The `build2` Repository Interface

Copyright © 2014-2018 Code Synthesis Ltd
Permission is granted to copy, distribute and/or modify this document under the terms of the MIT License.

Revision `0.8`, September 2018
This revision of the document describes the `build2` repository interface `0.8.x` series.

Table of Contents

Preface

This document describes `brep`, the `build2` package repository web interface. For the command line interface of `brep` utilities refer to the **`brep-load(1)`**, **`brep-clean(1)`**, and **`brep-migrate(1)`** man pages.

1 Package Submission

The package submission functionality allows uploading of package archives as well as additional, repository-specific information via the HTTP `POST` method using the `multipart/form-data` content type. The implementation in `brep` only handles uploading as well as basic verification (checksum, duplicates) expecting the rest of the submission and publishing logic to be handled by a separate entity according to the repository policy. Such an entity can be notified by `brep` about a new submission as an invocation of the *handler program* (as part of the HTTP request) and/or via email. It could also be a separate process that monitors the upload data directory.

The submission request without any parameters is treated as the submission form request. If `submit-form` is configured, then such a form is generated and returned. Otherwise, such a request is treated as an invalid submission (missing parameters).

For each submission request `brep` performs the following steps.

1. Verify submission size limit.

   The submission form-data payload size must not exceed `submit-max-size`.

2. Verify the required `archive` and `sha256sum` parameters are present.

   The `archive` parameter must be the package archive upload while `sha256sum` must be its 64 characters SHA256 checksum calculated in the binary mode.

3. Verify other parameters are valid manifest name/value pairs.

   The value can only contain printable ASCII characters as well as tab (`\t`), carriage return (`\r`), and line feed (`\n`).

4. Check for a duplicate submission.

   Each submission is saved as a subdirectory in the `submit-data` directory with a 12-character abbreviated checksum as its name.

5. Save the package archive into a temporary directory and verify its checksum.

   A temporary subdirectory is created in the `submit-temp` directory, the package archive is saved into it using the submitted name, and its checksum is calculated and compared to the submitted checksum.

6. Save the submission request manifest into the temporary directory.

   The submission request manifest is saved as `request.manifest` into the temporary subdirectory next to the archive.

7. Make the temporary submission directory permanent.

   Move/rename the temporary submission subdirectory to `submit-data` as an atomic operation using the 12-character abbreviated checksum as its new name. If such a directory already exist, then this is a duplicate submission.

8. Invoke the submission handler program.

   If `submit-handler` is configured, invoke the handler program passing to it additional arguments specified with `submit-handler-argument` (if any) followed by the absolute path to the submission directory.

   The handler program is expected to write the submission result manifest to `stdout` and terminate with the zero exit status. A non-zero exit status is treated as an internal error. The handler program's `stderr` is logged.

   Note that the handler program should report temporary server errors (service overload, network connectivity loss, etc.) via the submission result manifest status values in the [500-599] range (HTTP server error) rather than via a non-zero exit status.

   The handler program assumes ownership of the submission directory and can move/remove it. If after the handler program terminates the submission directory still exists, then it is handled by `brep` depending on the handler process exit status and the submission result manifest status value. If the process has terminated abnormally or with a non-zero exit status or the result manifest status is in the [500-599] range (HTTP server error), then the directory is saved for troubleshooting by appending the `.fail` extension followed by a numeric extension to its name (for example, `ff5a1a53d318.fail.1`). Otherwise, if the status is in the [400-499] range (HTTP client error), then the directory is removed. If the directory is left in place by the handler or is saved for troubleshooting, then the submission result manifest is saved as `result.manifest` into this directory, next to the request manifest and archive.

If `submit-handler-timeout` is configured and the handler program does not exit in the allotted time, then it is killed and its termination is treated as abnormal.

If the handler program is not specified, then the following submission result manifest is implied:

```
status: 200
message: package submission is queued
reference: <abbrev-checksum>
```

9. Send the submission email.

   If `submit-email` is configured, send an email to this address containing the submission request manifest and the submission result manifest.

10. Respond to the client.

    Respond to the client with the submission result manifest and its `status` value as the HTTP status code.

Check violations (max size, duplicate submissions, etc) that are explicitly mentioned above are always reported with the submission result manifest. Other errors (for example, internal server errors) might be reported with unformatted text, including HTML.

If the submission request contains the `simulate` parameter, then the submission service simulates the specified outcome of the submission process without actually performing any externally visible actions (e.g., publishing the package, notifying the submitter, etc). Note that the package submission email (`submit-email`) is not sent for simulated submissions.

Pre-defined simulation outcome values are `internal-error-text`, `internal-error-html`, `duplicate-archive`, and `success`. The simulation outcome is included into the submission request manifest and the handler program must at least handle `success` but may recognize additional outcomes.

1.1 Submission Request Manifest

The submission request manifest starts with the below values and in that order optionally followed by additional values in the unspecified order corresponding to the custom request parameters.

```
archive: <name>
sha256sum: <sum>
timestamp: <date-time>
[simulate]: <outcome>
[client-ip]: <string>
[user-agent]: <string>
```

The `timestamp` value is in the ISO-8601 `<YYYY>-<MM>-<DD>T<hh>:<mm>:<ss>Z` form (always UTC). Note also that `client-ip` can be IPv4 or IPv6.

1.2 Submission Result Manifest

The submission result manifest starts with the below values and in that order optionally followed by additional values if returned by the handler program. If the submission is successful, then the `reference` value must be present and contain a string that can be used to identify this submission (for example, the abbreviated checksum).

```
status: <http-code>
message: <string>
[reference]: <string>
```

2 Package CI

The CI functionality allows submission of package CI requests as well as additional, repository-specific information via the HTTP `GET` and `POST` methods using the `application/x-www-form-urlencoded` or `multipart/form-data` parameters encoding. The implementation in `brep` only handles reception as well as basic parameter verification expecting the rest of the CI logic to be handled by a separate entity according to the repository policy. Such an entity can be notified by `brep` about a new CI request as an invocation of the *handler program* (as part of the HTTP request) and/or via email. It could also be a separate process that monitors the CI data directory.

The CI request without any parameters is treated as the CI form request. If `ci-form` is configured, then such a form is generated and returned. Otherwise, such a request is treated as an invalid CI request (missing parameters).

For each CI request `brep` performs the following steps.

1. Verify the required `repository` and optional `package` parameters.

   The `repository` parameter is the remote `bpkg` repository location that contains the packages to be tested. If one or more `package` parameters are present, then only the specified packages are tested. If no `package` parameters are specified, then all the packages present in the repository (but excluding complement repositories) are tested.

   Each `package` parameter can specify either just the package name, in which case all the versions of this package present in the repository will be tested, or both the name and version in the `<name>/<version>` form (for example, `libhello/1.2.3`.

2. Verify other parameters are valid manifest name/value pairs.

   The value can only contain printable ASCII characters as well as tab (`\t`), carriage return (`\r`), and line feed (`\n`).

3. Generate CI request id and create request directory.

   For each CI request a unique id (UUID) is generated and a request subdirectory is created in the `ci-data` directory with this id as its name.

4. Save the CI request manifest into the request directory.

   The CI request manifest is saved as `request.manifest` into the request subdirectory created on the previous step.

5. Invoke the CI handler program.

   If `ci-handler` is configured, invoke the handler program passing to it additional arguments specified with `ci-handler-argument` (if any) followed by the absolute path to the CI request directory.

   The handler program is expected to write the CI result manifest to `stdout` and terminate with the zero exit status. A non-zero exit status is treated as an internal error. The handler program's `stderr` is logged.

   Note that the handler program should report temporary server errors (service overload, network connectivity loss, etc.) via the CI result manifest status values in the [500-599] range (HTTP server error) rather than via a non-zero exit status.

   The handler program assumes ownership of the CI request directory and can move/remove it. If after the handler program terminates the request directory still exists, then it is handled by `brep` depending on the handler process exit status and the CI result manifest status value. If the process has terminated abnormally or with a non-zero exit status or the result manifest status is in the [500-599] range (HTTP server error), then the directory is saved for troubleshooting by appending the `.fail` extension to its name. Otherwise, if the status is in the [400-499] range (HTTP client error), then the directory is removed. If the directory is left in place by the handler or is saved for troubleshooting, then the CI result manifest is saved as `result.manifest` into this directory, next to the request manifest.

   If `ci-handler-timeout` is configured and the handler program does not exit in the allotted time, then it is killed and its termination is treated as abnormal.

If the handler program is not specified, then the following CI result manifest is implied:

```
status: 200
message: CI request is queued
reference: <request-id>
```

6. Send the CI request email.

   If `ci-email` is configured, send an email to this address containing the CI request manifest and the CI result manifest.

7. Respond to the client.

   Respond to the client with the CI result manifest and its `status` value as the HTTP status code.

Check violations that are explicitly mentioned above are always reported with the CI result manifest. Other errors (for example, internal server errors) might be reported with unformatted text, including HTML.

If the CI request contains the `simulate` parameter, then the CI service simulates the specified outcome of the CI process without actually performing any externally visible actions (e.g., testing the package, publishing the result, etc). Note that the CI request email (`ci-email`) is not sent for simulated requests.

Pre-defined simulation outcome values are `internal-error-text`, `internal-error-html`, and `success`. The simulation outcome is included into the CI request manifest and the handler program must at least handle `success` but may recognize additional outcomes.

2.1 CI Request Manifest

The CI request manifest starts with the below values and in that order optionally followed by additional values in the unspecified order corresponding to the custom request parameters.

```
id: <request-id>
repository: <url>
[package]: <name>[/<version>]
timestamp: <date-time>
[simulate]: <outcome>
[client-ip]: <string>
[user-agent]: <string>
```

The `package` value can be repeated multiple times. The `timestamp` value is in the ISO-8601 `<YYYY>-<MM>-<DD>T<hh>:<mm>:<ss>Z` form (always UTC). Note also that `client-ip` can be IPv4 or IPv6.

2.2 CI Result Manifest

The CI result manifest starts with the below values and in that order optionally followed by additional values if returned by the handler program. If the CI request is successful, then the `reference` value must be present and contain a string that can be used to identify this request (for example, the CI request id).

```
status: <http-code>
message: <string>
[reference]: <string>
```