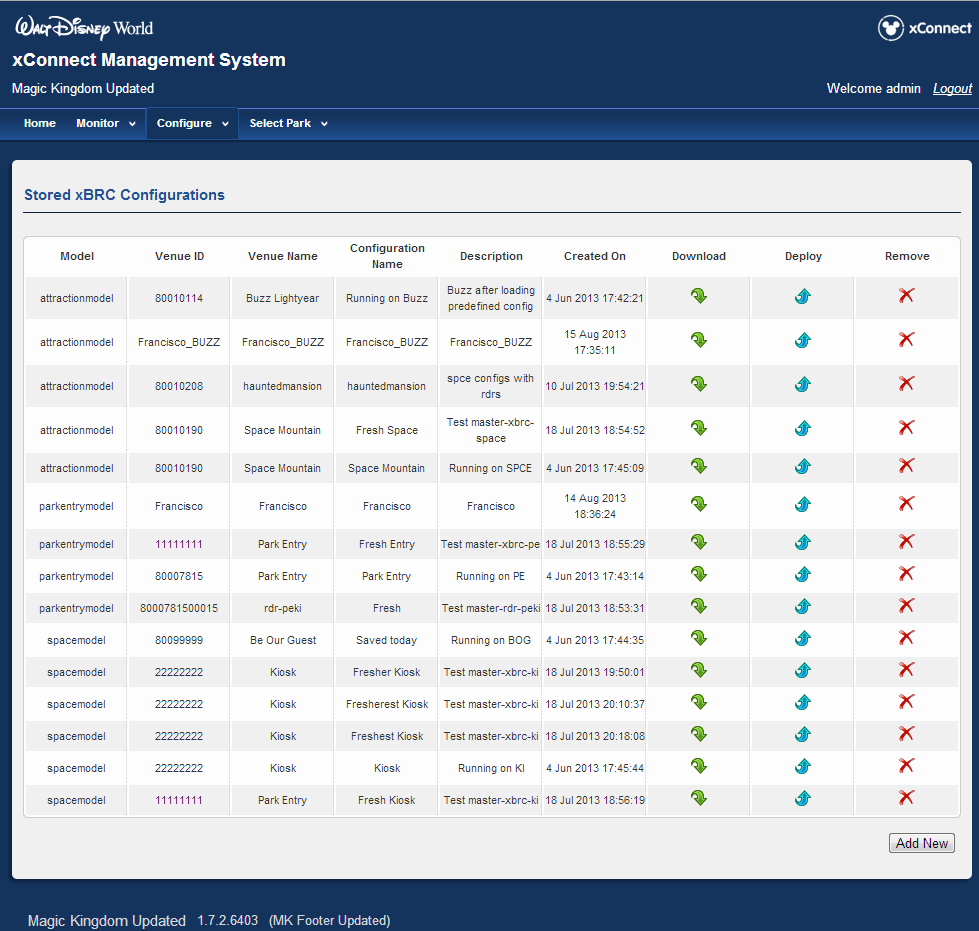


Figure - Stored xBRC Configurations Page

Please refer to the *xBRC Interface Control Document*for the description of the xBRC Configuration REST interface.

# xBRC Properties

The following figure shows the xBRC Properties page.

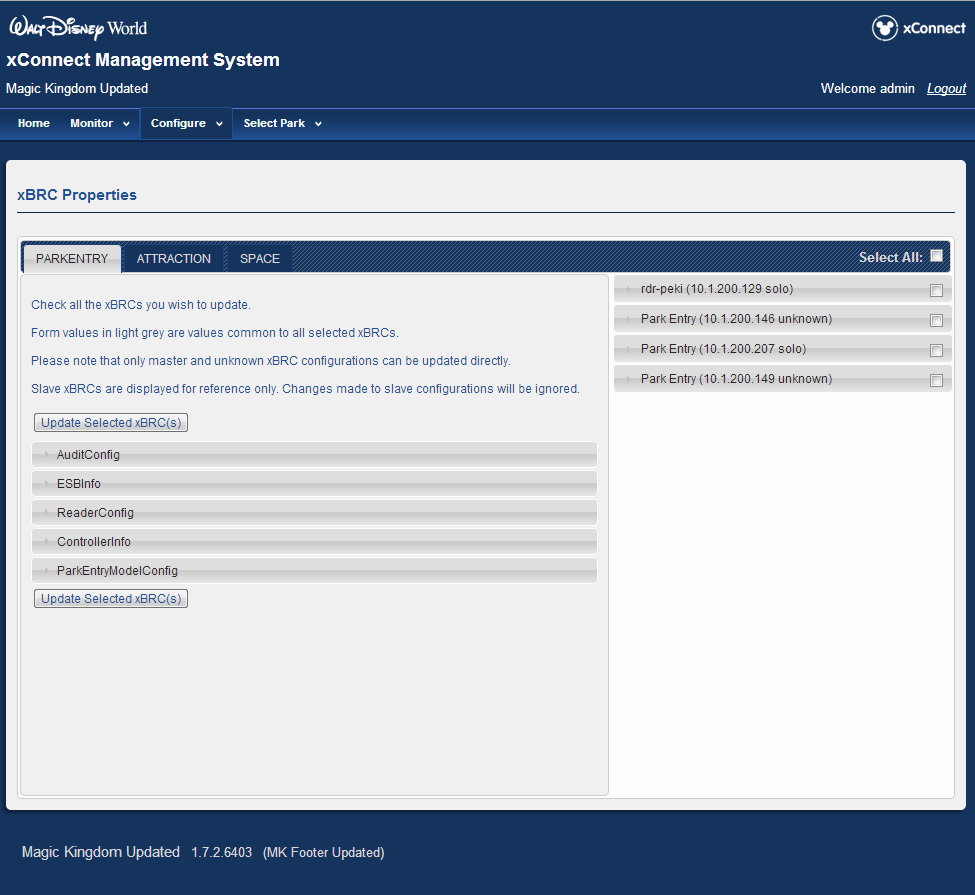


Figure - xBRC Properties Page

The xBRC Properties functionality is implemented using REST /properties call to the xBRC. The individual properties for the xBRCs are stored in the Mayhem.Config MySQL table on each of the xBRC machines. The xBRMS Web Service retrieves the properties from all xBRCs that are selected by the user using the checkboxes on the right side of the xBRC Properties Page. Once the properties are retrieved they are grouped by model (Space, Attraction, Parkentry). For each xBRC with a matching model all properties are compared to see if they are the same. If the property is the same for multiple xBRCs then the property value is displayed in the corresponding edit box. If the property is not the same for multiple xBRCs then a “Do Not Change” text is displayed instead. This mechanism allows one to modify some properties making them the same for all selected xBRCs while preserving the unique values of other properties not being modified by the user.

# REST Query Interface

Refer to the *xBRMS Interface Control Document* for the description of the REST Query Interface.

# Events and Auditing

Events and auditing subsystem maintains a centralized database of security and system health events for all xConnect components. The database enables responsible personal to examine functionality of the system over a period of time and find answers to such questions as who did what and when, as well as what happened to the system and why. The database can be queried either directly or via specialized reporting tools that will be developed over time.

Besides providing the historical data the system should be able to take real-time actions in response to critical systems events. For example, it can send an SMS or email to an administrator when a certain event occurs. Such functionality can be built later on top of the infrastructure described in this document.

## Components

The following xConnect components are producers of the audit events:

* xBRMS
* xBRMSUI
* xBRC
* xBRC UI
* xGreeter App (xBRMS)
* IDMS (via xBRMS)
* JmsListener (via xBRMS)

### xBRMS

Global xBRMS is the owner of the enterprise database. No matter in what place in the system an event occurred it will ultimately appear this database. The global xBRMS periodically collects events from park xBRMSs, which in turn pull them from xBRCs.

In addition to the describe collection model, any xConnect component can push events to either park or global xBRMS directly via “PUT@XBRMS/rest/audit/push” endpoint. Currently, only xBRMS UI uses this method.

xBRMS itself can generate audit data on:

* User logins, logouts, secure token requests and token invalidations (Greeter App)
* Initial health status of monitored components (xBRCs, IDMS, JmsListener, Park xBRMSs)
* Change in health status of monitored components (xBRCs, IDMS, JmsListener, Park xBRMSs)
* Access to all protected endpoints.

### xBRMS UI

Since xBRMS UI does not have its own database for events caching, it pushes generated events to the global xBRMS. xBRMS UI generates audit data on users' logins and logouts.

### XGreeter App, IDMS, JmsListener

Currently, these components neither push nor cache audit events for later collection by xBRMS. xBRMS itself generates events on their behalf in the following situations:

* xGreeter App requests a secure token for user authentication from xBRMS
* xBRMS detects a change in health status of one of the monitored systems (IDMS, JmsListener, or xBRC). In this case, xBRMS generates an event as if it was originated on the monitored system.

### xBRC

xBRC caches events in the Mayhem database. It generates audit data on:

* Authenticated or anonymous access
* Read, write, update and delete operations in the Mayhem database

### xBRC UI

Writes directly to the Mayhem database. Generates audit data on user logins and logouts.

## Authentication and Authorization

User access to all protected xConnect components requires authentication. The authentication process is performed by the xConnect Authentication Gateway (xAG), which delegates authentication operations to a Keystone server. As a result of successful authentication, a secure token is created. The token includes a list of roles and functional abilities granted to the user.

When one xConnect component calls another, it authenticates to it using HTTP Basic Authentication and service account credentials. If the call is made on behalf of a user, it additionally passes the user’s secure token, thus effectively impersonating the user. Besides the roles and functional abilities, the token contains a user session ID, which once set never changes until the user logs out or the user’s HTTP session expires. As a result, if all xConnect components are configured for SSO, i.e., use the same xAG server, all audit events generated by different systems in response to a user action will be marked with the same user name and session ID.

xBRC authentication mechanism is different from the one used by the servlet-based web applications like xBRMS, xBRMS UI or xBRC UI. In xBRC:

* “Fast” endpoints used by the readers are fully excluded from the authentication and audit activity
* Some endpoints used by systems external to xConnect do not require authentication.

All xBRC endpoints are described in xbrc-resources.xml file. xBRC looks for this file in /etc/nge/config directory. If the file is not present xBRC checks the default location /user/share/xbrc. The file entries look like this:

<resource method="GET" path="/currentconfiguration">

<audit>true</audit>

<authenticate>true</authenticate>

<ssl>false</ssl>

</resource>

The file makes it possible to enable/disable audit or authentication on a single endpoint or on a set of endpoints. Note that SSL attribute is not currently used.

## Event Categories

All events belong to one of two classes: *audit* or *health*.

The primary focus of the audit events is accountability. The audit events are marked with the user and session IDs. The latter can be used to correlate user-initiated operations spanning multiple systems.

*Health* events carry data related to the system operational status. They are not associated with real users and do not have session ID.

The following table summarizes the events classification:

|  |  |  |
| --- | --- | --- |
| **Event Type** | **Event Category** | **Description** |
| AUDIT\_SUCCESS | AUDIT\_FAILURE | LOGIN | User logon event |
| AUDIT\_SUCCESS |AUDIT\_FAILURE | LOGOUT | User logout event. |
| AUDIT\_SUCCESS | AUDIT\_FAILURE | READ | Information retrieval event generated by a data provider, e.g., xBRC. |
| AUDIT\_SUCCESS | AUDIT\_FAILURE | WRITE | Information creation or modification event generated by a data provider. |
| AUDIT\_SUCCESS | AUDIT\_FAILURE | ACCESS | Data access event typically generated by a front end system, e.g., a RESTful web service. On success, it may be followed by READ or WRITE events generated by a data provider. |
| FATAL | [Application specific] | Denote a total system failure. Not currently used. |
| ERROR | [Application specific] | Error condition, which may degrade system functionality or even result in total system failure. |
| WARN | [Application specific] | Unexpected or abnormal condition, which may or may not degrade system functionality. |
| INFO | [Application specific] | Informational event. |
| ERROR | WARN | INFO | STATUS | Initial health status of a monitored system. xBRMS generate this event on startup. |
| ERROR | WARN | INFO | STATUS\_CHANGE | Change in health status of a monitored system. |

## Events Message Format

All events include data to uniquely identify:

* Event source (application class, instance ID, etc.)
* Event date and time
* User (audit events only)
* User session (audit events only)

The audit WRITE events additionally contain details of a change request. The description includes a unique property ID, and a new property value. The property ID is similar to a REST resource ID. The following table specifies the detailed format of an event message.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| id | BIGINT | Private event ID local to the system owning the events cache. |
| aggregationId | BIGINT | Aggregation ID. This ID gets assigned to an event by the event aggregator as it pulls data from a child node. This ID equals the private ID of the event on the child node. |
| eventType | VARCHAR(16) | There are six event types: AUDIT\_SUCCESS, AUDIT\_FAILURE, FATAL, ERROR, WARN, INFO |
| eventCategory | VARCHAR(32) | For audit events is one of the following: READ, WRITE, ACCESS, LOGIN, or LOGOUT. For health events the category is application specific. |
| DateTime | DATETIME | Data and time of the event. |
| applicationClass | VARCHAT(16) | Unique application name, e.g. xBRMS, xBRMSUI, xBRC, xBRCUI, xGreeterApp, xi, IDMS, etc. |
| applicationId | INT | Application instance ID, e.g., park ID for xBRMS or venue ID for xBRC |
| sourceAddress | VARCHAR(128) | Host name/IP address of the event source. |
| sourceVirtualAddress | VARCHAR(128) | Host name/VIP address of the event source. |
| userId | VARCHAR(32) | User ID (Audit events only) |
| userSessionId | VARCHAR(32) | HTTP session ID (Audit events only) |
| description | VARCHAR(256) | Optional description. |
| resourceId | VARCHAR(128) | Optional resource ID (Property path for audit WRITE events) |
| resourceData | VARCHAR(max) | Optional type/specific data, e.g., new property value for audit WRITE events, or error message for “ERROR-STATUS\_CHANGE” events. |
| clientAddress | VARCHAR(256) | Host name or IP address of the client that made the request. |
| collectedFromXbrmsAt | VARCHAR(256) | Address of the event collector (typically park xBRMS) from where the event was collected. |

## Description Column

*Descriptio*n is a special column for a user-friendly message about an event. The text of the column is processed using a template that contains placeholders for other fields in the event. Each combination of “<eventType>/[eventCategory]” can have a separate template specified in the environment.properties file via *com.disney.xband.common.lib.audit.desc.template.[0-20]* property. By default the *description* field is processed using the following templates. Note that *%<column name>* is a placeholder for an event field from the above table and *<eventType>/[eventCategory]* is a pattern that tells what events the template applies to:

1. AUDIT\_SUCCESS/ACCESS: User %userId got access to end-point %resourceId
2. AUDIT\_FAILURE/ACCESS: User %userId failed to get access to end-point %resourceId
3. AUDIT\_SUCCESS/LOGIN: User %userId was successfully authenticated by xAG on host %sourceAddress
4. AUDIT\_FAILURE/LOGIN: User %userId failed to authenticate to xAG on host %sourceAddress
5. AUDIT\_SUCCESS/LOGOUT: User %userId successfully logged out from xConnect on host %sourceAddress
6. AUDIT\_SUCCESS/WRITE: User %userId modified the following %applicationClass property on host %sourceAddress: %resourceData
7. AUDIT\_FAILURE/WRITE: User %userId failed to modify an %applicationClass property on host %sourceAddress
8. ERROR/STATUS: Initial status of %applicationClass on host %sourceAddress is %message
9. WARN/STATUS: Initial status of %applicationClass on host %sourceAddress is %message
10. INFO/STATUS: Initial status of %applicationClass on host %sourceAddress is %message
11. ERROR/STATUS\_CHANGE: Status of %applicationClass on host %sourceAddress changed from %message
12. WARN/STATUS\_CHANGE: Status of %applicationId on host %sourceAddress changed from %message
13. INFO/STATUS\_CHANGE: Status of %applicationId on host %sourceAddress changed from %message
14. ERROR/: %applicationId on host %sourceId reported an error: %message
15. WARN/: %applicationId on host %sourceId issued a warning: %message
16. INFO/: %applicationId on host %sourceId said: %message

## Events Collection and Aggregation

Events collection and aggregation mechanism is hierarchical.

Initially generated events are cached in a private storage of an event provider, e.g., MySQL database in the case of xBRC or MS SQL Server database in case of xBRMS.

Event collectors (xBRMSs) periodically pull the stored events from the managed elements and write them in their own databases. Upon successful retrieval the event collectors delete the collected events on the remote system up to the last successfully saved event.

The global xBRMS periodically pulls the events from the park xBRMSs.

xConnect components that do not have a local storage, e.g,. xBRMS UI, or do not participate in the management hierarchy, e.g., xi, can publish events to the event collectors directly.

The following picture illustrates the events collection and aggregation architecture.



## Cleanup Procedure

Normally, audit events are automatically deleted from the event providers on successful retrieval by the event collectors. If for some reason no event collector pulls events from an event provider, e.g., because the provider is not being monitored by any collector, the cleanup procedure may kick in when the number of cached events exceeds an allowed maximum. The cleanup procedure checks the number of cached events every 3 hours. If the number exceeds a limit (10000 by default), 1/3 of the oldest records get deleted automatically. Such a cleanup procedure is currently used by xBRCs and park xBRMSs.

The global xBRMS uses a different cleanup procedure which does not have a limit on the number of stored events. Instead, it automatically deletes records that have been kept in the enterprise database for more than a certain number of days (14 by default).

## Event Interceptors

Events interceptors can make xConnect components take immediate action when an audit event flowing through them satisfies certain criteria. One of the possible actions is deleting events from the normal flow. In this case, the interceptor acts as a filter.

Currently, there is only one event filter implementation - com.disney.xband.common.lib.audit.interceptors.SimpleFilter

This filter is registered during audit system initialization. It can be disabled or its default parameters changed through the environment.properties file.

The filter reduces the amount of “noise” in the audit database, e.g., the activity of the event collectors themselves. It takes parameters in the following format (BNF):

Params := Param | Param “,” Params

Param := SimpleParam | ComplexParam

SimpleParam := <Resource ID regular expression> // Filter out events with the matching “rid” field

ComplexParam := EventTypeSpec “:” EventCategorySpec “:” UserIdSpec “:” Interval // Filter out events for the same resource within a specified interval in seconds

EventTypeSpec := EventType | “\*”

EventCategorySpec := EventCategory | “\*”

UserIdSpec := UserId | “\*”

Interval := INTEGER // if Interval <= 0 events frequency will be ignored

On xBRC the filter uses the following default parameters:

“AUDIT\_SUCCESS:ACCESS::,

AUDIT\_FAILURE:ACCESS::120,

AUDIT\_SUCCESS:ACCESS:xconnect-service:,

AUDIT\_SUCCESS:READ:xconnect-service:,

AUDIT\_SUCCESS:WRITE:xconnect-service:”

On xBRMS the parameters default to:

“AUDIT\_SUCCESS:ACCESS::,

AUDIT\_FAILURE:ACCESS::120,

AUDIT\_SUCCESS:ACCESS:xconnect-service:”

## Configuration properties

Most of the configuration properties used by the audit subsystem on xBRCs can be changed either through the xBRC UI or directly in the Mayhem database.

The same audit properties can be changed on xBRMS systems through the xBRMS database.

// Is Audit enabled?

@PersistName("isEnabled")

// Maximum number of records to keep in the events cache. On reaching this number 1/3 of the records

// get automatically deleted. This setting does not apply to the enterprise (global) database.

@PersistName("keepInCacheEventsMax")

// Maximum number of days to keep audit events in the enterprise (global) database.

@PersistName("keepInGlobalDbDaysMax")

// How often event collectors pull events from event providers.

@PersistName("pullIntervalSecs")

// See AuditEvent.Type (AUDIT\_FAILURE, AUDIT\_SUCCESS, FATAL, ERROR, WARN, INFO)

@PersistName("level")

For the convenience of developers and testers the following properties can also be specified in the environment.properties file, in which case they override those in xBRMS database:

nge.xconnect.audit.enabled

nge.xconnect.audit.keepInCacheEventsMax

nge.xconnect.audit.keepInGlobalDbDaysMax

nge.xconnect.audit.pullIntervalSecs

nge.xconnect.audit.level

There are also a few properties that can be changed only through the environment.properties file:

nge.xconnect.audit.batchSizeMax // How many events try to pull in a single collect operation

com.disney.xband.common.lib.audit.interceptor.class.[0-9] // Full name of the event interceptor classes

com.disney.xband.common.lib.audit.interceptor.params.[0-9] // Corresponding parameters to the event interceptors

Note that if at least one event interceptor is specified in the environment.properties file the default interceptor will not be registered on startup.

# UI Web Interface

Refer to the xBRMS User Guide document for details on the *xBRMS Web User Interface*.