Rapture API

Library Reference



Copyright © 2014 Incapture Technologies, LLC. All Rights Reserved.

Unless otherwise noted, text, images and layout of this publication are the exclusive property of Incapture Technologies LLC and may not be copied or distributed, in whole or in part, without the express written consent of Incapture Technologies LLC.

Incapture® and Rapture® are registered trademarks of Incapture Technologies LLC and/or its affiliated companies. The Incapture Technologies Logo is a service mark of Incapture Technologies LLC.

This document is for informational purposes and does not set forth any warranty, express or implied, concerning any products or services offered by Incapture Technologies LLC or its affiliate companies.

**Document version 1.01**, October 2014

Contents

[Contents 3](#_Toc398124079)

[Introduction 4](#_Toc398124080)

[Data Types 5](#_Toc398124081)

[Admin 41](#_Toc398124082)

[Async 49](#_Toc398124083)

[Audit 51](#_Toc398124084)

[Blob 53](#_Toc398124085)

[Bootstrap 56](#_Toc398124086)

[Decision 58](#_Toc398124087)

[Document 64](#_Toc398124088)

[Entitlement 69](#_Toc398124089)

[Environment 73](#_Toc398124090)

[Event 75](#_Toc398124091)

[Feature 78](#_Toc398124092)

[Fields 80](#_Toc398124093)

[Fountain 82](#_Toc398124094)

[Index 84](#_Toc398124095)

[Lock 85](#_Toc398124096)

[Mailbox 87](#_Toc398124097)

[Notification 88](#_Toc398124098)

[Pipeline 91](#_Toc398124099)

[Question 94](#_Toc398124100)

[Relationship 95](#_Toc398124101)

[Repository 98](#_Toc398124102)

[Runner 99](#_Toc398124103)

[Schedule 109](#_Toc398124104)

[Script 113](#_Toc398124105)

[Series 117](#_Toc398124106)

[Sheet 124](#_Toc398124107)

[Table 132](#_Toc398124108)

[User 133](#_Toc398124109)

[Appendix: List of API Data Types and Calls 135](#_Toc398124110)

# Introduction

The server-based Rapture platform is designed for creating and handling distributed applications in a cloud environment.

This API Reference manual contains the details for all non-deprecated API methods, organized by the specialty APIs.

## Audience

This manual is intended for developers and software architects using Rapture at client companies.

## Supported Hardware and Software

This manual applies to versions 1.1.16 of the Rapture Platform and to later versions.

## Exception Handling in the Rapture API

Although some of the API methods are hard-coded to return a single value regardless of what happens during their exection, many others are listed as returning (in other words, throwing) a possible exception.

There are three major categories of exceptions, corresponding to HTTP result codes as follows:

1. **Unauthorized (code 401).**   
   This exception is thrown when a user attempts to call any method from the API before having logged in.
2. **Forbidden (code 403).**  
   This exception is thrown when a user does not have adequate privilges to call a particular method. The Entitlements API handles privileges.
3. **Server Error (code 500).**  
   This exception is thrown when something goes wrong with Rapture internally. Examples could include an unexpected error or a failed call to a database.

# Data Types

The data types in this chapter are unique to Rapture and are used in other APIs that are documented in this manual.

## ApiVersion

Syntax:

type ApiVersion(@package=rapture.common.version) {

String major;

String minor;

}

**Type description:** Version of the current API.

## AppConfig

Syntax:

type AppConfig(@package=rapture.common) {

String name;

String title;

String vendor;

String homePage;

String longDescription;

String shortDescription;

String iconLocation;

String splashLocation;

String jarFile;

String mainClass;

}

**Type description:** Contains configuration details for hosted apps.

## AppInstanceConfig

Syntax:

type AppInstanceConfig(@package=rapture.common) {

String name;

String appConfig;

List(String) arguments;

Map(String, String) properties;

String apiKey;

}

**Type description:** Contains information about an active instance of a hosted app.

## AppRepoSettings

Syntax:

type AppRepoSettings(@package=rapture.common) {

String codeBase;

String raptureBase;

}

**Type description:** Describes a repository for apps.

## AppStatus

Syntax:

type AppStatus(@package=rapture.common) {

String name;

WorkOrderURI workOrderURI;

WorkOrderExecutionState overallStatus;

Long lastUpdated;

}

**Type description:** The status of a Rapture app.

## AppStatusDetails

Syntax:

type AppStatusDetails(@package=rapture.common.dp) {

AppStatus appStatus;

Map<String, List<StepRecord>> workerIdToSteps = new HashMap<String, List<StepRecord>>();

String logURI;

Map<String, String> extraContextValues;

}

**Type description:** Contains additional data about the app status.

## AppStatusGroup

Syntax:

type AppStatusGroup(@package=rapture.common) {

String name;

Map<WorkOrderURI, AppStatus> idToStatus;

}

**Type description:** Used for multiple work orders that share the same AppStatus name.

## AuditLogEntry

**Syntax:**

type AuditLogEntry(@package=rapture.common.model) {

String category;

Date when;

String message;

String user;

String logId;

}

**Type description**: Defines an audit result to be logged.

## AuditLogConfig

Syntax:

type AuditLogConfig(@package=rapture.common) {

String name;

String config;

}

**Type description:** Stores the config information for an audit log.

## BlobContainer

Syntax:

type BlobContainer(@package=rapture.common) {

Map(String, String) headers;

ByteArray content;

}

**Type description:** Describes a blob repository.

## BlobRepoConfig

**Syntax:**

type BlobRepoConfig(@package=rapture.common.model) {

String description;

String config;

String authority;

String metaConfig;

}

**Type description:** Defines a blob repository.

## CallingContext

Syntax:

type CallingContext(@package=rapture.common) {

String user;

String context;

String salt;

Map<String, String> metadata;

Boolean valid;

}

**Type description:** Identifies the user making an API call. The metadata map allows API clients to associate values with a particular session (such as user preferences or client location).

## CategoryQueueBindings

Syntax:

type CategoryQueueBindings(@package=rapture.common) {

String name;

Map(String, Set(String)) bindings;

}

**Type description:** Describes bound queues in a task exchange pipeline.

## CommitObject

Syntax:

type CommitObject(@package=rapture.common.repo) {

String treeRef;

String user;

Date when;

String comment;

String changes;

List(String) docReferences;

List(String) treeReferences;

}

**Type description:** Describes an object that can be committed to version history.

## ContentEnvelope

Syntax:

type ContentEnvelope(@package=rapture.common) {

Map(String, String) headers;

Object content;

}

**Type description:** Describes the content of a repository.

## CreateResponse

Syntax:

type CreateResponse(@package=rapture.common) {

Boolean isCreated;

RaptureURI uri;

String message;

}

**Type description:** An object returned by public API calls that create things. If the object was created, the URI is returned. Otherwise, the message contains the reason the object was not created.

## DocumentMetadata

Syntax:

type DocumentMetadata(@package=rapture.common.model) {

int ver;

Date writeTime;

String user;

String comment;

Boolean deleted = false;

}

**Type description:** Contains the metadata, if any, in a Rapture document.

## DocumentObject

Syntax:

type DocumentObject(@package=rapture.common.repo) {

String content;

}

**Type description:** Defines a document object.

## DocumentRepoConfig

Syntax:

type DocumentRepoConfig(@package=rapture.common.model) {

String description;

String config;

String authority;

FountainURI fountainURI;

Boolean strictCheck = true;

Set<IndexScriptPair> indexes;

Set<FullTextIndexScriptPair> fullTextIndexes;

String updateQueue;

RaptureDocConfig documentRepo;

}

**Type description:** Describes config info for a repository that stores Rapture documents.

## DocumentWithMeta

Syntax:

type DocumentWithMeta(@package=rapture.common.model) {

String displayName;

DocumentMetadata metaData;

String content;

}

**Type description:** Contains a Rapture document and its metadata.

## EnvironmentInfo

Syntax:

type EnvironmentInfo(@package=rapture.common) {

String name;

String motd;

Map(String, String) properties = new HashMap<String, String>();

}

**Type description:** Contains Rapture’s environment variables and the message of the day (MOTD).

## ErrorWrapper

**Syntax:**

type ErrorWrapper(@package=rapture.common) {

String id;

Integer status;

String message;

}

**Type description:** Contains information associated with an error captured by the Rapture system.

## ExchangeDomain

Syntax:

type ExchangeDomain(@package=rapture.common) {

String name;

String config;

}

**Type description:** Contains configuration details for a pipeline exchange.

## ExecutionContext

Syntax:

type ExecutionContext(@package=rapture.common.dp) {

WorkOrderURI workOrderURI; //the uri of the associated workorder

Map<String, String> data;

}

**Type description:** The execution context for a work order.

## FeatureConfig

Syntax:

type FeatureConfig(@package=rapture.common) {

Map(String, FeatureVersion) depends;

String description;

String feature;

FeatureVersion version;

}

**Type description:** Holds config information for the main object used by the Feature API.

## FeatureManifest

Syntax:

type FeatureManifest(@package=rapture.common) {

List(FeatureManifestItem) contents;

Map(String, FeatureVersion) depends;

String description;

String feature;

FeatureVersion version;

}

**Type description:** This is a more detailed variant of FeatureConfig that details the complete contents of a version of a feature instance

## FeatureManifestItem

Syntax:

type FeatureManifestItem(@package=rapture.common) {

String uri;

String hash;

}

**Type description:** Contains a single entry in the contents of a Feature manifest.

## FeatureTransportItem

Syntax:

type FeatureTransportItem(@package=rapture.common) {

String uri;

String content;

String hash;

}

**Type description:** The FeatureTransportItem is an internal class used by the FeatureInstaller. It carries the encoded form of a Rapture object between the FeatureInstaller and the Rapture server.

## FeatureVersion

Syntax:

type FeatureVersion(@package=rapture.common) {

int major;

int minor;

int release;

}

**Type description:** Holds version information for a Rapture feature.

## HooksConfig

Syntax:

type HooksConfig(@package=rapture.common.hooks) {

Map(String, SingleHookConfig) idToHook;

}

**Type description:** Config data for event hooks. Refer to the Event API for more details.

## IndexConfig

Syntax:

type IndexConfig(@package=rapture.common.model) {

String name;

String config;

}

**Type description:** Config info used by the Index API.

## JavaInvocable

Syntax:

type JavaInvocable(@package=rapture.common.dp) {

}

**Type description:** An invocable piece of Java code.

## JobErrorAck

Syntax:

type JobErrorAck(@package=rapture.common) {

JobURI jobURI;

Long execCount;

JobErrorType errorType; // the type of error that was acknowledged

Long timestamp; // when this was acknowledged

String user; // who acknowledged this

}

**Type description:** Acknowledgement of a job failure or delay

## JobLink

Syntax:

type JobLink(@package=rapture.common) {

String from;

String to;

}

**Type description:** The link between two jobs.

## JobLinkStatus

Syntax:

type JobLinkStatus(@package=rapture.common) {

String from;

String to;

Integer level;

Date lastChange;

}

**Type description:** The status of the link between two jobs

## LastJobExec

Syntax:

type LastJobExec(@package=rapture.common) {

JobURI jobURI;

JobType jobType;

Long execCount;

JobExecStatus status = JobExecStatus.WAITING;

Date nextRunDate;

Map<String, String> passedParams;

String execDetails = ""; //stores additional details, to be interpreted based on jobType

}

**Type description:** Contains a copy of the previous JobExec for a RaptureJob. This data is used to retrieve statuses of jobs that were previously executed, so that the ScheduleManager doesn't have to inefficiently sift through all jobs to determine statuses.

## LicenseInfo

Syntax:

type LicenseInfo(@package=rapture.common) {

String companyName = "Unlicensed";

String salt;

Boolean developer = true;

Long expirationTimestamp;

}

**Type description:** Determines whether the license of the current instance of Rapture is valid.

## LockHandle

Syntax:

type LockHandle(@package=rapture.common) {

String lockName;

String handle;

String lockHolder;

}

**Type description:** Handle to a Rapture lock.

## NotificationInfo

Syntax:

type NotificationInfo(@package=rapture.common) {

String id;

String content;

String reference;

Long epoch;

Date when;

String contentType;

String who;

}

**Type description:** Contains all relevant data for a notification.

## NotificationResult

Syntax:

type NotificationResult(@package=rapture.common) {

Long currentEpoch;

List(String) references;

}

**Type description:** The object returned by a notification.

## PipelineTaskStatus

Syntax:

type PipelineTaskStatus(@package=rapture.common) {

PipelineTaskState state;

String taskId;

String relatedTaskId;

Date creationTime;

Date startExecutionTime;

Date endExecutionTime;

int suspensionCount;

List(String) output;

}

**Type description:** Contains details about the current tasks on a pipeline.

## PropertyBasedSemaphoreConfig

Syntax:

type PropertyBasedSemaphoreConfig(@package=rapture.common.dp) {

Integer maxAllowed;

String propertyName;

}

**Type description:** Configuration for the property-based semaphore strategy

## QCallback

Syntax:

type QCallback(@package = rapture.common.dp.question) {

String uuid;

WorkOrderURI workerURI; // should be qualified with #workerId

}

**Type description:** Callback information for a question object.

## QDetail

Syntax:

type QDetail(@package = rapture.common.dp.question) {

String prompt;

String kind; // this is the type of the data, but ‘type’ is a reserved word

}

**Type description:** Additional detail for a question object.

## QNotification

Syntax:

type Question(@package = rapture.common.dp.question) {

QuestionURI questionURI;

QTemplateURI qtemplateURI;

String answer;

String priority;

Map(String, String) mapping;

ReplyProgress progress; // UNSTARTED, STARTED, FINISHED

}

**Type description:** Used when notifications are needed for question objects.

## QTemplate

Syntax:

type QTemplate(@package = rapture.common.dp.question) {

QTemplateURI qtemplateURI;

List(UserURI) quorum; // should include users and groups

AnswerRule rule; // FIRST, MAJORITY, PLURALITY, ...

String prompt;

List(String) options; // e.g. { "Yes", "No", "Cancel" }

List(QDetail) form; // fill in the blanks question list the blanks with prompts here

List(String) reports; // reserved

Long timeout; // Default timeout value. Can be overridden.

}

**Type description:** Template data for a Question object, as defined in the Question API.

## Question

Syntax:

type Question(@package = rapture.common.dp.question) {

QuestionURI questionURI;

QTemplateURI qtemplateURI;

String answer;

String priority;

Map(String, String) mapping;

ReplyProgress progress; // UNSTARTED, STARTED, FINISHED

}

**Type description:** Main data for a Question API object.

## QuestionSearch

Syntax:

type QuestionSearch(@package = rapture.common.dp.question) {

//expect this to grow

String user; // the user or group that's being asked

ReplyProgress progress;

String sortOrder;

Long timeStamp;

}

**Type description:** Contains search results for questions

## RaptureActivity

Syntax:

type RaptureActivity(@package=rapture.common) {

String id;

String otherId;

String message;

Long progress;

Long maxProgress;

Boolean requestFinish = false;

Date lastSeen;

Long expiresAt;

Boolean finished = false;

}

**Type description:** An activity

## RaptureApplicationDefinition

Syntax:

type RaptureApplicationDefinition(@package=rapture.common) {

String name;

String description;

String version;

}

**Type description:** Describes an application that this Rapture instance recognizes.

## RaptureApplicationInstance

Syntax:

type RaptureApplicationInstance(@package=rapture.common) {

String name;

String appName;

String description;

String serverGroup;

String timeRangeSpecification;

Integer retryCount = 0;

String parameters;

String apiUser;

String lockedBy;

Boolean oneShot = false;

Boolean finished = false;

String status;

Date lastStateChange;

}

**Type description:** Describes an active application.

## RaptureApplicationStatus

Syntax:

type RaptureApplicationStatus(@package=rapture.common.model) {

String appName;

String theDate;

String instanceId;

String overrideApplicationPath;

RaptureApplicationStatusStep status;

String lastMessage;

List(String) messages;

Map(String, String) inputConfig;

Map(String, String) outputConfig;

}

**Type description:** A complete description of an application’s status.

## RaptureApplicationStatusStep

Syntax:

type RaptureApplicationStatusStep (@package=rapture.common.model) {

String INITIATED;

String PICKEDUP;

String PREPROCESSING;

String RUNNING;

String POSTPROCESSING;

String COMPLETED;

String FAILED;

}

**Type description:** Takes one of the values in the strings to describe where an Application is in its execution.

## RaptureAuthority

Syntax:

type RaptureAuthority(@package=rapture.common) {

String name;

}

**Type description:** Identifies the current authority.

## RaptureCommit

Syntax:

type RaptureCommit(@package=rapture.common.model) {

String who;

Date when;

String comment;

String changes;

String reference;

List(String) docReferences;

List(String) treeReferences;

}

**Type description:** Stores info about a specific change made to a repository.

## RaptureContextInfo

Syntax:

type RaptureContextInfo(@package=rapture.common) {

String sessionId;

String authority;

String perspective;

}

**Type description:** Holds the details of a particular context

## RaptureCubeResult

**Syntax:**

type RaptureCubeResult(@package=rapture.common) {

List<String> groupNames;

List<String> columnNames;

List<RaptureCubeRow> rows;

}

**Type description:** Contains the result of a FilterCubeView.

**Example (if helpful):**

## RaptureDNCursor

**Syntax:**

type RaptureDNCursor(@package=rapture.common) {

String authority;

List<String> displayNames;

String continueContext;

Boolean finished;

}

**Type description:** Defines a displayNameQuery cursor.

## RaptureDocConfig

Syntax:

type RaptureDocConfig(@package=rapture.common.model) {

String authority;

String config;

}

**Type description:** Describes config info for a single Rapture document.

## RaptureEntitlement

Syntax:

type RaptureEntitlement(@package=rapture.common.model) {

String name;

EntitlementType entType;

Set<String> groups;

}

**Type description:** The base object used by the entitlements API.

## RaptureEntitlementGroup

Syntax:

type RaptureEntitlementGroup(@package=rapture.common.model) {

String name;

Set<String> users;

String dynamicEntitlementClassName;

}

**Type description:** A named collection of users who share any entitlements assigned to the group, as long as they remain members of the group.

## RaptureEvent

Syntax:

type RaptureEvent(@package=rapture.common.model) {

String uriFullPath;

Set(RaptureEventScript) scripts;

Set(RaptureEventMessage) messages;

Set(RaptureEventNotification) notifications;

Set(RaptureEventWorkflow) workflows;

}

**Type description:** Describes the main object used by the Event API.

## RaptureExchange

Syntax:

type RaptureExchange(@package=rapture.common.model) {

String domain;

String name;

RaptureExchangeType exchangeType;

List(RaptureExchangeQueue) queueBindings;

}

**Type description:** A RaptureExchange is the coordination point for a task-based pipeline. Clients put RapturePipelineTask instances onto an exchange, which then routes that task to a set of queues that are then consumed. This class defines the config of an exchange.

|  |
| --- |
|  |

## RaptureField

**Syntax:**

type RaptureField(@package=rapture.common) {

String authority;

String category;

String name;

String longName;

String description;

String units;

RaptureGroupingFn groupingFn = RaptureGroupingFn.SUM;

List<RaptureFieldBand> bands;

Set<RaptureFieldPath> fieldPaths;

}

**Type description:** A RaptureField is the definition of a concept in Rapture, referenced within a type or a series of types.

## RaptureFolderInfo

Syntax:

type RaptureFolderInfo(@package=rapture.common) {

String name;

Boolean isFolder;

}

**Type description:** Information about a folder or a file in a repository.

## RaptureFountainConfig

Syntax:

type RaptureFountainConfig(@package=rapture.common) {

String name;

String config;

String authority;

}

**Type description:** Holds the config info for a fountain (as described in the fountain API in this manual).

## RaptureFullTextIndexConfig

Syntax:

type RaptureFullTextIndexConfig(@package=rapture.common) {

String name;

String config;

String authority;

}

**Type description:** Holds the config info for a Rapture index.

## RaptureInstanceCapabilities

Syntax:

type RaptureInstanceCapabilities(@package=rapture.common) {

String server;

String instanceName;

Map<String, Object> capabilities;

}

**Type description:** Contains info about the capabilities that exist in the current Rapture system.

## RaptureIPWhiteList

Syntax:

type RaptureIPWhiteList(@package=rapture.common) {

List<String> ipWhiteList = new ArrayList<String>();

}

**Type description:** Contains IP addresses that are allowed to connect to the current Rapture platform.

## RaptureJob

Syntax:

type RaptureJob(@package=rapture.common) {

JobURI jobURI;

Long upcomingExecCount = new Long(0);

String description;

ScriptURI scriptURI;

String cronSpec;

String timeZone = "America/New\_York";

Map<String, String> params;

Boolean autoActivate = true;

Boolean activated = true;

JobType jobType;

Integer maxRuntimeMinutes = -1;

String appStatusNamePattern; //appstatus name pattern, will be interpreted and passed in to workflow

}

**Type description:** Contains info for a script-driven job in Rapture.

## RaptureJobExec

Syntax:

type RaptureJobExec(@package=rapture.common) {

JobURI jobURI;

JobType jobType;

Long execCount;

JobExecStatus status = JobExecStatus.WAITING;

Date nextRunDate;

Map<String, String> passedParams;

String execDetails = ""; //stores additional details, to be interpreted based on jobType

}

**Type description:** Contains info about an active or queued job.

## RaptureLibraryDefinition

Syntax:

type RaptureLibraryDefinition(@package=rapture.common) {

String name;

String description;

String version;

}

**Type description:** Describes a third-party library being used in this Rapture instance.

## RaptureLockConfig

Syntax:

type RaptureLockConfig(@package=rapture.common) {

String name;

String config;

String authority;

String pathPosition;

}

**Type description:** Metadata used by objects in the Lock API.

## RaptureMailMessage

**Syntax:**

type RaptureMailMessage(@package=rapture.common.model) {

String id;

String authority;

String documentPath;

String content;

Date when;

String user;

}

**Type description:** A mailbox message, usually posted by an external user.

## RaptureNetwork

Syntax:

type RaptureNetwork(@package=rapture.common.model) {

String networkId;

String networkName;

}

**Type description:** Holds basic information about the network that Rapture can access.

## RaptureNotificationConfig

Syntax:

type RaptureNotificationConfig(@package=rapture.common.model) {

String name;

String config;

String purpose;

}

**Type description:** Config info for the main object in the Notification API.

## RaptureOperation

Syntax:

type RaptureOperation(@package=rapture.common.model) {

String opName;

String paramDef;

String scriptName;

}

**Type description:** Describes the main object used by the Operation API

## RaptureParameter

Syntax:

type RaptureParameter(@package=rapture.common) {

String name;

RaptureParameterType parameterType;

}

**Type description:** Parameters that should be passed to a script.

## RapturePipelineTask

Syntax:

type RapturePipelineTask(@package=rapture.common) {

PipelineTaskStatus status;

PipelineTaskType taskType;

Integer priority;

List<String> categoryList;

String taskId;

String content;

String contentType;

Long epoch;

}

**Type description:** Represents a task that has been submitted to the Rapture pipeline. Includes the task’s status, type, and categories associated with it.

## RaptureProcessGroup

Syntax:

type RaptureProcessGroup (@package=rapture.common) {

String name;

Boolean autoAssign = false;

Map(String, String) capabilities;

Map(String, List(Queues)) queuesPerAuthority;

}

**Type description:** Defines the process group of Rapture.

## RaptureProcessInstance

Syntax:

type RaptureProcessInstance (@package=rapture.common) {

String processId;

String processGroupName;

String instanceName;

Date lastSeen;

RaptureProcessState state;

long totalTasksServiced;

float serviceRate;

float recentRate;

}

**Type description:** Stores metadata about an active process in Rapture.

## RaptureQueryResult

**Syntax:**

type RaptureQueryResult(@package=rapture.common) {

List<JsonContent> rows;

}

**Type description:** A return value from a query.

## RaptureRelation

Syntax:

type RaptureRelation(@package=rapture.common) {

String uri;

List<RaptureRelationship> outgoingRelationships;

List<RaptureRelationship> incomingRelationships;

RaptureURI targetURI;

}

**Type description:** Describes outgoing and incoming Rapture Relationships, as described in the Relationship API.

## RaptureRelationship

Syntax:

type RaptureRelationship(@package=rapture.common) {

RaptureURI fromURI;

RaptureURI toURI;

String label;

Long createDateUTC = System.currentTimeMillis();

String user;

Map<String,String> properties;

String uri;

UUID uuid = UUID.randomUUID();

}

**Type description:** The main object in the Relationship API.

## RaptureRelationshipRegion

Syntax:

type RaptureRelationshipRegion(@package=rapture.common) {

RaptureURI centerNode;

Long depth;

List<RaptureURI> nodes;

List<RaptureRelationship> relationships;

}

**Type description:** Describes a cluster of relationship nodes with an arbitrary central node.

## RaptureRemote

Syntax:

type RaptureRemote(@package=rapture.common) {

String name;

String description;

String url;

String apiKey;

String optionalPass;

}

**Type description:** Defines a remote instance of Rapture.

## RaptureRunnerConfig

Syntax:

type RaptureRunnerConfig(@package=rapture.common) {

Map(String, String) config;

}

**Type description:** Config info for the current RaptureRunner implementation.

## RaptureRunnerInstanceStatus

Syntax:

type RaptureRunnerInstanceStatus(@package=rapture.common) {

String serverGroup;

String appInstance;

String appName;

String status;

Date lastSeen;

Boolean needsRestart = false;

}

**Type description:** Describes the state of a currently running instance of RaptureRunner.

## RaptureRunnerStatus

Syntax:

type RaptureRunnerStatus(@package=rapture.common) {

String serverName;

Map<String, RaptureRunnerInstanceStatus> statusByInstanceName;

}

**Type description:** Contains status info for all instances of RaptureRunner.

## RaptureScript

Syntax:

type RaptureScript (@package=rapture.common) {

String name;

String script;

RaptureScriptLanguage language;

RaptureScriptPurpose purpose;

String authority;

List(RaptureParameter) parameters;

}

**Type description:** Defines a script used to run a Rapture job.

## RaptureScriptLanguage

Syntax:

type RaptureScriptLanguage (@package=rapture.common) {

String RUBY;

String JAVASCRIPT;

String PYTHON;

}

**Type description:** One of the scripting languages compatible with Rapture.

## RaptureScriptPurpose

Syntax:

type RaptureScriptPurpose (@package=rapture.common) {

String INDEXGENERATOR;

String MAP;

String FILTER;

String OPERATION;

String PROGRAM;

String LINK;

}

**Type description:** Describes the script’s functionality.

## RaptureSearchResult

Syntax:

type RaptureSearchResult(@package=rapture.common) {

String displayName;

}

**Type description:** Searches performed by any API calls are stored in this type.

## RaptureServerGroup

Syntax:

type RaptureServerGroup(@package=rapture.common) {

String name;

String description;

Integer jmxPort;

Set(String) inclusions;

Set(String) exclusions;

Set(String) libraries;

}

**Type description:** Metadata for a Rapture server group.

## RaptureServerInfo

Syntax:

type RaptureServerInfo(@package=rapture.common.model) {

String serverId;

String name;

}

**Type description:** Holds basic information about the server running Rapture.

## RaptureServerStatus

Syntax:

type RaptureServerStatus(@package=rapture.common.model) {

String serverId;

Long status;

String statusMessage;

Date lastSeen = new Date();

}

**Type description:** Holds status information for a Rapture server.

## RaptureSheet

Syntax:

type RaptureSheet(@package=rapture.common) {

String authority;

String sheetStore;

String name;

}

**Type description:** Describes a single Rapture sheet. Refer to the Sheet API for additional details.

## RaptureSheetCell

Syntax:

type RaptureSheetCell(@package=rapture.common) {

int row;

int column;

String data;

Long epoch;

}

**Type description:** Data for one cell in a sheet. Refer to the Sheet API for additional details.

## RaptureSheetDisplayCell

Syntax:

type RaptureSheetDisplayCell(@package=rapture.common) {

String data;

String style;

}

**Type description:** Data for how to display a given cell. Refer to the Sheet API for additional details.

## RaptureSheetDisplayForm

Syntax:

type RaptureSheetDisplayForm(@package=rapture.common) {

List(List(RaptureSheetDisplayCell)) cells;

List(RaptureSheetStyle) styles;

}

**Type description:** Data for how to display a list of cells. Refer to the Sheet API for additional details.

## RaptureSheetNote

Syntax:

type RaptureSheetNote(@package=rapture.common) {

String id;

String note;

String who;

Date when;

}

**Type description:** A note that can be attached to a sheet.

## RaptureSheetRange

Syntax:

type RaptureSheetRange(@package=rapture.common) {

String name;

int startRow;

int endRow;

int startColumn;

int endColumn;

}

**Type description:** Specifies any range of cells within one sheet.

## RaptureSheetRow

Syntax:

type RaptureSheetRow(@package=rapture.common) {

List(RaptureSheetCell) cells;

}

**Type description:** Specifies a given row in a sheet.

## RaptureSheetScript

Syntax:

type RaptureSheetScript(@package=rapture.common) {

String name;

String script;

}

**Type description:** A Reflex script stored within a sheet and able to reference the current sheet.

## RaptureSheetStatus

Syntax:

type RaptureSheetStatus(@package=rapture.common) {

List(RaptureSheetCell) cells;

Long epoch;

}

**Type description:** Used for determining the updates since a fixed point in time (the epoch field).

## RaptureSheetStyle

Syntax:

type RaptureSheetStyle(@package=rapture.common) {

String name;

}

**Type description:** Stores CSS information for cells in a Rapture sheet.

## RaptureSnippet

Syntax:

type RaptureSnippet (@package=rapture.common) {

String name;

String authority;

String snippet;

}

**Type description:** Defines a portion of a Rapture script.

## RaptureTableConfig

Syntax:

type RaptureTableConfig(@package=rapture.common) {

String name;

String config;

String authority;

}

**Type description:** Holds the config info for a Rapture table, as described in a later section.

## RaptureUser

Syntax:

type RaptureUser(@package=rapture.common.model) {

String username;

String emailAddress;

String salt;

String hashPassword;

String description;

Boolean inactive;

Boolean apiKey;

Boolean hasRoot;

}

**Type description:** Defines a user account for the Rapture system.

## ReflexREPLSession

Syntax:

type ReflexREPLSession(@package=rapture.common) {

String id;

List(REPLVariable) vars;

String partialLine;

List(String) functionDecls;

Date lastSeen;

}

**Type description:** Describes a read-eval-print loop in Reflex.

## RelationshipRepoConfig

Syntax:

type RelationshipRepoConfig(@package=rapture.common.model) {

String description;

String config;

String authority;

}

**Type description:** Config information for a repository for relationships

## REPLVariable

Syntax:

type REPLVariable(@package=rapture.common) {

String name;

String serializedVar;

}

**Type description:** Defines a variables used in a read-eval-print loop.

## RepoConfig

Syntax:

type RepoConfig (@package=rapture.common.model) {

String name;

String config;

}

**Type description:** Holds the config info for a repository.

## ScriptResult

Syntax:

type ScriptResult(@package=rapture.common) {

String returnValue;

List(String) output;

}

**Type description:** Returns the variable name and output of a given script.

## SemaphoreAcquireResponse

Syntax:

type SemaphoreAcquireResponse(@package=rapture.common) {

Boolean isAcquired;

RaptureURI acquiredURI;

Set<RaptureURI> existingStakeholderURIs;

}

**Type description:** A response returned when trying to acquire a semaphore lock. It indicates whether the lock was acquired and identifies any existing stakeholders.

## SemaphoreLock

Syntax:

type SemaphoreLock(@package=rapture.common) {

String lockKey;

Set<RaptureURI> stakeholderURIs;

}

**Type description:** Controls access to a resource using the standard semaphore model in programming.

## SeriesDoubles

Syntax:

type SeriesDoubles(@package=rapture.common) {

List(String) column;

List(Double) value;

}

**Type description:** For casting SeriesValues to doubles

## SeriesRepoConfig

Syntax:

type SeriesRepoConfig(@package=rapture.common) {

String description;

String config;

String authority;

String seriesName;

String sampleColumn;

}

**Type description:** Configuration info for a repository that stores series data.

## SeriesStrings

Syntax:

type SeriesStrings(@package=rapture.common) {

List(String) column;

List(String) value;

}

**Type description:** For casting SeriesValues to strings

## ServerCategory

Syntax:

type ServerCategory(@package=rapture.common) {

String name;

String description;

}

**Type description:** Associates a server category with a comment.

## SheetRepoConfig

Syntax:

type SheetRepoConfig(@package=rapture.common) {

String description;

String config;

String authority;

}

**Type description:** Describes a sheet repository. Refer to the Sheet API for additional details.

## Step

Syntax:

type Step(@package=rapture.common.dp) {

String name; //this is not a URI, just a String with this step's name

String description; //this is a description of the step

String executable; // workflow:// script:// qtemplate:// $RETURN:value $SLEEP:275

// $varName or #literal -- always switch on and discard first character

Map<String,String> view;

List<Transition> transitions;

String categoryOverride; //category associated with this step, if it overrides the workflow category

}

**Type description:** An executable step with config data.

## StepRecord

Syntax:

type StepRecord(@package=rapture.common.dp) {

WorkflowURI stepURI; //fully qualified workflowURI with stepName (e.g. //myProj/myWorkflow#myStep)

String name; //short step name, e.g. myStep

Long startTime;

Long endTime;

String retVal;

String hostname;

WorkOrderExecutionState status; //status of this step

}

**Type description:** A detailed step execution list, containing start and finish times and short step name

## TableQuery

Syntax:

type TableQuery(@package=rapture.common) {

List(TableSelect) fieldTests;

List(String) fieldReturns;

List(TableColumnSort) sortFields;

int skip;

int limit;

}

**Type description:** Describes a query made against a table.

## TableQueryResult

Syntax:

type TableQueryResult(@package=rapture.common) {

List(String) columnNames;

List(String) columnTypes;

List(List(String)) rows;

}

**Type description:** A subsection of a table that matches a given query.

## TableRecord

Syntax:

type TableRecord(@package=rapture.common) {

String keyName;

Map(String, Object) fields;

}

**Type description:** Describes a table object.

## TimedEventRecord

Syntax:

type TimedEventRecord(@package=rapture.common) {

String eventName;

String eventContext;

Date when;

Date end;

String infoContext; // Usually a color to differentiate these with other entries

}

**Type description:** Describes a scheduled event.

## TimeProcessorStatus

Syntax:

type TimeProcessorStatus(@package=rapture.common) {

String network;

Long when;

String processingServer;

}

**Type description:** Records when and where an event (such as a recurring task) was processed.

## Transition

Syntax:

type Transition(@package=rapture.common.dp) {

String name;

String targetStep; // set to $RETURN:returnCode for a terminal step. the return code is the name of the

// transition to take in the step in the calling context (if any)

// targetStep is assumed to be the stepName, not a full URI

}

**Type description:** Points to the next step in a given sequence.

## TreeObject

Syntax:

type TreeObject(@package=rapture.common.repo) {

Map(String, String) trees;

List(DocumentBagReference) documents;

}

**Type description:** Defines a tree object.

## TypeArchiveConfig

Syntax:

type TypeArchiveConfig(@package=rapture.common) {

String authority;

String typeName;

Boolean useScript = false;

String scriptName = "";

Long versionsToKeep = new Long(-1);

Long timeRangeToKeepInDays = new Long(-1);

}

**Type description:** Used when archiving data in a type

## UpcomingJobExec

Syntax:

type UpcomingJobExec(@package=rapture.common) {

JobURI jobURI;

   JobType jobType;

   Long execCount;

   JobExecStatus status = JobExecStatus.WAITING;

   Date nextRunDate;

   Map<String, String> passedParams = new HashMap<String, String>();

   String execDetails = ""; //stores additional details, to be interpreted based on jobType

}

**Type description:** Contains a copy of the upcoming JobExec for a RaptureJob. It is used by the ScheduleManager to determine whether something needs to run. This way the ScheduleManager doesn't have to inefficiently sift through all jobs to make this decision.

## Worker

Syntax:

type Worker(@package=rapture.common.dp) {

WorkOrderURI workOrderURI; //the uri of the associated workorder

String id;

// the view of the current step, overlayed by things such as $varName or #literal -- always switch

// on and discard first character

Map<String,String> viewOverlay = new HashMap<String, String>();

// fully qualified workflowURI with stepName (e.g. //myProj/myWorkflow#myStep)

List<WorkflowURI> stack;

List<Map<String,String>> localView);

List<StepExecutionRecord> stepExecutionRecords = new ArrayList<StepExecutionRecord>(); //deprecated

List<StepRecord> stepRecords;

WorkerExecutionState status; // RUNNING, READY, BLOCKED, CANCELLED, FINISHED

String detail; // subscription id if waiting, thread psuedo-id if running

String effectiveUser;

Integer priority;

CallingContext callingContext;

String activityId = null; // If present, the activityId this worker reports general status to

String myActivityId = null; // Usually the workflow uri

List<String> appStatusNameStack = new ArrayList<String>(); //appstatus uri being used here, interpreted

}

**Type description:** Workers keep track of progress when executing a work order. Each worker handles one thread of execution.

## Workflow

Syntax:

type Workflow(@package=rapture.common.dp) {

WorkflowURI workflowURI;

//the semaphore type and config below is used when creating Work Orders

SemaphoreType semaphoreType;

String semaphoreConfig;

List<Step> steps;

String startStep; // name of the step to start on by default

String category = "alpha"; // name of the category associated with this workflow. defaults

// to "alpha", which is the category associated with RaptureAPIServer and

// RaptureComputeServer

Map<String, String> view = new HashMap<String, String>();

String defaultAppStatusNamePattern; //appstatus name pattern, to be used if nothing passed in to workorder

}

**Type description:** Defines a workflow. Each workflow is a flowchart of WorkflowSteps and Transitions.

## WorkflowBasedSemaphoreConfig

Syntax:

type WorkflowBasedSemaphoreConfig(@package=rapture.common.dp) {

Integer maxAllowed;

}

**Type description:** Config for the workflow-based semaphore strategy

## WorkflowExecStatus

Syntax:

type WorkflowExecsStatus(@package=rapture.common) {

List<WorkflowJobExecDetails> failed;

List<WorkflowJobExecDetails> ok;

List<WorkflowJobExecDetails> overrun;

List<WorkflowJobExecDetails> success;

}

**Type description:** The status for a workflow-based job that's already completed

## WorkflowJobDetails

Syntax:

type WorkflowJobDetails(@package=rapture.common) {

String workOrderURI;

}

**Type description:** The workorderId, which is stored in a RaptureJobExec for a workflow-based job

## WorkflowJobExecDetails

Syntax:

type WorkflowJobExecDetails(@package=rapture.common) {

JobURI jobURI;

Long execCount;

Map<String, String> parameters;

Map<String, String> passedParams = new HashMap<String, String>(); //optional override

WorkflowURI workflowURI;

WorkOrderURI workOrderURI;

String workOrderID; //a shorter version of workOrderURI -- just the id at the end, useful for display

Long startDate;

Long lastUpdated;

WorkOrderExecutionState workOrderStatus;

JobExecStatus jobStatus;

String prettyStatus; //a human-readable status, aggregating jobStatus and workOrderStatus

Long overrunMillis = 0L; //how much we have overrun, or 0 if not overrun yet

Integer maxRuntimeMinutes = -1;

String notes; // optional notes about this workflow's status (e.g. "Still running")

JobErrorAck errorAck; // set if this is in error state and someone acknowledged the error

}

**Type description:** The status for an individual workflow-based job execution - either upcoming or in the past

## WorkOrder

Syntax:

type WorkOrder(@package=rapture.common.dp) {

WorkOrderURI workOrderURI;

WorkflowURI workflowURI; //uri of Workflow used when creating this WorkOrder

List<String> workerIds;

Integer priority;

Long startTime; //epoch in milliseconds

Long endTime = new Long(-1); //epoch in milliseconds

SemaphoreType semaphoreType = SemaphoreType.UNLIMITED;

String semaphoreConfig;

WorkOrderExecutionState status;

}

**Type description:** Each time a Workflow is executed, the execution is identified and tracked by a matching WorkOrder.

## WorkOrderCancellation

Syntax:

type WorkOrderCancellation(@package=rapture.common.dp) {

WorkOrderURI workOrderURI;

Long time;

}

**Type description:** A request to cancel a work order

## WorkOrderDebug

Syntax:

type WorkOrderDebug(@package=rapture.common.dp) {

WorkOrder order;

List<Worker> workers;

ExecutionContext context;

}

**Type description:** Contains extra information about a work order, useful mainly for debugging.

## WorkOrderSearch

Syntax:

type WorkOrderSearch(@package=rapture.common.dp) {

WorkerExecutionState status;

List<WorkOrderURI> workOrderURIs;

Long startTimeBegin; //in milliseconds

Long startTimeEnd; //in milliseconds

Long endTimeBegin; //in milliseconds

Long endTimeEnd; //in milliseconds

}

**Type description:** Used to store values for searching work orders

## WorkOrderStatus

Syntax:

type WorkOrderStatus(@package=rapture.common.dp) {

List<String> workerIds;

WorkOrderExecutionState status; // ACTIVE, CANCELLING, CANCELLED, FINISHED, ERROR

}

**Type description:** The status of a work order.

## XferDocumentAttribute

Syntax:

type XferDocumentAttribute(@package=rapture.common) {

String attributeType;

String key;

String value;

}

**Type description:** Holds a key/value pair for an attribute of a document object.

## XferSeriesValue

Syntax:

type XferSeriesValue(@package=rapture.common) {

String column;

String value;

}

**Type description:** This object holds the data for a series value that was requested, without specifying what type to present data as.

|  |
| --- |
|  |
|  |  |
|  |  |

# Admin

The Admin API is used to manipulate and access the low level entities in Rapture. Typically the methods in this API are only used during significant setup events in a Rapture environment.

## addIPToWhiteList

**Syntax:** Boolean addIPToWhiteList(String ipAddress)

**Parameters:** ipAddress: an IP address

**Returns:** true or exception

**Method description:** Use this method to add an IP address to a white list of allowed IP addresses that can log in to this Rapture environment. Once a white list is set, only IP addresses in this list can access Rapture. By default there are no whitelist IP addresses defined, so all IP addresses are allowed.

## addMetadata

**Syntax:** Boolean addMetadata(Map<String, String> values, Boolean overwrite)

**Method description:** This function adds values to the metadata field of the CallingContext. It's used to hold values specific to this connection. Because metadata is set by the caller, the values cannot be considered entirely trustworthy, so private or secure data such as passwords shouldn't be stored here. If overwrite is false and an entry already exists then an exception should be thrown.

## addRemote

**Syntax:** RaptureRemote addRemote(String name, String description, String url, String apiKey, String optP)

**Parameters:** name: the local name for the remote system

description: free text describing the remote system

url: an http address for the remote system

ApiKey: a token to use for connecting to the remote

optP: a password to use for connecting to the remote system.

**Returns:** an object describing the remote

## addTemplate

**Syntax:** Boolean addTemplate(String name, String template, Boolean overwrite)

**Parameters:** name: the name of the template

template: the text of the template

overwrite: call will fail if template exists unless this is set to true

**Returns:** true or exception

**Method description:** This function adds a template to the Rapture system. A template is a simple way of registering predefined configs that can be used to automatically generate configs for repositories, queues, and the like. Templates use the popular StringTemplate library for merging values into a text template.

## addUser

**Syntax:** Boolean addUser(String userName, String description, String hashPassword, String email)

**Parameters:** user: the name of the account to create

description: free text to associate with the account

hashPassword: an MD5 hash of the password

email: an email address to associate with the account

**Returns:** true or exception

**Method description:** This method adds a user to the Rapture environment. The user will be in no entitlement groups by default.

## clearRemote

**Syntax:** clearRemote(String raptureURI)

**Method description:** This method reverses a previously defined association between two repositories.

## copyDocumentRepo

**Syntax:** Boolean copyDocumentRepo(String srcAuthority, String targAuthority, Boolean wipe)

**Parameters:** srcAuthority: the first component of the source repo URI without slashes

targAuthority: the first component of the target repo URI without slashes

wipe: a flag to clear content from the target before propagating data

**Returns:** true or exception

**Method description:** Copies the data from one DocumentRepo to another. The target repository is wiped out beforehand if 'wipe' is set to true. The target must already exist when this method is called.

## deleteUser

**Syntax:** boolean deleteUser(String userName)

**Parameters:** the name of the user account to be disabled.

**Returns:** true or exception

**Method description:** This method removes a user from this Rapture system. The user is removed from all entitlement groups also. The actual user definition is retained and marked as inactive (so the user cannot login), because the user may still be referenced in audit trails and the change history in type repositories.

## destroyUser

**Parameters:** the name of the user account to be destroyed.

**Returns:** true or exception

**Method description:** This method destroys a user record. The user must have been previously disabled using 'deleteUser' before this method may be called. This is a severe method that should only be used in non-production machines or to correct an administrative error in creating an account with the wrong name before that account has been used. References to the missing user may still exist, and may not display properly in some UIs.

## doesUserExist

**Syntax:** Boolean doesUserExist(String userName)

**Parameters:** user: the name of the account to examine

**Returns:** whether the user account exists

**Method description:** This API call can be used to determine whether a given user exists in the Rapture system. Only system administrators can use this API call.

## generateApiUser

**Syntax:** RaptureUser generateApiUser(String prefix, String description)

**Parameters:** prefix: some human-readable characters to make the API key easier to identify

description: free text to associate with the API key

**Returns:** The complete API key

**Method description:** Generates an API user, for use in connecting to Rapture in a relatively opaque way using a shared secret. An API can be used in place of a normal username to log in without a password.

## getAllUsers

**Syntax:** List(RaptureUser) getAllUsers()

**Returns:** a list of objects describing user accounts

**Method description:** This method retrieves all of the registered users in the system, including those whose accounts are disabled.

## getEnvironmentName

**Syntax:** String getEnvironmentName()

**Returns:** the cluster name

**Method description:** Returns a name previously associated with the cluster

## getEnvironmentProperties

Syntax: Map<String, String> getEnvironmentProperties()

**Returns:** property map

**Method description:** Returns a map of properties specifying how the current cluster is displayed in the UI.

## getIPWhiteList

**Syntax:** List(String) getIPWhiteList()

**Returns:** A list of IP addresses

**Method description:** Use this method to return the IP white list

## getMOTD

**Syntax:** String getMOTD()

**Returns:** text of the MOTD

**Method description:** Retrieves the message of the day.

## getRemotes

**Syntax:** List(RaptureRemote) getRemotes()

**Returns:** objects describing all the remotes in a given cluster

**Method description:** Remotes are Rapure systems outside of the current cluster. The cluster communicates with them through API keys.

**Method description:** Remotes are used to connect one Rapture cloud environment to another. Each remote manages a connection to another Rapture cloud through the http API. This method adds a new remote to this system.

## getRepoConfig

**Syntax:** List(RepoConfig) getRepoConfig()

**Method description:** This method shows the configuration of all data stores without type-specific information. In practice, users generally prefer to get more specific information based on the type of store (by calling methods such as getBlobRepoConfig, getSeriesRepoConfig, etc.)

## getSessionsForUser

**Syntax:** List(CallingContext) getSessionsForUser(String user)

**Parameters:** user: The user’s account

**Returns:** The CallingContext contains session information passed to normal API calls from a given session.

**Method description:** When a user logs into Rapture they create a transient session. This method is a way of retrieving all of the sessions for a given user.

## getSystemProperties

**Syntax:** Map(String, String) getSystemProperties(List(String) keys)

**Parameters:** keys: A list of property names.

**Returns:** A map from the key names to the property values.

**Method description:** This function retrieves the system properties in use for this instance of Rapture. Because system properties are often used to control external connectivity, a client can determine the inferred connectivity endpoints by using this API call. Each API endpoint can have different properties. This method retrieves the properties for the specific API endpoint from which the request was made. The system properties cannot be modified through the API; the administrator sets them as part of the general setup of the Rapture system.

## getTemplate

**Syntax:** String getTemplate(String name)

**Parameters:** name: name of the template

**Returns:** the template definition without substitutions

**Method description:** This method returns the definition of a template.

## getUser

**Syntax:** RaptureUser getUser(String userName)

**Parameters:** user: the name of the account to retrieve

**Returns:** An object describing the user’s account

**Method description:** Retrieves a single user given the user’s name.

## ping

**Syntax:** Boolean ping()

**Method description:** A general-purpose function that tests (or refreshes) the API connection to Rapture with no side effects.

## pullRemote

**Syntax:** Boolean pullRemote(String raptureURI)

**Method description:** If this type has a remote defined, use it to sync this repository with that of the other.

## removeIPFromWhiteList

**Syntax:** Boolean removeIPFromWhiteList(String ipAddress)

**Parameters:** ipAddress: an IP address

**Returns:** true or exception

**Method description:** Use this method to remove an IP address from a white list

## removeRemote

**Syntax:** Boolean removeRemote(String name)

**Parameters:** name: the local name of the remote to remove

**Method description:** This method removes a previously created remote.

## resetUserPassword

**Syntax:** Boolean resetUserPassword(String userName, String newHashPassword)

**Parameters:** username: the account to modify

newhashPassword: an MD5 hash of the new password.

**Returns:** true or exception

**Method description:** This method gives an administrator the ability to reset the password of a user. The user will have the new password in the parameter set.

## restoreUser

**Parameters:** the name of the user account to be restored.

**Returns:** true or exception

**Method description:** This method restores a user that has been deleted.

## retrieveArchiveConfig

**Syntax:** TypeArchiveConfig retrieveArchiveConfig(String raptureURI)

**Method description:** Retrieve the archive config for an authority

## runBatchScript

**Syntax:** String runBatchScript(String script)

**Parameters:** script: the text of the script

**Returns:** the return value of the script

**Method description:** This method runs a batch script at the target site

## runTemplate

**Syntax:** String runTemplate(String name, String parameters)

**Parameters:** name: name of the template

parameters: parameters to be passed into the template

**Returns:** The result of substituting parameters in the template

**Method description:** This method executes a template, replacing parts of the template with the passed parameters to create a new string.

## setEnvironmentName

**Syntax:** Boolean setEnvironmentName(String name)

**Parameters:** name: the cluster name

**Returns:** true or exception

**Method description:** Associates a human-readable name with the current cluster (for example, ‘Testing’ or ‘Production’).

## setEnvironmentProperties

**Syntax:** Boolean setEnvironmentProperties(Map<String, String> properties)

**Parameters:** properties

**Returns:** true or exception

**Method description:** Set the properties of this environment. Usually for displaying the name (e.g. BANNER\_COLOR)

## setMOTD

**Syntax:** Boolean setMOTD(String message)

**Parameters:** text of the MOTD, or an empty string if there is no message

**Returns:** true or exception

**Method description:** Set the MOTD (message of the day) for this environment. Setting to a zero length string implies that there is no message of the day

## setRemote

**Syntax:** Boolean setRemote(String raptureURI, String remote, String remoteURI)

**Method description:** Once a remote has been defined, it can be used to synchronize one repository with another. This method defines how one (local) repository is connected to a remote type.

The binding is made between the combination of an authority, repository, and perspective from one system to another. The previously defined remote handles the synchronization tasks.

## storeArchiveConfig

**Syntax:** Boolean storeArchiveConfig(String raptureURI, TypeArchiveConfig config)

**Method description:** Sets the archive config for a type

## updateRemoteApiKey

**Syntax:** Boolean updateRemoteApiKey(String name, String apiKey)

**Parameters:** name: the local name of the remote to modify.

apiKey: the new token to use for connecting to the remote system

**Returns:**

**Method description:** This method updates the user API key used by a given remote. This API key is for a remote system, not the Rapture system from which this API call originates.

## updateUserEmail

**Syntax:** Boolean updateUserEmail(String userName, String newEmail)

**Parameters:** username: the name of the user account to modify

newEmail: The email address to associate with this account.

**Returns:** true or exception

**Method description:** Changes the address to which mail for this Rapture account is sent, replacing the old email address for this account.

# Async

The Async API works closely with the Pipeline API, putting tasks onto the Pipeline for future execution. The most often used call is one where a Reflex script is scheduled for execution, using one of the asyncReflex\* calls.

## asyncBatchSave

**Syntax:** List(String) asyncBatchSave(List(String) displayNames, List(String) contents)

**Method description:** Batch save - put a whole series of saves on the pipeline queue so that the saves are performed in an asynchronous manner.

## asyncExecuteRemote

Syntax: String asyncExecuteRemote(String serverGroup, String appName, String parameters)

**Method description:** Schedule (for immediate execution) a request to the RaptureRunner environment to launch a one-shot application that performs a task.

The call will create a temporary ApplicationInstance to run, and the state of the execution can be monitored through the normal asyncStatus API call. Return value is the Task ID.

## asyncOperation

**Syntax:** String asyncOperation(String typeURI, String operationName, String ctx, String parameters)

**Method description:** Runs an operation in an asynchronous manner. Operations are attached to types and are named. The context parameter is the context to the authority, and in this case (to be corrected) the parameters are passed as a string instead of a map.

## asyncReflexReference

**Syntax:** String asyncReflexReference(String scriptURI, Map(String, String) parameters)

**Method description:** Runs a script that has already been loaded onto Rapture in an asynchronous manner. The script is named through its authority and title. As with asyncReflexScript, the parameters in the parameter map are passed to the script upon invocation, and the return value from this function is a handle that can be used to determine the ultimate status of this task.

## asyncReflexScript

**Syntax:** String asyncReflexScript(String reflexScript, Map(String, String) parameters)

**Method description:** Runs a passed script in an asynchronous manner - returning a unique reference (a handle) that can be used in other calls to retrieve the status of a job. The return value is known as a task id. The second parameter is the set of parameters that are passed to the script on execution.

## asyncSave

**Syntax:** String asyncSave(String displayName, String content)

**Method description:** Saves a document in an asynchronous manner. This is effectively deferring the save of a document.

## asyncStatus

**Syntax:** PipelineTaskStatus asyncStatus(String taskId)

**Method description:** Retrieves the status of a given task. Will return null if the task id is not known to the environment.

## executeFolderQueryScript

**Syntax:** Long executeFolderQueryScript(String raptureURI, int depth, String scriptRef)

**Method description:** Runs a folderQuery, but with each value returned with the folderQuery execute a script asynchronously. Returns the number of tasks scheduled.

# Audit

The Audit API provides a way to create special logs that contain permanent records of activity in a Rapture system. Internally Rapture uses a system audit log for recording important events that take place in a Rapture environment. Users (or applications) can create their own custom audit logs for the same purpose.

The API provides a way of creating and removing these logs, along with a simple way of recording log entries. A final API call gives the caller the ability to retrieve log entries.

## createAuditLog

**Syntax:** Boolean createAuditLog(String name, String config)

**Parameters:** name: the name of the audit

config: a string containing the configuration syntax for this audit log

**Returns:** true or exception

**Method description:** This method creates a new audit log, given a name and a config string. The config string defines the implementation to be used to store the audit entries.

## deleteAuditLog

**Syntax:** Boolean deleteAuditLog(String logURI)

**Parameters:** the URI of the log to be deleted

**Returns:** true or exception

**Method description:** Removes a previously created audit log.

## doesAuditLogExist

**Syntax:** Boolean doesAuditLogExist(String logURI)

**Parameters:** logURI: the URI of the log to be looked up.

**Returns:** whether the log exists at this URI.

**Method description:** This method checks whether an audit log exists at the specified URI. The log must have been created using createAuditLog.

## getAuditLog

**Syntax:** AuditLogConfig getAuditLog(String logURI)

**Parameters:** the URI of the log to retrieve

**Returns:** Config data for the log

**Method description:** Retrieves the config information for an audit log.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String prefix)

**Parameters:** a prefix to a log name that can be used for searching

**Returns:** a list of RaptureFolderInfo objects with all the children.

**Method description:** This method searches for audit logs whose name follows the pattern “prefix/anything\_else/under/here”, where “prefix” is the argument that is passed in.

## getEntriesSince

**Syntax:** List(AuditLogEntry) getEntriesSince(String logURI, AuditLogEntry when)

**Parameters**: logURI: the uri of the log whose entries we want to retrieve

when: an AuditLogEntry whose “when” field, which specifies the Date, has been set to the earliest date from which we want to retrieve logs

**Returns:** list of the log entries retrieved.

**Method description:** This method retrieves any entries since a given entry was retrieved. The date of this audit entry is used to determine the start point of the query.

## getRecentLogEntries

**Syntax:** List(AuditLogEntry) getRecentLogEntries(String logURI, int count)

**Parameters:** logURI: the uri of the log whose entries we want to retrieve

count: the maximum number of entries to return

**Returns:** list of the log entries retrieved.

**Method description:** This method retrieves previously registered log entries, given a maximum number of entries to return.

## setup

**Syntax:** Boolean setup(Boolean force)

**Parameters:** force: whether the setup should run if it has already run in the past. Some of the operations in this method should be run only the first time Rapture ever starts. Setting this parameter to true will override previous definitions.

**Returns:** true if the setup steps ran, false if setup ran earlier

**Method description:** Sets up anything needed for audit to run properly. This method should be called from the \_startup.rfx script. This call is used internally by Rapture on startup, and is normally called only for debugging purposes.

## writeAuditEntry

**Syntax:** Boolean writeAuditEntry(String logURI, String category, int level, String message)

**Parameters:** logURI: the uri of the log where this should write

category: the category of the log (e.g. ERROR, INFO, etc)

level: the log level, e.g. 1, 2, 3. Another way of slicing the priority of this log, in addition to category.

message: the message to be written

**Returns:** true or exception

**Method description:** Writes an audit entry to the log specified by the URI.

# Blob

The Blob API is used to manipulate large opaque objects that have names (displaynames) like other data but do not have any insight to be gained from their contents from within Rapture. The RESTful API can be used to efficiently download or upload a blob as a stream.

## appendToBlob

**Syntax:** Boolean appendToBlob(String blobURI, ByteArray content)

**Returns:** true or exception

**Method description:** Appends to a blob created with createBlob.

## createBlob

**Syntax:** Boolean createBlob(String blobURI, String contentType)

**Parameters:** blobURI: the URI where the blob is to be created

contentType: the MIME type of the stored data, eg “text/plain”

**Returns:** true or exception

**Method description:** Creates a blank blob that is ready to be appended to. If the blob already exists, this call will not affect it.

## createBlobRepo

**Syntax:** Boolean createBlobRepo(String blobRepoURI, String config, String metaConfig)

**Parameters:** blobRepoURI: Identifies the blob repository. The expected format is “blob://name”

config: defines the Repository Configuration.   
The expected format is “BLOB { } USING *STORE* { }” where *STORE* must be one of the supported backing stores, such as MEMORY or MONGO.

metaConfig: Unlike other repositories, the Blob Repository takes a second configuration argument. The second argument is used to store the blob metadata, such as the MIME type. The expected format is “REP { } USING *STORE* { }” where *STORE* must be one of the supported backing stores.

**Returns:** true or exception

**Method description:** Creates a repository for unstructured data

## deleteBlob

**Syntax:** Boolean deleteBlob(String blobURI)

**Returns:** true or exception

**Method description:** Removes a blob from the backing store. There is no undo.

## deleteBlobRepo

**Syntax:** Boolean deleteBlobRepo(String repoURI)

**Parameters:** the URI identifying the blob repository

**Returns:** true or exception

**Method description:** This method removes a blob Repository and its data from the Rapture system. There is no undo.

## doesBlobRepoExist

**Syntax:** Boolean doesBlobRepoExist(String repoURI)

**Parameters:** the URI identifying the blob repository to be examined

**Returns:** whether the repository was found at the given URI.

**Method description:** This API call can be used to determine whether a specified repository exists.

## getAllBlobRepoConfigs

**Syntax:** List(BlobRepoConfig) getAllBlobRepoConfigs()

**Returns:** a list of repository config objects

**Method description:** Retrieves a collection of objects that contain the configuration information for all the defined blob repositories.

## getAllChildrenMap

**Syntax:** Map(String,RaptureFolderInfo) getAllChildrenMap(String blobURI)

**Method description:** Returns full pathnames for an entire subtree as a map of the path to RFI.

## getBlob

**Syntax:** BlobContainer getBlob(String blobURI)

**Method description:** Retrieves a blob in and its metadata. The blob is represented as a byte array.

## getBlobRepoConfig

**Syntax:** BlobRepoConfig getBlobRepoConfig(String blobRepoURI)

**Parameters:** the URI identifying the blob repository

**Returns:** A repository config object for the blob

**Method description:** Retrieves an object that contains repository definition information for a specific blob repository.

## getBlobSize

**Syntax:** Long getBlobSize(String blobURI)

**Method description:** Retrieves the number of bytes in a blob.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String blobURI)

**Method description:** Return a list of full display names of the paths below this one. Ideally optimized depending on the repo.

## getMetaData

**Syntax:** Map(String,String) getMetaData(String blobURI)

**Method description:** Retrieves all metadata associated with a blob.

## storeBlob

**Syntax:** Boolean storeBlob(String blobURI, ByteArray content, String contentType)

**Returns:** true or exception

**Method description:** Combines the functionality of createBlob and appendToBlob. If a blob already exists at the URI, it is overwritten.

# Bootstrap

The Bootstrap API is used to setup an initial Rapture environment and to migrate existing bootstrap repositories to a new repository format.

## addScriptClass

**Syntax:** Boolean addScriptClass(String keyword, String className)

**Method description:** All scripts that are run by Rapture are passed a set of helper instances that can be used by the script. The helpers are locked to the entitlement context of the calling user. This method sets the name of such a class in this context. It is primarily an internal function, defined during startup, as the class provided must be accessible by the main Rapture application.

## getConfigRepo

**Syntax:** String getConfigRepo()

**Method description:** Retrieves the current settings of the config repository.

## getEphemeralRepo

**Syntax:** String getEphemeralRepo()

**Method description:** Retrieves the settings of the ephemeral repository.

## getScriptClasses

**Syntax:** Map(String, String) getScriptClasses()

**Method description:** This method retrieves all previously defined script classes for this system

## getSettingsRepo

**Syntax:** String getSettingsRepo()

**Method description:** Retrieves the settings of of the setting repository.

## migrateConfigRepo

**Syntax:** Boolean migrateConfigRepo(String newConfig)

**Parameters:**

**Returns:**

**Method description:** This method is used to migrate the top level Config repository to a new config. This task takes place in the background, and once completed the config repository is switched to the new config. Any changes to config up to this point may be lost.

## migrateEphemeralRepo

**Syntax:** Boolean migrateEphemeralRepo(String newConfig)

**Method description:** This method is used to migrate the top level Ephemeral repository to a new config. This task takes place in the background, and once completed the config repository is switched to the new config. Any changes to config up to this point may be lost.

## migrateSettingsRepo

**Syntax:** Boolean migrateSettingsRepo(String newConfig)

**Method description:** This method is used to migrate the top level Settings repository to a new config. This task takes place in the background, and once completed the config repository is switched to the new config. Any changes to config up to this point may be lost.

## restartBootstrap

**Syntax:** Boolean restartBootstrap()

**Method description:** After changing the definition of any bootstrap repository, Rapture will need to be restarted. This method will restart Rapture.

## removeScriptClass

**Syntax:** Boolean removeScriptClass(String keyword)

**Method description:** This method removes a previously defined script class.

## setConfigRepo

**Syntax:** Boolean setConfigRepo(String config)

**Method description:** The config repository is used to store general config information about entities in Rapture. These entities include users, types, indices, queues and the like.

## setEphemeralRepo

**Syntax:** Boolean setEmphemeralRepo(String config)

**Method description:** The ephemeral repository is used to store information that does not need to survive a restart of Rapture. It normally holds information such as sessions, and its config is usually based around a shared non-versioned memory model.

## setSettingsRepo

**Syntax:** Boolean setSettingsRepo(String config)

**Method description:** The settings repository is used to store general low-level settings in Rapture.

# Decision

The Decision Process API is used to manage and control decision processes and decision packets.

## addErrorToContext

**Syntax:** Boolean addErrorToContext(String workerURI, ErrorWrapper errorWrapper)

**Returns:** true or exception

**Method description:** Adds an error to the context of a particular worker. The workerURI is a workOrderURI with the element set to the worker id.

## addStep

**Syntax:** Boolean addStep(String workflowURI, Step step)

**Returns:** true or exception

**Method description:** Adds a new step to an existing workflow initially containing the specified transitions.

## addTransition

**Syntax:** Boolean addTransition(String workflowURI, String stepName, Transition transition)

**Returns:** true or exception

**Method description:** Adds a new Transition to a workflow.

## cancelWorkOrder

**Syntax:** Boolean cancelWorkOrder(String workOrderURI)

**Returns:** true or exception

**Method description:** Requests cancellation of a work order. This method returns immediately once the cancellation is recorded, but the individual workers may continue for some time before stopping, depending on the type of step being executed.

## createWorkOrder

**Syntax:** String createWorkOrder(String workflowURI, Map<String, String> contextMap)

**Parameters:** workflowURI: the URI of the workflow

contextMap: a mapping of context variable names to their values

**Returns:** the URI of the WorkOrder created.

**Method description:** Executes a workflow. If there is a defaultAppStatusUriPattern set for this Workflow, then it will be used for the appstatus URI. Otherwise, no appstatus will be created.

## createWorkOrderP

**Syntax:** CreateResponse createWorkOrderP(String workflowURI, Map<String, String> contextMap, String appStatusUriPattern)

**Parameters:** workflowURI: the URI of the workflow

contextMap: a mapping of context variable names to their values

appStatusUriPattern: a template pattern to be used for the URI. It can use variables defined in the contextMap, or in the view overlay of this workflow. For example %//auth/${someVar}.

**Returns:** the URI of the WorkOrder created.

**Method description:** Creates and executes a workflow. Same as [createWorkOrder](#_createWorkOrder_1), but the appStatusUriPattern is passed as an explicit argument instead of using the default appStatusUriPattern (if one has been set).

Note that the app status allows the Workflow and its output to be accessed via the web interface; workflows without an app status are not accessible in this way.

## defineWorkflow

**Syntax:** Boolean defineWorkflow(Workflow workflow)

**Parameters:** a workflow definition

**Returns:** true or exception

**Method description:** Creates a workflow initially containing the specified nodes and transitions. If a workflow with the passed in URI already exists, an exception is thrown.

## deleteWorkflow

Syntax: Boolean deleteWorkflow(String workflowURI)

**Parameters:**

**Returns:** true or exception

**Method description:** Deletes a workflow.

## getAllWorkflows

**Syntax:** List<Workflow> getAllWorkflows()

**Returns:** a list of workflow objects

**Method description:** Returns all workflow definitions.

## getAppStatuses

**Syntax:** List<AppStatus> getAppStatuses(String prefix)

**Method description:** Gets app statuses by prefix.

## getAppStatusDetails

Syntax: List<AppStatusDetails> getAppStatusDetails(String prefix, List<String> extraContextValues)

**Method description:** Gets detailed app status info by prefix. Also returns any context values requested in the second argument.

## getCancellationDetails

**Syntax:** WorkOrderCancellation getCancellationDetails(String workOrderURI)

**Method description:** gets details for the cancellation for a workOrder -- or null if not cancelled.

## getContextValue

**Syntax:** String getContextValue(String workerURI, String varAlias)

**Method description:** Gets a value in the context, as json. The workerURI is a workOrderURI with the element set to the worker id.

## getErrorsFromContext

**Syntax:** List<ErrorWrapper> getErrorsFromContext(String workerURI)

**Method description:** Gets the errors from the context for a given worker. The workerURI is a workOrderURI with the element set to the worker id.

## getStepCategory

**Syntax:** String getStepCategory(String stepURI)

**Parameters:** a workflowURI qualified with a step name

**Returns:** the category name

**Method description:** Gets the category associated with a step. This is the step's own categoryOverride, if present, or otherwise the category associated with the entire workflow.

## getWorker

**Syntax:** Worker getWorker(String workOrderURI, String workerId)

**Method description:** Gets the status object associated with a single worker.

## getWorkflow

**Syntax:** Workflow getWorkflow(String workflowURI)

**Parameters:** a URI from which to retrieve the workflow

**Returns:** the workflow definition, or null if not found

**Method description:** Returns a workflow definition, or null if not found.

## getWorkflowChildren

**Syntax:** List(RaptureFolderInfo) getWorkflowChildren(String workflowURI)

**Parameters:** the URI of the parent folder

**Returns:** directory entries for the children of the URI.

**Method description:** Returns a list of full display names of the paths below this one. Ideally optimized depending on the repo.

## getWorkflowStep

**Syntax:** Step getWorkflowStep(String stepURI)

**Parameters:** a workflowURI qualified with a step name

**Returns:** the step definition, or null if not found.

**Method description:** Returns a step definition, or null if not found.

**Example:**

#decision.getWorkflowStep(**“//myProj/myWorkflow#myStep”)**

## getWorkOrder

**Syntax:** WorkOrder getWorkOrder(String workOrderURI)

**Method description:** Gets the top-level status object associated with the work order.

## getWorkOrderChildren

**Syntax:** List(RaptureFolderInfo) getWorkOrderChildren(String parentPath)

**Parameters:** the URI of the parent folder

**Returns:** directory entries for the children of the URI.

**Method description:** Return a list of full display names of the paths below this one. Ideally optimized depending on the repo.

## getWorkOrderDebug

**Syntax:** WorkOrderDebug getWorkOrderDebug(String workOrderURI)

**Method description:** Gets the detailed context information for a work order in progress

## getWorkOrdersByDay

**Syntax:** List<WorkOrder> getWorkOrdersByDay(Long startTimeInstant)

**Parameters:** startTimeInstant: any instant in the target day as determined by UTC time zone

**Returns:** Objects containing work orders that started on the target day. Orders that carried over from the previous day are not included.

**Method description:** Gets the WorkOrder objects starting on a given day.

## getWorkOrderStatus

**Syntax:** WorkOrderStatus getWorkOrderStatus(String workOrderURI)

**Parameters:** the URI of the WorkOrder.

**Method description:** Gets the status of a workOrder.

## putWorkflow

**Syntax:** Boolean putWorkflow(Workflow workflow)

**Parameters:** a workflow definition

**Returns:** true or exception

**Method description:** Create or update a workflow to contain only the specified nodes and transitions.

## releaseWorkOrderLock

**Syntax:** Boolean releaseWorkOrderLock(String workOrderURI)

**Parameters:** the URI of the WorkOrder.

**Returns:** true or exception

**Method description:** Releases the lock associated with this WorkOrder. This method should only be used by admins, in case there was an unexpected problem that caused a WorkOrder to finish or die without releasing the lock.

## removeStep

**Syntax:** Boolean removeStep(String workflowURI, String stepName)

**Returns:** true or exception

**Method description:** Removes a step from a workflow.

## removeTransition

Syntax: Boolean removeTransition(String workflowURI, String stepName, String transitionName)

**Returns:** true or exception

**Method description:** Removes a transition from a workflow.

## setContextLiteral

**Syntax:** Boolean setContextLiteral(String workerURI, String varAlias, String literalValue)

**Method description:** Sets a literal in the context. Whatever is stored will be returned literally during a read. The workerURI is a workOrderURI with the element set to the worker ID.

## setContextLink

**Syntax:** Boolean setContextLink(String workerURI, String varAlias, String expressionValue)

## setWorkOrderFountainConfig

**Syntax:** Boolean setWorkOrderFountainConfig(String config, Boolean force)

**Method description:** Defines the fountain config for work order items.

## wasCancelCalled

**Syntax:** Boolean wasCancelCalled(String workOrderURI)

**Method description:** Returns true if cancelWorkOrder was called.

## writeWorkflowAuditEntry

**Syntax:** Boolean writeWorkflowAuditEntry(String workOrderURI, String message, Boolean error)

**Parameters:** workOrderURI: the URI of the work order

message: human-readable text

error: flag to show if the message represents an error

**Returns:** true or exception

**Method description:** Writes an audit entry related to a workOrder. Messages may be INFO or ERROR based on the boolean fourth parameter.

# Document

The Document API is used for manipulating document objects in Rapture. Documents are normally stored in JSON format.

## addDocumentAttribute

**Syntax:** String addDocumentAttribute(String attributeURI, String value)

**Parameters:** attributeURI: the location of the attribute

value: the attribute’s contents

**Returns:** null on success, or exception (not the same as for addDocumentAttributes)

**Method description:** Adds a single attribute to an existing document.

## addDocumentAttributes

**Syntax:** String addDocumentAttributes(String attributeURI, List(String) keys, List(String) values)

**Parameters:** attributeURI: the location of the attribute

keys: a list of identifiers for each attribute

values: a list of the attributes’ contents

**Returns:** the supplied URI, or an exception

**Method description:** Adds multiple attributes in key/value pairs to an existing document.

## archiveVersions

**Syntax:** Boolean archiveVersions(String repoURI, int versionLimit, long timeLimit, Boolean ensureVersionLimit)

**Method description:** Archives older versions of a repository.

## attachFountainToDocumentRepo

**Syntax:** DocumentRepoConfig attachFountainToDocumentRepo(String documentRepoURI, String fountainConfig)

**Method description:** This method creates a fountain and attaches it to a document repository. This way, when a document containing an autoid string is created that autoid will be replaced with a unique id.

## batchGet

**Syntax:** List(String) batchGet(List(String) docURIs)

**Method description:** Returns a list of contents (null for those that do not exist) given a list of display names. Note that ordering is not guaranteed.

## batchExist

**Syntax:** List(Boolean) batchExist(List(String) docURIs)

**Parameters:** docURIs: A list of URIs. Can be empty, but must not be null.

**Returns:** A list of boolean values corresponding to whether each URI exists.

**Method description:** This method sorts the input list and removes duplicates, so the size of the returned list may not match the size of docURIs. Also, the argument cannot be an immutable list.

## batchPutContent

**Syntax:** Boolean batchPutContent(List(String) docURIs, List(String) contents)

**Parameters:** The lists of display names and contents must be of the same size; they can be empty but may not be null.

**Returns:** true or exception

**Method description:** Puts a series of documents in a batch form. Refer to <putContent> for details.

## batchRenameContent

**Syntax:** Boolean batchRenameContent(String authority, String comment, List(String) fromDocURIs, List(String) toDocURIs)

**Parameters:** The lists of display names and contents must be of the same size; they can be empty but may not be null.

**Returns:** true or exception

**Method description**: Renames a series of documents in a batch form. See [renameContent](#_renameContent_1).

## createDocumentRepo

**Syntax:** Boolean createDocumentRepo(String raptureURI, String config)

**Returns:** true or exception

**Method description:** A DocumentRepository is used to store JSON docs. This method creates and configures the repository for an authority.

## deleteContent

**Syntax:** Boolean deleteContent(String docURI)

**Returns:** true or exception

**Method description:** Removes a document from the system.

## deleteDocumentRepo

**Syntax:** Boolean deleteDocumentRepo(String repoURI)

Parameters:

**Returns:** true or exception

**Method description:** This method removes a documentRepository and its data from the Rapture system. There is no undo.

## doesDocumentRepoExist

**Syntax:** Boolean doesDocumentRepoExist(String raptureURI)

**Returns:** true or false

**Method description:** This API call can be used to determine whether a given type exists in a given authority.

## folderQuery

**Syntax:** List(String) folderQuery(String docURI, int depth)

**Parameters:** Starting point, depth limit

**Returns:** A list of folder

**Method description:** Locates all folders under a named location, down to the specified depth.

## getAllChildrenMap

**Syntax:** Map<String, RaptureFolderInfo> getAllChildrenMap(String docURI)

**Parameters:** Starting folder

**Returns:** URIs of all documents and folders below the starting point.

## getAllDocumentRepoConfigs

**Syntax:** List(DocumentRepoConfig) getAllDocumentRepoConfigs()

**Returns:** List of document repository configuration strings. The order is unspecified.

## getAttachedFountain

**Syntax:** RaptureFountainConfig getAttachedFountain(String documentRepoURI)

**Parameters:** Location of the document repository

**Returns:** Any fountain configuration associated with this repository, or null if there isn't one.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String docURI)

**Parameters:** Starting folder

**Returns:** A list of full display names of the paths below this one. Ideally optimized depending on the repo.

## getContent

**Syntax:** String getContent(String docURI)

**Parameters:** Location of the document repository

**Returns:** The content of a document as a string.

## getDocumentAttribute

**Syntax:** XferDocumentAttribute getDocumentAttribute(String attributeURI)

**Parameters:** A URI that specifies a document attribute

**Returns:** A single attribute for the given URI, attributeType, and key e.g. displayName/$attributeType/key

## getDocumentAttributes

**Syntax:** List(XferDocumentAttribute) getDocumentAttributes(String attributeURI)

**Parameters:** A URI that specifies a document attribute

**Returns:** A list of all known attributes for the given URI attributeType, and key e.g. displayName/$attributeType/key

## getdocumentRepoConfig

**Syntax:** DocumentRepoConfig getDocumentRepoConfig(String docRepoURI)

**Parameters:** Location of the document

**Returns:** The configuration string for the document repository for a given authority.

## getDocumentRepositoryStatus

**Syntax:** Map<String, String> getDocumentRepositoryStatus(String docRepoURI)

**Parameters:** Location of the document

**Returns:** Key/value pairs with any available information about a repository.

## getFountainURI

**Syntax:** String getFountainURI(String documentRepoURI)

**Returns:** The URI of the fountain that belongs to this document repository.

**Method description:** Note that every repository has a fountain URI, even if no fountain is attached to it.

## getMetaContent

**Syntax:** DocumentWithMeta getMetaContent(String docURI)

**Returns:** A structure containing both the document contents and the document’s metadata, including version and user information.

**Method description:** If the storage does not support metadata, this method returns a dummy object.

## getMetaData

**Syntax:** DocumentMetadata getMetaData(String docURI)

**Returns:** The metadata for a given document, including version and user information.

**Method description:** Does not retrieve the document content.

## putContent

**Syntax:** String putContent(String docURI, String content)

**Parameters:** docURI: URI specifying a document, or the attribute of a document.

value: String specifying document content or attribute value.

**Returns:** URI specifying the document, or the attribute of the document. Note that the return value may not be identical to the parameter.

**Method description:** Stores a document in the Rapture system.

## putContentWithVersion

**Syntax:** Boolean putContentWithVersion(String docURI, String content, int currentVersion)

**Method description:** Attempts to put the content into the repository, but fails if the repository supports versioning and the current version of the document stored does not match the version passed. A version of zero implies that the document should not exist. The purpose of this call is for a client to be able to call getMetaContentP to retrieve an existing document, modify it, and save the content back, using the version number in the metadata of the document. If another client has modified the data since it was loaded, this call will return false, indicating that the save was not possible.

## removeDocumentAttribute

**Syntax:** Boolean removeDocumentAttribute(String attributeURI)

**Parameters:** URI specifying an attribute of a document

**Returns:** true on success. On error may return false or throw an exception (depending on the repository implementation)

**Method description:** Removes a document attribute. Can be used to remove all attributes for a given type as well if the key argument is null.

## removeFolder

**Syntax:** List(String) removeFolder(String docURI, Boolean force)

**Method description:** Removes a folder and its contents; does not remove non-empty folders unless force is set to true. Returns what has been removed.

## renameContent

**Syntax:** Boolean renameContent(String fromDocURI, String toDocURI)

**Parameters:** fromDocURI: the old URI

toDocURI: the new URI

**Method description:** Renames a document by getting and putting it on the system without transferring the data back to the client.

## revertDocument

**Syntax:** DocumentWithMeta revertDocument(String docURI)

**Method description:** Reverts this document back to the previous version by taking the previous version and making a new version.

## validate

**Syntax:** Boolean validate(String raptureURI)

**Method description:** Validates repository; requires write permission because it can cause files/tables to be created on first use.

# Entitlement

Entitlements are a very important part of the security of Rapture, and the Entitlement API is the way in which information about these entitlements is updated. The API is of course protected by the same entitlements system, so care must be taken to not remove your own entitlement to this API through the use of this API.

### Concepts/Terminology

User - A user represents a person who is making calls to Rapture or an application that is making calls to Rapture. A user is a single entity with a username/password who needs access to Rapture.

Group - A group represents a collection of users.

Entitlement - An entitlement is a named permission that has associated with it 0 or more groups.  If an entitlement has no groups associated with it, it is essentially open and any defined user in Rapture can access it. If an entitlement has at least 1 group associated with it, any user wishing to access the resource protected by this entitlement, must be a member of one of the associated groups.

Each API call within Rapture is associated with an entitlement path, and when users wish to execute that API call they are checked to see if they are a member of that entitlement (by seeing which groups they are members of). Some API calls have dynamic entitlements, where the full name of the entitlement is derived from the URI of the object that the method uses. For example, a method that writes a document to a specific URI can use that URI as part of the entitlement.

If an entitlement with the specified name exists, then it is used; otherwise the full entitlement path is truncated one part at a time until an entitlement is found.

## addGroupToEntitlement

**Syntax:** RaptureEntitlement addGroupToEntitlement(String entitlementName, String groupName)

**Parameters:** the name of the entitlement and the group to add

**Returns:** the entitlement object just modified.

**Method description:** This method is used to add an entitlement group to an entitlement.

## addEntitlement

**Syntax:** RaptureEntitlement addEntitlement(String entitlementName, String initialGroup)

**Parameters:** entitlementName: The name of the entitlement to be added, e.g. “myapi/read”

initialGroup: The initial group that should be allowed to have this entitlement. This is not required, but it is recommended, otherwise you could end up with an entitlement that nobody can access.

**Returns:** the entitlement object that has just been added.

**Method description:** This method adds a new entitlement, specifying an initial group that should be assigned to this entitlement. The reason for assigning an initial group is to prevent lock out.

## addEntitlementGroup

**Syntax:** RaptureEntitlementGroup addEntitlementGroup(String groupName)

**Parameters:** name of the existing entitlement group to add

**Returns:** the entitlement group object just added

**Method description:** This method adds a new entitlement group to the system.

## addUserToEntitlementGroup

**Syntax:** RaptureEntitlementGroup addUserToEntitlementGroup(String groupName, String userName)

**Parameters:** the names of both the entitlement group and of the user to add to the group

**Returns:** the entitlement group object just modified

**Method description:** This method adds a user to an existing entitlement group. The user will then have all of the privileges (entitlements) associated with that group.

## deleteEntitlement

**Syntax:** Boolean deleteEntitlement(String entitlementName)

**Parameters:** the name of the entitlement to delete

**Returns:** true or exception

**Method description:** This method removes an entitlement entirely from the system.

## deleteEntitlementGroup

**Syntax:** Boolean deleteEntitlementGroup(String groupName)

**Parameters:** the name of the entitlement group to delete

**Returns:** true or exception

**Method description:** This method removes an entitlement group from the system.

## getEntitlement

**Syntax:** RaptureEntitlement getEntitlement(String entitlementName)

**Parameters:** the name of the entitlement to be retrieved.

**Returns:** the entitlement object, or null if not found.

**Method description:** Retrieves a single entitlement.

## getEntitlementByAddress

**Syntax:** RaptureEntitlement getEntitlementByAddress(String entitlementURI)

**Parameters:** the URI of the entitlement. Entitlement URIs use the “entitlement://” protocol, followed by the entitlement’s name, which can be in a path format (such as ‘user/read’).

**Returns:** the entitlement object, or null if not found.

**Method description:** Retrieves a single entitlement by using its URI.

## getEntitlementGroup

**Syntax:** RaptureEntitlement getEntitlementGroup(String groupName)

**Returns:** the entitlement group object, or null if not found.

**Method description:** Retrieves a single entitlement group.

## getEntitlementGroupByAddress

**Syntax:** RaptureEntitlementGroup getEntitlementGroupByAddress(String groupURI)

**Returns:** the entitlement group object, or null if not found.

**Method description:** Retrieves an entitlement group from its URI.

## getEntitlementGroups

**Syntax:** List(RaptureEntitlementGroup) getEntitlementGroups()

**Returns:** a list of all entitlement group objects

**Method description:** This method returns all of the entitlement groups defined in the Rapture environment.

## getEntitlements

**Syntax:** List(RaptureEntitlement) getEntitlements()

**Returns:** a list of entitlement objects

**Method description:** This method is used to retrieve all of the entitlements defined in Rapture.

## removeGroupFromEntitlement

**Syntax:** RaptureEntitlement removeGroupFromEntitlement(String entitlementName, String groupName)

**Parameters:** the name of the entitlement and the group to remove

**Returns:** the entitlement object just modified.

**Method description:** Removes the entitlement from the given group

## removeUserFromEntitlementGroup

**Syntax:** RaptureEntitlementGroup removeUserFromEntitlementGroup(String groupName, String userName)

**Parameters:** the names of both the entitlement group and of the user to add to the group

**Returns:** the entitlement group object just modified

**Method description**: Removes a user from an existing entitlement group

# Environment

A data center can contain multiple instances of Rapture – such as staging, production, or testing – that are nearly identical. In fact, a single piece of hardware can host multiple instances. In this case we cannot rely on host names or IP addresses to uniquely identify systems. The Environment API generates and assigns UIDs to the Rapture instance and to its component servers. Servers in the same network share the same bootstrap configuration and low level connectivity to data source configuration.

Although one topology for a Rapture network is a collection of homogeneous servers sharing the same data and messaging infrastructure, an alternate topology would be a collection of heterogeneous servers connected together through synchronization. A server in this topology is considered an “appliance” - it is essentially a Rapture network with one member. In this configuration the Rapture kernel needs to handle tasks that are normally performed by other servers. Appliance Mode can be enabled or disabled as necessary.

## getApplianceMode

**Syntax:** Boolean getApplianceMode()

**Returns:** Whether the instance is in appliance mode

## getLicenseInfo

**Syntax:** LicenseInfo getLicenseInfo()

**Returns:** An object with information about the current Rapture license.

**Method description:** License information includes the name of the licensee, expiration date, and whether the license is a developer license.

## getNetworkInfo

**Syntax:** RaptureNetwork getNetworkInfo()

**Returns:** The unique identifier and name for this Rapture instance.

## getServers

**Syntax:** List(RaptureServerInfo) getServers()

**Returns:** A list of the unique identifier and name for all Rapture servers in the network.

## getServerStatus

**Syntax:** List(RaptureServerStatus) getServerStatus()

**Returns:** A list of status objects.

**Method description:** For each of the servers in the network, gets the last reported state. This includes a numerical status, a human readable message, and a Date object indicating the time that the status was last updated.

## getThisServer

**Syntax:** RaptureServerInfo getThisServer()

**Returns:** The unique identifier and name for this Rapture server instance.

## setApplianceMode

**Syntax:** Boolean setApplianceMode(Boolean mode)

**Parameters:** whether to turn appliance mode on or off.

**Returns:** true or exception

**Method description:** Configures the instance into or out of appliance mode.

## setNetworkInfo

**Syntax:** RaptureNetwork setNetworkInfo(RaptureNetwork network)

**Parameters:** An object specifying the network name and unique identifier.

**Returns:** An object whose network information matches the passed parameter.

## setThisServer

**Syntax:** RaptureServerInfo setThisServer(RaptureServerInfo info)

**Parameters:** An object specifying the server name and unique identifier.

**Returns:** An object whose server information matches the passed parameter.

# Event

Events are used to coordinate large-scale activity in Rapture. The process is relatively simple - a caller assigns any number of scripts to a named event (simply a unique path), and when the event is fired all attached scripts are scheduled for execution. Some events are internally managed (system events) and other events can be user created and managed.

## attachMessageToEvent

**Syntax:** Boolean attachMessageToEvent(String eventURI, String name, String pipeline, Map(String, String) params)

**Returns:** true or exception

**Method description:** This method is used to attach a message to an event. When the event is fired a message is sent to the pipeline with content based on the context of the event and parameters passed to this call.

## attachNotificationToEvent

**Syntax:** Boolean attachNotificationToEvent(String eventURI, String name, String notification, Map(String, String) params)

**Returns:** true or exception

**Method description:** This method is used to attach a notification to an event. When the event is fired a message is sent to the notification with content based on the context of the event and parameters passed to this call.

## attachScriptToEvent

**Syntax:** Boolean attachScriptToEvent(String eventURI, String scriptURI, Boolean performOnce)

**Returns:** true or exception

**Method description:** This method is used to attach a script to an event. A final parameter signals whether this script should be detached from the event when it is fired.

## attachWorkflowToEvent

**Syntax:** Boolean attachWorkflowToEvent(String eventURI, String name, String workflowUri, Map(String, String) params)

**Returns:** true or exception

**Method description:** This method is used to attach a workflow (dp) to an event. When the event is fired an instance of the workflow is started.

## delete

**Syntax:** Boolean delete(String eventURI)

**Returns:** true or exception

**Method description:** This method removes an event (and any attached scripts) from the system. If the event is fired at a later point nothing will happen because there would be no scripts attached.

## fireEvent

**Syntax:** Boolean fireEvent(String eventURI, String associatedURI, String eventContext)

**Returns:** true or exception

**Method description:** This method fires an event, scheduling any attached scripts to run. The optional displayName and context parameters are passed to the script when fired.

## get

**Syntax:** RaptureEvent get(String eventURI)

**Method description:** This method is used to retrieve information about an event (primarily the scripts attached to it).

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String eventURIPrefix)

**Method description:** Return a list of full display names of the paths below this one. Ideally optimized depending on the repo.

## put

**Syntax:** Boolean put(RaptureEvent event)

**Returns:** true or exception

**Method description:** This method puts an event in the system.

## removeMessageFromEvent

**Syntax:** Boolean removeMessageFromEvent(String eventURI, String name)

**Returns:** true or exception

**Method description:** This method reverses the message attachment, using the same name as passed in the origin attachMessage call

## removeNotificationFromEvent

**Syntax:** Boolean removeNotificationFromEvent(String eventURI, String name)

**Returns:** true or exception

**Method description:** This method removes the notification attachment, using the same name as passed in the origin attachMessage call.

## removeScriptFromEvent

**Syntax:** removeScriptFromEvent(String eventURI, String scriptURI)

**Returns:** true or exception

**Method description:** This method detaches a script from the event.

## removeWorkflowFromEvent

**Syntax:** Boolean removeWorkflowFromEvent(String eventURI, String name)

**Parameters:** eventURI: URI specifying an event

name: name of workflow supplied previously

**Returns:** true or exception

**Method description:** This method removes the notification attachment, using the same name as passed in the original [attachWorkflowToEvent](#_attachWorkflowToEvent_1) call

# Feature

The feature API is used to manipulate information about stored features in the system. A feature is a set of scripts, type and data definitions.

## doesFeatureNeedToBeInstalled

**Syntax:** Boolean doesFeatureNeedToBeInstalled(FeatureConfig feature)

**Parameters:** Header information for a feature.

**Returns:** Whether the feature is outdated or missing.

**Method description:** Checks the version number of the feature (if already installed) to determine the return value. No content checking of the feature is performed.

## downloadURI

**Syntax:** FeatureTransportItem downloadURI(String uri)

**Parameters:** uri: the URI of a Rapture object or other data

**Returns:** The encoded contents of the Rapture object or other data

**Method description:** Get the encoding for a Rapture object given its URI.

## getFeature

Syntax: FeatureConfig getFeature(String featureURI)

**Parameters:** The URI of the feature.

**Returns:** Brief information about each feature.

**Method description:** Retrieves a feature by name.

## getFeatureManifest

**Syntax:** FeatureManifest getFeatureManifest(String manifestURI)

**Parameters:** the URI of the feature manifest object.

**Returns:** Detailed information about the feature, including its content list.

**Method description:** Retrieves the manifest for a feature.

## getInstalledFeatures

**Syntax:** List(FeatureConfig) getInstalledFeatures()

**Returns:** Brief information about each feature.

**Method description:** Lists features in the system.

## installFeature

**Syntax:** Boolean installFeature(FeatureManifest manifest, Map(String,FeatureTransportItem)payload)

**Parameters:** manifest: the contents of the feature

payload: a map between feature URI keys and the contents of the feature

**Returns:** true or exception

**Method description:** Installs the feature and updates the registry.

## installFeatureItem

**Syntax:** Boolean installFeatureItem(String featureName, FeatureTransportItem item)

**Parameters:** featureName: the name of the feature

item: a single line from the corresponding manifest

**Returns:** true or exception

**Method description:** Installs a single feature item; used for streaming installs.

## removeFeatureManifest

**Syntax:** Boolean removeFeatureManifest(String manifestURI)

**Parameters:** manifestURI: the URI of the feature

**Returns:** true or exception

**Method description:** Removes Feature Manifest but does not uninstall any referenced items.

## uninstallFeature

**Syntax:** Boolean uninstallFeature(String name)

**Parameters:** name: the feature to uninstall

**Returns:** true or exception

**Method description:** Removes the contents of a feature.

## uninstallFeatureItem

**Syntax:** boolean uninstallFeatureItem(FeatureManifest manifest, FeatureTransportItem item)

**Parameters:** manifest: reserved

item: the line item to delete

**Returns:** true or exception

**Method description:** Removes one line item from the server.

## unrecordFeature

**Syntax:** Boolean unrecordFeature(String name)

**Parameters:** name: the feature to hide

**Returns:** true or exception

**Method description:** Removes Feature from the installed list (note: does not delete manifest)

## verify

**Syntax:** Map(String,String) verify(String feature)

**Parameters:** feature: the name of the feature to verify

**Returns:** A map of line items that do not match the manifest. The key is the URI and the value is a message about what is wrong (such as *missing* or *changed*).

**Method description:** Verifies that the contents of a feature match the hashes in the manifest.

# Fields

Fields are well known concepts in Rapture that are parts of documents. By defining a field and its relationship to the data within a type, information can be retrieved from a document without the need to transfer the complete document back to a client. The methods in this API are used to both define fields and retrieve information from a document.

|  |
| --- |
|  |
|  |  |

## delete

**Syntax:** Boolean delete(String fieldURI)

**Parameters:** fieldURI: URI specifying a field

**Returns:** true on success, false or exception on error depending on implementation

**Method description:** Removes a field definition.

## exists

**Syntax:** Boolean exists(String fieldURI)

**Parameters:** fieldURI: uri specifying a field

**Returns:** true if the field definition exists, false otherwise.

**Method description:** Checks whether a field definition with the given URI exists. Note that this method is implemented by calling get, so checking for existence before getting is inefficient.

## get

**Syntax:** RaptureField get(String fieldURI)

**Parameters:** fieldURI: uri specifying a field

**Returns:** RaptureField object, or null if it does not exist.

**Method description:** Retrieves the field definition.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String authority)

**Parameters:** authority: the URI specifying an initial document folder.

**Returns:** List of full display names of the paths below the supplied folder. Ideally optimized depending on the repo.

## put

**Syntax:** Boolean put(String fieldURI, RaptureField field)

**Parameters:** fieldURI: uri specifying a field

**Returns:** true on success, false or exception on error depending on implementation

**Method description:** Creates or replaces the field definition.

## retrieveFieldsFromContent

**Syntax:** List(String) retrieveFieldsFromContent(String docURI, String content, List(String) fields)

**Parameters:** docURI: the URI specifying a document

content: Content of the document

fields: List of fields to retrieve from the document. May be empty but not null.

**Returns:** A list of values referenced by the fields. Note that there is not a simple 1:1 mapping between the returned list and the list of fields supplied as a parameter

**Method description:** Behaves similarly to retrieveFieldsFromDocument, except that the supplied content is first added to the document cache, overwriting any previous values.

## retrieveFieldsFromDocument

**Syntax:** List(String) retrieveFieldsFromDocument(String docURI, List(String) fields)

**Parameters:** docURI: uri specifying a document

fields: List of fields to retrieve from the document. May be empty but not null.

**Returns:** List of values referenced by the fields. Note that there is not a simple 1:1 mapping between the returned list and the list of fields supplied as a parameter.

# Fountain

A fountain is a unique number generator - once defined it can be used to create unique ids that can be attached to documents or entities. Fountains can be attached to a document repository so that new documents created in that repository can optionally have unique ids.

## createFountain

**Syntax:** RaptureFountainConfig createFountain(String fountainURI, String config)

**Parameters:** fountainURI: URI for the fountain to be created

config: the desired information defining the storage to be used for the fountain

**Returns:** An object for the fountain just created.

**Method description:** This method is used to define a new fountain in a given authority. The config parameter defines the storage to be used for managing the fountain.

## deleteFountain

**Syntax:** Boolean deleteFountain(String fountainURI)

**Parameters:** URI for the fountain to be deleted

**Returns:** true or exception

**Method description:** This method is used to delete a previously defined fountain.

## doesFountainExist

**Syntax:** Boolean doesFountainExist(String fountainURI)

**Parameters:** the URI to check.

**Returns:** True if a fountain was found.

## getFountain

**Syntax:** RaptureFountainConfig getFountain(String fountainURI)

**Parameters:** the URI for the fountain to be retrieved

**Returns:** Configuration info for the fountain

**Method description:** Retrieves a single fountain config given its name.

## getFountains

**Syntax:** List(RaptureFountainConfig) getFountains(String authority)

**Parameters:** authority: the authority that needs to be matched against.

**Returns:** A list of configuration objects for all matching fountains.

**Method description:** Gets a list of fountains that have the given authority. Each fountain has a URI, and the authority is part of the URI. Fountains whose URIs have an authority that matches the passed parameter will be returned.

|  |
| --- |
|  |

## incrementFountain

**Syntax:** String incrementFountain(String fountainURI, Long amount)

**Parameters:** fountainURI: URI for the fountain to be incremented

amount: the value to add to the fountain’s ID.

**Method description:** This method is used to increment the fountain and returns a string that corresponds to the newly generated id.

## resetFountain

**Syntax:** Boolean resetFountain(String fountainURI, Long count)

**Parameters:** fountainURI: URI for the fountain to be reset

count: the new value for the fountain’s ID.

**Returns:** true or exception

**Method description:** This method can be used to reset a fountain to a new id - all future requests will start from this new point.

# Index

One of Rapture’s key features is that it is not restricted to any particular backing store; it can leverage multiple database technologies at the same time. As a result, the underlying implementation might not support a fast searchable index. To address this issue, each backing store can provide its own index mechanism. The intention is that the implementation can use native indexing where available, but that the indexing interface is abstracted.

Note that only Mongo, Memory and Postgres implementations are available as of this release of the API. There is no indexing implementation for Cassandra.

## createIndex

**Syntax:** IndexConfig createIndex(String indexURI, String config)

**Parameters:** indexURI: The repository to be indexed

config: Defines which fields in the object need to be indexed.

**Returns:** An object describing the index.

**Method description:** Generates a new index for the repository. Note that objects are indexed only when they are written. Any objects already in the repository are not automatically indexed; they need to be read and written back.

## deleteIndex

**Syntax:** Boolean deleteIndex(String indexURI)

**Parameters:** The URI of the repository that no longer needs to be indexed.

**Returns:** true or exception

## getIndex

**Syntax:** IndexConfig getIndex(String indexURI)

**Parameters:** the URI of the index to be retrieved.

**Returns:** An object containing the config information, or null if no index is found

**Method description:** Get the config for a specified index.

## queryIndex

**Syntax:** TableQueryResult queryIndex(String indexURI, String query)

**Parameters:** indexURI: the repository to search

query: the query criteria

**Returns:** A table of all values matching the query.

**Method description:** queryIndex uses a simple structure of the form SELECT [DISTINCT] field [,field...] WHERE condition [, condition…] [ORDER BY field [DESC] ]

**Example:** SELECT DISTINCT foo, bar WHERE baz = “wibble” ORDER BY foo

# Lock

The Lock API contains functionality for working with semaphore locks.

## acquireLock

**Syntax:** LockHandle acquireLock(String providerURI, String lockName, long secondsToWait, long secondsToKeep)

**Method description:** Acquires a lock. Returns a LockHandle, which you need to pass to releaseLock when calling it. If unable to acquire the lock, returns null.

|  |
| --- |
|  |

## acquireLockWithContext

**Syntax:** LockHandle acquireLockWithContext(String providerURI, String lockName, String localContext, long secondsToWait, long secondsToKeep)

**Method description:** Similar to acquireLock, but you can add additional "context", which makes the owner of this lock current user + context.

## breakLock

**Syntax:** Boolean breakLock(String providerURI, String lockName)

**Method description:** This is a dangerous variant of releaseLock that will kick someone else off the lock queue.

## createLockProvider

**Syntax:** RaptureLockConfig createLockProvider(String providerURI, String config, String pathPosition)

**Method description:** Creates a lock provider.

## deleteLockProvider

**Syntax:** Boolean deleteLockProvider(String providerURI)

**Method description:** Deletes a lock provider by its URI.

## doesLockProviderExist

**Syntax:** Boolean doesLockProviderExist(String providerURI)

**Returns:** true if a lock provider was found.

## getLockProvider

**Syntax:** RaptureLockConfig getLockProvider(String providerURI)

**Method description:** Gets a lock provider by its URI.

## getLockProvidersForAuthority

**Syntax:** List(RaptureLockConfig) getLockProvidersForAuthority(String providerURI)

**Method description:** Retrieves the lock providers for a given authority.

## releaseLock

**Syntax:** Boolean releaseLock(String providerURI, String lockName, LockHandle lockHandle)

**Method description:** Releases a lock.

## releaseLockWithContext

**Syntax:** Boolean releaseLockWithContext(String providerURI, String lockName, LockHandle lockHandle, String localContext)

**Method description:** Releases a lock with additional context appended.

## setupDefaultProviders

**Syntax:** Boolean setupDefaultProviders(Boolean force)

**Method description:** Sets up any lock providers needed by Rapture by default, should be called from any startup scripts. If force is set to true, it will force the config to be set up again, even if it already exists.

|  |
| --- |
|  |

# Mailbox

Each Rapture environment has a single mailbox, although it is divided into logical parts by authority and category. Remote users (usually remote systems) can submit items to a mailbox, whereupon an event is signaled to allow for any processing of that item. Typical processing validates the mailbox content and creates real entities within the local system (e.g. an incoming order is converted into a real order if valid). When processed the category of an item can be changed to 'Done' to ensure it isn't reprocessed.

## getMailboxMessages

**Syntax:** List(RaptureMailMessage) getMailboxMessages(String mailboxURI)

**Parameters:** the category to get messages for

**Returns:** A list of messages

**Method description:** Retrieves all mailbox messages for a category.

## moveMailboxMessage

**Syntax:** Boolean moveMailboxMessage(String mailboxMessageURI, String newMailboxURI)

**Parameters:** mailboxMessageURI: the category of the message

newMailboxURI: the new category to publish to

**Returns:** true or exception

**Method description:** Moves a message from one category to another.

## postMailboxMessage

**Syntax:** String postMailboxMessage(String mailboxURI, String content)

**Parameters:** mailboxURI: the category to publish to

message: the content

**Returns:** the URI of the published message

**Method description:** This method is used to post a message onto a category (for an authority).

## setMailboxStorage

**Syntax:** Boolean setMailboxStorage(String mailboxConfig, String fountainConfig)

**Parameters:** mailboxConfig: storage-specific parameters for the mailbox

fountainConfig: storage-specific parameters for a fountain (q.v. [fountain.createFountain](#_createFountain_1))

**Returns:** true or exception

**Method description:** Defines the config for mailbox storage.

# Notification

The notification API is used as a means for transferring notifications between interested parties. In most cases a notification provider maintains a list of messages, and each message is associated with an ever increasing 'epoch number'. A client can retrieve the latest epoch number from a provider and then poll for changes since that epoch - all updates since that point can be returned.

Within the notification API is the activity API, which gives clients and programs the ability to record messages about changes and status for common activity running in the system.

|  |
| --- |
|  |
|  |  |
|  |  |

## createNotificationProvider

**Syntax:** RaptureNotificationConfig createNotificationProvider(String notificationName, String config, String purpose)

**Method description:** This method creates a definition of a notification provider.

## deleteNotificationProvider

**Syntax:** Boolean deleteNotificationProvider(String notificationName)

**Returns:** true or exception

**Method description:** This method removes a notification provider and all its content.

## doesNotificationProviderExist

**Syntax:** Boolean doesNotificationProviderExist(String notificationName)

**Method description:** Indicates whether a notification provider with notificationName was found.

## finishActivity

**Syntax:** finishActivity(String id, String myId, String message)

**Method description:** This method marks an activity as finished.

## getActivities

**Syntax:** List(RaptureActivity) getActivities()

**Method description:** This method retrieves information about the current activities in the system.

## getAllNotificationProviders

**Syntax:** List(RaptureNotificationConfig) getAllNotificationProviders()

**Method description:** This method retrieves the notification providers in use at this Rapture system.

## getChanges

**Syntax:** NotificationResult getChanges(String notificationName, Long lastEpochSeen)

**Method description:** This method returns the changes seen on a notification since an epoch. A client would then update its latest epoch by using the value in the notification result.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String notificationNamePrefix)

**Method description:** Returns a list of full display names of the paths below this one. Ideally optimized depending on the repo.

## getEpoch

**Syntax:** Long getEpoch(String notificationName)

**Method description:** This method retrieves the current epoch number for a given notification point.

## getNotification

**Syntax:** NotificationInfo getNotification(String notificationName, String id)

**Method description:** This method returns a notification message given its id.

## getNotificationProvider

**Syntax:** RaptureNotificationConfig getNotificationProvider(String notificationName)

**Method description:** This method returns the low level config for a given notification provider.

## getNotificationProviders

**Syntax:** List(RaptureNotificationConfig) getNotificationProviders(String purpose)

**Method description:** Notification providers have an associated purpose - this method returns only those providers that match the given purpose.

## publishNotification

**Syntax:** String publishNotification(String notificationName, String referenceId, String content, String contentType)

**Method description:** This method pushes a notification to a provider.

|  |
| --- |
|  |

## recordActivity

**Syntax:** String recordActivity(String myId, String message, Long progress, Long maxProgress, Long secondToExpire)

**Method description:** This method records an activity currently taking place. It returns a unique id that can be used to update the status of the activity.

## requestAbortActivity

**Syntax:** Boolean requestAbortActivity(String id, String myId)

**Method description:** This method is used to request that an activity finish. Only when an activity calls updateActivity will it receive such a request.

## updateActivity

**Syntax:** updateActivity(String id, String myId, String message, Long progress, Long maxProgress, Long secondToExpire)

**Method description:** This method updates the status of an activity. The return value is true if another process has requested that this activity should finish. The id and myId need to match those provided in the recordActivity call.

|  |
| --- |
|  |

# Pipeline

The Pipeline API is used to configure the Rapture System pipeline for running tasks within a cluster of Rapture Servers.

## broadcastMessageToCategory

**Syntax:** Boolean broadcastMessageToCategory(RapturePipelineTask task)

**Method description:** This message will be broadcasted to all servers belonging to the category specified in the RapturePipelineTask object. If no category is specified, an error is thrown.

## broadcastMessageToAll

**Syntax:** Boolean broadcastMessageToAll(RapturePipelineTask task)

**Method description:** This message will be broadcasted to all servers connected to the pipeline system.

## deregisterExchangeDomain

**Syntax:** Boolean deregisterExchangeDomain(String domainURI)

**Returns:** true or exception

**Method description:** Removes an exchange domain.

## deregisterPipelineExchange

**Syntax:** Boolean deregisterPipelineExchange(String name)

**Method description:** Removes an exchange.

## drainPipeline

**Syntax:** Boolean drainPipeline(String exchange)

**Returns:** true or exception

**Method description:** Drain an exchange - remove all messages.

## getBoundExchanges

**Syntax:** List(CategoryQueueBindings) getBoundExchanges(String category)

**Method description:** Lists all bound exchanges for a category.

## getExchange

**Syntax:** RaptureExchange getExchange(String name)

**Method description:** Retrieves an exchange object by name.

## getExchangeDomains

**Syntax:** List(String) getExchangeDomains()

**Method description:** Retrieves all registered exchange domains.

## getExchanges

**Syntax:** List(String) getExchanges()

**Method description:** Retrieves all registered exchanges

## getLatestTaskEpoch

**Syntax:** Long getLatestTaskEpoch()

**Method description:** On the task information, get the latest epoch (the maximum message id).

## getServerCategory

**Syntax:** List(String) getServerCategories()

**Method description:** Returns all server categories.

## getStatus

**Syntax:** PipelineTaskStatus getStatus(String taskId)

**Method description:** Get status for a published RapturePipelineTask.

## publishMessageToCategory

**Syntax:** Boolean publishMessageToCategory(RapturePipelineTask task)

**Method description:** Publishes a message. This message will be published to the category specified in the RapturePipelineTask object.

If no category is specified, an error is thrown. This type of message should be handled by only one of the servers belonging to this category; in other words, it is not a broadcast.

|  |
| --- |
|  |

## queryTasks

**Syntax:** List(RapturePipelineTask) queryTasks(TableQuery query)

**Method description:** Query for pipeline statuses.

## registerExchangeDomain

**Syntax:** Boolean registerExchangeDomain(String domainURI, String config)

**Returns:** true or exception

**Method description:** Registers a new exchange domain.

## removeServerCategory

**Syntax:** Boolean removeServerCategory(String category)

**Method description:** Deletes a given category.

## setupStandardCategory

**Syntax:** Boolean setupStandardCategory(String category)

**Returns:** true or exception

**Method description**: Sets up the default queue-exchanges and bindings for a given category

# Question

The question API allows control input to a decision process from clients (typically humans -- automated steps are better served by scripts).

|  |
| --- |
|  |

## answerQuestion

**Syntax:** Boolean answerQuestion(String questionURI, String response, Map(String,Object) data)

**Method description:** Attaches an answer to the given question.

## askQuestion

**Syntax:** String askQuestion(String qTemplateURI, Map(String,String) variables, String callback)

**Method description:** Asks a question using a template. The callback may not contain slashes.

## defineTemplate

**Syntax:** Boolean defineTemplate(String qTemplateURI, QTemplate template)

**Method description:** Defines or redefines a question template.

## getQNotificationURIs

**Syntax:** List(String) getQNotificationURIs(QuestionSearch search)

**Method description:** Gets just the URIs for a questionSearch.

## getQNotifications

**Syntax:** List(QNotification) getQNotifications(QuestionSearch search)

**Method description:** Lists what questions are pending for a given condition.

## getQuestion

**Syntax:** Question getQuestion(String questionURI)

**Method description:** Retrieves a question for a URI.

## getTemplate

**Syntax:** QTemplate getTemplate(String qTemplateURI)

**Method description:** Retrieves a question template

# Relationship

The Relationship API contains low-level methods for making connections between data and metadata. It can be used to track provenance, model association, or to impose a secondary organization of data.

## createRelationship

**Syntax:** String createRelationship(String relationshipAuthorityURI, String fromURI, String toURI, String label, Map(String,String) properties)

**Parameters:** relationshipAuthorityURI: the URI of the relationship repository

fromURI: the URI of the ‘from’ item in the relation

toURI: the URI of the ‘to’ item in the relation

label: the type of relationship

properties: details about the relationship

**Returns:** the URI of the relationship created

**Method description:** Stores a relationship link and return its URI

## createRelationshipRepo

**Syntax:** Boolean createRelationshipRepo(String relationshipRepoURI, String config)

**Parameters:** relationshipRepoURI: the name of the repository to create

config: the storage-specific configuration

**Returns:** true or exception

**Method description:** Creates a repository in which to store relationship information

## deleteRelationship

**Syntax:** Boolean deleteRelationship(String relationshipURI)

**Parameters:** the URI of the relationship

**Returns:** true or exception

## deleteRelationshipRepo

**Syntax:** Boolean deleteRelationshipRepo(String repoURI)

**Parameters:** repoURI: the URI of the repository to be deleted

**Returns:** true or exception

**Method description:** This method removes a Relationship Repository and its data from the Rapture system. There is no undo.

## doesRelationshipRepoExist

**Syntax:** Boolean doesRelationshipRepoExist(String repoURI)

**Parameters:** repoURI: the URI to be checked.

**Returns:** true if the repository is found at the URI

**Method description:** This API call can be used to determine whether a given type exists in a given authority.

## getAllRelationshipRepoConfigs

**Syntax:** List(RelationshipRepoConfig) getAllRelationshipRepoConfigs()

**Returns:** A list of configuration objects

**Method description:** Retrieves relationship repositories.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String prefix)

**Parameters:** the top level of the URI to search

**Returns:** A list of matching relationships

**Method description:** Gets children (nodes) in a relationship store. Relationship stores are only one level deep, so this API exists only to provide a consistent API as other repository types. This method is not useful except when making generic repository browsers.

## getInboundRelationships

**Syntax:** List(RaptureRelationship) getInboundRelationships(String relationshipRepoURI, String toURI)

**Parameters:** RelationshipRepoURI**:** the URI of the relationship repository

toURI: the URI of the item being searched for.

**Returns:** A list of relationships with the searched URI as the “to” side

**Method description:** Get all the relationship links with the specified Rapture resource as the "to" half of the link.

## getLabeledRelationships

**Syntax:** List(RaptureRelationship) getLabledRelationships(String relationshipRepoURI, String relationshipLabel)

**Parameters:** relationshipRepoURI**:** the URI of the relationship repository

relationshipLabel: the type of relationship to search for

**Returns:** A list of matching relationships

**Method description:** Gets all the relationship links with the specified label.

## getOutboundRelationships

**Syntax:** List(RaptureRelationship) getOutboundRelationships(String relationshipRepoURI, String fromURI)

**Parameters:** RelationshipRepoURI**:** the URI of the relationship repository

fromURI: the URI of the item being searched for.

**Returns:** A list of relationships with the searched URI as the “from” side

**Method description:** Gets all the relationship links with the specified Rapture resource as the "from" half of the link.

## getRelationship

**Syntax:** RaptureRelationship getRelationship(String relationshipURI)

**Parameters:** the URI of the relationship

**Returns:** the details of the relationship

**Method description:** Retrieves a relationship link.

## getRelationshipCenteredOn

**Syntax:** RaptureRelationshipRegion getRelationshipCenteredOn(String relationshipNodeURI, Map(String, String) options)

**Parameters:** relationshipNodeURI**:** the starting point for a search

options: the search options

**Returns:** A list of search results

**Method description:** Gets relationships from a given node.

## getRelationshipRepoConfig

**Syntax:** RelationshipRepoConfig getRelationshipRepoConfig(String relationshipRepoURI)

**Parameters:** relationshipRepoURI: the URI of the repository

**Returns:** The configuration information for the repo

**Method description:** Retrieves relationshipo repository information.

# Repository

The Repo API is used to interact with documents in the system.

## deleteContent

**Syntax:** Boolean deleteContent(String raptureURI, String comment)

**Method description:** Removes a document from the system.

## getContent

**Syntax:** ContentEnvelope getContent(String raptureURI)

**Method description:** Retrieves the content given a documentURI.

## putContent

**Syntax:** Boolean putContent(String raptureURI, Object content, String comment)

**Method description:** Stores the content supplied at the document URI given. Existing content will be overwritten at this URI, but the previous version may be stored if the underlying repository supports it.

|  |
| --- |
|  |

# Runner

The Runner API is used to manage application definitions for a Rapture environment that will be started and managed by the RaptureRunner application

|  |
| --- |
|  |
|  |  |

## addGroupInclusion

**Syntax:** RaptureServerGroup addGroupInclusion(String name, String inclusion)

**Parameters:** name: The name of the server group

inclusion: The value of the inclusion, e.g. ‘myhost002’, or ‘\*’

**Returns:** The modified server group object

**Method description:** Add a server group “inclusion.” An inclusion is a hostname where this server group should run. By default, this is set to \*, which means run everywhere. Adding an inclusion makes it so that this server group will run only on certain servers.

## addGroupExclusion

**Syntax:** RaptureServerGroup addGroupExclusion(String name, String exclusion)

**Parameters:** name: The name of the server group

exclusion: The value of the exclusion, e.g. ‘badhost002’, or ‘\*’

**Returns:** The modified server group object

**Method description:** Add a server group “exclusion.” An exclusion is a hostname where this server group should not run. By default, this is set to empty, which means run on every host specified in inclusions. It makes more sense to add an exclusion if this server group has a wildcard (\*) for inclusions. See also addGroupInclusion.

## addLibraryToGroup

**Syntax:** RaptureServerGroup addLibraryToGroup(String serverGroup, String libraryName)

**Parameters:** serverGroup: The name of the server group

libraryName: The name of the library

**Returns:** The server group object to which the library was added

**Method description:** Associates a library with a server group.

## changeApplicationStatus

**Syntax:** RaptureApplicationStatus changeApplicationStatus(String applicationURI, RaptureApplicationStatusStep statusCode, String message)

**Parameters**: applicationURI: The uri of the application status object to update

statusCode: The new status code

message: The new message to set

**Returns:** The updated status object

**Method description:** Updates the status of an app’s instance.

## cleanRunnerStatus

**Syntax:** Boolean cleanRunnerStatus(int ageInMinutes)

**Parameters:** Any statuses older than the age parameter will be deleted

**Returns:** true if the method could acquire the lock and clean old statuses.

**Method description:** Cleans out old status information, older than the passed parameter in minutes. It acquires a lock based on the server name, same as [recordRunnerStatus](#_recordRunnerStatus_1) and [markForRestart](#_markForRestart_1).

## createApplicationDefinition

**Syntax:** RaptureApplicationDefinition createApplicationDefinition(String name, String ver, String description)

**Parameters:** name: The name of the application

ver: The version of the application

description: A human-friendly description of the application

**Returns:** The application definition object just created

**Method description:** Create an application definition.

## createApplicationInstance

**Syntax:** RaptureApplicationInstance createApplicationInstance(String name, String description, String serverGroup, String appName, String timeRange, int retryCount, String parameters, String apiUser)

**Parameters:** name: The name of the schedule, aka “application instance”

description: A human-friendly description for this

serverGroup: The server group to which the application is associated

appName: The application name that will run

timeRange: The time range when this application should run. Normally this should be “\* \*” (no quotes)

retryCount: If the application fails to start, how many times it should retry. 3 is a reasonable value

parameters: Any parameters to pass to the application on startup. Usually this is an empty string

apiUser: The user who this should run as. Typically this should be “api”

**Returns:** The new application instance object

**Method description:** Adds an association between an application and a server group. This is the way to tell Rapture that a certain application needs to run (or be scheduled to run at given hours) as part of a server group. The association with the server group exists for two reasons:

1. It allows the application to run on certain hosts and not others (see [addGroupInclusion](#_addGroupInclusion_1) for more details).
2. It allows the application to have certain libraries on the class path, if those libraries are associated with the server group (see [addLibraryToGroup](#_addLibraryToGroup_1)).

## createLibraryDefinition

**Syntax:** RaptureLibraryDefinition createLibraryDefinition(String name, String ver, String description)

**Parameters:** name: the name of the library

ver: the version of the library

description: the human-friendly description

**Returns:** The new library definition object

**Method description:** Creates a Library definition. See also [getAllLibraryDefinitions](#_getAllLibraryDefinitions_1).

## createServerGroup

**Syntax:** RaptureServerGroup createServerGroup(String name, String description)

**Parameters:** name: name of the new server group

description: a human-readable description of the server group

**Returns:** A new server group object

**Method description:** Creates a new server group.

## getAllApplicationDefinitions

**Syntax:** List<RaptureApplicationDefinition> getAllApplicationDefinitions()

**Returns:** A list of application definition objects

**Method description:** Returns a list of all the applications defined in Rapture, which Rapture Runner knows about, including their versions. This is the list of applications that Rapture is aware of, but it does not necessarily run everything. To get a list of what will be running, look at getAllApplicationInstances.

## getAllApplicationInstances

**Syntax:** List<RaptureApplicationInstance> getAllApplicationInstances()

**Returns:** A list of application instance objects

**Method description:** Retrieves all the application instances defined in Rapture. This is really the list of schedule entries, meaning every application-server group combination that is scheduled to run.

## getAllServerGroups

**Syntax:** List<RaptureServerGroup> getAllServerGroups()

**Returns:** A list of server group objects

**Method description:** Returns all server groups defined in Rapture.

## getAllLibraryDefinitions

**Syntax:** List<RaptureLibraryDefinition> getAllLibraryDefinitions()

**Method description:** Get a list of all libraries defined in Rapture. These are also known as Rapture add-ons, or plugins.

## getApplicationsForServer

**Syntax:** List(RaptureApplicationInstance) getApplicationsForServer(String serverName)

**Parameters:** the server name or hostname

**Returns:** A list of application objects that will run on this server

**Method description**: Returns a list of applications that should run on a specific host (aka server). Servers are defined in inclusions; see [addGroupInclusion](#_addGroupInclusion_2) for more details. All applications that will run on a given server will be returned. Applications belonging to a server group that includes all servers via the “\*” wildcard will also be returned.

## getApplicationsForServerGroup

**Syntax:** List(String) getApplicationsForServerGroup(String serverGroup)

**Parameters:** the server group name

**Returns:** A list of application objects that will run on this server

**Method description:** Returns a list of application instance (aka schedule) names that are configured to run as part of a specific server group.

## getApplicationStatus

**Syntax:** RaptureApplicationStatus getApplicationStatus(String applicationURI)

**Parameters:** The URI of the application whose status we want

**Returns:** A status object that shows the current state of the app.

## getApplicationStatusDates

**Syntax:** List(String) getApplicationStatusDates()

**Returns:** A list of dates for which statuses exist.

**Method description:** Returns all dates that contain statuses.

## getApplicationStatuses

Syntax: List(RaptureApplicationStatus) getApplicationStatuses(String date)

**Parameters:** A string representing the target date

**Returns:** A list of status objects set on the given date

**Method description:** Lists the apps that are interesting, given a QBE template (empty strings have default behavior).

## getCapabilities

**Syntax:** Map<String, RaptureInstanceCapabilities> getCapabilities(String serverName, List<String> instanceName)

**Parameters:** serverName: The server’s name, aka hostname, where the app is running

instanceNames: The name of the RaptureApplicationInstance

**Returns:** a map of the instance name to capabilities that instance has.

**Method description:** Returns the capabilities for one or more instance running on the specified host. See also [recordInstanceCapabilities](#_recordInstanceCapabilities_1).

## getRunnerConfig

**Syntax:** RaptureRunnerConfig getRunnerConfig()

**Returns:** A RaptureRunner Config object

**Method description:** Returns the RaptureRunnerConfig object, which contains the values of the variables configured via [setRunnerConfig](#_setRunnerConfig_1).

## getRunnerServers

**Syntax:** List(String) getRunnerServers()

**Returns:** A list of host names

**Method description:** Gets a list of all the known server names (aka hostnames). This is determined by finding where a RaptureRunner is currently running or has run in the past and recorded a status (which has not been deleted), whether it be up or down.

## getRunnerStatus

**Syntax:** RaptureRunnerStatus getRunnerStatus(String serverName)

**Parameters:** the hostname

**Returns:** A runner status object

**Method description:** Get a RaptureRunnerStatus object for one specific host, which is a map of the statuses of all instances on a specific host.

## getServerGroup

**Syntax:** RaptureServerGroup getServerGroup(String name)

**Parameters:** the name of the server group to find

**Returns:** The server group object, or null if not found.

## markForRestart

**Syntax:** Boolean markForRestart(String serverName, String name)

**Parameters:** serverName: The server’s name, aka hostname, where the app is running

name: The name of the RaptureApplicationInstance

**Returns:**

**Method description:** Marks a running instance as needing reboot. If an application is not found as running on the specified server, nothing is done. This is specifically intended for “Restart”, not “Start if not already running.” This acquires a lock based on the server name, same as [recordRunnerStatus](#_recordRunnerStatus_2) and [cleanRunnerStatus](#_cleanRunnerStatus_1).

## recordInstanceCapabilities

**Syntax:** Boolean recordInstanceCapabilities(String serverName, String instanceName, Map<String, Object> capabilities)

**Parameters:** serverName: The server’s name, aka hostname, where the app is running

instanceName: The name of the RaptureApplicationInstance object to update

capabilities: A Map of the capabilities to store, usually string-to-string map

**Returns:** true or exception

**Method description:** Each RaptureApplicationInstance has certain “capabilities” associated with it. These could be queried by other apps if necessary (see [getCapabilities](#_getCapabilities_1)). For example, the RaptureAPIServer has a capability to handle api calls, and it posts its api uri, including port, as a capability, that other apps can retrieve if they want to contact the api directly. This method will record capabilities for a given instance.

## recordRunnerStatus

**Syntax:** Boolean recordRunnerStatus(String serverName, String serverGroup, String appInstance, String appName, String status)

**Parameters:** serverName: The server’s name, aka hostname, where the app is running

serverGroup: The name of the serverGroup associated with this RaptureApplicationInstance

appInstance: The name of the RaptureApplicationInstance object to update

appName: The name of the app to update

status: The value of the status

**Returns:** true if a lock can be acquired and updated

**Method description:** records the status of an app’s instance by acquiring a lock based on the server name, similar to the behavior of [cleanRunnerStatus](#_cleanRunnerStatus_2) and [markForRestart](#_markForRestart_2).

## recordStatusMessage

**Syntax:** Boolean recordStatusMessages(String applicationURI, List(String) messages)

**Parameters:** applicationURI: The uri of the application status to update

messages: A list of strings of messages to be associated with this application status

**Returns:** True if a status is found for the given URI

**Method description:** Adds a message to a running app instance.

## removeApplicationDefinition

**Syntax:** Boolean removeApplicationDefinition(String name)

**Parameters:** The name of the application

**Returns:** true or exception

**Method description:** Removes an application definition and any references to it in server groups.

## removeApplicationInstance

**Syntax:** Boolean removeApplicationInstance(String name, String serverGroup)

**Parameters:** name: The name of the application instance to remove

serverGroup: The name of the server group this application instance is associated with

**Returns:** true or exception

**Method description:** Deletes an application instance from Rapture.

## removeGroupInclusion

**Syntax:** RaptureServerGroup removeGroupInclusion(String name, String inclusion)

**Parameters:** name: The name of the server group

inclusion: The value of the inclusion, e.g. ‘myhost002’, or ‘\*’

**Returns:** The modified server group object

**Method description:** Removes a server group inclusion. Refer to [addGroupInclusion](#_addGroupInclusion_3) for more details.

## removeGroupEntry

**Syntax:** RaptureServerGroup removeGroupEntry(String name, String entry)

**Parameters:** name: The name of the server group

entry: The value of the inclusion or exclusion, e.g. ‘somehost002’

**Returns:** The modified server group object

**Method description:** Removes either an exclusion or inclusion from a server group. Refer to [removeGroupInclusion](#_removeGroupInclusion_1) or [removeGroupExclusion](#_removeGroupExclusion_1) for more details.

## removeGroupExclusion

**Syntax:** RaptureServerGroup removeGroupExclusion(String name, String exclusion)

**Parameters:** name: The name of the server group

exclusion: The value of the exclusion, e.g. ‘badhost002’, or ‘\*’

**Returns:** The modified server group object

**Method description:** Removes a server group exclusion. Refer to addGroupExclusion for more details

## removeLibraryDefinition

**Syntax:** Boolean removeLibraryDefinition(String name)

**Parameters:** the name of the library

**Returns:** true if found and deleted, false if not found, throws exception if there was an error deleting

**Method description:** Removes a library definition (and any references in server groups). See also [getAllLibraryDefinitions](#_getAllLibraryDefinitions_2).

## removeLibraryFromGroup

**Syntax:** RaptureServerGroup removeLibraryFromGroup(String serverGroup, String libraryName)

**Parameters:** serverGroup: The name of the server group

libraryName: The name of the library

**Returns:** The server group object from which the library was removed

**Method description:** Removes the library’s association with the server group.

## removeRunnerConfig

**Syntax:** Boolean removeRunnerConfig(String name)

**Parameters:** name: The name of the variable to be removed

**Returns:** true if any config variables previously existed, false otherwise

**Method description:** Removes a variable from the Runner config.

## removeServerGroup

**Syntax:** Boolean removeServerGroup(String name)

**Parameters:** name of the server group to be removed

**Returns:** true if found and deleted, false if not found, throws exception if there was an error deleting

**Method description:** Removes a server group (and all of its application definitions).

## retrieveApplicationDefinition

**Syntax:** RaptureApplicationDefinition retrieveApplicationDefinition(String name)

**Parameters:** the name of the application definition

**Returns:** the application definition object, or null if not found

**Method description:** Retrieves an application definition. See also [createApplicationDefinition](#_createApplicationDefinition_1).

## retrieveApplicationInstance

**Syntax:** RaptureApplicationInstance retrieveApplicationInstance(String name, String serverGroup)

**Parameters:** name: The name of the application instance to retrieve

serverGroup: The name of the server group this application instance is associated with

**Returns:** The application object associated with this instance and server group, or null if not found.

**Method description:** Gets an application instance object that has been defined in Rapture.

## retrieveLibraryDefinition

**Syntax:** RaptureLibraryDefinition retrieveLibraryDefinition(String name)

**Parameters:** the name of the library

**Returns:** The library definition object, or null otherwise

**Method description:** Retrieves an existing Library definition. See also [getAllLibraryDefinitions](#_getAllLibraryDefinitions_3).

## runApplication

**Syntax:** RaptureApplicationStatus runApplication(String appName, String queueName, Map(String, String) parameterInput, Map(String, String) parameterOutput)

**Parameters:** appName: The name of the application to run

queueName: deprecated, not used

parameterInput: A map of input parameters

parameterOutput: A map where outputs will be placed

**Returns:** A status object that shows the state of the app after it has executed.

**Method description:** Starts a batch/single process (ultimately to replace the oneshot calls).

## runCustomApplication

**Syntax:** RaptureApplicationStatus runCustomApplication(String appName, String queueName, Map(String, String) parameterInput, Map(String, String) parameterOutput, String customApplicationPath)

**Parameters:** appName: The name of the application to run

queueName: deprecated, not used

parameterInput: A map of input parameters

parameterOutput: A map where outputs will be placed

customApplicationPath: The path where the application is located.

**Returns:** A status object that shows the state of the app after it has executed.

**Method description:** Starts a batch/single process (ultimately to replace the oneshot calls). Allows you to specify any application path.

## setRunnerConfig

**Syntax:** Boolean setRunnerConfig(String name, String value)

**Parameters:** name: The name of the variable. Accepted values are APPSOURCE or MODSOURCE

value: The value of the variable, normally a filesystem path or URI

**Returns:** true or exception

**Method description:** Set a config variable available in RaptureRunner. The config variables understood are APPSOURCE and MODSOURCE, and they specify the location of the apps and libraries controlled by RaptureRunner.

## terminateApplication

**Syntax:** RaptureApplicationStatus terminateApplication(String applicationURI, String reasonMessage)

**Parameters:** applicationURI: The uri of the application status object

reasonMessage: The message to be added to the status object, when terminating

**Returns:** null

**Method description:** Cancels execution of the app. [To be implemented; test before using.]

## updateApplicationVersion

**Syntax:** RaptureApplicationDefinition updateApplicationVersion(String name, String ver)

**Parameters:** name: The name of the application to update

ver: The new version number

**Returns:** The modified application definition object

**Method description:** Updates a version of an application. This modifies the corresponding store RaptureApplicationDefinition

## updateLibraryVersion

**Syntax:** RaptureLibraryDefinition updateLibraryVersion(String name, String ver)

**Parameters:** name: The name of the library to update

ver: The new version number

**Returns:** The modified library definition object.

# Schedule

The methods in this API control the scheduler function in a Rapture implementation.

## ackJobError

**Syntax:** JobErrorAck ackJobError(String jobURI, Long execCount, String jobErrorType)

**Parameters:** - a jobURI, specifying the unique uri for this job

-execCount, the execCount that is used to identify the execution that failed, which we want to acknowledge

- The error type that we want to acknowledge (e.g “Overrun” or “Failed”)

**Method description:** Acknowledges a job failure, storing the acknowledgment in Rapture. This information is returned when retrieving job statuses. See also [getWorkflowExecsStatus](#getWorkflowExecStatus).

## activateJob

**Syntax:** Boolean activateJob(String jobURI, Map(String, String) extraParams)

**Parameters:** - jobURI: a jobURI, specifying the unique uri for this job

-extraParams: any additional parameters to be passed to the job the next time it runs. These are temporary: they are only passed to the one upcoming execution. To change parameters passed in during subsequent executions, you need to change the job definition.

**Method description:** Activates a job (usually that is not auto-activate). This means that the job will now be picked up by the scheduler and executed at whatever time it is configured to run.

## createJob

**Syntax:** RaptureJob createJob(String jobURI, String description, String scriptURI, String cronExpression, String timeZone, Map<String,String> jobParams, Boolean autoActivate)

|  |
| --- |
|  |

**Parameters:**

- jobURI: a unique uri to identify this job

- description: a human-readable description

- scriptURI: The uri of the script to run

- cronExpression: when the job should run, follows the Unix cron format

- timeZone: the timezone when the job should run, uses the tz format for timezone names (<http://en.wikipedia.org/wiki/List_of_tz_database_time_zones>)

- jobParams: any parameters to pass to the script when running it

- autoActivate: whether the job should be scheduled and run as specified in the cron or if it should be only on-demand (see [runJobNow](#_runJobNow_1))

**Returns:** The RaptureJob object that was created

**Method description:** Create a new job. The scriptURI should point to a RaptureScript.

A job needs to be activated for it can be available for execution. A job can be either auto-activate (i.e. it is activated, then de-activated while it runs, then activated on completion. OR it can be not-auto-activate, whereupon it needs to be activated manually, by either a predecessor job (a job that has this job as a dependency) or manually via the activate schedule API call.

|  |
| --- |
|  |

## createWorkflowJob

**Syntax:** RaptureJob createWorkflowJob(String jobURI, String description, String workflowURI, String cronExpression, String timeZone, Map<String,String> jobParams, Boolean autoActivate, int maxRuntimeMinutes, String appStatusNamePattern)

**Parameters:**

- jobURI: a unique uri to identify this job

-description: a human-readable description

-workflowURI: The uri of the workflow to run

-cronExpression: when the job should run, follows the Unix cron format

- timeZone: the timezone when the job should run, uses the tz format for timezone names (<http://en.wikipedia.org/wiki/List_of_tz_database_time_zones>)

- jobParams: any parameters to pass to the script when running it

- autoActivate: whether the job should be scheduled and run as specified in the cron or if it should be only on-demand (see [runJobNow](#_runJobNow_2))

-maxRuntimeMinutes: the maximum number of minutes that the job is expected to take. Anything above this is marked as overrun (see [getWorkflowExecStatus](#getWorkflowExecStatus))

- appStatusNamePattern: The “appstatus” name to be set when this workflow runs, used to group audit log messages output by this workorder as well as group the workorder for viewing purposes

**Returns:** The RaptureJob object that was created

**Method description:** Creates a new Workflow-based job. The executableURI should point to a Workflow. A WorkOrder will be created when the job is executed. The jobParams will be passed in to the Workflow as the contextMap.

The maxRuntimeMinutes will be used to throw alerts when the job runs longer than expected. A job needs to be activated for it to be available for execution. A job can be either auto-activate (i.e. it is activated, then de-activated while it runs, then activated on completion. OR it can be not-auto-activate, whereupon it needs to be activated manually, by either a predecessor job (a job that has this job as a dependency) or manually via the activate schedule API call.

|  |
| --- |
|  |

## deactivateJob

**Syntax:** Boolean deactivateJob(String jobURI)

**Parameters:** - jobURI: The unique uri used to identify this job

**Returns:** true

**Method description:** Turns off a job’s schedule-based execution. See also: [activateJob](#_activateJob_1), [runJobNow](#_runJobNow_2).

## deleteJob

**Syntax:** Boolean deleteJob(String jobURI)

**Parameters:** jobURI: unique URI used to identify the job

**Returns:** true if job was found and deleted, false otherwise

**Method description:** Removes a job from the system.

## getCurrentWeekTimeRecords

**Syntax:** List(TimedEventRecord) getCurrentWeekTimeRecords(int weekOffsetfromNow)

**Returns:** a List of TimedEventRecord objects, which are the TimeServer scheduled events

**Method description:** For TimeServer, gets a list of scheduled events for this week (starts on Sunday, use offset to look at next week).

## getJobExecs

**Syntax:** List(RaptureJobExec) getJobExecs(String jobURI, int start, int count, Boolean reversed)

Parameters: jobURI: the unique uri used to identify this job

- start: the number of the execution where we want to start counting

- count: how many job executions this should return

- reversed: whether we should start counting from the end and count backwards; if true, a start value of 0 means start from the very last job execution, if false, 0 means start from the very first job execution.

**Method description:** Retrieves a list of job executions in a given range.

## getJobs

**Syntax:** List(String) getJobs()

**Returns:** A list of strings representing each job URI

**Method description:** Retrieves all of the JobURI addresses of the jobs in the system.

## getNextExec

**Syntax:** RaptureJobExec getNextExec(String jobURI)

**Parameters:** -jobURI: a unique uri used to identify this job

**Returns:** the RaptureJobExec for the next execution, or null if none found

**Method description:** Gets the next execution time for a given job.

## getUpcomingJobs

**Syntax:** List<RaptureJobExec> getUpcomingJobs()

**Returns:** A list of RaptureJobExec objects with all the upcoming job executions currently scheduled in Rapture.

**Method description:** Retrieves all of the upcoming jobs in the system.

## getWorkflowExecsStatus

**Syntax:** WorkflowExecsStatus getWorkflowExecsStatus()

**Method description:** Retrieves the status of all current workflow-based job executions. This looks into the last execution as well as upcoming execution for all scheduled jobs. The return object contains a list of jobs that succeeded, failed, are overrun, or are ok (i.e. either scheduled to start in the future or currently running but not overrun). For failed or overrun jobs, information is also returned as to whether the failure/overrun has been acknowledged. See also [ackJobError](#_ackJobError_1).

## resetJob

**Syntax:** Boolean resetJob(String jobURI)

**Parameters:** A job uri used to uniquely identify this job

**Returns:** true

**Method description:** Removes the upcoming scheduled execution of this job and schedules it to run according to the cron in the job configuration.

## retrieveJob

**Syntax:** RaptureJob retrieveJob(String jobURI)

**Parameters:** job uri used to uniquely identify this job

**Returns:** A RaptureJob object with this job, or null if none found at that URI

**Method description:** Retrieves the definition of a job given its URI.

## retrieveJobExec

**Syntax:** RaptureJobExec retrieveJobExec(String jobURI, Long execCount)

**Parameters:** - job uri used to uniquely identify this job

- execCount: the count of the execution we want to retrieve

**Returns:** A RaptureJobExec with that count, if one exists, or null if none exists

**Method description:** Retrieves the execution of a job.

## runJobNow

**Syntax:** Boolean runJobNow(String jobURI, Map(String, String) extraParams)

**Parameters:** jobURI: job uri used to uniquely identify this job

- extraParams: any additional parameters to pass to the executable when running it

**Returns:** true

**Method description:** Tries to schedule this job to run as soon as possible.

# Script

The Scripting API is used to define and manage Reflex scripts that are used within Rapture.

## archiveOldREPLSessions

**Syntax:** Boolean archiveOldREPLSessions(Long ageInMinutes)

**Parameters:** ageInMinutes: anything older than this will be archived.

**Returns:** true

**Method description:** Archives/deletes old Reflex REPL sessions.

## checkScript

**Syntax:** String checkScript(String scriptURI)

**Parameters:** scriptURI: a uri used to uniquely identify this script

**Returns:** A string with all of the error messages, or empty string if none

**Method description:** Parses the script and returns any error messages from the parsing process. If the String returned is empty the script is valid Reflex.

## createREPLSession

**Syntax:** String createREPLSession()

**Returns:** The ID of the REPL session

**Method description:** Create a Reflex REPL session that can be written to. These sessions eventually die if not used or killed.

## createScript

**Syntax:** RaptureScript createScript(String scriptURI, RaptureScriptLanguage language, RaptureScriptPurpose purpose, String script)

**Parameters:** scriptURI: a uri used to uniquely identify this script

- language: The language of the script, e.g. REFLEX

- purpose: the purpose of the script, e.g. PROGRAM

- script: The body of the script

**Returns:** a RaptureScript object for the script that was created

**Method description:** Create a script in the system.

## createScriptLink

**Syntax:** Boolean createScriptLink(String fromScriptURI, String toScriptURI)

**Parameters:** fromScriptURI: The new symbolic link URI

toScriptURI: The uri of the real script

**Returns:** true

**Method description:** Create a symbolic link to a script in the system.

## createSnippet

**Syntax:** RaptureSnippet createSnippet(String snippetURI, String snippet)

**Parameters:** -snippetURI: The uri for the new snippet

- snippet: The body/code of the snippet

**Returns:** A RaptureSnippet object created for the new snippet

**Method description:** Creates a code snippet and stores it in Rapture.

## deleteScript

**Syntax:** Boolean deleteScript(String scriptURI)

**Parameters:** scriptURI: a uri used to uniquely identify this script

**Returns:** true if found and deleted, false otherwise

**Method description:** Remove the script from the system.

## deleteSnippet

**Syntax:** Boolean deleteSnippet(String snippetURI)

**Parameters:** snippetURI: unique uri used to identify the snippet

**Returns:** true if found and deleted, false otherwise

**Method description:** Deletes a snippet by its URI.

## destroyREPLSession

**Syntax:** Boolean destroyREPLSession(String sessionId)

**Parameters:** the session id for this REPL session

**Returns:** true if found and deleted, false otherwise

**Method description:** Kill an existing Reflex REPL session. See also [createREPLSesson](#_createREPLSession_1).

## doesScriptExist

**Syntax:** Boolean doesScriptExist(String scriptURI)

**Parameters:** scriptURI: a uri used to uniquely identify this script

**Returns:** true if it exists, false otherwise

**Method description:** Returns whether the given script exists or not.

## evaluateREPL

**Syntax:** String evaluateREPL(String sessionId, String line)

**Parameters:** sessionId: Unique id to identify a given session

- line: the line to add

**Returns:** The output of the execution

**Method description:** Add a line to the current Reflex session, returns what the parser/evaluator says.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String scriptURI)

**Parameters:** scriptURI: a URI whose children we want to return

**Returns:** A list of RaptureFolderInfo objects, with the structure of folders and scripts that exists at any level at or below the passed-in URI

**Method description:** Return a list of full display names of the paths below this one. Ideally optimized depending on the repo.

## getSnippet

**Syntax:** RaptureSnippet getSnippet(String snippetURI)

**Parameters:** snippetURI: unique uri to identify this snippet

**Returns:** a RaptureSnippet object, or null if none found at this URI

**Method description:** Retrieves a snippet by its URI.

## getSnippetChildren

**Syntax:** List(RaptureFolderInfo) getSnippetChildren(String prefix)

**Parameters:** - prefix: a uri whose children we want to return

**Returns:** A list of RaptureFolderInfo objects, with the structure of folders and scripts that exists at any level at or below the passed in URI

**Method description:** Returns all children snippets with a given prefix.

## getScript

**Syntax:** RaptureScript getScript(String scriptURI)

**Parameters:** scriptURI: a uri used to uniquely identify this script

**Returns:** A RaptureScript object for that URI, or null if none found

**Method description:** Retrieve the contents of a script.

## getScriptNames

**Syntax:** List(String) getScriptNames(String scriptURI)

**Parameters:** scriptURI: a uri with an authority, under which we want to retrieve all scripts

**Returns:** a list of strings with script names for all scripts in the uri specified in the authority

**Method description:** Retrieve all of the scripts within an authority.

## putScript

**Syntax:** RaptureScript putScript(String scriptURI, RaptureScript script)

**Parameters:** -scriptURI: a uri used to uniquely identify this script

- script: a RaptureScript object with the already defined script we want to put at that URI

**Returns:** The RaptureScript object that was passed in.

**Method description:** Store a script in the system. See also [createScript](#_createScript_1).

## removeFolder

**Syntax:** List(String) removeFolder(String scriptURI)

**Parameters:** scriptURI: a uri of the folder to remove

**Returns:** A list of all the contents of the folder that were removed

**Method description:** Removes a folder from the script area, and its contents.

## removeScriptLink

**Syntax:** Boolean removeScriptLink(String fromScriptURI)

**Parameters:** fromScriptURI: the uri of the link

**Returns:** true if found and deleted, false otherwise

**Method description:** Removes a symbolic link to a script in the system.

## runScript

**Syntax:** String runScript(String scriptURI, Map(String, String) parameters)

**Parameters:** scriptURI: a uri used to uniquely identify this script

- parameters: parameters to pass in o the script

**Returns:** The return value of the script

**Method description:** Runs a script in the Rapture environment.

## runScriptExtended

**Syntax:** ScriptResult runScriptExtended(String scriptURI, Map(String, String) parameters)

**Parameters:** scriptURI: a uri used to uniquely identify this script

- parameters: parameters to pass in o the script

**Returns:** A ScriptResult object, which contains the return code as well as output of that script.

**Method description:** Runs a script and return its result as a ScriptResult object.

## setScriptParameters

**Syntax:** RaptureScript setScriptParameters(String scriptURI, List(RaptureParameter) parameters)

**Parameters:** scriptURI: a uri used to uniquely identify this script

- parameters: parameters to pass in to the script. The type RaptureParameter allows you to specify both paramater name and type

**Returns:** the RaptureScript whose parameters were just set

**Method description:** Set the parameters that should be used for a script. This isn’t a set of actual parameter values to pass in to the script, but rather the list of argument that can be passed in to the script.

# Series

For manipulating time series objects.

## addDoubleToSeries

**Syntax:** Boolean addDoubleToSeries(String seriesURI, String columnKey, Double columnValue)

**Parameters:** seriesURI: URI of the series to edit

columnKey: the column/date of the new item

columnValue: the new item

**Returns:** true or exception

**Method description:** Adds one point of floating-point data to a series.

## addDoublesToSeries

**Syntax:** Boolean addDoublesToSeries(String seriesURI, List(String) columns, List(Double) values)

**Parameters:** seriesURI: URI of the series to edit

columnKey: a list of the columns/dates of the new data

values: the new data

**Returns:** true or exception

**Method description:** Adds a list of floating-point data points to a series.

## addLongToSeries

**Syntax:** Boolean addLongToSeries(String seriesURI, String columnKey, Long columnValue)

**Parameters:** seriesURI: URI of the series to edit

columnKey: the column/date of the new item

columnValue: the new item

**Returns:** true or exception

**Method description:** Adds one point of type long to a series.

## addLongsToSeries

**Syntax:** Boolean addLongsToSeries(String seriesURI, List(String) columns, List(Long) values)

**Parameters:** seriesURI: URI of the series to edit

columnKey: a list of the columns/dates of the new data

values: the new data

**Returns:** true or exception

**Method description:** Adds a list of long integer points to a series.

## addStringToSeries

**Syntax:** Boolean addStringToSeries(String seriesURI, String columnKey, String columnValue)

**Parameters:** seriesURI: URI of the series to edit

columnKey: the column/date of the new item

columnValue: the new item

**Returns:** true or exception

**Method description:** Adds one string point to a series.

## addStringsToSeries

**Syntax:** Boolean addStringsToSeries(String seriesURI, List(String) columns, List(String) values)

**Parameters:** seriesURI: URI of the series to edit

columnKey: a list of the columns/dates of the new data

values: the new data

**Returns:** true or exception

**Method description:** Adds a list of string points to a series.

## addStructureToSeries

**Syntax:** Boolean addStructureToSeries(String seriesURI, String columnKey, String jsonColumnValue)

**Parameters:** seriesURI: URI of the series to edit

columnKey: the column/date of the new item

columnValue: the new item

**Returns:** true or exception

**Method description:** Adds one point containing a JSON-encoded structure to a series.

## addStructuresToSeries

**Syntax:** Boolean addStructuresToSeries(String seriesURI, List(String) columns, List(String) jsonValues)

**Parameters:** seriesURI: URI of the series to edit

columnKey: a list of the columns/dates of the new data

values: the new data

**Returns:** true or exception

**Method description:** Adds a list of points containing JSON-encoded structures to a series.

## createSeriesRepo

**Syntax:** Boolean createSeriesRepo(String seriesURI, String config)

**Parameters:** seriesURI**:** the URI of the repository to create

config: The syntax of the config paramter is:

SREP { } USING *type* { }  
where *type* must be one of MEMORY, CASSANDRA, or MONGO.

**Returns:** true or exception

**Method description:** Creates a repository for series data.

## deleteSeriesRepo

**Syntax:** Boolean deleteSeriesRepo(String repoURI)

**Parameters:** the URI of the repository to delete.

**Returns:** true or exception

**Method description:** This method removes a Series Repository and its data from the Rapture system. There is no undo.

## doesSeriesRepoExist

**Syntax:** Boolean doesSeriesRepoExist(String seriesURI)

**Parameters:** the URI of the repository to examine

**Returns:** true if found

## dropAllPointsFromSeries

**Syntax:** Boolean dropAllPointsFromSeries(String seriesURI)

**Parameters:** The series to delete

**Returns:** true or exception

**Method description:** Removes all points in a series, then removes the series from the directory listing for its parent folder.

## dropPointsFromSeries

**Syntax:** Boolean dropPointsFromSeries(String seriesURI, List(String) columns)

**Parameters:** seriesURI: URI of the series to edit

columnKey: a list of the columns/dates of the data to remove

**Returns:** true or exception

**Method description:** Deletes a list of points from a series.

## executeSeriesProgram

**Syntax:** List(XferSeriesValue) executeSeriesProgram(String program, List(String) args)

**Parameters:** program: the full text of a series program

args: arguments for the formal parameters (if any)

**Returns:** the output series of the program

**Method description:** Executes a series function program and returns its default output.

## executeSeriesProgramQuietly

**Syntax:** Boolean executeSeriesProgramQuietly(String program, List(String) args)

**Parameters:** program: the full text of a series program

args: arguments for the formal parameters (if any)

**Returns:** true or exception

**Method description:** Executes a series function program and returns success status only.

## getAllChildrenMap

**Syntax:** Map(String,RaptureFolderInfo) getAllChildrenMap(String seriesURI)

**Parameters:** The URI to examine

**Returns:** A map of all children, determined recursively.

**Method description:** Returns full pathnames for an entire subtree as a map of the path to RFI.

## getAllFromRange

**Syntax:** List(XferSeriesValue) getAllFromRange(String seriesURI, String startColumn, String endColumn)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

endColumn: the column/date not to read beyond

**Returns:** All points in the specified range

**Method description:** Gets all points from a range in a series.

## getAllFromRangeAsDoubles

**Syntax:** SeriesDoubles getAllFromRangeAsDoubles(String seriesURI, String startColumn, String endColumn)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

endColumn: the column/date not to read beyond

**Method description:** Same as getAllFromRange but casts the points to doubles.

## getAllFromRangeAsStrings

**Syntax:** SeriesStrings getAllFromRangeAsStrings(String seriesURI, String startColumn, String endColumn)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

endColumn: the column/date not to read beyond

**Method description:** Same as getAllFromRange but casts the points to strings.

## getAllPoints

**Syntax:** List(XferSeriesValue) getAllPoints(String seriesURI)

**Parameters:** The series to examine

**Returns:** A list of all data points in the series

**Method description:** If the series size is less than the maximum batch size (one million points by default), this returns all points in a list. If the series is larger, an exception is thrown.

## getAllPointsAsDoubles

**Syntax:** SeriesDoubles getAllPointsAsDoubles(String seriesURI)

**Parameters:** The series to examine

**Returns:** A list of all data points in the series

**Method description:** Gets all points and casts them to doubles.

## getAllPointsAsStrings

**Syntax:** SeriesStrings getAllPointsAsStrings(String seriesURI)

**Parameters:** The series to examine

**Returns:** A list of all data points in the series

**Method description:** Gets all points and casts them to strings.

## getAllSeriesRepoConfigs

**Syntax:** List(SeriesRepoConfig) getAllSeriesRepoConfigs()

**Returns:** A list of all series repositories.

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String seriesURI)

**Parameters:** The URI to examine

**Returns:** The directory listing of the folder

**Method description:** Gets all the immediate children of a particular series path, including both series and folders.

## getLastPoint

**Syntax:** XferSeriesValue getLastPoint(String seriesURI)

**Parameters:** The series to examine

**Returns:** The data point

**Method description:** Retrieves the last point in a series.

## getPoints

**Syntax:** List(XferSeriesValue) getPoints(String seriesURI, String startColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

maxNumber: the page size

**Returns:** up to one page of data

**Method description:** Gets one page of data from the series.

## getPointsAsDoubles

**Syntax:** SeriesDoubles getPointsAsDoubles(String seriesURI, String startColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

maxNumber: the page size

**Returns:** the column/date values as parallel lists

**Method description:** Same as getPoints and casts returned data to doubles.

## getPointsAsStrings

**Syntax:** SeriesStrings getPointsAsStrings(String seriesURI, String startColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

maxNumber: the page size

**Method description:** Same as getPoints and casts returned data to strings.

## getPointsReverse

**Syntax:** List(XferSeriesValue) getPointsReverse(String seriesURI, String startColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

maxNumber: the page size

**Returns:** up to one page of data

**Method description:** Gets one page of data and reverses the normal sort order.

## getRange

**Syntax:** List(XferSeriesValue) getRange(String seriesURI, String startColumn, String endColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

endColumn: the column/date not to read beyond

maxNumber: the page size

**Returns:** up to one page of data

**Method description:** Gets one page of data from a series range.

## getRangeAsDoubles

**Syntax:** SeriesDoubles getRangeAsDoubles(String seriesURI, String startColumn, String endColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

endColumn: the column/date not to read beyond

maxNumber: the page size

**Method description:** Same as getRange but casts the points as doubles.

## getRangeAsStrings

**Syntax:** SeriesStrings getRangeAsStrings(String seriesURI, String startColumn, String endColumn, int maxNumber)

**Parameters:** seriesURI: the URI of the series

startColumn: the column/date to start from

endColumn: the column/date not to read beyond

maxNumber: the page size

**Method description:** Same as getRange but casts the points to strings.

## getSeriesRepoConfig

**Syntax:** SeriesRepoConfig getSeriesRepoConfig(String seriesURI)

**Parameters:** the URI of the repository to examine

**Returns:** The repository’s config, or null if the repository was not found.

## removeFolder

**Syntax:** Boolean removeFolder(String seriesURI)

**Parameters:** the URI of the folder to remove

**Returns:** true or exception

**Method description:** Recursively removes all series repositories that are children of the given URI.

## validateSeriesRepo

**Syntax:** Boolean validateSeriesRepo(String seriesURI)

**Parameters:** the URI of the repository to instantiate

**Returns:** true or exception

**Method description:** Forces instantiation of the repository.

# Sheet

For manipulating Rapture sheet objects.

## cloneSheet

**Syntax:** Boolean cloneSheet(String sheetURI, String newSheetURI)

**Parameters:** - sheetURI: the source sheet to be cloned

- newShetURI: The URI of the clone sheet that will be created

**Returns:** true

**Method description:** Copies everything fom one sheet to another. This is currently only allowed within sheets that have the same authority.

## createNote

**Syntax:** RaptureSheetNote createNote(String sheetURI, RaptureSheetNote note)

**Parameters:** sheetURI: the uri used to uniquely identify this sheet

- note: The RaptureSheetNote object that will be associated with the sheet

**Returns:** The note that was passed in as an argument

**Method description:** Adds a note to a sheet. One sheet can have multiple notes associated with it. A note is a text note with information about who created it and when.

## createRange

**Syntax:** RaptureSheetRange createRange(String sheetURI, String rangeName, RaptureSheetRange range)

**Parameters:** -sheetURI: the uri used to uniquely identify this sheet

-rangeName: the name to set for the range. If the RaptureSheetRange object already has a name assigned, it will get overridden.

- range: The range object specifying the columns and rows

**Method description:** Creates a range in the sheet. A range is a named set of sequential rows and columns.

## createSheet

**Syntax:** RaptureSheet createSheet(String sheetURI)

**Parameters:** sheetURI: the uri used to uniquely identify this sheet

**Returns:** The new RaptureSheet that was created.

**Method description:** Create an empty sheet at the given URI.

## createSheetRepo

**Syntax:** Boolean createSheetRepo(String sheetURI, String config)

**Parameters:** sheetURI: the URI for the repository

config: the config for the repo. Possible values:

i) SHEET {} USING MONGODB { prefix = “someprefix” }

ii) - SHEET {} USING MEMORY{}

iii) - SHEET {} USING CSV { }

**Returns:** true

**Method description:** Creates a sheet repository.

## createStyle

**Syntax:** RaptureSheetStyle createStyle(String sheetURI, String styleName, RaptureSheetStyle style)

**Parameters**: sheetURI: the uri of the sheet

styleName: the name used to identify this style

style: The RaptureSheetStyle object that contains details about the style

**Returns:** The style object that was passed in as an argument

**Method description:** Creates a style for this sheet. This can be used by GUIs when displaying the sheet.

## createScript

**Syntax:** RaptureSheetScript createScript(String sheetURI, String scriptName, RaptureSheetScript script)

**Parameters:** sheetURI: The sheet uri

- scriptName: The name to use for the script

- script: The RaptureSheetScript object with the details of the script, such as the body and description.

**Method description:** Creates a Reflex script and associates it with a given sheet. This script gets embedded into the sheet, i.e. it cannot be accessed as a standard Rapture script using the Script API’s retrieveScript method. See also [runScriptOnSheet](#_runScriptOnSheet_1).

## deleteCell

Syntax: Boolean deleteCell(String sheetURI, int row, int column, int dimension)

**Parameters:** - sheetURI: The uri of the sheet

- row: The row of the cell

- column: The column of the cell

- dimension: The dimension where this cell lives. Equivalent to a “tab” in the sheet, from a GUI standpoint.

**Method description:** Deletes data sotred in a given cell.

## deleteColumn

**Syntax:** Boolean deleteColumn(String sheetURI, int column)

**Parameters:** - sheetURI: The uri of the sheet

- column: The column to delete

**Returns:** true

**Method description:** Deletes an entire column in the sheet, in all dimensions.

## deleteRow

**Syntax:** Boolean deleteRow(String sheetURI, int row)

**Parameters:** - sheetURI: The uri of the sheet

- row: The row to delete

**Returns:** true

**Method description:** Deletes an entire row in the sheet, in all dimensions.

## deleteSheet

**Syntax:** RaptureSheet deleteSheet(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** The RaptureSheet that was deleted, if one was found, or null otherwise

**Method description:** Deletes a sheet and all its contents.

## deleteSheetRepo

**Syntax:** Boolean deleteSheetRepo(String repoURI)

**Parameters:** repoURI: The uri of the repo to delete.

**Returns:** True if found and deleted, false otherwise.

**Method description:** This method removes a Sheet Repository and its data from the Rapture system. There is no undo. It does not delete the sheets stored in the repo, so calling this while there are still sheets inside this repo will cause a memory leak.

## doesSheetExist

**Syntax:** Boolean doesSheetExist(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** true if the sheet exists, false otherwise

**Method description:** Checks whether a sheet exists at a given URI.

## doesSheetRepoExist

**Syntax:** Boolean doesSheetRepoExist(String repoURI)

**Parameters:** repoURI: The uri of the repo to check

**Returns:** true if it exists, false otherwise

**Method description:** This APIP call can be used to determine whether a given type exists in a given authority.

## generateScriptToRecreateSheet

**Syntax:** String generateScriptToRecreateSheet(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** A String containing a Reflex script that, when run, will recreate this exact sheet

**Method description:** Returns a String with Reflex code that can be used to create the entire sheet. This is useful if you want to clone a sheet onto a different instance of Rapture, or store it for installation as a feature.

## getAllCells

**Syntax:** RaptureSheetStatus getAllCells(String sheetURI, int dimension, Long epoch)

**Parameters:** - sheetURI: The URI of the sheet

**Returns:** A RaptureSheetStatus object, which contains all the cells as well as the epoch that was used when retrieving these cells.

**Method description:** Return all the cells in the sheet. The epoch is the caller’s best-known understanding as to the state of its knowledge of the sheet. Starting at 0 retrieves every cell. The latest epoch is returned in the response to this call. Once you know the epoch, you can call this method to retrieve the updates since that epoch.

## getAllNotes

**Syntax:** List(RaptureSheetNote) getAllNotes(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** A list of all the notes associated with this sheet

**Method description:** Get all the notes associated with this sheet. See also [createNote](#_createNote_1).

## getAllRanges

**Syntax:** List(RaptureSheetRange) getAllRanges(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** A list of all the ranges associated with this sheet.

**Method description:** Get all the ranges of interest that are marked on this sheet. See also: [createRange](#_createRange_1).

## getAllScripts

**Syntax:** List(RaptureSheetScript) getAllScripts(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** A list of all the scripts associated with this sheet.

**Method description:** Get all the scripts associated with this sheet. See also: [createScript](#sheet_createScript).

## getAllSheetRepoConfigs

**Syntax:** List(SheetRepoConfig) getAllSheetRepoConfigs()

**Returns:** A list of all the sheet repo configs defined in Rapture

**Method description:** Gets the metadata for all sheets.

## getAllStyles

**Syntax:** List(RaptureSheetStyle) getAllStyles(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** A list of all the styles associated with this sheet.

**Method description:** Gets all the style objects associated with this sheet. See also [createStyle](#_createStyle_1).

## getChildren

**Syntax:** List(RaptureFolderInfo) getChildren(String sheetURI)

**Parameters:** - sheetURI: The URI where to start the search

**Returns:** A list of RaptureFolderInfo objects at or below the URI that was passed in

**Method description:** Returns a list of full display names of the paths below this one.

## getNamedSheetCell

**Syntax:** String getNamedSheetCell(String sheetURI, String rangeName, int dimension)

**Parameters:** - sheetURI: The uri of the sheet

- rangeName: The name of the range

- dimension: The dimension where this cell lives. Equivalent to a “tab” in the sheet, from a GUI standpoint.

**Returns:** The value of the cell, or empty string if there is no range with that name

**Method description:** Gets the contents of the cell in the first row, of the first column, of a range on a cell, in the specified dimension.

## getSheetAsDisplay

**Syntax:** RaptureSheetDisplayForm getSheetAsDisplay(String sheetURI)

**Parameters:** - sheetURI: The uri of the sheet

**Returns:** A RaptureSheetDisplayForm object, which is a wrapper around the formatted cells and styles

**Method description:** Get this sheet in a display form - filling in the blanks, and setting the format where appropriate, and passing a list of the formats in play. This should be sufficient for a client side renderer to do the necessary work with no further calls needed.

## getSheetCell

**Syntax:** String getSheetCell(String sheetURI, int row, int column, int dimension)

**Parameters:**

- sheetURI: The uri of the sheet

- row: The row of the cell

- column: The column of the cell

- dimension: The dimension where this cell lives. You can think of this as a “tab” in the sheet, from a gui standpoint.

**Returns:** The value of the cell, or null if not set

**Method description:** Retrieves the data in a cell.

## getSheetRangeByCoords

**Syntax:** List(RaptureSheetRow) getSheetRangeByCoords(String sheetURI, int startRow, int startColumn, int endRow, int endColumn)

**Parameters:** - sheetURI: The uri of the sheet

- startRow: The first row of the rectangle

- startColumn: The first column of the rectangle

- endRow: The last row of the rectangle

- endColumn: The last column of the rectangle

**Returns:** A list of RaptureSheetRow objects, each of which contains all the coordinates and values for cells in one row

**Method description:** Return the values for a rectangle of sheet cells. The coordinates of the rectangle are specified in the inputs. This only returns values in dimension 0 (the default dimension).

## getSheetRangeByName

**Syntax:** List(RaptureSheetRow) getSheetRangeByName(String sheetURI, String rangeName)

**Parameters:** - sheetURI: The uri of the sheet

- rangeName: The name of the range that we want to retrieve

**Returns:** A list of RaptureSheetRow objects, each of which contains all the coordinates and values for cells in one row

**Method description:** Get the contents of a specified range on a sheet.

## getSheetRepoConfig

**Syntax:** SheetRepoConfig getSheetRepoConfig(String sheetURI)

**Parameters:** - sheetURI: The uri of the repository

**Returns:** A SheetRepoConfig object with the details for this repository

**Method description:** Fetches the sheet repository’s config metadata.

## getSheetScript

**Syntax:** RaptureSheetScript getSheetScript(String sheetURI, String scriptName)

**Parameters:** - sheetURI: The uri of the sheet

- scriptName: The name of the script

**Returns:** The RaptureSheetScript object, or null if none found

**Method description:** Gets a particular script associated with this sheet.

## removeNote

**Syntax:** Boolean removeNote(String sheetURI, String noteId)

**Parameters:** - sheetURI: The uri of the sheet

- noteId: The name of the note

**Returns:** true if found and removed, false otherwise

**Method description:** Deletes a note associated with this sheet.

## removeRange

**Syntax:** Boolean removeRange(String sheetURI, String rangeName)

**Returns:** true if found and removed, false otherwise

**Method description:** Deletes a range associated with this sheet.

## removeStyle

**Syntax:** Boolean removeStyle(String sheetURI, String styleName)

**Parameters:** - sheetURI: The uri of the sheet

- styleName: the name of the style

**Returns:** true if found and removed, false otherwise

**Method description:** Removes a style associated with this sheet.

## removeScript

**Syntax:** Boolean removeScript(String sheetURI, String scriptName)

**Parameters:** -sheetURI: The uri of the sheet

- scriptName: The name of the script

**Returns:** true if found and removed, false otherwise

**Method description:** Removes a script associated with this sheet.

## renderSheet

**Syntax:** Boolean renderSheet(String sheetURI, String blobURI)

**Parameters:** -sheetURI: The uri of the sheet

-blobURI: The Blob URI where the pdf should be stored, as a blob

**Returns:** true

**Method description:** Renders the sheet to a PDF document, and saves the PDF as a blob in Rapture.

## runScriptOnSheet

**Syntax:** Boolean runScriptOnSheet(String sheetURI, String scriptName)

**Parameters:** - sheetURI: The uri of the sheet

- scriptName: The name of the script

**Returns:** true

**Method description:** Runs a script that’s associated with this sheet.

## setBlock

**Syntax:** Boolean setBlock(String sheetURI, int startRow, int startColumn, List(String) values, int height, int width, int dimension)

**Parameters:** - sheetURI: The uri of the sheet

- startRow: The first row to set

- startColumn: The first column to set

- values: A 1-dimensional list of values to set. The values are wrapped into multiple rows based on the width

- height: This is ignored

- width: The width of the rectangle of cells that is being set

- dimension: The dimension where this cell lives. You can think of this as a “tab” in the sheet, from a GUI standpoint.

**Returns:** true

**Method description:** Sets values in bulk for a rectangle of cells. The inputs are specified as a 1-dimensional list, and the list is wrapped based on the width parameter. See also [setBulkSheetCell](#_setBulkSheetCell_1).

## setBulkSheetCell

**Syntax:** Boolean setBulkSheetCell(String sheetURI, int startRow, int startColumn, List(List(String)) values, int dimension)

**Parameters:** - sheetURI: The uri of the sheet

- startRow: The first row to set

- startColumn: The first column to set

- values: A 2-dimensional list of values to set. The size of this determines how many cells will be set.

- dimension: The dimension where this cell lives. You can think of this as a “tab” in the sheet, from a GUI standpoint.

**Returns:** true

**Method description:** Sets values in bulk for a rectangle of cells. See also: [setBlock](#_setBlock_1).

## setNamedSheetCell

**Syntax:** String getNamedSheetCell(String sheetURI, String rangeName, String value, int dimension)

**Parameters:** - sheetURI: The uri of the sheet

- rangeName: the name of the range

- value: the value to set

- dimension: The dimension where this cell lives. You can think of this as a “tab” in the sheet, from a GUI standpoint.

**Returns:** The value that was passed in, or an empty string if the range is not found

**Method description:** Assigns data to the cell on the first row and first column of a given sheet, on the specified dimension.

## setSheetCell

**Syntax:** String setSheetCell(String sheetURI, int row, int column, String value, int dimension)

**Parameters:**

- sheetURI: The uri of the sheet

- row: The row of the cell

- column: The column of the cell

- dimension: The dimension where this cell lives. You can think of this as a “tab” in the sheet, from a GUI standpoint.

**Returns:** The value that was passed in

**Method description:** Assigns the value passed to the cell.

# Table

For manipulating Rapture tables. Tables are data warehouses in Rapture that are used primarily internally to manage things such as indexes on other data. The terms index and table are used interchangeably in Rapture, which might be confusing.

## createTable

**Syntax:** RaptureTableConfig createTable(String indexURI, String config)

**Parameters:** -indexURI: The uri for this table (aka index)

- config: The config for this table. For example:

TABLE {} USING MONGODB { prefix="aPrefix" }

**Returns:** The RaptureTableConfig object that was created.

**Method description:** Creates a RaptureTableConfig object with the specified configuration and stores it in the config repo.

|  |
| --- |
|  |

## deleteTable

**Syntax:** Boolean deleteTable(String indexURI, TableQuery query)

**Parameters:**  -indexURI: The uri for this table (aka index)

**Method description:** Deletes a RaptureTableConfig from the config repo.

## getTable

**Syntax:** RaptureTableConfig getTable(String indexURI)

**Parameters:** -indexURI: The uri for this table (aka index)

**Returns:** a RaptureTableConfig object for the uri, or null if none are found

**Method description:** Gets the config for a specified table.

## getTablesForAuthority

**Syntax:** List(RaptureTableConfig) getTablesForAuthority(String authority)

**Parameters:** authority: The authority to query

**Returns:** A list of RaptureTableConfig objects that share this authority

**Method description:** Gets the configs for all tables in a given authority.

## queryTable

**Syntax:** List(TableRecord) queryTable(String indexURI, TableQuery query)

**Parameters:** -indexURI: The uri for this table (aka index)

-query: The TableQuery object that will be used to run thq query

**Returns:** A list of TableRecord objects, i.e. results found based on the query

**Method description:** Runs a TableQuery on a given table/index uri.

# User

The user API contains methods for users to query and manipulate their own accounts and sessions.

|  |
| --- |
|  |

## changeMyPassword

**Syntax:** RaptureUser changeMyPassword(String oldHashPassword, String newHashPassword)

**Parameters:** MD5s of both old and new passwords

**Returns:** the updated user record

**Method description:** Changes the password of the current user.

## getPreference

**Syntax:** String getPreference(String category, String name)

**Parameters:** category: the preference category

name: the name of the preference to retrieve

**Returns:** the value of the preferences

**Method description:** Retrieves app preferences.

## getPreferenceCategories

**Syntax:** List(String) getPreferenceCategories()

**Returns:** The list of categories

**Method description:** This method will list the categories of preferences available for a user.

## getPreferencesInCategory

**Syntax:** List(String) getPreferencesInCategory(String category)

**Parameters:** category: the preference category

**Returns:** a list of all preferences in the category

**Method description:** This method will list the preference documents in a category.

## getServerApiVersion

**Syntax:** ApiVersion getServerApiVersion()

**Returns:** the API version

**Method description:** Returns the API version currently in use.

## getWhoAmI

**Syntax:** RaptureUser getWhoAmI()

**Returns:** A user account object

**Method description:** Returns account information for the current user.

## logoutUser

**Syntax:** Boolean logoutUser()

**Returns:** true or exception

**Method description:** Logs out the active user and terminates the current session.

## removePreference

Syntax: Boolean removePreference(String category, String name)

**Parameters:** category: the preference category

name: the name of the preference to remove

**Returns:** true or exception

**Method description:** Removes a previously stored preference.

## storePreference

**Syntax:** Boolean storePreference(String category, String name, String content)

**Parameters:** category: the preference category

name: the name of the preference to set

content: the value to set the preference to

**Returns:** true or exception

**Method description:** Stores application preferences for current user.

## updateMyDescription

**Syntax:** RaptureUser updateMyDescription(String description)

**Parameters:** free user text

**Returns:** the updated user record

**Method description:** Updates the description for the current user.

# Appendix: List of API Data Types and Calls

**Data Types**

ApiVersion

AppConfig

AppInstanceConfig

AppRepoSettings

AppStatus

AppStatusDetails

AppStatusGroup

AuditLogEntry

AuditLogConfig

BlobContainer

BlobRepoConfig

CallingContext

CategoryQueueBindings

CommitObject

ContentEnvelope

CreateResponse

DocumentMetadata

DocumentObject

DocumentRepoConfig

DocumentWithMeta

EnvironmentInfo

ErrorWrapper

ExchangeDomain

ExecutionContext

FeatureConfig

FeatureManifest

FeatureManifestItem

FeatureTransportItem

FeatureVersion

HooksConfig

IndexConfig

JavaInvocable

JobErrorAck

JobLink

JobLinkStatus

LastJobExec

LicenseInfo

LockHandle

NotificationInfo

NotificationResult

PipelineTaskStatus

PropertyBasedSemaphoreConfig

QCallback

QDetail

QNotification

QTemplate

Question

QuestionSearch

RaptureActivity

RaptureApplicationDefinition

RaptureApplicationInstance

RaptureApplicationStatus

RaptureApplicationStatusStep

RaptureAuthority

RaptureCommit

RaptureContextInfo

RaptureCubeResult

RaptureDNCursor

RaptureDocConfig

RaptureEntitlement

RaptureEntitlementGroup

RaptureEvent

RaptureExchange

RaptureField

RaptureFolderInfo

RaptureFountainConfig

RaptureFullTextIndexConfig

RaptureInstanceCapabilities

RaptureIPWhiteList

RaptureJob

RaptureJobExec

RaptureLibraryDefinition

RaptureLockConfig

RaptureMailMessage

RaptureNetwork

RaptureNotificationConfig

RaptureOperation

RaptureParameter

RapturePipelineTask

RaptureProcessGroup

RaptureProcessInstance

RaptureQueryResult

RaptureRelation

RaptureRelationship

RaptureRelationshipRegion

RaptureRemote

RaptureRunnerConfig

RaptureRunnerInstanceStatus

RaptureRunnerStatus

RaptureScript

RaptureScriptLanguage

RaptureScriptPurpose

RaptureSearchResult

RaptureServerGroup

RaptureServerInfo

RaptureServerStatus

RaptureSheet

RaptureSheetCell

RaptureSheetDisplayCell

RaptureSheetDisplayForm

RaptureSheetNote

RaptureSheetRange

RaptureSheetRow

RaptureSheetScript

RaptureSheetStatus

RaptureSheetStyle

RaptureSnippet

RaptureTableConfig

RaptureUser

ReflexREPLSession

RelationshipRepoConfig

REPLVariable

RepoConfig

ScriptResult

SemaphoreAcquireResponse

SemaphoreLock

SeriesDoubles

SeriesRepoConfig

SeriesStrings

ServerCategory

SheetRepoConfig

Step

StepRecord

TableQuery

TableQueryResult

TableRecord

TimedEventRecord

TimeProcessorStatus

Transition

TreeObject

TypeArchiveConfig

UpcomingJobExec

Worker

Workflow

WorkflowBasedSemaphoreConfig

WorkflowExecStatus

WorkflowJobDetails

WorkflowJobExecDetails

WorkOrder

WorkOrderCancellation

WorkOrderDebug

WorkOrderSearch

WorkOrderStatus

XferDocumentAttribute

XferSeriesValue

**Admin**

addIPToWhiteList

addMetadata

addRemote

addTemplate

addUser

clearRemote

copyDocumentRepo

deleteUser

destroyUser

doesUserExist

generateApiUser

getAllUsers

getEnvironmentName

getEnvironmentProperties

getIPWhiteList

getMOTD

getRemotes

getRepoConfig

getSessionsForUser

getSystemProperties

getTemplate

getUser

ping

pullRemote

removeIPFromWhiteList

removeRemote

resetUserPassword

restoreUser

retrieveArchiveConfig

runBatchScript

runTemplate

setEnvironmentName

setEnvironmentProperties

setMOTD

setRemote

storeArchiveConfig

updateRemoteApiKey

updateUserEmail

**Async**

asyncBatchSave

asyncExecuteRemote

asyncOperation

asyncReflexReference

asyncReflexScript

asyncSave

asyncStatus

executeFolderQueryScript

**Audit**

createAuditLog

deleteAuditLog

doesAuditLogExist

getAuditLog

getChildren

getEntriesSince

getRecentLogEntries

setup

writeAuditEntry

**Blob**

appendToBlob

createBlob

createBlobRepo

deleteBlob

deleteBlobRepo

doesBlobRepoExist

getAllBlobRepoConfigs

getAllChildrenMap

getBlob

getBlobRepoConfig

getBlobSize

getChildren

getMetaData

storeBlob

**Bootstrap**

addScriptClass

getConfigRepo

getEphemeralRepo

getScriptClasses

getSettingsRepo

migrateConfigRepo 66

migrateEphemeralRepo

migrateSettingsRepo

restartBootstrap

removeScriptClass

setConfigRepo

setEphemeralRepo

setSettingsRepo

**Decision**

addErrorToContext

addStep

addTransition 68

cancelWorkOrder

createWorkOrder

createWorkOrderP

defineWorkflow

deleteWorkflow

getAllWorkflows

getAppStatuses

getAppStatusDetails

getCancellationDetails

getContextValue

getErrorsFromContext

getStepCategory

getWorker

getWorkflow

getWorkflowChildren

getWorkflowStep

getWorkOrder

getWorkOrderChildren

getWorkOrderDebug

getWorkOrdersByDay

getWorkOrderStatus

putWorkflow

releaseWorkOrderLock

removeStep

removeTransition

setContextLiteral

setContextLink

setWorkOrderFountainConfig

wasCancelCalled

writeWorkflowAuditEntry

**Document**

addDocumentAttribute

addDocumentAttributes

archiveVersions

attachFountainToDocumentRepo

batchGet

batchExist

batchPutContent

batchRenameContent

createDocumentRepo

deleteContent

deleteDocumentRepo

doesDocumentRepoExist

folderQuery

getAllChildrenMap

getAllDocumentRepoConfigs

getAttachedFountain

getChildren

getContent

getDocumentAttribute

getDocumentAttributes

getdocumentRepoConfig

getDocumentRepositoryStatus

getFountainURI

getMetaContent

getMetaData

putContent

putContentWithVersion

removeDocumentAttribute

removeFolder

renameContent

revertDocument

validate

**Entitlement**

Concepts/Terminology

addGroupToEntitlement

addEntitlement

addEntitlementGroup

addUserToEntitlementGroup

deleteEntitlement

deleteEntitlementGroup

getEntitlement

getEntitlementByAddress

getEntitlementGroup

getEntitlementGroupByAddress

getEntitlementGroups

getEntitlements

removeGroupFromEntitlement

removeUserFromEntitlementGroup

**Environment**

getApplianceMode

getLicenseInfo

getNetworkInfo

getServers

getServerStatus

getThisServer

setApplianceMode

setNetworkInfo

setThisServer

**Event**

attachMessageToEvent

attachNotificationToEvent

attachScriptToEvent

attachWorkflowToEvent

delete

fireEvent

get

getChildren

put

removeMessageFromEvent

removeNotificationFromEvent

removeScriptFromEvent

removeWorkflowFromEvent

**Feature**

doesFeatureNeedToBeInstalled

downloadURI

getFeature

getFeatureManifest

getInstalledFeatures

installFeature

installFeatureItem

removeFeatureManifest

uninstallFeature

uninstallFeatureItem

unrecordFeature

verify

**Fields**

delete

exists

get

getChildren

put

retrieveFieldsFromContent

retrieveFieldsFromDocument

**Fountain**

createFountain

deleteFountain

doesFountainExist

getFountain

getFountains

incrementFountain

resetFountain

**Index**

createIndex

deleteIndex

getIndex

queryIndex

**Lock**

acquireLock

acquireLockWithContext

breakLock

createLockProvider

deleteLockProvider

doesLockProviderExist

getLockProvider

getLockProvidersForAuthority

releaseLock

releaseLockWithContext

setupDefaultProviders

**Mailbox**

getMailboxMessages

moveMailboxMessage

postMailboxMessage

setMailboxStorage

**Notification**

createNotificationProvider

deleteNotificationProvider

doesNotificationProviderExist

finishActivity

getActivities

getAllNotificationProviders

getChanges

getChildren

getEpoch

getNotification

getNotificationProvider

getNotificationProviders

publishNotification

recordActivity

requestAbortActivity

updateActivity

**Pipeline**

broadcastMessageToCategory

broadcastMessageToAll

deregisterExchangeDomain

deregisterPipelineExchange

drainPipeline

getBoundExchanges

getExchange

getExchangeDomains

getExchanges

getLatestTaskEpoch

getServerCategory

getStatus

publishMessageToCategory

queryTasks

registerExchangeDomain

removeServerCategory

setupStandardCategory

**Question**

answerQuestion

askQuestion

defineTemplate

getQNotificationURIs

getQNotifications

getQuestion

getTemplate

**Relationship**

createRelationship

createRelationshipRepo

deleteRelationship

deleteRelationshipRepo

doesRelationshipRepoExist

getAllRelationshipRepoConfigs

getChildren

getInboundRelationships

getLabeledRelationships

getOutboundRelationships

getRelationship

getRelationshipCenteredOn

getRelationshipRepoConfig

**Repository**

deleteContent

getContent

putContent

**Runner**

addGroupInclusion

addGroupExclusion

addLibraryToGroup

changeApplicationStatus

cleanRunnerStatus

createApplicationDefinition

createApplicationInstance

createLibraryDefinition

createServerGroup

getAllApplicationDefinitions

getAllApplicationInstances

getAllServerGroups

getAllLibraryDefinitions

getApplicationsForServer

getApplicationsForServerGroup

getApplicationStatus

getApplicationStatusDates

getApplicationStatuses

getCapabilities

getRunnerConfig

getRunnerServers

getRunnerStatus

getServerGroup

markForRestart

recordInstanceCapabilities

recordRunnerStatus

recordStatusMessage

removeApplicationDefinition

removeApplicationInstance

removeGroupInclusion

removeGroupEntry

removeGroupExclusion

removeLibraryDefinition

removeLibraryFromGroup

removeRunnerConfig

removeServerGroup

retrieveApplicationDefinition

retrieveApplicationInstance

retrieveLibraryDefinition

runApplication

runCustomApplication

setRunnerConfig

terminateApplication

updateApplicationVersion

updateLibraryVersion

**Schedule**

ackJobError

activateJob

createJob

createWorkflowJob

deactivateJob

deleteJob

getCurrentWeekTimeRecords

getJobExecs

getJobs

getNextExec

getUpcomingJobs

getWorkflowExecsStatus

resetJob

retrieveJob

retrieveJobExec

runJobNow

**Script**

archiveOldREPLSessions

checkScript

createREPLSession

createScript

createScriptLink

createSnippet

deleteScript

deleteSnippet

destroyREPLSession

doesScriptExist

evaluateREPL

getChildren

getSnippet

getSnippetChildren

getScript

getScriptNames

putScript

removeFolder

removeScriptLink

runScript

runScriptExtended

setScriptParameters

**Series**

addDoubleToSeries

addDoublesToSeries

addLongToSeries

addLongsToSeries

addStringToSeries

addStringsToSeries

addStructureToSeries

addStructuresToSeries

createSeriesRepo

deleteSeriesRepo

doesSeriesRepoExist

dropAllPointsFromSeries

dropPointsFromSeries

executeSeriesProgram

executeSeriesProgramQuietly

getAllChildrenMap

getAllFromRange

getAllFromRangeAsDoubles

getAllFromRangeAsStrings

getAllPoints

getAllPointsAsDoubles

getAllPointsAsStrings

getAllSeriesRepoConfigs

getChildren

getLastPoint

getPoints

getPointsAsDoubles

getPointsAsStrings

getPointsReverse

getRange

getRangeAsDoubles

getRangeAsStrings

getSeriesRepoConfig

removeFolder

validateSeriesRepo

**Sheet**

cloneSheet

createNote

createRange

createSheet

createSheetRepo

createStyle

createScript

deleteCell

deleteColumn

deleteRow

deleteSheet

deleteSheetRepo

doesSheetExist

doesSheetRepoExist

generateScriptToRecreateSheet

getAllCells

getAllNotes

getAllRanges

getAllScripts

getAllSheetRepoConfigs

getAllStyles

getChildren

getNamedSheetCell

getSheetAsDisplay

getSheetCell

getSheetRangeByCoords

getSheetRangeByName

getSheetRepoConfig

getSheetScript

removeNote

removeRange

removeStyle

removeScript

renderSheet

runScriptOnSheet

setBlock

setBulkSheetCell

setNamedSheetCell

setSheetCell

**Table**

createTable

deleteTable

getTable

getTablesForAuthority

queryTable

**User**

changeMyPassword

getPreference

getPreferenceCategories

getPreferencesInCategory

getServerApiVersion

getWhoAmI

logoutUser

removePreference

storePreference

updateMyDescription