

Ethereum Improvement Proposals

All Core Networking Interface ERC Meta Informational

 This EIP had no activity for at least 6 months.

EIP-1319: Smart Contract Package Registry Interface <>

Author	Piper Merriam, Christopher Gewecke, g. nicholas d'andrea, Nick Gheorghita
Discussions-To	https://github.com/ethereum/EIPs/issues/1319
Status	Stagnant
Type	Standards Track
Category	ERC
Created	2018-08-13

Table of Contents

- [Simple Summary](#)
- [Abstract](#)
- [Motivation](#)
- [Specification](#)
 - [Examples](#)
 - [Write API Specification](#)
 - [Events](#)
 - [Read API Specification](#)
- [Rationale](#)
- [Backwards Compatibility](#)

- [Implementation](#)
- [Copyright](#)

Simple Summary

A standard interface for smart contract package registries.

Abstract

This EIP specifies an interface for publishing to and retrieving assets from smart contract package registries. It is a companion EIP to [1123](#) which defines a standard for smart contract package manifests.

Motivation

The goal is to establish a framework that allows smart contract publishers to design and deploy code registries with arbitrary business logic while exposing a set of common endpoints that tooling can use to retrieve assets for contract consumers.

A clear standard would help the existing EthPM Package Registry evolve from a centralized, single-project community resource into a decentralized multi-registry system whose constituents are bound together by the proposed interface. In turn, these registries could be ENS name-spaced, enabling installation conventions familiar to users of `npm` and other package managers.

Examples

```
$ ethpm install packages.zeppelin.eth/Ownership
```

```
const SimpleToken = await web3.packaging
    .registry('packages.ethpm.eth')
    .getPackage('simple-token')
    .getVersion('^1.1.5');
```

Specification

The specification describes a small read/write API whose components are mandatory. It allows registries to manage versioned releases using the conventions of [semver](#) without imposing this as a requirement. It assumes registries will share the following structure and conventions:

- a **registry** is a deployed contract which manages a collection of **packages**.
- a **package** is a collection of **releases**

- a **package** is identified by a unique string name and unique bytes32 **packageId** within a given **registry**
- a **release** is identified by a `bytes32` **releaseId** which must be unique for a given package name and release version string pair.
- a **releaseId** maps to a set of data that includes a **manifestURI** string which describes the location of an [EIP 1123 package manifest](#). This manifest contains data about the release including the location of its component code assets.
- a **manifestURI** is a URI as defined by [RFC3986](#) which can be used to retrieve the contents of the package manifest. In addition to validation against RFC3986, each **manifestURI** must also contain a hash of the content as specified in the [EIP-1123](#).

Examples

Package Names / Release Versions

```
"simple-token" # package name  
"1.0.1"       # version string
```

Release IDs

Implementations are free to choose any scheme for generating a **releaseId**. A common approach would be to hash the strings together as below:

```
// Hashes package name and a release version string  
function generateReleaseId(string packageName, string version)  
    public  
    view  
    returns (bytes32 releaseId)  
{  
    return keccak256(abi.encodePacked(packageName, version));  
}
```

Implementations **must** expose this id generation logic as part of their public `read` API so tooling can easily map a string based release query to the registry's unique identifier for that release.

Manifest URIs

Any IPFS or Swarm URI meets the definition of **manifestURI**.

Another example is content on GitHub addressed by its SHA-1 hash. The Base64 encoded content at this hash can be obtained by running:

```
$ curl https://api.github.com/repos/:owner/:repo/git/blobs/:file_sha

# Example
$ curl https://api.github.com/repos/rstallman/hello/git/blobs/ce013625030ba8dba90
```

The string "hello" can have its GitHub SHA-1 hash independently verified by comparing it to the output of:

```
$ printf "blob 6\000hello\n" | sha1sum
> ce013625030ba8dba906f756967f9e9ca394464a
```

Write API Specification

The write API consists of a single method, `release`. It passes the registry the package name, a version identifier for the release, and a URI specifying the location of a manifest which details the contents of the release.

```
function release(string packageName, string version, string manifestURI) public
returns (bytes32 releaseId);
```

Events

VersionRelease

MUST be triggered when `release` is successfully called.

```
event VersionRelease(string packageName, string version, string manifestURI)
```

Read API Specification

The read API consists of a set of methods that allows tooling to extract all consumable data from a registry.

```
// Retrieves a slice of the list of all unique package identifiers in a registry.
// `offset` and `limit` enable paginated responses / retrieval of the complete set
function getAllPackageIds(uint offset, uint limit) public view
returns (
    bytes32[] packageIds,
    uint pointer
);
```

```

// Retrieves the unique string `name` associated with a package's id.
function getPackageName(bytes32 packageId) public view returns (string packageName)

// Retrieves the registry's unique identifier for an existing release of a package
function getReleaseId(string packageName, string version) public view returns (bytes32 releaseId)

// Retrieves a slice of the list of all release ids for a package.
// `offset` and `limit` enable paginated responses / retrieval of the complete set
function getAllReleaseIds(string packageName, uint offset, uint limit) public view
    returns (
        bytes32[] releaseIds,
        uint pointer
    );

// Retrieves package name, release version and URI location data for a release id
function getReleaseData(bytes32 releaseId) public view
    returns (
        string packageName,
        string version,
        string manifestURI
    );

// Retrieves the release id a registry *would* generate for a package name and version
// when executing a release.
function generateReleaseId(string packageName, string version)
    public
    view
    returns (bytes32 releaseId);

// Returns the total number of unique packages in a registry.
function numPackageIds() public view returns (uint totalCount);

// Returns the total number of unique releases belonging to the given packageName
function numReleaseIds(string packageName) public view returns (uint totalCount);

```

Pagination

`getAllPackageIds` and `getAllReleaseIds` support paginated requests because it's possible that the return values for these methods could become quite large. The methods should return a `pointer` that points to the next available item in a list of all items such that a caller can use it to pick up from where the previous request left off. (See [here](#) for a discussion of the merits and

demerits of various pagination strategies.) The `limit` parameter defines the maximum number of items a registry should return per request.

Rationale

The proposal hopes to accomplish the following:

- Define the smallest set of inputs necessary to allow registries to map package names to a set of release versions while allowing them to use any versioning schema they choose.
- Provide the minimum set of getter methods needed to retrieve package data from a registry so that registry aggregators can read all of their data.
- Define a standard query that synthesizes a release identifier from a package name and version pair so that tooling can resolve specific package version requests without needing to query a registry about all of a package's releases.

Registries may offer more complex `read` APIs that manage requests for packages within a semver range or at `latest` etc. This EIP is agnostic about how tooling or registries might implement these. It recommends that registries implement [EIP-165](#) and avail themselves of resources to publish more complex interfaces such as [EIP-926](#).

Backwards Compatibility

No existing standard exists for package registries. The package registry currently deployed by EthPM would not comply with the standard since it implements only one of the method signatures described in the specification.

Implementation

A reference implementation of this proposal is in active development at the EthPM organization on GitHub [here](#).

Copyright


Copyright and related rights waived via [CC0](#).

Citation

Please cite this document as:

Piper Merriam, Christopher Gewecke, g. nicholas d'andrea, Nick Gheorghita, "EIP-1319: Smart Contract Package Registry Interface," *Ethereum Improvement Proposals*, no. 1319, August 2018. [Online serial]. Available: <https://eips.ethereum.org/EIPS/eip-1319>.

Ethereum Improvement Proposals

Ethereum Improvement Proposals  [ethereum/EIPs](https://github.com/ethereum/EIPs) Ethereum Improvement Proposals (EIPs) describe standards for the Ethereum platform, including core protocol specifications, client APIs, and contract standards.