Balrog Documentation

Release 2.74

Mozilla

Sep 10, 2019

CONTENTS

1 Contributing to Balrog **3**
- 1.1 Requirements . . . 3
- 1.2 Cloning the Repository . . . 3
- 1.3 Usage . . . 3
- 1.4 Running Tests . . . 4
- 1.5 Finding Bugs to Solve . . . 4
- 1.6 IRC best practices . . . 4
- 1.7 Git workflow . . . 5
- 1.8 Submitting your work . . . 5

2 Running without Docker Compose **7**
- 2.1 Creating a database . . . 7
- 2.2 Environment Variable . . . 7

3 Code Overview **9**

4 Database Model **11**
- 4.1 Rules . . . 11
- 4.2 Releases . . . 14
- 4.3 Permissions . . . 14
- 4.4 User Roles . . . 15
- 4.5 Required Signoffs . . . 15
- 4.6 History Tables . . . 16
- 4.7 Releases History . . . 16
- 4.8 Scheduled Changes . . . 16

5 Admin API **19**
- 5.1 Rules . . . 19
- 5.2 Releases . . . 22
- 5.3 Users . . . 26
- 5.4 Product Required Signoffs . . . 28
- 5.5 Permissions Required Signoffs . . . 29
- 5.6 Scheduled Changes . . . 30
- 5.7 Others . . . 32

6 Blobs **35**
- 6.1 App Release . . . 35
- 6.2 GMP/System Addons . . . 37
- 6.3 Superblobs . . . 37

7 Balrog Agent **39**

8 Developer Documentation (Auslib Package) 41
8.1 auslib.blobs . . . 41
8.2 auslib.db . . . 49
8.3 auslib.util . . . 66
8.4 auslib.web . . . 68
8.5 auslib.AUS . . . 68
8.6 auslib.config . . . 69
8.7 auslib.dockerflow . . . 69
8.8 auslib.errors . . . 70
8.9 auslib.global_state . . . 70
8.10 auslib.log . . . 70

9 Indices and tables 73

Python Module Index 75

Index 77

Balrog is the software that runs the server side component of the update system used by Firefox and other Mozilla products. It is the successor to AUS (Application Update Service), which did not scale to our current needs nor allow us to adapt to more recent business requirements. Balrog helps us ship updates faster and with much more flexibility than we've had in the past.

Contents:

CHAPTER

ONE

CONTRIBUTING TO BALROG

If you like to get involved in the development of Balrog, there're lots of areas where we could use some help.

1.1 Requirements

These instructions assume you have the following installed (if not, you can follow the links for installation instructions).

- Git
- Docker
- Tox (Optional)

1.2 Cloning the Repository

- Fork the Balrog Repository on GitHub.
- Clone the fork using

```
$   git clone git@github.com:<user-id>/balrog.git
```

By creating a fork, you are able to generate a *pull request* so that the changes can be easily seen and reviewed by other community members.

1.3 Usage

Run the following command to create and run the necessary images:

```
$ docker-compose up
```

After it completes, you should be able to access

- The admin interface at https://localhost:8010
- The public interface at http://localhost:9010, for example http://localhost:9010/update/3/Firefox/33.0/20141202185629/Darwin_x86_64-gcc3-u-i386-x86_64/en-US/release/default/default/default/update.xml?force=1

You'll need to use the "Sign in..." button to do anything useful with the admin interface, which will ask you to sign in with a third party provider (eg: gmail, github). Once you've done that, run the following to create a local admin user to gain write access:

```
$ export LOCAL_ADMIN=<email address you signed in with>
$ docker-compose run balrogadmin create-local-admin
```

By default, Balrog will import a copy of the production database, so you can more easily replicate issues found in production. Also by default, releases history will be read from the production Google Cloud Storage buckets, but you will not be able to write to it. If you need to do so for some reason, you will need to create your own buckets, and provide a *google.json* file in the root of the repository that can read and write to your buckets. More details on that can be found in *docker-compose.yml*.

1.4 Running Tests

It is a good idea to run all tests to see if Balrog is running properly.

To execute all tests, run

```
$ ./run-tests.sh
```

For executing test only for backend, run

```
$ ./run-tests.sh backend
```

For executing test only for frontend, run

```
$ ./run-tests.sh frontend
```

In case you can't or don't want to run tests within Docker for some reason, tests can also run by using tox

```
$ tox
```

Tests run fine on any posix-like environment, but are only run regularly within the Docker image, so it's possible to have failures that aren't related to Balrog code or your changes when running directly with tox.

1.5 Finding Bugs to Solve

To start with, it's recommended that you look at a Good First Bug. Once you're more comfortable with Balrog, we've got a long list of other bugs ready to be worked on.

Once you have decided the work a bug you can comment on the Bug about any questions you have related to it. You can also ask in irc://irc.mozilla.org/#balrog for help.

1.6 IRC best practices

If you want to message someone directly within the channel, precede your message with their nick:. This will alert the person that they have a message especially for them, and makes it easier for the person receiving the message to read it in a busy channel. For example:

```
johnsmith_: your message/query
```

1.7 Git workflow

When you want to start contributing, you should create a branch from master. This allows you to work on different projects at the same time.

```
$   git checkout master
$   git checkout -b topic-branch
```

Once you're done with your changes, you'll need to describe those changes in the commit message.

1.8 Submitting your work

Once you have made the required changes, make sure all the tests still pass. Then, you should submit your work with a pull request to master. Make sure you reference the Bug number on the Pull Request. Now, you have to wait for the review.

Once your code has been positively reviewed, it will be deployed shortly after. So if you want feedback on your code, but it's not ready to be deployed, you should note it in the pull request.

CHAPTER

TWO

RUNNING WITHOUT DOCKER COMPOSE

Generally, this is only done in production. For local development, Docker Compose is always recommended.

2.1 Creating a database

Balrog's database is controlled through sqlalchemy-migrate. To initialize a new Balrog database, run the following:

```
docker run --entrypoint python mozilla/balrog /app/scripts/manage-db.py -d DBURI␣
↪create
docker run -e "DBURI=<database uri>" -e "LOCAL_ADMIN=<email address of initial admin>
↪" mozilla/balrog create-local-admin
```

Similarly, to upgrade the schema of an existing Balrog database, run the following:

```
docker run --entrypoint python mozilla/balrog /app/scripts/manage-db.py -d DBURI␣
↪upgrade
```

See the "Environment Variables" section below for DBURI format. If your testing out local changes that affect database creation or upgrades, you should replace "mozilla/balrog" with your local image.

2.2 Environment Variable

The following environment variables are required by the Balrog WSGI apps:

- **DBURI** - The database to use, in the format: driver://user:password@host/database.
- **SECRET_KEY** - A pseudorandom string to use when generating CSRF tokens. Only used for the admin app.

These are optional:

- **LOG_LEVEL** - Controls the python level the app logs at. Set to INFO by default.
- **LOG_FORMAT** - Controls the log format. If unset, mozlog format (json) will be used. Can be overridden with "plain" to log simple plain-text messages. The former is recommended for production, the latter for local development.
- **NOTIFY_TO_ADDR** - An address to send an e-mail to if a Rule or Permission changes. Unset by default, and only used for the admin app. If set, the following additional variables are required:
  - **SMTP_HOST, SMTP_PORT, SMTP_USERNAME, SMTP_PASSWORD** - Information about the SMTP relay to send mail through.
  - **NOTIFY_FROM_ADDR** - The "from" address to use when sending mail.

CHAPTER

THREE

CODE OVERVIEW

Balrog's code is divided into the following parts:

- Blobs: These contain most of the brains (business logic) behind Balrog. They know how to validate new data coming into the system and translate existing data into useful responses to update requests.
- Database Abstraction Layer:This layer sits between the actual database and the applications. It defines the database schema, performs permissions checking, and ensures all changes are written to history tables. Higher level code should never touch the database directly - they should always go through this layer.
- User-facing application: The entry point to requests from applications looking for updates.
- Admin API: A simple RESTful API that allows the Admin UI and automation to make changes to Balrog's database.
- The Admin UI : A human-friendly interface to manage updates.
- The Balrog Agent: A long running process that is responsible for enacting Schedule Changes.

CHAPTER

FOUR

DATABASE MODEL

Balrog's model centers around two concepts: Rules and Releases. When a request for an update from an application is received it is matched up against the rules. Once the correct rule has been found, it contains a pointer to a Release, which contains all of the metadata needed to construct a proper update response. Rules and Releases are described in greater detail below:

4.1 Rules

The most important part of Balrog to understand is its rules. When a request comes in it is matched against Balrog's rules to find the one that best suits it (more on this in *How are requests match to a rule?*). Once found, Balrog looks at that rule's "mapping", or "fallbackMapping", which points to a release that has the required information to serve an update back to the client. Without any rules, Balrog will never serve an update. With badly configured rules Balrog could do bad things like serve Firefox updates to Thunderbird users.

4.1.1 What's in a rule?

Each rule has multiple columns. They all fall into one of the following Category:

- **Matchable** : These correspond to information provided in the update request, and are used to filter out rules that don't apply to the request
- **Decision** : These are also used to filter rules, but do not correspond to information in the request
- **Response** : these contain information that ends up in the response
- **Info** : Informational columns, not used as part of serving updates

Following tables show columns according to different Categories:

Category	Attribute	Description	Matching Logic	Examples
Decision	backgroundRate	The percentage of background update requests that if specified. Generally, this is used as a throttle to increase or decrease the rate at which the majority of users receive the latest update.	N/A	Any number 0 to 100
	Priority	The priority of the rule, relative to other rules. If multiple rules match an incoming request based on the Matchable columns, the rule with the highest priority is chosen.	N/A	Any number, by convention positive integers.
Info	Alias	The id of the rule. This id is necessary to make changes to the rule through the REST API.	N/A	"firefox-release-betatest", "firefox-nightly"
	Comment	A string describing the purpose of the rule. Not always necessary for obvious rules.	N/A	Any string
	id	The id of the rule. This id is necessary to make changes to the rule through the REST API.	N/A	Autoincrementing integer
Matchable	buildID	The build ID of the application requesting an update.	Exact string match or operator plus buildid to compare the incoming one against	"201410010830" or "<201512010830"
	buildTarget	The "build target" of the application requesting an update. This is usually related to the target platform the app was built for.	Exact string match only	"Darwin_x86_64-gcc3-u-i386-x86_64" or "flame-kk-userdebug"
	channel	The update channel of the application request an update.	Exact string match or a string with "*" character to glob	"nightly" or "beta*"
	distribution	The partner distributions names that the application must send in order for the rule to match or "default" if the application is not a partner build. A comma separated list may be used to list multiple distributions	Exact string match or comma separated list of distributions to do an exact match on	"default", "yahoo" or "mozilla1, mozilla2"
	distVersion	The version of the partner distribution of the application requesting an update or "default" if the application is not a partner build.	Exact string match only	"default" or "1.19"
	headerArchitecture	The architecture of the OS of the client as guessed based on build target. This field is mostly deprecated now that this information is included in the build target.	Exact string match only	"PPC" and "Intel" are the only possible values
	locale	The locale of the application requesting an update.	Exact string match or comma separated list of locales to do an exact match on	"de" or "en-US,en-GB,id"
	osVersion	The OS Version of the application requesting an update. This field is primarily used to point desupported operating systems to their last supported build.	Simplified boolean string match. '&&' ANDs terms which '' ORs them. Terms are matched using partial strings.	"Windows_NT 5.0" or "Darwin 6.[0-9]+" or "Windows 7," or "Windows && (websense-"

4.1.2 How are requests match to a rule?

Most of the Matchable database fields are present as distinct parts of the update URL. For example, most update requests will send a URL in the following format

```
/update/6/<product>/<version>/<buildID>/<buildTarget>/<locale>/<channel>/<osVersion>/
↪<systemCapabilities>/<distribution>/<distVersion>/update.xml?force=1
```

There are a few special cases to consider:

- systemCapabilities contains comma separated data and breaks down into multple database columns (instructionSet, memory, jaws)
- headerArchitecture is extracted from the User-Agent header
- mig64 is optional, and comes from the query string instead of the path

The following logic is used to figure out which rule a request matches, and how to respond:

- Retrieve all rules where product, buildTarget, distribution, and distVersion are (each) unspecified, or match the request with a simple string match.
- Discard any rules where the rule specifies a channel, version, buildID, osVersion, any part of systemCapabilities, and/or locale, and that doesn't match the request. The method for each match is described in the table above.
  - The channel has special handling to try "falling back" to a simpler channel, for example a request with release-cck-foo will also consider rules for 'release'. This only applies to channels containing '-cck-'.
- Sort the remaining rules by priority, and keep the one with highest.
- The rule's value for backgroundRate modifies the response
  - if the request has a query parameter force=1 then the background rate is ignored, and all requests will be served using the release in Mapping
  - if force is absent then backgroundRate is the percentage of requests which will be served using Mapping
  - the remaining requests will be served fallbackMapping, if that is specified on the rule, otherwise nothing.
- The Release, combined with the update_type specified by the rule, is used to construct an XML response with the details of the update.

4.1.3 Rules example

Rules are usually set up like this, in increasing order of priority:

- The lowest priority rule is the main path, providing the latest release for a channel
- Special cases are slightly higher, e.g. whatsnew pages for some locales
- Watersheds are higher again, to ensure that older release update to the watershed first. The older the watershed the higher the priority, so that X –> Y –> Z is preserved.
- The oldest operating system deprecations are highest priority.

Here is a simplified set of rules for Firefox on the release channel, with a throttled main release, a Windows-specific watershed, and the deprecation of Windows 98 along time ago. All other values unspecified, except for update_type being 'minor' for all rules.

Priority	Product	Channel	Version	OS Version	Mapping	Fallback Mapping	Background Rate
400	Firefox	release*		Windows_98	No-Update		100
300	Firefox	release	< 43.0.1	Windows_NT	Firefox-43.0.1-build1		100
100	Firefox	release			Firefox-51.0.1-build3	Firefox-50.1.0-build2	25

The first two rules are static, while the last has the two mapping values updated as new releases are created. Future watersheds would be placed with priority below 300, while special cases are closer to 100.

4.2 Releases

To Balrog, a "release" is data about a related set of builds. This does _not_ match up with the concept of a "release" being on the "beta", "release" or "esr" channel elsewhere. In Balrog, each set of nightlies on any branch is considered a release.

While there's no enforced format on release names, there are a few conventions that we use:

- Nightly-style builds submit to releases named by product and branch. Each nightly generally submits to two different releases, one "dated" (eg: Firefox-mozilla-central-nightly-20150513010203) and one "latest" (eg: Firefox-mozilla-central-nightly-latest).
- Release-style builds submit to releases named by product, version number, and build number, eg: Firefox-38.0-build1
- GMP blobs are created by hand and generally named with the version of each plugin they contain in the name, eg: GMP-20150423-CDM-v4-OpenH264-v1.4

4.3 Permissions

The permissions table is a simple list of usernames and the ACLs(Access Control Lists) that they have. A user could be an "admin", giving them write access to everything, or could have one or more specific permissions. These specific ACLs let us do things such as give B2G folks access to Balrog without the risk of them or their tools accidentally messing up Firefox updates.

The table below describe all possible permissions:

Object	Action	Options	Comments
admin	No supported actions	products - If specified, the user can perform any actions on Rules or Releases that affect the specified products.	An admin user with no options specified has completely unrestricted access to Balrog
rule	create	products - If specified, the user only has permission for the object and action if the changes they are making only affect the product specified.	
	modify		
	delete		
release	create		
	modify		
	delete		
release_read_only	set		
	unset		
release_locale	modify		
required_signoff	create		
	modify		
	delete		
permission	create	No supported options.	
	modify		
	delete		
scheduled_change	enact	No supported options.	Only the Balrog Agent should be granted this permission.

4.4 User Roles

Users may hold any number of Roles. Roles are used when signing off on Scheduled Changes.

Roles and Permissions are not directly related - assigning a User a Role does not inherently grant them any Permissions.

4.5 Required Signoffs

Some types of changes to Balrog's database require more than one person to approve them before they can be done. The Required Signoffs tables specify how many signoffs are needed from different Roles for each type of change. For example, a change may required 3 signoffs from users that hold the "releng" Role as well as 1 signoff from a user that holds the "relman" role.

Changes to Required Signoffs tables generally require signoff as well. If you are adding, modifying, or removing signoff requirements for something that already has signoff requirements, you must obtain signoff to do so. For example, if a change requires 2 signoffs from users who hold the "releng" Role, and you want to also require signoff from 1 user who holds the "relman" Role, you must get signoff from 2 "releng" users first. The one exception to this is that if you are adding a new signoff requirement for something that doesn't require any signoff yet, you do not need any signoff to do so.

You cannot require more signoffs than a Role has users. Eg: if only 3 users hold the "releng" Role, you cannot require 4 "releng" signoffs for anything. Similarly, if 3 "releng" signoffs are currently required for something, and 3 users hold that Role, you cannot remove that Role from any user.

Changes that require signoff will either be Product changes or Permissions changes. Required Signoffs for each are managed independently, and described in more detail below.

4.5.1 Product Required Signoffs

Changes that directly affect updates that clients receive (the Rules and Releases tables) are considered Product changes. Our paranoia level for changes to these varies greatly depending on the Product and Channel. Eg: we're far more concerned about changes to Firefox's release channel than we are about Thunderbird's nightly channel. Because of this, we specify Required Signoffs for these with a product and channel combination.

Any changes to a Rule that would affect a product and channel combination specified in this table will require signoff. This includes Rules that don't specify a product or channel at all (because that is treated as a wildcard).

Releases which are mapped to be a Rule's mapping or fallbackingMapping field require the same signoffs as the Rule. Releases that are not mapped to by a rule never require any signoff. It's important that they are inspected before mapping to them for the first time.

If a change affects more than one product and channel combination, *all* affected combinations' required signoffs will be combined to find the full set of required. For example, if Firefox's release channel requires 3 signoffs from "relman" and Firefox's beta channel requires 2 signoffs from "releng", a change to a Rule that affects both channels will require 3 signoffs from "relman" and 2 from "releng". Changing a Release that is mapped to by Rules on the "release" and "beta" channel would also require the same signoffs.

4.5.2 Permissions Required Signoffs

Changes to the Permissions table may also require signoff. These Required Signoffs are specified by product, which most Permissions support as an option. Changing a Permission that affects the named product wil require the signoffs from the Roles specified in this table. Changing a Permission that does not specify a product will require signoff the signoffs from *all* Roles specified in this table, because such Permissions grant access to all products. This includes the "admin" Permission and "permission" Permission, which are often used without a product specified.

4.6 History Tables

Change attribution and recording is embedded deeply into Balrog. The rules, permissions, required signoffs, and all associated scheduled changes tables have a corresponding history table that records the time a change was made and who made it. This allows us to look back in time when debugging issues, attribute changes to people (aka blame), and quickly roll back bad changes.

4.7 Releases History

We also store history for changes to the releases table, but due to its immense size and data type, it is stored in Google Cloud Storage instead of the database. We use two separate buckets for it: one for nightly releases, which has a short expiration window and another for non-nightly releases, which is kept forever.

4.8 Scheduled Changes

Some tables (Rules, Releases, and Permissions) support having changes to them scheduled in advance. Tables with Scheduled Changes enabled will have additional related tables to store the necessary information about them.

The primary Scheduled Changes table stores the desired new version of the object and the user who scheduled it. The Conditions table stores information about when to enact the Scheduled Change. Finally, the Signoffs table stores

information about who (if anybody) has signed off on the Scheduled Change. All of these tables have their own History tables too.

Permissions for Scheduled Changes are inherited from their asociated base table. Eg: to scheduled a change to a Rule, you must have permission to modify that Rule directly. No special permission is required on top of that.

CHAPTER

FIVE

ADMIN API

Balrog's Admin app provides an API that allows for retrieval and modification of Rules, Releases, and Permissions. This page documents all of the available endpoints, their parameters, and responses. POST and PUT requests may submit parameters as multipart form data or json.

5.1 Rules

5.1.1 /rules

GET

Returns all of the Rules in Balrog's database inside of a JSON Object in the following format:

```
{
    "count": 2,
    "rules": [
            {
                "rule_id": 1,
                    ...
            },
            {
                "rule_id": 2,
                ...
            }
    ]
}
```

POST

Creates a new rule with the provided values. The following parameters are supported:

- priority (required)
- backgroundRate (required)
- mapping
- alias
- product
- version

- buildID
- channel
- locale
- distribution
- buildTarget
- osVersion
- distVersion
- comment
- headerArchitecture

For detailed information on parameters see *Rules*

5.1.2 /rules/<id_or_alias>

GET

Returns the entire rule identified by the id or alias given in JSON format. Eg:

```
{
    "rule_id": 3,
    "product": "Firefox",
    ...
}
```

POST

Modifies the rule identified by the id or alias given according to the parameters given. The following parameters are supported:

- data_version (required)
- priority
- backgroundRate
- mapping
- alias
- product
- version
- buildID
- channel
- locale
- distribution
- buildTarget
- osVersion
- distVersion

- comment
- headerArchitecture

For detailed information on parameters see *Rules*

DELETE

Deletes the rule identified by the id or alias given. The following parameters are supported:

- data_version (required)

5.1.3 /rules/<id_or_alias>/revisions

GET

Returns previous versions of the rule identified by the id or alias given in a JSON Object in the following format:

```
{
    "count": 2,
    "rules": [
    {
        "id": 1,
        "change_id": 4,
        "timestamp": 1451610061000,
        "changed_by": "jane",
        "product": "Firefox",
        ...
    },
    {
        "id": 1,
        "change_id": 4,
        "timestamp": 1451610061000,
        "changed_by": "jane",
        "product": "Firefox",
        ...
    }
    ]
}
```

This endpoint supports pagination. If "page" and "limit" are present in the query args, a slice of the revisions is returned instead of the full history. Eg: if the page is "2" and the limit is "5", the 6th through 10th revisions would be returned. "count" is not affected by pagination - it will always return the total number of revisions that exist.

POST

Reverts the rule identified by the given id (alias is not supported here) to the version identified by the change_id given in the request body. The request body must be a JSON object containing a "change_id" key.

5.1.4 /rules/columns/<column>

GET

Returns a JSON Object containing the unique values for the given column. For example, /rules/columns/product would return something like:

```
{
  "count": 10,
  "product": [
    "Firefox",
    "Graphene",
    "Thunderbird",
    "MetroFirefox",
    "Horizon",
    "B2G",
    "GMP",
    "Fennec",
    "SystemAddons",
    "B2GDroid"
  ]
}
```

5.2 Releases

5.2.1 /releases

GET

Returns a JSON Object containing metadata about Releases in Balrog's database. Due to its size, the actual Release "blob" is never returned from this endpoint. There are a few query arguments that affect its response. If no arguments are provided, it returns information about all of the Releases in the database in the following format:

```
{
  "releases": [
    {
      "name": "Firefox-34.0-build1",
      "product": "Firefox",
      "data_version": 4,
      "read_only": null
    },
    {
      "name": "Fennec-34.0-build1",
      "product": "Fennec",
      "data_version": 43,
      "read_only": true
      },
      ...
  ]
}
```

If "product" is passed, only Releases with the given product name will be returned. If "name_prefix" is passed, only Releases whose name starts with the given prefix will be returned. If "names_only" is set to true, the response changes format and provides a list of all of the Release names in the database:

```
{
  "names": [
    "Firefox-34.0-build1",
    "Fennec-34.0-build1",
    ...
   ]
}
```

POST

Creates a new Release with the provided values. The following parameters are supported:

- name (required)
- product (required)
- blob (required)

5.2.2 /releases/<release>

GET

Returns the "data" portion of the named Release, which is a JSON Object. If "pretty" is present in the query string and set to true, it will be pretty formatted. For example:

```
{
  "name": "Firefox-mozilla-central-nightly-latest",
  "schema_version": 4,
  "platforms": {
    "WINNT_x86-msvc": {
    ...
    }
  }
}
```

PUT

Overwrites the named Release with the data given. The "blob" field is completely overridden with the new one, not updated. If the Release does not exist, it is created. The following parameters are supported:

- name (required)
- product (required)
- blob (required)
- data_version (required if the Release already exists)

POST

Updates the named Release with the data given. The "blob" field is updated with the new one instead of being completely overridden. If the Release does not exist, it is created. The following parameters are supported:

- product (required)

- data (required)
- data_version (required if the Release already exists)
- hashFunction
- schema_version
- copyTo
- alias

DELETE

Deletes the named Release. The following parameters are supported:

- data_version (required)

5.2.3 /releases/<release>/read_only

GET

Returns whether or not the named Release is marked as read_only. Eg:

```
{
  "read_only": true
}
```

5.2.4 /releases/<release>/builds/<platform>/<locale>

GET

Returns the platform+locale specific data of the named Release, which is a JSON Object. Eg:

```
{
  "buildID": "20160329030246",
  "appVersion": "48.0a1",
  "displayVersion": "48.0a1",
  "platformVersion": "48.0a1",
  "partials": [
    {
        "fileUrl": "https://mozilla-nightly-updates.s3.amazonaws.com/mozilla-central/
↪20160329030246/Firefox-mozilla-central-48.0a1-win32-de-20160327030437-
↪20160329030246.partial.mar?versionId=uIza17vCjTuL6XVvCvtpzlVVQSelUdJm",
        "from": "Firefox-mozilla-central-nightly-20160327030437",
        "hashValue":
↪"0d36245eedef3bfce927339ee89da58400f8afa5a8cc8b4323f7407660f291bbfa1f00527665d5f16614de679723b874d9
↪",
        "filesize": 10388948
    }
  ],
  "completes": [
    {
        "fileUrl": "https://mozilla-nightly-updates.s3.amazonaws.com/mozilla-central/
↪20160329030246/Firefox-mozilla-central-48.0a1-win32-de.complete.mar?
↪versionId=sdNQURDy9.8GH3P4SLdO1V.XtA9MLIzu",
```

(continues on next page)

(continued from previous page)

```
        "from": "*",
        "hashValue":
↪"981082f1b7f5264d88aa017f45362aac362990842b82a0934e70506c1536304b0fda6beb229b7ef56b153d71b69669cc92
↪",
        "filesize": 53656493
    }
  ]
}
```

PUT

Sets or unsets the read_only flag of the named Release. The following parameters are supported:

- name (required)
- data_version (required)
- read_only

5.2.5 /releases/<release>/revisions

GET

Returns previous versions of the named Release in a JSON Object in the following format:

```
{
  "count": 1
  "rules": [
    {
      "id": 1,
      "change_id": 4,
      "timestamp": 1451610061000,
      "changed_by": "jane",
      "product": "Firefox",
      ...
    }
  ]
}
```

This endpoint supports pagination. If "page" and "limit" are present in the query args, a slice of the revisions are returned instead of the full history. Eg: if the page is "2" and the limit is "5", the 6th through 10th revisions would be returned. "count" is not affected by pagination - it will always return the total number of revisions that exist.

POST

Reverts the named Release to the version identified by the change_id given in the request body. The request body must be a JSON object containing a "change_id" key.

5.2.6 /releases/columns/<column>

GET

Returns a JSON Object containing the unique values for the given column. For example, /releases/columns/product would return something like: vpn

```
{
  "count": 10,
  "product": [
    "Firefox",
    "Graphene",
    "Thunderbird",
    "MetroFirefox",
    "Horizon",
    "B2G",
    "GMP",
    "Fennec",
    "SystemAddons",
    "B2GDroid"
  ]
}
```

5.3 Users

5.3.1 /users

GET

Returns all of the users known to Balrog inside of a JSON Object in the following format:

```
{
  "users": [
    "bhearsum@mozilla.com",
    "ffxbld",
    "nthomas@mozilla.com",
    ...
  ]
}
```

Note that Balrog only tracks permissions, not accounts, so this list does not include users who are able to log in, but have no permissions to change anything.

5.3.2 /users/<username>/permissions

GET

Returns all of the permissions that the given username has been granted in a JSON Object in the following format:

```
{
  "/releases/:name": {
    "data_version": 1,
    "options": {
      "method": "POST,
```

(continues on next page)

(continued from previous page)

```
      "product": [
        "Firefox",
        "Fennec"
      ]
    }
  },
    ...
}
```

5.3.3 /users/<username>/permissions/<permission>

GET

Returns the details of the named permission for the username given in a JSON Object in the following format:

```
{
  "data_version": 1,
  "options": {
    "method": "POST,
    "product": [
      "Firefox",
      "Fennec"
    ]
  }
}
```

PUT

Overwrites the details of named permission for the username given. If the permission does not exist, it is created. The following parameters are supported:

- data_version (required if the permission already exists)
- options

POST

Overwrites the details of named permission for the username given. The following parameters are supported:

- data_version (required)
- options

DELETE

Deletes the named permission for the username given. The following parameters are supported:

- data_version (required)

5.3.4 /users/<username>/roles

GET

Returns all of the roles that the given username holds in a JSON Object in the following format:

```
{
  "roles": [
    "qa",
    "releng"
  ]
}
```

5.3.5 /users/<username>/roles/<role>

PUT

Grants the given username the given role. If the user already holds that role, this is a no-op.

DELETE

Revokes the given role from the given username. The "data_version" parameter must be provided.

5.4 Product Required Signoffs

Endpoints to view and create new *Product Required Signoffs*. In most cases, these will be managed with *Scheduled Changes* instead, because they themselves require signoff.

5.4.1 /required_signoffs/product

GET

Returns all of the *Product Required Signoffs*. Example response:

```
{
  "count": 2,
  "required_signoffs": [
    {
      "product": "Firefox",
      "channel": "release",
      "role": "releng",
      "signoffs_required": 2,
      "data_version": 1
    },
    {
      "product": "Firefox",
      "channel": "release",
      "role": "relman",
      "signoffs_required": 1,
      "data_version": 1
```

(continues on next page)

(continued from previous page)

```
    }
  ]
}
```

POST

Create a new Product Required Signoff. “product”, “channel”, “role”, and “signoffs_required” are all required. If the product and channel provided already require signoff, a 400 will be returned (you must use a Scheduled Change and meet the existing signoff requirements to modify Required Signoffs for things that already require it).

5.5 Permissions Required Signoffs

Endpoints to view and create new *Permissions Required Signoffs*. In most cases, these will be managed with *Scheduled Changes* instead, because they themselves require signoff.

5.5.1 /required_signoffs/permissions

GET

Returns all of the *Permissions Required Signoffs*. Example response:

```
{
  "count": 2,
  "required_signoffs": [
    {
      "product": "Firefox",
      "role": "releng",
      "signoffs_required": 3,
      "data_version": 1
    },
    {
      "product": "SystemAddons",
      "role": "gofaster",
      "signoffs_required": 2,
      "data_version": 1
    }
  ]
}
```

POST

Create a new Permissions Required Signoff. “product”, “role”, and “signoffs_required” are all required. If the product provided already requires a signoff, a 400 will be returned (you must use a Scheduled Change and meet the existing signoff requirements to modify Required Signoffs for things that already require it).

5.6 Scheduled Changes

Endpoints to create and manage *Scheduled Changes* and Signoffs. Each type of object that supports Scheduled Changes has its own set of endpoints. These objects are:

- rules (*Rules*)
- releases (*Releases*)
- permissions (*Permissions*)
- required_signoffs/product (*Product Required Signoffs*)
- required_signoffs/permissions (*Permissions Required Signoffs*)

5.6.1 /scheduled_changes/<object>

GET

Returns the Scheduled Changes for the named object. If the query arg "all" evaluates to True, Scheduled Changes that have been enacted will be returned along with pending ones. If "all" evaluates to False, only active Scheduled Changes will be returned. Example response:

```
{
    "count": 2,
    "scheduled_changes": [
    {
        "sc_id": 1,
        "when": 100000,
        "complete": True,
        "scheduled_by": "janet",
        "signoffs": {
            "janet": "relman"
        },
        # base attributes follow...
    },
    {
        "sc_id": 2,
        "when": 20000000000,
        "complete": False,
        "scheduled_by": "charlie",
        "signoffs": {
            "charlie": "releng",
            "janet": "relman"
        },
        # base attributes follow...
    }
    ]
}
```

POST

Creates a new Scheduled Change for the named object. The following parameters are supported:

- when
- telemetry_product

- telemetry_channel
- telemetry_uptake

Either "when" or the full set of "telemetry" parameters must be provided as a condition. Details of the object to be created or modified are also required. Eg, "product", "channel", "priority", "backgroundRate" and "mapping" might be included if a Scheduled Change for a Rule was being created.

5.6.2 /scheduled_changes/<object>/<sc_id>

POST

Modifies an existing Scheduled Change for the named object. Supported parameters are the same as /scheduled_changes/<object> POST.

DELETE

Deletes the given Scheduled Change for the named object. "data_version" must be provided.

5.6.3 /scheduled_changes/<object>/<sc_id>/enact

POST

Enacts the given Scheduled Change for the named object. This endpoint should only be used by the *Balrog Agent*.

5.6.4 /scheduled_changes/<object>/<sc_id>/signoffs

POST

Signs off on the given Scheduled Change for the named object. "role" must be provided in the request body.

DELETE

Removes an existing Signoff from the Scheduled Change for the named object.

5.6.5 /scheduled_changes/<object>/<sc_id>/<id>/revisions

GET

Returns previous versions of the scheduled change identified by the id given in a JSON Object in the following format:

```
{
    "count": 2,
    "scheduled_changes": [
    {
        "change_id": 4,
        "timestamp": 1451610061000,
        "changed_by": "jane",
        "sc_id": 1,
```

(continues on next page)

(continued from previous page)

```
        "complete": False,
        "when": 1000000000,
        "scheduled_by": "jane",
        "signoffs": {
            "jane": "relman"
        },
        # base attributes follow...
    },
    {
        "change_id": 4,
        "timestamp": 1451610061000,
        "changed_by": "jane",
        "sc_id": 1,
        "complete": False,
        "when": 2000000000,
        "scheduled_by": "jane",
        "signoffs": {
            "jane": "relman"
        },
        # base attributes follow...
    }
    ]
}
```

This endpoint supports pagination. If "page" and "limit" are present in the query args, a slice of the revisions is returned instead of the full history. Eg: if the page is "2" and the limit is "5", the 6th through 10th revisions would be returned. "count" is not affected by pagination - it will always return the total number of revisions that exist.

POST

Reverts the scheduled change identified by the given sc_id to the version identified by the change_id given in the request body. The request body must be a JSON object containing a "change_id" key.

5.7 Others

5.7.1 /csrf_token

GET

Returns an empty response with a valid CSRF token in the X-CSRF-Token header.

5.7.2 /history/view/<object>/<change_id>/<field>

GET

Returns the value of the named field from the named object at the specified change_id.

5.7.3 /history/diff/<object>/<change_id>/<field>

GET

Returns a diff of the value of the named field from the named object at the specified change_id vs. the previous change to that object.

CHAPTER

SIX

BLOBS

Balrog's Blobs are classes that encapsulate the logic pertaining to a type of Release. Blobs are responsible for defining their own data structure format (with a jsonschema), the response format, and making decisions about when a response should be returned. This page documents the Blobs we have in Balrog, what they're used for, and what responses from them look like.

6.1 App Release

App Release blobs are used for Gecko Application updates - generally anything that receives updates through MAR files (eg: Firefox, Fennec, Thunderbird). Each App Release Blob contains all of the metadata for all platforms and locales of a particular release. There are many versions of the App Release blob to support the various Firefox versions, and implement various Balrog features.

6.1.1 Schema 1

Legacy format to support Firefox 1.5 through 3.6.x and Thunderbird 1.5 to 3.1.y.

6.1.2 Schema 2

Compatible with Firefox and Thunderbird 4.0 and above.

Supports client side changes from https://bugzilla.mozilla.org/show_bug.cgi?id=530872

Changes from V1:

- appv and extv become appVersion, platformVersion, and displayVersion
- Added actions, billboardURL, openURL, notificationURL, alertURL, showPrompt, showNeverForVersion, isOSUpdate
- Removed oldVersionSpecialCases

6.1.3 Schema 3

Compatible with Firefox and Thunderbird 4.0 and above.

This was an internal change to add multiple partials support to Balrog. It replaces the "partial" and "complete" properties at the local level with new "partials" and "completes" properties.

6.1.4 Schema 4

Compatible with Firefox and Thunderbird 4.0 and above.

This was an internal change to add support for pushing release channel builds to the beta channel. It replaces the top level "fileUrls", "bouncerProducts", and "ftpFilenames" properties with a single, unified "fileUrls" properties.

Here is an example that shows the appreleaseExample, with all of its platforms and locales.

6.1.5 Schema 5

Compatible with Firefox and Thunderbird 19.0 and above.

Driven by a client side change made in https://bugzilla.mozilla.org/show_bug.cgi?id=813322 that added support for the optional "promptWaitTime" attribute.

6.1.6 Schema 6

Compatible with Firefox and Thunderbird 51.0 and above.

This version removes support for platformVersion, billboardURL, licenseURL, version, and extensionVersion which are no longer supported by the client.

6.1.7 Schema 7

This schema was never used and no longer exists. It was briefly added to support the backgroundInterval attribute which was never used.

6.1.8 Schema 8

Compatible with Firefox and Thunderbird 51.0 and above.

This was an internal change to add support for the binary transparency attributes (https://bugzilla.mozilla.org/show_bug.cgi?id=1384296).

6.1.9 Schema 9

Compatible with Firefox and Thunderbird 51.0 and above.

This was an internal change to allow the App Release blob to make decisions about when to return most <update> parameters. Most notably, it allows us to return (or not) What's New Pages based on incoming client information. This work happened in https://bugzilla.mozilla.org/show_bug.cgi?id=1400016.

Here is an example that shows a single appreleasev9Example that supports both WNP and no-WNP updates.

6.1.10 Response Format

The response for App Releases is as follows (different versions of the App Release blob use different <update> line parameters, but the general structure is consistent):

```
<updates>
    <update type="minor" displayVersion="43.0.2" appVersion="43.0.2" platformVersion=
↪"43.0.2" buildID="20151221130713" detailsURL="https://www.mozilla.org/en-US/firefox/
↪43.0.2/releasenotes/">
        <patch type="complete" URL="http://download.mozilla.org/?product=firefox-43.0.
↪2-complete&os=osx&lang=en-US&force=1" hashFunction="sha512" hashValue=
↪"781478556846b719ebc906a8a9613a421e24449b4456c4ccee990e878b3be9fb0478a78821a499a4c1f1a76d75078acf3
↪" size="80329415"/>
        <patch type="partial" URL="http://download.mozilla.org/?product=firefox-43.0.
↪2-partial-41.0.2&os=osx&lang=en-US&force=1" hashFunction="sha512" hashValue=
↪"6edd0803e36a03117e12a36e9fc8941e8f6321071fb00c7e8489f67b332d1cbfa95d00218e5c1b61115752fc0aecde8b2
↪" size="39520883"/>
    </update>
</updates>
```

Some responses may have only a type="complete" patch line, but if an <update> line is included it must have at least a <patch> block whose type="complete".

6.2 GMP/System Addons

GMP and System Addon updates are very similar in that they both provide a list of the latest version of addons/plugins that Firefox should install. Although the payloads they point to are different, the data structure and response are exactly the same, so they share a Blob. Here is an example that shows a gmpExample that serves OpenH264 and CDM updates.

6.2.1 Response Format

The response format for GMP and System Addons is as follows:

```
<updates>
    <addons>
        <addon id="gmp-eme-adobe" URL="https://cdmdownload.adobe.com/firefox/win/x86/
↪primetime_gmp_win_x86_gmc_30527.1.zip" hashFunction="sha512" hashValue=
↪"d0077885971419a5db8e8ab9f0cb2cac236be98497aa9b6f86ff3b528788fc01a755a8dd401f391f364ff6e586204a766e
↪" size="3696996" version="15"/>
        <addon id="gmp-gmpopenh264" URL="http://ciscobinary.openh264.org/openh264-
↪win32-2706e36bf0a8b7c539c803ed877148c005ffca59.zip" hashFunction="sha512" hashValue=
↪"45124a776054dcfc81bfc65ad4ff85bd65113900c86f98b70917c695cd9d8924d9b0878da39d14b2af5708029bc0346be
↪" size="341180" version="1.5.3"/>
    </addons>
</updates>
```

Similar to App Release blobs, not all <addon> lines are required, but if an <addons> block is present, at least one <addon> must be inside of it. There is no limit to the number of <addon> blocks that can be in a response, but each one must have a unique "id".

6.3 Superblobs

Superblobs are used to serve multiple updates for GMP and System Addons releases. They point at the releases that should be severed for the respective query.

6.3.1 Superblob Example

```
{
  "products": [
    "CDM",
    "OpenH264",
    "Widevine"
  ],
  "name": "GMP-Superblob",
  "schema_version": 4000
}
```

CHAPTER

SEVEN

BALROG AGENT

The Balrog Agent is a long running process that is responsible for monitoring and enacting *Scheduled Changes*. The Agent is a separate application that runs in its own Docker container. In order to ensure it cannot bypass permissions or history (accidentally or on purpose), the Agent acts as a client to the Admin App rather than interacting with the database directly. This means that it must be granted the "scheduled_change" permission to function.

In the future, the Agent may also act as a client to Telemetry, Crash Stats, or other systems that contain data that we may want to schedule changes with.

CHAPTER

EIGHT

DEVELOPER DOCUMENTATION (AUSLIB PACKAGE)

8.1 auslib.blobs

8.1.1 auslib.blobs.apprelease

class auslib.blobs.apprelease.**DesupportBlob**(***kwargs*)
Bases: *auslib.blobs.base.Blob*

This blob is used to inform users that their OS is no longer supported. This is available on the client side since Firefox 24 (bug 843497).

The XML should look like this (whitespace for clarity & consistency only): <?xml version="1.0"?> <updates>

<update type="major" unsupported="true" detailsURL="http://moreinfo"> </update>

</updates>

containsForbiddenDomain(*product*, *whitelistedDomains*)

getInnerFooterXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

jsonschema = 'desupport.yml'

shouldServeUpdate(*updateQuery*)
Should be implemented by subclasses. In the event that it's not, False is the safest thing to return (it will fail closed instead of failing open).

class auslib.blobs.apprelease.**MultipleUpdatesXMLMixin**
Bases: object

class auslib.blobs.apprelease.**NewStyleVersionsMixin**
Bases: object

getAppVersion(*platform*, *locale*)

getApplicationVersion(*platform*, *locale*)
For v2 schema, appVersion really is the app version

getDisplayVersion(*platform*, *locale*)

getPlatformVersion(*platform*, *locale*)

class auslib.blobs.apprelease.**ProofXMLMixin**
Bases: object

class auslib.blobs.apprelease.**ReleaseBlobBase**(***kwargs*)
Bases: *auslib.blobs.base.Blob*

containsForbiddenDomain(*product*, *whitelistedDomains*)
Returns True if the blob contains any file URLs that contain a domain that we're not allowed to serve updates to.

getBuildID(*platform*, *locale*)

getInnerFooterXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)
This method, along with getHeaderXML and getFooterXML are the entry point for update XML creation for all Gecko app blobs. However, the XML and underlying data has changed over time, so there is a lot of indirection and calls factored out to subclasses. Below is a brief description of the flow of control that should help in understanding this code. Inner methods that are shared between blob versions live in Mixin classes so that they can be easily shared. Inner methods that only apply to a single blob version live on concrete blob classes (but should be moved if they need to be shared in the future). * getInnerXML, getFooterXML and getHeaderXML called by web layer,

live on this base class. The V1 blob class override them to support bug 1113475, but still calls the base class one to do most of the work.

** **_getUpdateLineXML() called to get information that is independent** of specific MARs. Most notably, version information changed starting with V2 blobs.

** **_getPatchesXML() called to get the information that describes** specific MARs. Where in the blob this information comes from changed significantly starting with V3 blobs.

*** **_getSpecificPatchXML() called to translate MAR information into** XML. This transformation in blob version independent, so it lives on the base class to avoid duplication.

**** **_getUrl() called to figure out what the MAR URL is for a** specific patch. This changed starting with V4 blobs. V3 and earlier use SeparatedFileUrlsMixin, V4 and later use UnifiedFileUrlsMixin.

* **_getFtpFilename/_getBouncerProduct called to substitute some** paths with real information. This is another part of the blob format that changed starting with V3 blobs. It was later deprecated in V4 and thus not used for UnifiedFileUrlsMixin blobs.

getLocaleData(*platform*, *locale*)

getLocaleOrTopLevelParam(*platform*, *locale*, *param*)

getPlatformData(*platform*)

getResolvedPlatform(*platform*)

matchesUpdateQuery(*updateQuery*)

shouldServeUpdate(*updateQuery*)
Should be implemented by subclasses. In the event that it's not, False is the safest thing to return (it will fail closed instead of failing open).

class auslib.blobs.apprelease.**ReleaseBlobV1**(***kwargs*)

Bases: *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.SingleUpdateXMLMixin*, *auslib.blobs.apprelease.SeparatedFileUrlsMixin*

This is the legacy format for apps based on Gecko 1.8.0 to 1.9.2, which translates to

- Firefox 1.5 to 3.6.x
- Thunderbird 1.5 to 3.1.y

It was deprecated by https://bugzilla.mozilla.org/show_bug.cgi?id=530872 during Gecko 2.0 development (aka 1.9.3).

createSnippets(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

getApplicationVersion(*platform*, *locale*)

We used extv as the application version for v1 schema, while appv may have been a pretty version for users to see

getAppv(*platform*, *locale*)

getExtv(*platform*, *locale*)

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

In order to update some older versions of Firefox without prompting them for add-on compatibility, we need to be able to modify the appVersion and extVersion attributes. bug 998721 and bug 1174605 have additional background on this. It would be nicer to do this in _getUpdateLineXML to avoid overriding this method, but that doesn't have access to the updateQuery to lookup the version making the request.

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current

release references

jsonschema = 'apprelease-v1.yml'

class auslib.blobs.apprelease.**ReleaseBlobV2**(***kwargs*)

Bases: *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.NewStyleVersionsMixin*, *auslib.blobs.apprelease.SingleUpdateXMLMixin*, *auslib.blobs.apprelease.SeparatedFileUrlsMixin*

Compatible with Gecko 1.9.3a3 and above, ie Firefox/Thunderbird 4.0 and above.

Client-side changes in https://bugzilla.mozilla.org/show_bug.cgi?id=530872

renamed or introduced several attributes in update.xml

Changed parameters from ReleaseBlobV1:

- appv, extv become appVersion, platformVersion, displayVersion

Added:

- actions, billboardURL, openURL, notificationURL, alertURL, showPrompt, showNeverForVersion, isOSUpdate

Removed:

- oldVersionSpecialCases

createSnippets(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current

release references

interpolable_ = ('billboardURL', 'openURL', 'notificationURL', 'alertURL')

jsonschema = 'apprelease-v2.yml'

optional_ = ('billboardURL', 'showPrompt', 'showNeverForVersion', 'actions', 'openURL'

class auslib.blobs.apprelease.**ReleaseBlobV3**(***kwargs*)

Bases: *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.NewStyleVersionsMixin*, *auslib.blobs.apprelease.MultipleUpdatesXMLMixin*, *auslib.blobs.apprelease.SeparatedFileUrlsMixin*

Compatible with Gecko 1.9.3a3 and above, ie Firefox/Thunderbird 4.0 and above.

This is an internal change to add functionality to Balrog.

Changes from ReleaseBlobV2:

- support multiple partials * remove "partial" and "complete" from locale level * add "partials" and "completes" to locale level, ftpFilenames, and bouncerProducts

createSnippets(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current

release references

interpolable_ = ('billboardURL', 'openURL', 'notificationURL', 'alertURL')

jsonschema = 'apprelease-v3.yml'

optional_ = ('billboardURL', 'showPrompt', 'showNeverForVersion', 'actions', 'openURL'

class auslib.blobs.apprelease.**ReleaseBlobV4**(***kwargs*)

Bases: *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.NewStyleVersionsMixin*, *auslib.blobs.apprelease.MultipleUpdatesXMLMixin*, *auslib.blobs.apprelease.UnifiedFileUrlsMixin*

Compatible with Gecko 1.9.3a3 and above, ie Firefox/Thunderbird 4.0 and above.

This is an internal change to add functionality to Balrog.

Changes from ReleaseBlobV3:

- Support pushing release builds to the beta channel with bouncer support (bug 1021026)

** **Combine fileUrls, bouncerProducts, and ftpFilenames into a larger data structure,** still called "fileUrls". (See below for a more detailed description.)

classmethod fromV3(*v3Blob*)

Creates a v4 blob based on the v3 blob given.

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current

release references

interpolable_ = ('billboardURL', 'openURL', 'notificationURL', 'alertURL')

jsonschema = 'apprelease-v4.yml'

optional_ = ('billboardURL', 'showPrompt', 'showNeverForVersion', 'actions', 'openURL'

class auslib.blobs.apprelease.**ReleaseBlobV5**(***kwargs*)

Bases: *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.NewStyleVersionsMixin*, *auslib.blobs.apprelease.MultipleUpdatesXMLMixin*, *auslib.blobs.apprelease.UnifiedFileUrlsMixin*

Compatible with Gecko 19.0 and above, ie Firefox/Thunderbird 19.0 and above.

Driven by a client-side change made in https://bugzilla.mozilla.org/show_bug.cgi?id=813322

Changes from ReleaseBlobV4:

- Support optional promptWaitTime attribute

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current release references

interpolable_ = ('billboardURL', 'openURL', 'notificationURL', 'alertURL')

jsonschema = 'apprelease-v5.yml'

optional_ = ('billboardURL', 'showPrompt', 'showNeverForVersion', 'actions', 'openURL'

class auslib.blobs.apprelease.**ReleaseBlobV6**(***kwargs*)

Bases: *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.NewStyleVersionsMixin*, *auslib.blobs.apprelease.MultipleUpdatesXMLMixin*, *auslib.blobs.apprelease.UnifiedFileUrlsMixin*

Compatible with Gecko 51.0 and above, ie Firefox/Thunderbird 51.0 and above.

Changes from ReleaseBlobV5:

- Removes support for platformVersion, billboardURL, licenseURL, version, and extensionVersion

For further information:

- https://bugzilla.mozilla.org/show_bug.cgi?id=1296685

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current release references

interpolable_ = ('openURL', 'notificationURL', 'alertURL')

jsonschema = 'apprelease-v6.yml'

optional_ = ('showPrompt', 'showNeverForVersion', 'actions', 'openURL', 'notificationU

class auslib.blobs.apprelease.**ReleaseBlobV8**(***kwargs*)

Bases: *auslib.blobs.apprelease.ProofXMLMixin*, *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.NewStyleVersionsMixin*, *auslib.blobs.apprelease.MultipleUpdatesXMLMixin*, *auslib.blobs.apprelease.UnifiedFileUrlsMixin*

Compatible with Gecko 51.0 and above, ie Firefox/Thunderbird 51.0 and above.

Changes from ReleaseBlobV6:

- Adds support for ProofXMLMixin (placed as first parameter for inheritance preference)

For further information:

- https://bugzilla.mozilla.org/show_bug.cgi?id=1384296

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current release references

interpolable_ = ('openURL', 'notificationURL', 'alertURL')

jsonschema = 'apprelease-v8.yml'

optional_ = ('showPrompt', 'showNeverForVersion', 'actions', 'openURL', 'notificationU

class auslib.blobs.apprelease.**ReleaseBlobV9**(***kwargs*)

Bases: *auslib.blobs.apprelease.ProofXMLMixin*, *auslib.blobs.apprelease.ReleaseBlobBase*, *auslib.blobs.apprelease.MultipleUpdatesXMLMixin*, *auslib.blobs.apprelease.UnifiedFileUrlsMixin*

Compatible with Gecko 51.0 and above, ie Firefox/Thunderbird 51.0 and above.

Changes from ReleaseBlobV8:

- Moved most <update> properties into new updateLine data structure

For further information:

- https://bugzilla.mozilla.org/show_bug.cgi?id=1400016

getApplicationVersion(*platform*, *locale*)

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current release references

interpolable_ = ('openURL', 'notificationURL', 'alertURL', 'detailsURL')

jsonschema = 'apprelease-v9.yml'

validate(**args*, ***kwargs*)

Raises a BlobValidationError if the blob is invalid.

class auslib.blobs.apprelease.**SeparatedFileUrlsMixin**

Bases: object

class auslib.blobs.apprelease.**SingleUpdateXMLMixin**

Bases: object

class auslib.blobs.apprelease.**UnifiedFileUrlsMixin**

Bases: object

8.1.2 auslib.blobs.base

class auslib.blobs.base.**Blob**(**args*, ***kwargs*)

Bases: dict

containsForbiddenDomain(*product*, *whitelistedDomains*)

getFooterXML()

Returns Returns the outer most footer. Returns the outer most header

getHeaderXML()

Returns Returns the outer most header. Returns the outer most header

getInnerFooterXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

getJSON()
Returns a JSON formatted version of this blob.

getReferencedReleases()

Returns Returns set of names of partially referenced releases that the current

release references

getResponseBlobs()

Returns Usually returns None. It the Blob is a systemaddons superblob, it returns the list of return blobs

getResponseProducts()

Returns Usually returns None. If the Blob is a SuperBlob, it returns the list of return products.

getSchema()

jsonschema = None

loadJSON(*data*)
Replaces this blob's contents with parsed contents of the json string provided.

processSpecialForceHosts(*url*, *specialForceHosts*, *force_arg*)

shouldServeUpdate(*updateQuery*)
Should be implemented by subclasses. In the event that it's not, False is the safest thing to return (it will fail closed instead of failing open).

validate(*product*, *whitelistedDomains*)
Raises a BlobValidationError if the blob is invalid.

exception auslib.blobs.base.**BlobValidationError**(*message*, *errors*, **args*, ***kwargs*)
Bases: ValueError

auslib.blobs.base.**createBlob**(*data*)
Takes a string form of a blob (eg from DB or API) and converts into an actual blob, taking care to notice the schema

auslib.blobs.base.**merge_dicts**(*ancestor*, *left*, *right*)
Perform a 3-way merge on dictonaries. We used to use an external library for this, but we replaced it with this because our merge can be a bit more liberal than the general case. Specifically:

- Lists are treated as sets (see above for details)
- A type mismatch of unicode vs string is OK as long as the text is the same (Any other type mismatches result in a failure to merge.)

auslib.blobs.base.**merge_lists**(**lists*)
Merge an arbitrary number of lists together, keeping only the unique items, and not worrying about order. This is essentially a hack to treat lists in blobs as sets. In an ideal world, that's what they'd be, but because we use jsonschema for validation, we cannot use proper sets.

8.1.3 auslib.blobs.gmp

class auslib.blobs.gmp.**GMPBlobV1**(***kwargs*)
Bases: *auslib.blobs.base.Blob*

containsForbiddenDomain(*product*, *whitelistedDomains*)
Returns True if the blob contains any file URLs that contain a domain that we're not allowed to serve updates to.

getInnerFooterXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

getPlatformData(*vendor*, *platform*)

getResolvedPlatform(*vendor*, *platform*)

getVendorsForPlatform(*platform*)

jsonschema = 'gmp.yml'

shouldServeUpdate(*updateQuery*)
Should be implemented by subclasses. In the event that it's not, False is the safest thing to return (it will fail closed instead of failing open).

validate(*product*, *whitelistedDomains*)
Raises a BlobValidationError if the blob is invalid.

8.1.4 auslib.blobs.superblob

class auslib.blobs.superblob.**SuperBlob**(***kwargs*)
Bases: *auslib.blobs.base.Blob*

containsForbiddenDomain(*product*, *whitelistedDomains*)

getInnerFooterXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Header specific to GMP and systemaddons superblob

getResponseBlobs()

Returns Blob names in case of systemaddon Blobs

getResponseProducts()

Returns Product in case of GMP supreblob

jsonschema = 'superblob.yml'

shouldServeUpdate(*updateQuery*)
Should be implemented by subclasses. In the event that it's not, False is the safest thing to return (it will fail closed instead of failing open).

8.1.5 auslib.blobs.systemaddons

class auslib.blobs.systemaddons.**SystemAddonsBlob**(***kwargs*)
Bases: *auslib.blobs.base.Blob*

containsForbiddenDomain(*product*, *whitelistedDomains*)
Returns True if the blob contains any file URLs that contain a domain that we're not allowed to serve updates to.

getAddonsForPlatform(*platform*)

getInnerFooterXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerHeaderXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

Returns Releases-specific header should be implemented for individual blobs

getInnerXML(*updateQuery*, *update_type*, *whitelistedDomains*, *specialForceHosts*)

getPlatformData(*addon*, *platform*)

getResolvedPlatform(*addon*, *platform*)

hasUpdates(*updateQuery*, *whitelistedDomains*)

jsonschema = 'systemaddons.yml'

shouldServeUpdate(*updateQuery*)
Should be implemented by subclasses. In the event that it's not, False is the safest thing to return (it will fail closed instead of failing open).

8.1.6 auslib.blobs.whitelist

8.2 auslib.db

class auslib.db.**AUSDatabase**(*dburi=None*, *mysql_traditional_mode=False*, *releases_history_buckets=None*, *releases_history_class=<class 'auslib.db.GCSHistory'>*)
Bases: object

begin()

create(*version=None*)

property dockerflow

downgrade(*version*)

property emergencyShutoffs

engine = None

getUserRoles(**args*, ***kwargs*)

hasPermission(**args*, ***kwargs*)

hasRole(**args*, ***kwargs*)

isAdmin(**args*, ***kwargs*)

isKnownUser(*username*)

migrate_repo = '/home/docs/checkouts/readthedocs.org/user_builds/mozilla-balrog/checko

property permissions

property permissionsRequiredSignoffs

property productRequiredSignoffs

property releases

reset()

property rules

setDburi(*dburi*, *mysql_traditional_mode=False*, *releases_history_buckets=None*, *releases_history_class=<class 'auslib.db.GCSHistory'>*)

Setup the database connection. Note that SQLAlchemy only opens a connection to the database when it needs to, however.

setDomainWhitelist(*domainWhitelist*)

setSystemAccounts(*systemAccounts*)

setupChangeMonitors(*relayhost*, *port*, *username*, *password*, *to_addr*, *from_addr*, *use_tls=False*, *notify_tables=None*)

upgrade(*version=None*)

class auslib.db.**AUSTable**(*db*, *dialect*, *historyClass=None*, *historyKwargs={}*, *versioned=True*, *scheduled_changes=False*, *scheduled_changes_kwargs={}*, *onInsert=None*, *onUpdate=None*, *onDelete=None*)

Bases: object

Base class for all AUS Tables. By default, all tables have a history table created for them, too, which mirrors their own structure and adds a record of who made a change, and when the change happened.

@param history: Whether or not to create a history table for this table. When True, a History object will be created for this table, and all changes will be logged to it. Defaults to True.

@type history: bool @param versioned: Whether or not this table is versioned. When True,

an additional 'data_version' column will be added to the Table, and its version increased with every update. This is useful for detecting colliding updates.

@type versioned: bool @param scheduled_changes: Whether or not this table should allow changes

to be scheduled. When True, two additional tables will be created: a $name_scheduled_changes, which will contain data needed to schedule changes to $name, and $name_scheduled_changes_history, which tracks the history of a scheduled change.

@type scheduled_changes: bool @param onInsert: A callback that will be called whenever an insert is

made to the table. It must accept the following 4 parameters:

- The table object the query is being performed on
- The type of query being performed (eg: INSERT)
- The name of the user making the change
- The query object that will be exeeceuted

If the callback raises an exception the change will be aborted.

@type onInsert: callable @param onDelete: See onInsert @type onDelete: callable @param onUpdate: See onInsert @type onUpdate: callable

count(*column='*'*, *where=None*, *transaction=None*)

delete(*where*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*)
Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

getEngine()

getRecentChanges(*limit=10*, *transaction=None*)

insert(*changed_by=None*, *transaction=None*, *dryrun=False*, ***columns*)
Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

select(*where=None*, *transaction=None*, ***kwargs*)
Perform a SELECT statement on this table. See AUSTable._selectStatement for possible arguments.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs.

@param transaction: A transaction object to add the update statement (and history changes) to. If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@rtype: sqlalchemy.engine.base.ResultProxy

update(*where*, *what*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*)
Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

class auslib.db.**AUSTransaction**(*engine*)
Bases: object

Manages a single transaction. Requires a connection object.

@param conn: connection object to perform the transaction on @type conn: sqlalchemy.engine.base.Connection

close()

commit()

execute(*statement*)

rollback()

exception auslib.db.**AlreadySetupError**
Bases: Exception

auslib.db.**BlobColumn**(*impl=<class 'sqlalchemy.sql.sqltypes.Text'>*)
BlobColumns are used to store Release Blobs, which are ultimately dicts. Release Blobs must be serialized before storage, and deserialized upon retrevial. This type handles both conversions. Some database engines (eg: mysql) may require a different underlying type than Text. The desired type may be passed in as an argument.

exception auslib.db.**ChangeScheduledError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

Raised when a Scheduled Change cannot be created, modified, or deleted for data consistency reasons.

class auslib.db.**CompatibleBooleanColumn**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

A Boolean column that is compatible with all of our supported database engines (mysql, sqlite). SQLAlchemy's built-in Boolean does not work because it creates a CHECK constraint that makes it impossible to downgrade a database with sqlalchemy-migrate.

impl
alias of sqlalchemy.sql.sqltypes.Integer

process_bind_param(*value*, *dialect*)
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying `TypeEngine` object, and from there to the DBAPI `execute()` method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** – the `Dialect` in use.

process_result_value(*value*, *dialect*)

Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying `TypeEngine` object, originally from the DBAPI cursor method `fetchone()` or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** – the `Dialect` in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

class auslib.db.**ConditionsTable**(*db*, *dialect*, *metadata*, *baseName*, *conditions*, *historyClass=<class 'auslib.db.HistoryTable'>*)

Bases: *auslib.db.AUSTable*

condition_groups = {'time': ('when',), 'uptake': ('telemetry_product', 'telemetry_ch

validate(*conditions*)

class auslib.db.**Dockerflow**(*db*, *metadata*, *dialect*)

Bases: *auslib.db.AUSTable*

getDockerflowEntry(*transaction=None*)

incrementWatchdogValue(*changed_by*, *transaction=None*, *dryrun=False*)

class auslib.db.**EmergencyShutoffs**(*db*, *metadata*, *dialect*)

Bases: *auslib.db.AUSTable*

delete(*where*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

getPotentialRequiredSignoffs(*affected_rows*, *transaction=None*)

insert(*changed_by*, *transaction=None*, *dryrun=False*, ***columns*)

Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

class auslib.db.**GCSHistory**(*db*, *dialect*, *metadata*, *baseTable*, *buckets*, *identifier_column*, *data_column*)

Bases: object

forDelete(*rowData*, *changed_by*, *trans*)

forInsert(*insertedKeys*, *columns*, *changed_by*, *trans*)

forUpdate(*rowData*, *changed_by*, *trans*)

getChange(*change_id=None*, *column_values=None*, *data_version=None*, *transaction=None*)

class auslib.db.**HistoryTable**(*db*, *dialect*, *metadata*, *baseTable*)

Bases: *auslib.db.AUSTable*

Represents a history table that may be attached to another AUSTable. History tables mirror the structure of their 'baseTable', with the exception that nullable and primary_key attributes are always overwritten to be True and False respectively. Additionally, History tables have a unique change_id for each row, and record the username making a change, and the timestamp of each change. The methods forInsert, forDelete, and forUpdate will generate appropriate INSERTs to the History table given appropriate inputs, and are documented below. History tables are never versioned, and cannot have history of their own.

forDelete(*rowData*, *changed_by*, *trans*)

Deletes cause a single row to be created, which only contains the primary key data. This represents that the row no longer exists.

forInsert(*insertedKeys*, *columns*, *changed_by*, *trans*)

Inserts cause two rows in the History table to be created. The first one records the primary key data and NULLs for other row data. This represents that the row did not exist prior to the insert. The timestamp for this row is 1 millisecond behind the real timestamp to reflect this. The second row records the full data of the row at the time of insert.

forUpdate(*rowData*, *changed_by*, *trans*)
Updates cause a single row to be created, which contains the full, new data of the row at the time of the update.

getChange(*change_id=None*, *column_values=None*, *data_version=None*, *transaction=None*)
Returns the unique change that matches the give change_id or combination of data_version and values for the specified columns. column_values is a dict that contains the column names that are versioned and their values. Ignores non primary key attributes specified in column_values.

getPointInTime(*timestamp*, *transaction=None*)

class auslib.db.**JSONColumn**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

JSONColumns are used for types that are deserialized JSON (usually dicts) in memory, but need to be serialized to text before storage. JSONColumn handles the conversion both ways, serialized just before storage, and deserialized just after retrieval.

impl
alias of sqlalchemy.sql.sqltypes.Text

process_bind_param(*value*, *dialect*)
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying TypeEngine object, and from there to the DBAPI execute() method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

process_result_value(*value*, *dialect*)
Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying TypeEngine object, originally from the DBAPI cursor method fetchone() or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

exception auslib.db.**MismatchedDataVersionError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

Raised when the data version of a scheduled change and its associated conditions row do not match after an insert or update.

exception auslib.db.**OutdatedDataError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

Raised when an update or delete fails because of outdated data.

exception auslib.db.**PermissionDeniedError**
Bases: Exception

class auslib.db.**Permissions**(*db*, *metadata*, *dialect*)
Bases: *auslib.db.AUSTable*

allPermissions defines the structure and possible options for all available permissions. Permissions can be limited to specific types of actions. Eg: granting the "rule" permission with "actions" set to ["create"] allows rules to be created but not modified or deleted. Permissions that relate to rules or releases can be further limited by product. Eg: granting the "release" permission with "products" set to ["GMP"] allows the user to modify GMP releases, but not Firefox.

allPermissions = {'admin': ['products'], 'emergency_shutoff': ['actions', 'products'

assertOptionsExist(*permission*, *options*)

assertPermissionExists(*permission*)

countAllUsers(*transaction=None*)

delete(*where*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*, *signoffs=None*)
Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

getAllPermissions(*transaction=None*)

getAllUsers(*transaction=None*)

getOptions(*username*, *permission*, *transaction=None*)

getPermission(*username*, *permission*, *transaction=None*)

getPotentialRequiredSignoffs(*affected_rows*, *transaction=None*)

getUserPermissions(*username*, *retrieving_as*, *transaction=None*)

getUserRoles(*username*, *transaction=None*)

grantRole(*username*, *role*, *changed_by*, *transaction=None*)

hasPermission(*username*, *thing*, *action*, *product=None*, *transaction=None*)

hasRole(*username*, *role*, *transaction=None*)

insert(*changed_by*, *transaction=None*, *dryrun=False*, *signoffs=None*, ***columns*)

Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@**param changed_by: The username of the person inserting the row. Required when** history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

isAdmin(*username*, *transaction=None*)

isKnownUser(*username*)

revokeRole(*username*, *role*, *changed_by=None*, *old_data_version=None*, *transaction=None*)

update(*where*, *what*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

class auslib.db.**PermissionsRequiredSignoffsTable**(*db*, *metadata*, *dialect*)

Bases: *auslib.db.RequiredSignoffsTable*

decisionColumns = ['product']

class auslib.db.**ProductRequiredSignoffsTable**(*db*, *metadata*, *dialect*)
Bases: *auslib.db.RequiredSignoffsTable*

decisionColumns = ['product', 'channel']

exception auslib.db.**ReadOnlyError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

Raised when a release marked as read-only is attempted to be changed.

class auslib.db.**Releases**(*db*, *metadata*, *dialect*, *history_buckets*, *historyClass*)
Bases: *auslib.db.AUSTable*

addLocaleToRelease(*name*, *product*, *platform*, *locale*, *data*, *old_data_version*, *changed_by*, *transaction=None*, *alias=None*)
Adds or update's the existing data for a specific platform + locale combination, in the release identified by 'name'. The data is validated before commiting it, and a ValueError is raised if it is invalid.

delete(*where*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*, *signoffs=None*)
Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

getLocale(*name*, *platform*, *locale*, *transaction=None*)

getPotentialRequiredSignoffs(*affected_rows*, *transaction=None*)

getReleaseBlob(*name*, *transaction=None*)

getReleaseInfo(*names=None*, *product=None*, *limit=None*, *transaction=None*, *nameOnly=False*, *name_prefix=None*)

getReleaseNames(***kwargs*)

getReleases(*name=None*, *product=None*, *limit=None*, *transaction=None*)

insert(*changed_by*, *transaction=None*, *dryrun=False*, *signoffs=None*, ***columns*)
Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

isMappedTo(*name*, *transaction=None*)

isReadOnly(*name*, *limit=None*, *transaction=None*)

localeExists(*name*, *platform*, *locale*, *transaction=None*)

setDomainWhitelist(*domainWhitelist*)

update(*where*, *what*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

class auslib.db.**RequiredSignoffsTable**(*db*, *dialect*)

Bases: *auslib.db.AUSTable*

RequiredSignoffsTables store and validate information about what types and how many signoffs are required for the data provided in *decisionColumns*. Subclasses are required to create a Table with the necessary columns, and add those columns names to *decisionColumns*. When changes are made to a RequiredSignoffsTable, it will look at its own rows to determine whether or not that change needs signoff.

decisionColumns = []

delete(*where*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

getPotentialRequiredSignoffs(*affected_rows*, *transaction=None*)

insert(*changed_by*, *transaction=None*, *dryrun=False*, *signoffs=None*, ***columns*)

Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

update(*where*, *what*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

validate(*columns*, *transaction=None*)

class auslib.db.**Rules**(*db*, *metadata*, *dialect*)

Bases: *auslib.db.AUSTable*

delete(*where*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

getOrderedRules(*where=None*, *transaction=None*)

Returns all of the rules, sorted in ascending order

getPotentialRequiredSignoffs(*affected_rows*, *transaction=None*)

getRule(*id_or_alias*, *transaction=None*)

Returns the unique rule that matches the give rule_id or alias.

getRulesMatchingQuery(*updateQuery*, *fallbackChannel*, *transaction=None*)

Returns all of the rules that match the given update query. For cases where a particular updateQuery channel has no fallback, fallbackChannel should match the channel from the query.

insert(*changed_by*, *transaction=None*, *dryrun=False*, *signoffs=None*, ***columns*)

Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

update(*where*, *what*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*, *signoffs=None*)

Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

class auslib.db.**ScheduledChangeTable**(*db*, *dialect*, *metadata*, *baseTable*, *conditions=('time', 'uptake')*, *historyClass=<class 'auslib.db.HistoryTable'>*)

Bases: *auslib.db.AUSTable*

A Table that stores the necessary information to schedule changes to the baseTable provided. A ScheduledChangeTable ends up mirroring the columns of its base, and adding the necessary ones to provide the schedule. By default, ScheduledChangeTables enable History on themselves.

auto_signoff(*changed_by*, *transaction*, *sc_id*, *dryrun*, *columns*)

delete(*where*, *changed_by=None*, *old_data_version=None*, *transaction=None*, *dryrun=False*)

Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

enactChange(*sc_id*, *enacted_by*, *transaction=None*)

Enacts a previously scheduled change by running update or insert on the base table.

insert (*changed_by*, *transaction=None*, *dryrun=False*, ***columns*)
Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

mergeUpdate (*old_row*, *what*, *changed_by*, *transaction=None*)
Merges an update to the base table into any changes that may be scheduled for the affected row. If the changes are unmergable (meaning: the scheduled change and the new version of the row modify the same columns), an UpdateMergeError is raised.

select (*where=None*, *transaction=None*, ***kwargs*)
Perform a SELECT statement on this table. See AUSTable._selectStatement for possible arguments.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs.

@param transaction: A transaction object to add the update statement (and history changes) to. If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@rtype: sqlalchemy.engine.base.ResultProxy

update (*where*, *what*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*)
Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

validate (*base_columns*, *condition_columns*, *changed_by*, *sc_id=None*, *transaction=None*)

class auslib.db.**SetSqlMode**
Bases: sqlalchemy.interfaces.PoolListener

connect(*dbapi_con*, *connection_record*)
Called once for each new DB-API connection or Pool's creator().

dbapi_con A newly connected raw DB-API connection (not a SQLAlchemy Connection wrapper).

con_record The _ConnectionRecord that persistently manages the connection

exception auslib.db.**SignoffRequiredError**
Bases: Exception

Raised when someone attempts to directly modify an object that requires signoff.

class auslib.db.**SignoffsTable**(*db*, *metadata*, *dialect*, *baseName*)
Bases: *auslib.db.AUSTable*

delete(*where*, *changed_by=None*, *transaction=None*, *dryrun=False*, *reset_signoff=False*)
Perform a DELETE statement on this table. See AUSTable._deleteStatement for a description of 'where'. To simplify versioning, this method can only delete a single row per invocation. If the where clause given would delete zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param changed_by: The username of the person deleting the row(s). Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the delete statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

insert(*changed_by=None*, *transaction=None*, *dryrun=False*, ***columns*)
Perform an INSERT statement on this table. See AUSTable._insertStatement for a description of columns.

@param changed_by: The username of the person inserting the row. Required when history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param transaction: A transaction object to add the insert statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

update(*where*, *what*, *changed_by=None*, *transaction=None*, *dryrun=False*)
Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

exception auslib.db.**TransactionError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

Raised when a transaction fails for any reason.

class auslib.db.**UTF8PrettyPrinter**(*indent=1*, *width=80*, *depth=None*, *stream=None*, ***, *compact=False*)
Bases: pprint.PrettyPrinter

Encodes strings as UTF-8 before printing to avoid ugly u" style prints. Adapted from http://stackoverflow.com/questions/10883399/unable-to-encode-decode-pprint-output

format(*object*, *context*, *maxlevels*, *level*)
Format object for a specific context, returning a string and flags indicating whether the representation is 'readable' and whether the object represents a recursive construct.

class auslib.db.**UnquotedStr**
Bases: str

exception auslib.db.**UpdateMergeError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

class auslib.db.**UserRoles**(*db*, *metadata*, *dialect*)
Bases: *auslib.db.AUSTable*

update(*where*, *what*, *changed_by*, *old_data_version*, *transaction=None*, *dryrun=False*)
Perform an UPDATE statement on this table. See AUSTable._updateStatement for a description of 'where' and 'what'. This method can only update a single row per invocation. If the where clause given would update zero or multiple rows, a WrongNumberOfRowsError is raised.

@param where: A list of SQLAlchemy clauses, or a key/value pair of columns and values. @type where: list of clauses or key/value pairs. @param what: Key/value pairs containing new values for the given columns. @type what: key/value pairs @param changed_by: The username of the person inserting the row. Required when

history is enabled. Unused otherwise. No authorization checks are done at this level.

@type changed_by: str @param old_data_version: Previous version of the row to be deleted. If this version doesn't

match the current version of the row, an OutdatedDataError will be raised and the delete will fail. Required when versioning is enabled.

@type old_data_version: int @param transaction: A transaction object to add the update statement (and history changes) to.

If provided, you must commit the transaction yourself. If None, they will be added to a locally-scoped transaction and committed.

@param dryrun: If true, this insert statement will not actually be run. @type dryrun: bool

@rtype: sqlalchemy.engine.base.ResultProxy

exception auslib.db.**WrongNumberOfRowsError**(**arg*, ***kw*)
Bases: sqlalchemy.exc.SQLAlchemyError

Raised when an update or delete fails because the clause matches more than one row.

auslib.db.**generate_random_string**(*length*)

auslib.db.**make_change_notifier**(*relayhost*, *port*, *username*, *password*, *to_addr*, *from_addr*, *use_tls*)

auslib.db.**make_change_notifier_for_read_only**(*relayhost*, *port*, *username*, *password*, *to_addr*, *from_addr*, *use_tls*)

auslib.db.**rows_to_dicts**(*rows*)
Converts SQL Alchemy result rows to dicts.

You might want this if you want to mutate objects (SQLAlchemy rows are immutable), or if you want to serialize them to JSON (SQLAlchemy rows get confused if you try to serialize them).

auslib.db.**send_email**(*relayhost*, *port*, *username*, *password*, *to_addr*, *from_addr*, *table*, *subj*, *body*, *use_tls*)

auslib.db.**verify_signoffs**(*potential_required_signoffs*, *signoffs*)
Determines whether or not something is signed off given: * A list of potential required signoffs * A list of signoffs that have been made

The real number of signoffs required is found by looking through the potential required signoffs and finding the highest number required for each role. If there are not enough signoffs provided for any of the groups, a SignoffRequiredError is raised.

8.3 auslib.util

8.3.1 auslib.util.cache

class auslib.util.cache.**MaybeCacher**
Bases: object

MaybeCacher is a very simple wrapper to work around the fact that we have two consumers of the auslib library (admin app, non-admin app) that require cache different things. Most notably, the non-admin app caches blobs and blob versions, while the admin app only caches blobs (because blob versions can change frequently). This class is intended to be instantiated as a global object, and then have caches created by consumers through calls to make_cache. Consumers that make changes (ie: the admin app) generally also set make_copies to True to get the cache to copy values on get/put to avoid the possibility of accidental cache pollution. For performance reasons, this should be disabled when not necessary.

If the cache given to get/put/clear/invalidate doesn't exist, these methods are essentially no-ops. In a world where bug 1109295 is fixed, we might only need to handle the caching case.

clear(*name=None*)

get(*name*, *key*, *value_getter=None*)
Returns the value of the specified key from the named cache. If value_getter is provided and no cache is found, or no value is found for the key, the return value of value_getter will be returned instead.

invalidate(*name*, *key*)

make_cache(*name*, *maxsize*, *timeout*)

property make_copies

put(*name*, *key*, *value*)

reset()

8.3.2 auslib.util.comparison

auslib.util.comparison.**get_op**(*pattern*)

auslib.util.comparison.**has_operator**(*value*)

auslib.util.comparison.**int_compare**(*value*, *compstr*)
Do a int comparison of a bare int with another, which may carry a comparison operator.

eg int_compare(1, '>2') is False

auslib.util.comparison.**string_compare**(*value*, *compstr*)
Do a string comparison of a bare string with another, which may carry a comparison operator.

eg string_compare('a', '>b') is False

auslib.util.comparison.**strip_operator**(*value*)

auslib.util.comparison.**version_compare**(*value*, *compstr*)
Do a version comparison between a string (representing a version), with another which may carry a comparison operator. A true version comparison is done.

eg version_compare('1.1', '>1.0') is True

8.3.3 auslib.util.thirdparty

auslib.util.thirdparty.**extendsyspath**()

8.3.4 auslib.util.timestamp

auslib.util.timestamp.**getMillisecondTimestamp**()

8.3.5 auslib.util.versions

class auslib.util.versions.**AncientMozillaVersion**(*vstring=None*)
Bases: distutils.version.StrictVersion

A version class that is slightly less restrictive than StrictVersion. Instead of just allowing "a" or "b" as prerelease tags, it allows any alpha. This allows us to support the once-shipped "3.6.3plugin1" and similar versions. It also supports versions w.x.y.z by transmuting to w.x.z, which is useful for versions like 1.5.0.x and 2.0.0.y

version_re = re.compile('^(\\d+) \\. (\\d+) \\. \\d (\\. (\\d+))\n ([a-zA-Z]+(\\d+))?$

class auslib.util.versions.**ModernMozillaVersion**(*vstring=None*)
Bases: distutils.version.StrictVersion

A version class that is slightly less restrictive than StrictVersion. Instead of just allowing "a" or "b" as prerelease tags, it allows any alpha. This allows us to support the once-shipped "3.6.3plugin1" and similar versions.

version_re = re.compile('^(\\d+) \\. (\\d+) (\\. (\\d+))?\n ([a-zA-Z]+(\\d+))?$', re.V

auslib.util.versions.**MozillaVersion**(*version*)

class auslib.util.versions.**PostModernMozillaVersion**(*vstring=None*)
Bases: distutils.version.StrictVersion

A version class that supports Firefox versions 5.0 and up, which may have "a1" but not "b2" tags in them

version_re = re.compile('^(\\d+) \\. (\\d+) (\\. (\\d+))?\n (a(\\d+))?$', re.VERBOSE)

auslib.util.versions.**decrement_version**(*version*)
Decrements a version to its 'previous' version by subtracting one from its last part. If the last part is 0, it is changed to 99, and the second last part is subtracted by one. This is repeated until subtraction happens or we run out of parts.

auslib.util.versions.**get_version_parts**(*version*)

auslib.util.versions.**increment_version**(*version*)
Increments a version to its 'next' version by adding one to the last part of the version.

8.4 auslib.web

8.4.1 auslib.web.public.base

auslib.web.public.base.**apply_security_headers**(*response*)

auslib.web.public.base.**contributejson**()

auslib.web.public.base.**fourohfour**(*error*)

auslib.web.public.base.**generic**(*error*)
Deals with any unhandled exceptions. If the exception is not a BadDataError, it will be sent to Sentry, and a 400 will be returned, because BadDataErrors are considered to be the client's fault. Otherwise, the error is just re-raised (which causes a 500).

auslib.web.public.base.**get_yaml**()

auslib.web.public.base.**robots**()

auslib.web.public.base.**set_cache_control**()

auslib.web.public.base.**unicode**(*error*)

8.5 auslib.AUS

class auslib.AUS.**AUS**
Bases: object

evaluateRules(*updateQuery*, *transaction=None*)

updates_are_disabled(*product*, *channel*, *transaction=None*)

class auslib.AUS.**ForceResult**(*name*, *query_value*)
Bases: object

Enumerated "result" class that represents a non-random result chosen by a caller.

auslib.AUS.**getFallbackChannel**(*channel*)

auslib.AUS.**isForbiddenUrl**(*url*, *product*, *whitelistedDomains*)

auslib.AUS.**isSpecialURL**(*url*, *specialForceHosts*)

8.6 auslib.config

class auslib.config.**AUSConfig**(*filename*)
Bases: object

getCaches()

getDburi()

getDomainWhitelist()

getLogLevel()

getLogfile()

loglevels = {'CRITICAL': 50, 'DEBUG': 10, 'ERROR': 40, 'INFO': 20, 'WARNING': 30}

required_options = {'database': ['dburi'], 'logging': ['logfile']}

validate()

class auslib.config.**AdminConfig**(*filename*)
Bases: *auslib.config.AUSConfig*

getPageTitle()

getSecretKey()

getSystemAccounts()

required_options = {'app': ['secret_key'], 'database': ['dburi'], 'logging': ['logf

class auslib.config.**ClientConfig**(*filename*)
Bases: *auslib.config.AUSConfig*

getSpecialForceHosts()

8.7 auslib.dockerflow

auslib.dockerflow.**create_dockerflow_endpoints**(*app*, *heartbeat_database_fn=<function _heartbeat_database_fn>*)

Wrapper that creates the endpoints required by CloudOps' Dockerflow spec: https://github.com/mozilla-services/Dockerflow. This gets used by both the admin and public apps. :param heartbeat_database_fn: Function that calls the database when reponding to /__heartbeat__. A database object is passed to this function.

If heartbeat_database_fn is None, a default function is be set. The default function writes in a dummy table. Even though we respond to GET, we do insert/update something in the database. This allows us to see if the connection to the database exists, is active, and if the credentials given are the correct ones. For more context see bug 1289178.

auslib.dockerflow.**get_version**(*version_file*)

auslib.dockerflow.**heartbeat_response**(*heartbeat_database_fn*)
Per the Dockerflow spec: Respond to /__heartbeat__ with a HTTP 200 or 5xx on error. This should depend on services like the database to also ensure they are healthy.

auslib.dockerflow.**lbheartbeat_response**()
Per the Dockerflow spec: Respond to /__lbheartbeat__ with an HTTP 200. This is for load balancer checks and should not check any dependent services.

8.8 auslib.errors

exception auslib.errors.**BadDataError**
Bases: Exception

Raised when a client sends data that appears to be invalid.

8.9 auslib.global_state

class auslib.global_state.**DbWrapper**
Bases: object

setDb(*dburi*, *releases_history_buckets=None*, *releases_history_class=None*)

8.10 auslib.log

class auslib.log.**BalrogLogger**(*name*, *level=0*)
Bases: logging.Logger

makeRecord(*name*, *level*, *fn*, *lno*, *msg*, *args*, *exc_info*, *func=None*, *extra=None*, *sinfo=None*)
A factory method which can be overridden in subclasses to create specialized LogRecords.

class auslib.log.**JsonLogFormatter**(*fmt=None*, *datefmt=None*, *logger_name='Balrog'*)
Bases: logging.Formatter

Log formatter that outputs machine-readable json.

This log formatter outputs JSON format messages that are compatible with Mozilla's standard heka-based log aggregation infrastructure.

See also: https://mana.mozilla.org/wiki/display/CLOUDSERVICES/Logging+Standard https://mana.mozilla.org/wiki/pages/viewpage.action?pageId=42895640

Adapted from: https://github.com/mozilla-services/mozservices/blob/master/mozsvc/util.py#L106

DEFAULT_SYSLOG_LEVEL = 7

EXCLUDED_LOGRECORD_ATTRS = {'args', 'asctime', 'created', 'exc_info', 'exc_text', 'fil

LOGGING_FORMAT_VERSION = '2.0'

SYSLOG_LEVEL_MAP = {10: 7, 20: 6, 30: 4, 40: 3}

format(*record*)
Map from Python LogRecord attributes to JSON log format fields

- from - https://docs.python.org/3/library/logging.html#logrecord-attributes

- to - https://mana.mozilla.org/wiki/pages/viewpage.action?pageId=42895640

auslib.log.**configure_logging**(*stream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>, formatter=<class 'auslib.log.JsonLogFormatter'>, format_='%(asctime)s - %(levelname)s - PID: %(process)s - Request: %(requestid)s - %(name)s.%(funcName)s#%(lineno)s: %(message)s', level=10*)

auslib.log.**safer_format_traceback**(*exc_typ*, *exc_val*, *exc_tb*)

Format an exception traceback into safer string. We don't want to let users write arbitrary data into our logfiles, which could happen if they e.g. managed to trigger a ValueError with a carefully-crafted payload. This function formats the traceback using "%r" for the actual exception data, which passes it through repr() so that any special chars are safely escaped.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

auslib.AUS, 68
auslib.blobs.apprelease, 41
auslib.blobs.base, 46
auslib.blobs.gmp, 48
auslib.blobs.superblob, 48
auslib.blobs.systemaddons, 49
auslib.config, 69
auslib.db, 49
auslib.dockerflow, 69
auslib.errors, 70
auslib.global_state, 70
auslib.log, 70
auslib.util.cache, 66
auslib.util.comparison, 67
auslib.util.thirdparty, 67
auslib.util.timestamp, 67
auslib.util.versions, 67
auslib.web.public.base, 68

INDEX

A

addLocaleToRelease() (*auslib.db.Releases method*), 58
AdminConfig (*class in auslib.config*), 69
allPermissions (*auslib.db.Permissions attribute*), 56
AlreadySetupError, 52
AncientMozillaVersion (*class in auslib.util.versions*), 67
apply_security_headers() (*in module auslib.web.public.base*), 68
assertOptionsExist() (*auslib.db.Permissions method*), 56
assertPermissionExists() (*auslib.db.Permissions method*), 56
AUS (*class in auslib.AUS*), 68
AUSConfig (*class in auslib.config*), 69
AUSDatabase (*class in auslib.db*), 49
auslib.AUS (*module*), 68
auslib.blobs.apprelease (*module*), 41
auslib.blobs.base (*module*), 46
auslib.blobs.gmp (*module*), 48
auslib.blobs.superblob (*module*), 48
auslib.blobs.systemaddons (*module*), 49
auslib.config (*module*), 69
auslib.db (*module*), 49
auslib.dockerflow (*module*), 69
auslib.errors (*module*), 70
auslib.global_state (*module*), 70
auslib.log (*module*), 70
auslib.util.cache (*module*), 66
auslib.util.comparison (*module*), 67
auslib.util.thirdparty (*module*), 67
auslib.util.timestamp (*module*), 67
auslib.util.versions (*module*), 67
auslib.web.public.base (*module*), 68
AUSTable (*class in auslib.db*), 50
AUSTransaction (*class in auslib.db*), 52
auto_signoff() (*auslib.db.ScheduledChangeTable method*), 62

B

BadDataError, 70
BalrogLogger (*class in auslib.log*), 70
begin() (*auslib.db.AUSDatabase method*), 49
Blob (*class in auslib.blobs.base*), 46
BlobColumn() (*in module auslib.db*), 52
BlobValidationError, 47

C

ChangeScheduledError, 52
clear() (*auslib.util.cache.MaybeCacher method*), 66
ClientConfig (*class in auslib.config*), 69
close() (*auslib.db.AUSTransaction method*), 52
commit() (*auslib.db.AUSTransaction method*), 52
CompatibleBooleanColumn (*class in auslib.db*), 52
condition_groups (*auslib.db.ConditionsTable attribute*), 53
ConditionsTable (*class in auslib.db*), 53
configure_logging() (*in module auslib.log*), 71
connect() (*auslib.db.SetSqlMode method*), 64
containsForbiddenDomain() (*auslib.blobs.apprelease.DesupportBlob method*), 41
containsForbiddenDomain() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
containsForbiddenDomain() (*auslib.blobs.base.Blob method*), 46
containsForbiddenDomain() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
containsForbiddenDomain() (*auslib.blobs.superblob.SuperBlob method*), 48
containsForbiddenDomain() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
contributejson() (*in module auslib.web.public.base*), 68
count() (*auslib.db.AUSTable method*), 50
countAllUsers() (*auslib.db.Permissions method*), 56
create() (*auslib.db.AUSDatabase method*), 49

create_dockerflow_endpoints() (*in module auslib.dockerflow*), 69
createBlob() (*in module auslib.blobs.base*), 47
createSnippets() (*auslib.blobs.apprelease.ReleaseBlobV1 method*), 43
createSnippets() (*auslib.blobs.apprelease.ReleaseBlobV2 method*), 43
createSnippets() (*auslib.blobs.apprelease.ReleaseBlobV3 method*), 44

D

DbWrapper (*class in auslib.global_state*), 70
decisionColumns (*auslib.db.PermissionsRequiredSignoffsTable attribute*), 57
decisionColumns (*auslib.db.ProductRequiredSignoffsTable attribute*), 58
decisionColumns (*auslib.db.RequiredSignoffsTable attribute*), 59
decrement_version() (*in module auslib.util.versions*), 68
DEFAULT_SYSLOG_LEVEL (*auslib.log.JsonLogFormatter attribute*), 70
delete() (*auslib.db.AUSTable method*), 50
delete() (*auslib.db.EmergencyShutoffs method*), 53
delete() (*auslib.db.Permissions method*), 56
delete() (*auslib.db.Releases method*), 58
delete() (*auslib.db.RequiredSignoffsTable method*), 59
delete() (*auslib.db.Rules method*), 61
delete() (*auslib.db.ScheduledChangeTable method*), 62
delete() (*auslib.db.SignoffsTable method*), 64
DesupportBlob (*class in auslib.blobs.apprelease*), 41
Dockerflow (*class in auslib.db*), 53
dockerflow() (*auslib.db.AUSDatabase property*), 49
downgrade() (*auslib.db.AUSDatabase method*), 49

E

EmergencyShutoffs (*class in auslib.db*), 53
emergencyShutoffs() (*auslib.db.AUSDatabase property*), 49
enactChange() (*auslib.db.ScheduledChangeTable method*), 62
engine (*auslib.db.AUSDatabase attribute*), 49
evaluateRules() (*auslib.AUS.AUS method*), 68
EXCLUDED_LOGRECORD_ATTRS (*auslib.log.JsonLogFormatter attribute*), 70
execute() (*auslib.db.AUSTransaction method*), 52
extendsyspath() (*in module auslib.util.thirdparty*), 67

F

ForceResult (*class in auslib.AUS*), 68
forDelete() (*auslib.db.GCSHistory method*), 54
forDelete() (*auslib.db.HistoryTable method*), 54
forInsert() (*auslib.db.GCSHistory method*), 54
forInsert() (*auslib.db.HistoryTable method*), 54
format() (*auslib.db.UTF8PrettyPrinter method*), 65
format() (*auslib.log.JsonLogFormatter method*), 70
forUpdate() (*auslib.db.GCSHistory method*), 54
forUpdate() (*auslib.db.HistoryTable method*), 54
fourohfour() (*in module auslib.web.public.base*), 68
fromV3() (*auslib.blobs.apprelease.ReleaseBlobV4 class method*), 44

G

GCSHistory (*class in auslib.db*), 54
generate_random_string() (*in module auslib.db*), 66
generic() (*in module auslib.web.public.base*), 68
get() (*auslib.util.cache.MaybeCacher method*), 67
get_op() (*in module auslib.util.comparison*), 67
get_version() (*in module auslib.dockerflow*), 69
get_version_parts() (*in module auslib.util.versions*), 68
get_yaml() (*in module auslib.web.public.base*), 68
getAddonsForPlatform() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
getAllPermissions() (*auslib.db.Permissions method*), 56
getAllUsers() (*auslib.db.Permissions method*), 56
getApplicationVersion() (*auslib.blobs.apprelease.NewStyleVersionsMixin method*), 41
getApplicationVersion() (*auslib.blobs.apprelease.ReleaseBlobV1 method*), 43
getApplicationVersion() (*auslib.blobs.apprelease.ReleaseBlobV9 method*), 46
getAppv() (*auslib.blobs.apprelease.ReleaseBlobV1 method*), 43
getAppVersion() (*auslib.blobs.apprelease.NewStyleVersionsMixin method*), 41
getBuildID() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getCaches() (*auslib.config.AUSConfig method*), 69
getChange() (*auslib.db.GCSHistory method*), 54
getChange() (*auslib.db.HistoryTable method*), 55

getDburi() (*auslib.config.AUSConfig method*), 69
getDisplayVersion() (*auslib.blobs.apprelease.NewStyleVersionsMixin method*), 41
getDockerflowEntry() (*auslib.db.Dockerflow method*), 53
getDomainWhitelist() (*auslib.config.AUSConfig method*), 69
getEngine() (*auslib.db.AUSTable method*), 51
getExtv() (*auslib.blobs.apprelease.ReleaseBlobV1 method*), 43
getFallbackChannel() (*in module auslib.AUS*), 69
getFooterXML() (*auslib.blobs.base.Blob method*), 46
getHeaderXML() (*auslib.blobs.base.Blob method*), 46
getInnerFooterXML() (*auslib.blobs.apprelease.DesupportBlob method*), 41
getInnerFooterXML() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getInnerFooterXML() (*auslib.blobs.base.Blob method*), 46
getInnerFooterXML() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
getInnerFooterXML() (*auslib.blobs.superblob.SuperBlob method*), 48
getInnerFooterXML() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
getInnerHeaderXML() (*auslib.blobs.apprelease.DesupportBlob method*), 41
getInnerHeaderXML() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getInnerHeaderXML() (*auslib.blobs.apprelease.ReleaseBlobV1 method*), 43
getInnerHeaderXML() (*auslib.blobs.base.Blob method*), 47
getInnerHeaderXML() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
getInnerHeaderXML() (*auslib.blobs.superblob.SuperBlob method*), 48
getInnerHeaderXML() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
getInnerXML() (*auslib.blobs.apprelease.DesupportBlob method*), 41
getInnerXML() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getInnerXML() (*auslib.blobs.base.Blob method*), 47
getInnerXML() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
getInnerXML() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
getJSON() (*auslib.blobs.base.Blob method*), 47
getLocale() (*auslib.db.Releases method*), 58
getLocaleData() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getLocaleOrTopLevelParam() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getLogfile() (*auslib.config.AUSConfig method*), 69
getLogLevel() (*auslib.config.AUSConfig method*), 69
getMillisecondTimestamp() (*in module auslib.util.timestamp*), 67
getOptions() (*auslib.db.Permissions method*), 56
getOrderedRules() (*auslib.db.Rules method*), 61
getPageTitle() (*auslib.config.AdminConfig method*), 69
getPermission() (*auslib.db.Permissions method*), 56
getPlatformData() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
getPlatformData() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
getPlatformData() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
getPlatformVersion() (*auslib.blobs.apprelease.NewStyleVersionsMixin method*), 41
getPointInTime() (*auslib.db.HistoryTable method*), 55
getPotentialRequiredSignoffs() (*auslib.db.EmergencyShutoffs method*), 54
getPotentialRequiredSignoffs() (*auslib.db.Permissions method*), 56
getPotentialRequiredSignoffs() (*auslib.db.Releases method*), 58
getPotentialRequiredSignoffs() (*auslib.db.RequiredSignoffsTable method*), 60
getPotentialRequiredSignoffs() (*auslib.db.Rules method*), 61
getRecentChanges() (*auslib.db.AUSTable method*), 51
getReferencedReleases() (*auslib.blobs.apprelease.ReleaseBlobV1 method*), 43

getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV2 method), 43
getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV3 method), 44
getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV4 method), 44
getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV5 method), 45
getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV6 method), 45
getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV8 method), 45
getReferencedReleases() (auslib.blobs.apprelease.ReleaseBlobV9 method), 46
getReferencedReleases() (auslib.blobs.base.Blob method), 47
getReleaseBlob() (auslib.db.Releases method), 58
getReleaseInfo() (auslib.db.Releases method), 58
getReleaseNames() (auslib.db.Releases method), 58
getReleases() (auslib.db.Releases method), 58
getResolvedPlatform() (auslib.blobs.apprelease.ReleaseBlobBase method), 42
getResolvedPlatform() (auslib.blobs.gmp.GMPBlobV1 method), 48
getResolvedPlatform() (auslib.blobs.systemaddons.SystemAddonsBlob method), 49
getResponseBlobs() (auslib.blobs.base.Blob method), 47
getResponseBlobs() (auslib.blobs.superblob.SuperBlob method), 48
getResponseProducts() (auslib.blobs.base.Blob method), 47
getResponseProducts() (auslib.blobs.superblob.SuperBlob method), 48
getRule() (auslib.db.Rules method), 61
getRulesMatchingQuery() (auslib.db.Rules method), 61
getSchema() (auslib.blobs.base.Blob method), 47
getSecretKey() (auslib.config.AdminConfig method), 69
getSpecialForceHosts() (auslib.config.ClientConfig method), 69
getSystemAccounts() (auslib.config.AdminConfig method), 69
getUserPermissions() (auslib.db.Permissions method), 56
getUserRoles() (auslib.db.AUSDatabase method), 49
getUserRoles() (auslib.db.Permissions method), 57
getVendorsForPlatform() (auslib.blobs.gmp.GMPBlobV1 method), 48
GMPBlobV1 (class in auslib.blobs.gmp), 48
grantRole() (auslib.db.Permissions method), 57

H

has_operator() (in module auslib.util.comparison), 67
hasPermission() (auslib.db.AUSDatabase method), 49
hasPermission() (auslib.db.Permissions method), 57
hasRole() (auslib.db.AUSDatabase method), 49
hasRole() (auslib.db.Permissions method), 57
hasUpdates() (auslib.blobs.systemaddons.SystemAddonsBlob method), 49
heartbeat_response() (in module auslib.dockerflow), 70
HistoryTable (class in auslib.db), 54

I

impl (auslib.db.CompatibleBooleanColumn attribute), 52
impl (auslib.db.JSONColumn attribute), 55
increment_version() (in module auslib.util.versions), 68
incrementWatchdogValue() (auslib.db.Dockerflow method), 53
insert() (auslib.db.AUSTable method), 51
insert() (auslib.db.EmergencyShutoffs method), 54
insert() (auslib.db.Permissions method), 57
insert() (auslib.db.Releases method), 58
insert() (auslib.db.RequiredSignoffsTable method), 60
insert() (auslib.db.Rules method), 61
insert() (auslib.db.ScheduledChangeTable method), 62
insert() (auslib.db.SignoffsTable method), 64
int_compare() (in module auslib.util.comparison), 67
interpolable_ (auslib.blobs.apprelease.ReleaseBlobV2 attribute), 44
interpolable_ (auslib.blobs.apprelease.ReleaseBlobV3 attribute), 44

interpolable_ (*auslib.blobs.apprelease.ReleaseBlobV4 attribute*), 44
interpolable_ (*auslib.blobs.apprelease.ReleaseBlobV5 attribute*), 45
interpolable_ (*auslib.blobs.apprelease.ReleaseBlobV6 attribute*), 45
interpolable_ (*auslib.blobs.apprelease.ReleaseBlobV8 attribute*), 46
interpolable_ (*auslib.blobs.apprelease.ReleaseBlobV9 attribute*), 46
invalidate() (*auslib.util.cache.MaybeCacher method*), 67
isAdmin() (*auslib.db.AUSDatabase method*), 49
isAdmin() (*auslib.db.Permissions method*), 57
isForbiddenUrl() (*in module auslib.AUS*), 69
isKnownUser() (*auslib.db.AUSDatabase method*), 49
isKnownUser() (*auslib.db.Permissions method*), 57
isMappedTo() (*auslib.db.Releases method*), 59
isReadOnly() (*auslib.db.Releases method*), 59
isSpecialURL() (*in module auslib.AUS*), 69

J

JSONColumn (*class in auslib.db*), 55
JsonLogFormatter (*class in auslib.log*), 70
jsonschema (*auslib.blobs.apprelease.DesupportBlob attribute*), 41
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV1 attribute*), 43
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV2 attribute*), 44
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV3 attribute*), 44
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV4 attribute*), 44
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV5 attribute*), 45
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV6 attribute*), 45
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV8 attribute*), 46
jsonschema (*auslib.blobs.apprelease.ReleaseBlobV9 attribute*), 46
jsonschema (*auslib.blobs.base.Blob attribute*), 47
jsonschema (*auslib.blobs.gmp.GMPBlobV1 attribute*), 48
jsonschema (*auslib.blobs.superblob.SuperBlob attribute*), 48
jsonschema (*auslib.blobs.systemaddons.SystemAddonsBlob attribute*), 49

L

lbheartbeat_response() (*in module auslib.dockerflow*), 70
loadJSON() (*auslib.blobs.base.Blob method*), 47
localeExists() (*auslib.db.Releases method*), 59
LOGGING_FORMAT_VERSION (*auslib.log.JsonLogFormatter attribute*), 70
loglevels (*auslib.config.AUSConfig attribute*), 69

M

make_cache() (*auslib.util.cache.MaybeCacher method*), 67
make_change_notifier() (*in module auslib.db*), 66
make_change_notifier_for_read_only() (*in module auslib.db*), 66
make_copies() (*auslib.util.cache.MaybeCacher property*), 67
makeRecord() (*auslib.log.BalrogLogger method*), 70
matchesUpdateQuery() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
MaybeCacher (*class in auslib.util.cache*), 66
merge_dicts() (*in module auslib.blobs.base*), 47
merge_lists() (*in module auslib.blobs.base*), 47
mergeUpdate() (*auslib.db.ScheduledChangeTable method*), 63
migrate_repo (*auslib.db.AUSDatabase attribute*), 49
MismatchedDataVersionError, 55
ModernMozillaVersion (*class in auslib.util.versions*), 67
MozillaVersion() (*in module auslib.util.versions*), 68
MultipleUpdatesXMLMixin (*class in auslib.blobs.apprelease*), 41

N

NewStyleVersionsMixin (*class in auslib.blobs.apprelease*), 41

O

optional_ (*auslib.blobs.apprelease.ReleaseBlobV2 attribute*), 44
optional_ (*auslib.blobs.apprelease.ReleaseBlobV3 attribute*), 44
optional_ (*auslib.blobs.apprelease.ReleaseBlobV4 attribute*), 44
optional_ (*auslib.blobs.apprelease.ReleaseBlobV5 attribute*), 45
optional_ (*auslib.blobs.apprelease.ReleaseBlobV6 attribute*), 45
optional_ (*auslib.blobs.apprelease.ReleaseBlobV8 attribute*), 46

OutdatedDataError, 56

P

PermissionDeniedError, 56
Permissions (*class in auslib.db*), 56
permissions() (*auslib.db.AUSDatabase property*), 50
permissionsRequiredSignoffs() (*auslib.db.AUSDatabase property*), 50
PermissionsRequiredSignoffsTable (*class in auslib.db*), 57
PostModernMozillaVersion (*class in auslib.util.versions*), 68
process_bind_param() (*auslib.db.CompatibleBooleanColumn method*), 52
process_bind_param() (*auslib.db.JSONColumn method*), 55
process_result_value() (*auslib.db.CompatibleBooleanColumn method*), 53
process_result_value() (*auslib.db.JSONColumn method*), 55
processSpecialForceHosts() (*auslib.blobs.base.Blob method*), 47
productRequiredSignoffs() (*auslib.db.AUSDatabase property*), 50
ProductRequiredSignoffsTable (*class in auslib.db*), 57
ProofXMLMixin (*class in auslib.blobs.apprelease*), 41
put() (*auslib.util.cache.MaybeCacher method*), 67

R

ReadOnlyError, 58
ReleaseBlobBase (*class in auslib.blobs.apprelease*), 42
ReleaseBlobV1 (*class in auslib.blobs.apprelease*), 42
ReleaseBlobV2 (*class in auslib.blobs.apprelease*), 43
ReleaseBlobV3 (*class in auslib.blobs.apprelease*), 44
ReleaseBlobV4 (*class in auslib.blobs.apprelease*), 44
ReleaseBlobV5 (*class in auslib.blobs.apprelease*), 44
ReleaseBlobV6 (*class in auslib.blobs.apprelease*), 45
ReleaseBlobV8 (*class in auslib.blobs.apprelease*), 45
ReleaseBlobV9 (*class in auslib.blobs.apprelease*), 46
Releases (*class in auslib.db*), 58
releases() (*auslib.db.AUSDatabase property*), 50
required_options (*auslib.config.AdminConfig attribute*), 69
required_options (*auslib.config.AUSConfig attribute*), 69
RequiredSignoffsTable (*class in auslib.db*), 59
reset() (*auslib.db.AUSDatabase method*), 50
reset() (*auslib.util.cache.MaybeCacher method*), 67
revokeRole() (*auslib.db.Permissions method*), 57
robots() (*in module auslib.web.public.base*), 68
rollback() (*auslib.db.AUSTransaction method*), 52
rows_to_dicts() (*in module auslib.db*), 66
Rules (*class in auslib.db*), 60
rules() (*auslib.db.AUSDatabase property*), 50

S

safer_format_traceback() (*in module auslib.log*), 71
ScheduledChangeTable (*class in auslib.db*), 62
select() (*auslib.db.AUSTable method*), 51
select() (*auslib.db.ScheduledChangeTable method*), 63
send_email() (*in module auslib.db*), 66
SeparatedFileUrlsMixin (*class in auslib.blobs.apprelease*), 46
set_cache_control() (*in module auslib.web.public.base*), 68
setDb() (*auslib.global_state.DbWrapper method*), 70
setDburi() (*auslib.db.AUSDatabase method*), 50
setDomainWhitelist() (*auslib.db.AUSDatabase method*), 50
setDomainWhitelist() (*auslib.db.Releases method*), 59
SetSqlMode (*class in auslib.db*), 63
setSystemAccounts() (*auslib.db.AUSDatabase method*), 50
setupChangeMonitors() (*auslib.db.AUSDatabase method*), 50
shouldServeUpdate() (*auslib.blobs.apprelease.DesupportBlob method*), 41
shouldServeUpdate() (*auslib.blobs.apprelease.ReleaseBlobBase method*), 42
shouldServeUpdate() (*auslib.blobs.base.Blob method*), 47
shouldServeUpdate() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
shouldServeUpdate() (*auslib.blobs.superblob.SuperBlob method*), 48
shouldServeUpdate() (*auslib.blobs.systemaddons.SystemAddonsBlob method*), 49
SignoffRequiredError, 64
SignoffsTable (*class in auslib.db*), 64
SingleUpdateXMLMixin (*class in auslib.blobs.apprelease*), 46
string_compare() (*in module auslib.util.comparison*), 67
strip_operator() (*in module auslib.util.comparison*), 67
SuperBlob (*class in auslib.blobs.superblob*), 48

SYSLOG_LEVEL_MAP (*auslib.log.JsonLogFormatter attribute*), 70
SystemAddonsBlob (*class in auslib.blobs.systemaddons*), 49

T

TransactionError, 65

U

unicode() (*in module auslib.web.public.base*), 68
UnifiedFileUrlsMixin (*class in auslib.blobs.apprelease*), 46
UnquotedStr (*class in auslib.db*), 65
update() (*auslib.db.AUSTable method*), 51
update() (*auslib.db.Permissions method*), 57
update() (*auslib.db.Releases method*), 59
update() (*auslib.db.RequiredSignoffsTable method*), 60
update() (*auslib.db.Rules method*), 61
update() (*auslib.db.ScheduledChangeTable method*), 63
update() (*auslib.db.SignoffsTable method*), 64
update() (*auslib.db.UserRoles method*), 65
UpdateMergeError, 65
updates_are_disabled() (*auslib.AUS.AUS method*), 68
upgrade() (*auslib.db.AUSDatabase method*), 50
UserRoles (*class in auslib.db*), 65
UTF8PrettyPrinter (*class in auslib.db*), 65

V

validate() (*auslib.blobs.apprelease.ReleaseBlobV9 method*), 46
validate() (*auslib.blobs.base.Blob method*), 47
validate() (*auslib.blobs.gmp.GMPBlobV1 method*), 48
validate() (*auslib.config.AUSConfig method*), 69
validate() (*auslib.db.ConditionsTable method*), 53
validate() (*auslib.db.RequiredSignoffsTable method*), 60
validate() (*auslib.db.ScheduledChangeTable method*), 63
verify_signoffs() (*in module auslib.db*), 66
version_compare() (*in module auslib.util.comparison*), 67
version_re (*auslib.util.versions.AncientMozillaVersion attribute*), 67
version_re (*auslib.util.versions.ModernMozillaVersion attribute*), 68
version_re (*auslib.util.versions.PostModernMozillaVersion attribute*), 68

W

WrongNumberOfRowsError, 66