Meteor Isobuild is a complete toolchain for building an app from its source code into a set of runnable programs. As part of that toolchain, we run the Meteor Packaging Server, which stores Meteor packages that have been published with `meteor publish` and keeps track of their usage.

Package Metadata

Meteor provides a DDP interface to get general information about packages and their versions that are available on the package server, at packages.meteor.com.

syncNewPackageData(syncToken)

This will return 500 or so records per collection from the package database. It may be useful to call this method multiple times, with syncTokens returned by the previous call, until it has returned all of the latest data.

Arguments:

- `syncToken` : (Object) the syncToken represents the current state of the client database. When the client database is empty, pass in the default syncToken (see below). Afterwards, pass in the syncToken returned by the previous call to syncNewData. (See below for details)

Returns an object with the following keys:

- `collections` : This is the package data, represented by an object with the following keys:
  - `packages` : up to 500 records from the `packages` collection (see: Metadata Collections)
  - `versions` : up to 500 records from the `versions` collection (see: Metadata Collections)
  - `builds` : up to 500 records from the `builds` collection (see: Metadata Collections)
  - `releaseTracks` : up to 500 records from the `releases` collection (see: Metadata Collections)
  - `releaseVersions` : up to 500 records from the `releaseVersions` collection (see: Metadata Collections)
- `syncToken` : (Object) a new syncToken, representing the result of this sync.
- `upToDate` : (Boolean) True, if this is the latest data. To make sure you`ve synced all of the data, call syncNewPackageData repeatedly until this is true.
- `resetData` : (Boolean) Some data has been deleted, and the resync process has begone from scratch. If this is true, delete the results from all previous syncs.

In addition to the sync method, the Meteor package server publishes the following dataset:

`defaults` : This dataset exposes the collection `defaults` and takes no arguments. It contains exactly one record, with the following keys:

- `syncToken` : the base sync token. Pass this in the first time that you run `syncNewData` .

`changes` : This dataset exposes the collection `changes` and takes no arguments. It contains several records, with the following keys:

- `collection` : (String) collection name, such as `packages` or `versions` (see: Metadata Collections)
- `changes` : (Number) Incremented every time a document in that collection is changed or added.

Package Use Statistics

Meteor aims to provide easy user access to statistics on package installs. Specifically, every day, at slightly past UTC midnight, we run a job to aggregate package adds over the previous UTC day and store it in a publicly available file. To get it, make an HTTP GET request to:

`packages.meteor.com/stats/v1/<YYYY-MM-DD>`

In the document, stats are represented as a series of JSON objects, separated by new lines, for example, as so:

```
{ "name": "tracker", "version": "1.0.1", "directAdds": 250, "totalAdds": 1000 }

{ "name": "iron:router", "version": "1.0.0", "directAdds": 200, "totalAdds": 800 }
```

The keys are as follows:

- `name` : (String) name of the package, such as `blaze` , `tracker` or `iron:router`
- `version` : (String) version being added.
- `directAdds` : (Number) number of times this package was *directly* added to an app in the 24 hour period, for the first time — that is, the number of apps where the app maintainer ran `meteor add <package name>` and, as a result, Meteor added @ to their app. In the example above, 200 people ran `meteor add iron:router` , adding `iron:router@1.0.0` to 200 apps.
- `totalAdds` : (Number) total number of times this package was added to an app in the 24 hour period, for the first time. Unlike the `directAdds` , this also includes the number of times that a package was added to an app as a dependency of a different package. For example, let's say someone runs `meteor add foo:bar` to add `foo:bar` , which depends on `tracker` to an app. Because foo:bar depends on `tracker` , `tracker` gets added to their app. The totalAdds number for `tracker` still gets incremented, but the `directAdds` number stays the same.

The packages server publishes a dataset called `stats` , that exposes one collection called `stats` and takes no arguments. It contains one record with the keys:

- `earliest` : (String) Earliest date for which stats are available, in the form `YYYY-MM-DD` .
- `latest` : (String) Latest date for which stats are available, in the form `YYYY-MM-DD` .

Metadata Collections

Package information comes in three collections, which we sync to the client with the DDP method `syncNewData` . That method looks at the latest time the record was updated, which means that, over a long period of time, the same record might be sent to the client more than once (with different contents). The collections are as follows.

Packages

Stores the general metadata about each package and contains one record per package. Some of its fields are as follows (there may be other, undocumented fields):

- `name` : (String) name of the package (ex: `iron:router`)
- `maintainers` : (Array of objects, containing key `username`) List of authorized package maintainers
- `homepage` : (String) the URL of the package homepage, as set with `meteor admin change-homepage`
- `lastUpdated` : (Date) the last time this record was updated

Versions

Stores the per-version metadata about each version of each package. Contains one record for each package version. Some of the fields are as follows (there may be other, undocumented fields):

- `packageName` : (String) name of the package
- `version` : (String) Version number.
- `description` : (String) short, one sentence description of this version, as defined in Package.describe

- `longDescription` : (String) a longer description, usually excerpted from README.md
- `git` : (String) this versions`s Git URL
- `source` : object representing the version`s source, with the following keys:
  - `url` : (String) URL containing the source tarball
  - `hash` : (String) SHA256 hash of the source tarball
- `readme` : object representing the version`s README file, with the following keys:
  - `url` : (String) URL the README contents
  - `hash` : (String) SHA256 hash of the README contents
- `dependencies` : object representing the dependencies of this version. The keys are package names, and the values are objects with keys:
  - `constraint` : (String) version constraint for that dependency. For example, `1.0.0` or `=1.0.1` .
  - `references` : an object with the following keys:
    - `arch` : (String) dependency target, one of: `os` , `browser.web` or `browser.cordova`
    - `weak` : (Boolean) True if this is a weak dependency for that target
    - `unordered` : (Boolean) True if this is an unordered dependency for that target
    - `implied` : (Boolean) True if this dependency is implied for that target
- `exports` : an array of objects, representing the variables this package version exports. Exports have the following keys:
  - `name` : (String) name of the exported variable, such as `Router`
  - `architectures` : Array of strings with values that are one of: `os` , `browser.web` or `browser.cordova`
- `unmigrated` : (Boolean) true if this version is NOT compatible with Meteor 0.9.0
- `debugOnly` : (Boolean) set to true if this is a debug-only package
- `publishedBy` : (Object, containing key `username`) user that published this package
- `published` : (Date) publication date of this version
- `lastUpdated` : (Date) the last time this record was updated
- `_id` : Mongo ID for this version record.

> *Note: Meteor package names can include periods. This means that the keys in the `dependencies` field may include periods, which is an invalid object schema in many databases, including Mongo.*

Builds

Stores one record for each OS build of each version of each package. A package version with no binary dependencies will only have one build. For a package with binary dependencies, a build record will be created whenever its publisher successfully runs `meteor publish-for-arch` . Some of the fields are as follows (there may be other, undocumented fields):

- `versionId` : The _id of the version record for this package name and version
- `buildArchitectures` : (String) the OS architectures for this build separated by `+` , such as `os+web.browser+web.cordova` .
- `build` : object showing where to find the Isopack for this package, version and architecture. Has the following keys:
  - `url` : (String) URL of the tarball
  - `tarballHash` : (String) SHA256 hash of the tarball

    - `treeHash` : (String) SHA256 hash of the contents of the tarball
- `builtBy` :(Object containing key `username`) user who ran `meteor publish` or `meteor publish-for-arch` for this architecture
- `buildPublished` : (Date) time of publication
- `lastUpdated` : (Date) last time this record was updated

Release Tracks

Stores one record for each release track. (For example, METEOR is a release track, with versions like `1.0` and `1.0.4-rc.0.). Records have the following keys:

- `name` : (String) name of the track, such as `METEOR`
- `maintainers` : (Array of objects with key `username`) array representing the release maintainers
- `lastUpdated` : (Date) last time this record was updated

Release versions

Stores one record for each release version. Records have the following keys:

- `track` : (String) name of the track, such as `METEOR`
- `version` : (String) release version, such as `1.0` or `1.0.1-albatross`
- `description` : (String) short description of the release version
- `recommended` : (Boolean) true for recommended releases
- `tool` : (String) version of the Meteor tool to be used in this release, such as `meteor-tool@1.0.39`
- `packages` : (Object, keyed from package name to version) Package versions included in this release.
- `orderKey` : (String) an order key for this release, automatically generated for semver-like release names. Meteor uses this to pick the correct release for `meteor update` .
- `published` : (Date) date of publication of this release version
- `publishedBy` : (Object with key `username`) user who published this release
- `lastUpdated` : (Date) last time this record was updated

> *Note: Meteor package names can include periods. This means that the keys in the `packages` field may include periods, which is an invalid object schema in many databases, including Mongo.*