EthPM Package Manifest Documentation

Piper Merriam, et al.

May 21, 2018

Contents:

CHAPTER 1

Overview

1.1 Background

These docs are meant to provide insight into the EVM Smart Contract Packaging Specification and facilitate implementation and adoption of these standards.

CHAPTER 2

Glossary

ABI The JSON representation of the application binary interface. See the official specification for more information.

Address A public identifier for an account on a particular chain

Bytecode The set of EVM instructions as produced by a compiler. Unless otherwise specified this should be assumed to be hexadecimal encoded, representing a whole number of bytes, and prefixed with a `'0x'`.

Bytecode can either be linked or unlinked. (see *Linking*)

Unlinked Bytecode The hexadecimal representation of a contract's EVM instructions that contains sections of code that requires *linking* for the contract to be functional.

The sections of code which are unlinked **must** be filled in with zero bytes.

Example: `0x606060405260e06000730000000000000000000000000000000000000000000000000000634d536f`

Linked Bytecode The hexadecimal representation of a contract's EVM instructions which has had all *Link References* replaced with the desired *Link Values*.

Example: `0x606060405260e06000736fe3600060405160200152604051816e060020a634d536f`

Contract Instance A contract instance a specific deployed version of a *Contract Type*.

All contract instances have an *Address* on some specific chain.

Contract Type Refers to a specific contract in the package source. This term can be used to refer to an abstract contract, a normal contract, or a library. Two contracts are of the same contract type if they have the same bytecode.

Example:

```
contract Wallet {
    ...
}
```

A deployed instance of the `Wallet` contract would be of of type `Wallet`.

Link Reference A location within a contract's bytecode which needs to be linked. A link reference has the following properties.

`offset` Defines the location within the bytecode where the link reference begins.

`length` Defines the length of the reference.

`name` (optional.) A string to identify the reference

Link Value A link value is the value which can be inserted in place of a *Link Reference*

Linking The act of replacing *Link References* with *Link Values* within some *Bytecode*.

Package Distribution of an application's source or compiled bytecode along with metadata related to authorship, license, versioning, et al.

For brevity, the term **Package** is often used metonymously to mean *Package Manifest*.

Package Manifest A machine-readable description of a package (See *Package Specification* for information about the format for package manifests.)

CHAPTER 3

Package Specification

This document defines the specification for a **Package**. The Package JSON document provides metadata about itself and in most cases should provide sufficient information about the packaged contracts and its dependencies to do bytecode verification of its contracts.

3.1 Guiding Principles

The Package specification makes the following assumptions about the document lifecycle.

1. Packages are intended to be generated programatically by package management software as part of the release process.
2. Packages will be consumed by package managers during tasks like installing package dependencies or building and deploying new releases.
3. Packages will typically **not** be stored alongside the source, but rather by package registries *or* referenced by package registries and stored in something akin to IPFS.

3.2 Keywords

3.2.1 RFC2119

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

- https://www.ietf.org/rfc/rfc2119.txt

3.2.2 Custom

Prefixed vs Unprefixed

A prefixed hexadecimal value begins with `'0x'`. Unprefixed values have no prefix. Unless otherwise specified, all hexadecimal values should be represented with the `'0x'` prefix.

Prefixed `0xdeadbeef`

Unprefixed `deadbeef`

Contract Name

The name found in the source code which defines a specific *Contract Type*. These names **must** conform to the regular expression `[a-zA-Z][-a-zA-Z0-9_]{0,255}`

There can be multiple contracts with the same contract name in a projects source files.

Contract Alias

This is a name used to reference a specific *Contract Type*. Contract aliases **must** be unique within a single *Package*.

The *contract alias* **must** use *one of* the following naming schemes.

- `<contract-name>`
- `<contract-name>[<identifier>]`

The `<contract-name>` portion **must** be the same as the *contract name* for this contract type.

The `[<identifier>]` portion **must** match the regular expression `\[[-a-zA-Z0-9]{1,256}\]`.

Contract Instance Name

A name which refers to a specific *Contract Instance* on a specific chain from the deployments of a single *Package*. This name **must** be unique across all other contract instances for the given chain. The name must conform to the regular expression `[a-zA-Z][a-zA-Z0-9_]{0,255}`

In cases where there is a single deployed instance of a given *Contract Type*, package managers **should** use the *contract alias* for that contract type for this name.

In cases where there are multiple deployed instances of a given contract type, package managers **should** use a name which provides some added semantic information as to help differentiate the two deployed instances in a meaningful way.

Identifier

A string matching the regular expression `[a-zA-Z][-_a-zA-Z0-9]{0,255}`

Package Name

A string matching the regular expression `[a-zA-Z][-_a-zA-Z0-9]{0,255}`

Content Addressable URI

Any URI which contains a cryptographic hash which can be used to verify the integrity of the content found at the URI.

The URI format is defined in RFC3986

It is **recommended** that tools support IPFS and Swarm.

Chain Definition

This definition originates from BIP122 URI.

A URI in the format `blockchain://<chain_id>/block/<block_hash>`

- `chain_id` is the unprefixed hexadecimal representation of the genesis hash for the chain.
- `block_hash` is the unprefixed hexadecimal representation of the hash of a block on the chain.

A chain is considered to match a chain definition if the the genesis block hash matches the `chain_id` and the block defined by `block_hash` can be found on that chain. It is possible for multiple chains to match a single URI, in which case all chains are considered valid matches

3.3 Format

The canonical format for the Package JSON document containing a single JSON object. Packages **must** conform to the following serialization rules.

- The document **must** be tightly packed, meaning no linebreaks or extra whitespace.
- The keys in all objects must be sorted alphabetically.
- Duplicate keys in the same object are invalid.
- The document **must** use UTF-8 encoding.
- The document **must** not have a trailing newline.

3.4 Document Specification

The following fields are defined for the Package. Custom fields may be included. Custom fields **should** be prefixed with `x-` to prevent name collisions with future versions of the specification.

3.4.1 EthPM Manifest Version: `manifest_version`

The `manifest_version` field defines the specification version that this document conforms to. Packages **must** include this field.

Required Yes

Key `manifest_version`

Type String

Allowed Values `2`

3.4.2 Package Name: `package_name`

The `package_name` field defines a human readable name for this package. Packages **must** include this field. Package names **must** begin with a lowercase letter and be comprised of only lowercase letters, numeric characters, and the dash character `'-'`. Package names **must** not exceed 214 characters in length.

Required Yes

Key `package_name`

Type String

Format **must** be a valid package name.

3.4.3 Package Meta: `meta`

The `meta` field defines a location for metadata about the package which is not integral in nature for package installation, but may be important or convenient to have on-hand for other reasons. This field **should** be included in all Packages.

Required No

Key `meta`

Type Object (String: *Package Meta* object)

3.4.4 Version: `version`

The `version` field declares the version number of this release. This value **must** be included in all Packages. This value **should** conform to the semver version numbering specification.

Required Yes

Key `version`

Type String

3.4.5 Sources: `sources`

The `sources` field defines a source tree that **should** comprise the full source tree necessary to recompile the contracts contained in this release. Sources are declared in a key/value mapping.

Key `sources`

Type Object (String: String)

Format

- Keys **must** be relative filesystem paths beginning with a `./`. Paths **must** resolve to a path that is within the current working directory.
- Values **must** conform to *one of* the following formats.
  - Source string.
  - When the value is a source string the key should be interpreted as a file path.
  - *Content Addressable URI*.
  - *If* the resulting document is a directory the key should be interpreted as a directory path.

- *If* the resulting document is a file the key should be interpreted as a file path.

3.4.6 Contract Types: `contract_types`

The `contract_types` field holds the *Contract Types* which have been included in this release. *Packages* **should** only include contract types that can be found in the source files for this package. Packages **should not** include contract types from dependencies.

Key `contract_types`

Type Object (String: *Contract Type Object*)

Format

- Keys **must** be valid *contract aliases*.
- Values **must** conform to the *Contract Type Object* definition.

Packages **should not** include abstract contracts in the contract types section of a release.

3.4.7 Deployments: `deployments`

The `deployments` field holds the information for the chains on which this release has *Contract Instances* as well as the *Contract Types* and other deployment details for those deployed contract instances. The set of chains defined by the BIP122 URI keys for this object **must** be unique.

Key `deployments`

Type Object (String: Object(String: *Contract Instance Object*))

Format

- Keys **must** be a valid BIP122 URI *chain definition*.
- Values **must** be objects which conform to the format:
  - Keys **must** be a valid *Contract Instance Name*.
  - Values **must** be a valid *Contract Instance Object*.

3.4.8 Build Dependencies: `build_dependencies`

The `build_dependencies` field defines a key/value mapping of ethereum packages that this project depends on.

Key `dependencies`

Type Object (String: String)

Format

- Keys **must** be valid *package names* matching the regular expression `[a-z][-a-z0-9]{0,213}`
- Values **must** be valid IPFS URIs which resolve to a valid Package

3.5 Definitions

Definitions for different objects used within the Package. All objects allow custom fields to be included. Custom fields **should** be prefixed with `x-` to prevent name collisions with future versions of the specification.

3.5.1 The *Link Reference* Object

A **|LinkReference| object** has the following key/value pairs. All link references are assumed to be associated with some corresponding bytecode.

Offsets: `offsets`

The `offsets` field is an array of integers, corresponding to each of the start positions where the link reference appears in the bytecode. Locations are 0-indexed from the beginning of the bytes representation of the corresponding bytecode. This field is invalid if it references a position that is beyond the end of the bytecode.

Required Yes

Type Array

Length: `length`

The `length` field is an integer which defines the length in bytes of the link reference. This field is invalid if the end of the defined link reference exceeds the end of the bytecode.

Required Yes

Type Integer

Name: `name`

The `name` field is a string which **must** be a valid *Identifier*. Any link references which **should** be linked with the same link value **should** be given the same name.

Required No

Type String

Format **must** conform to the *Identifier* format.

3.5.2 The *Link Value* Object

A **|LinkValue| object** is defined to have the following key/value pairs.

Offsets: `offsets`

The `offsets` field defines the locations within the corresponding bytecode where the `value` for this link value was written. These locations are 0-indexed from the beginning of the bytes representation of the corresponding bytecode.

Required Yes

Type Integer

Format Array of integers, where each integer **must** conform to all of the following:

- be greater than or equal to zero
- strictly less than the length of the unprefixed hexadecimal representation of the corresponding bytecode.

Type: `type`

The `type` field defines the `value` type for determining what is encoded when *linking* the corresponding bytecode.

Required Yes

Type String

Allowed Values

- `'literal'` for bytecode literals
- `'reference'` for named references to a particular *Contract Instance*

Value: `value`

The `value` field defines the value which should be written when *linking* the corresponding bytecode.

Required Yes

Type String

Format determined based on `type`:

Type `literal` For static value literals (e.g. address), value **must** be a *byte string*

Type `reference` To reference the address of a *Contract Instance* from the current package the value should be the name of that contract instance.

- This value **must** be a valid contract instance name.
- The chain definition under which the contract instance that this link value belongs to must contain this value within its keys.
- This value **may not** reference the same contract instance that this link value belongs to.

To reference a contract instance from a *Package* from somewhere within the dependency tree the value is constructed as follows.

- Let `[p1, p2, .. pn]` define a path down the dependency tree.
- Each of `p1, p2, pn` **must** be valid package names.
- `p1` **must** be present in keys of the `build_dependencies` for the current package.
- For every `pn` where `n > 1`, `pn` **must** be present in the keys of the `build_dependencies` of the package for `pn-1`.
- The value is represented by the string `<p1>:<p2>:<...>:<pn>:<contract-instance>` where all of `<p1>`, `<p2>`, `<pn>` are valid package names and `<contract-instance>` is a valid *contract name*.
- The `<contract-instance>` value **must** be a valid *contract instance name*.
- Within the package of the dependency defined by `<pn>`, all of the following must be satisfiable:
  - There **must** be *exactly* one chain defined under the `deployments` key which matches the chain definition that this link value is nested under.
  - The `<contract-instance>` value **must** be present in the keys of the matching chain.

3.5.3 The *Bytecode* Object

A bytecode object has the following key/value pairs.

Bytecode: `bytecode`

The `bytecode` field is a string containing the `0x` prefixed hexadecimal representation of the bytecode.

Required Yes

Type String

Format *0x* prefixed hexadecimal.

Link References: `link_references`

The `link_references` field defines the locations in the corresponding bytecode which require *linking*.

Required No

Type Array

Format All values **must** be valid *Link Reference objects*

This field is considered invalid if *any* of the *Link References* are invalid when applied to the corresponding `bytecode` field, *or* if any of the link references intersect.

Intersection is defined as two link references which overlap.

Link Dependencies: `link_dependencies`

The `link_dependencies` defines the *Link Values* that have been used to link the corresponding bytecode.

- Required: No
- Type: Array
- Format: All values **must** be valid *Link Value objects*

Validation of this field includes the following:

- No two link value objects may contain any of the same values for `offsets`.
- Each *link value object* **must** have a corresponding *link reference object* under the `link_references` field.
- The length of the resolved `value` **must** be equal to the `length` of the corresponding *Link Reference*.

3.5.4 The *Package Meta* Object

The *Package Meta* object is defined to have the following key/value pairs.

Authors: `authors`

The `authors` field defines a list of human readable names for the authors of this package. Packages **may** include this field.

Required No

Key `authors`

Type List of Strings

License: `license`

The `license` field declares the license under which this package is released. This value **should** conform to the SPDX format. Packages **should** include this field.

Required No

Key `license`

Type String

Description: `description`

The `description` field provides additional detail that may be relevant for the package. Packages **may** include this field.

Required No

Key `description`

Type String

Keywords: `keywords`

The `keywords` field provides relevant keywords related to this package.

Required No

Key `keywords`

Type List of Strings

Links: `links`

The `links` field provides URIs to relevant resources associated with this package. When possible, authors **should** use the following keys for the following common resources.

`website` Primary website for the package.

`documentation` Package Documentation

`repository` Location of the project source code.

Key `links`

Type Object (String: String)

3.5.5 The *Contract Type* Object

A *Contract Type* object is defined to have the following key/value pairs.

Contract Name: `contract_name`

The `contract_name` field defines the *contract name* for this *Contract Type*.

Required If the *contract name* and *contract alias* are not the same.

Type String

Format **must** be a valid *contract name*.

Deployment Bytecode: `deployment_bytecode`

The `deployment_bytecode` field defines the bytecode for this *Contract Type*.

Required No

Type Object

Format **must** conform to *the Bytecode Object* format.

Runtime Bytecode: `runtime_bytecode`

The `runtime_bytecode` field defines the unlinked `'0x'` prefixed runtime portion of *Bytecode* for this *Contract Type*.

Required No

Type Object

Format **must** conform to *the Bytecode Object* format.

ABI: `abi`

Required No

Type List

Format see https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI#json

Natspec: `natspec`

Required No

Type Object

Format The Merged *UserDoc* and *DevDoc*

- UserDoc
- DevDoc

Compiler: `compiler`

Required No

Type Object

Format **must** conform to *the Compiler Information object* format.

3.5.6 The *Contract Instance* Object

A **|ContractInstance| Object** is defined to have the following key/value pairs.

Contract Type: `contract_type`

The `contract_type` field defines the *Contract Type* for this *Contract Instance*. This can reference any of the contract types included in this *Package or* any of the contract types found in any of the package dependencies from the `build_dependencies` section of the *Package Manifest*.

Required Yes

Type String

Format **must** conform to one of the following formats

To reference a contract type from this Package, use the format `<contract-alias>`.

- The `<contract-alias>` value **must** be a valid *contract alias*.
- The value **must** be present in the keys of the `contract_types` section of this Package.

To reference a contract type from a dependency, use the format `<package-name>:<contract-alias>`.

- The `<package-name>` value **must** be present in the keys of the `build_dependencies` of this Package.
- The `<contract-alias>` value **must** be be a valid *contract alias*.
- The resolved package for `<package-name>` must contain the `<contract-alias>` value in the keys of the `contract_types` section.

Address: `address`

The `address` field defines the *Address* of the *Contract Instance*.

Required Yes

Type String

Format Hex encoded `'0x'` prefixed Ethereum address matching the regular expression `0x[0-9a-fA-F]{40}`.

Transaction: `transaction`

The `transaction` field defines the transaction hash in which this *Contract Instance* was created.

Required No

Type String

Format `0x` prefixed hex encoded transaction hash.

Block: `block`

The `block` field defines the block hash in which this the transaction which created this *contract instance* was mined.

Required No

Type String

Format `0x` prefixed hex encoded block hash.

Runtime Bytecode: `runtime_bytecode`

The `runtime_bytecode` field defines the runtime portion of bytecode for this *Contract Instance*. When present, the value from this field supersedes the `runtime_bytecode` from the *Contract Type* for this *Contract Instance*.

Required No

Type Object

Format **must** conform to *the Bytecode Object* format.

Every entry in the `link_references` for this bytecode **must** have a corresponding entry in the `link_dependencies` section.

Compiler: `compiler`

The `compiler` field defines the compiler information that was used during compilation of this *Contract Instance*. This field **should** be present in all *Contract Types* which include `bytecode` or `runtime_bytecode`.

Required No

Type Object

Format **must** conform to the *Compiler Information Object* format.

3.5.7 The *Compiler Information* Object

The `compiler` field defines the compiler information that was used during compilation of this *Contract Instance*. This field **should** be present in all contract instances that locally declare `runtime_bytecode`.

A *Compiler Information* object is defined to have the following key/value pairs.

Name `name`

The `name` field defines which compiler was used in compilation.

Required Yes

Key `type`:

Type String

Version: `version`

The `version` field defines the version of the compiler. The field **should** be OS agnostic (OS not included in the string) and take the form of either the stable version in semver format or if built on a nightly should be denoted in the form of `<semver>-<commit-hash>` ex: `0.4.8-commit.60cc1668`.

Required Yes

Key `version`:

Type String

Settings: `settings`

The `settings` field defines any settings or configuration that was used in compilation. For the `'solc'` compiler, this **should** conform to the Compiler Input and Output Description.

Required No

Key `settings`:

Type Object

3.5.8 BIP122 URIs

BIP122 URIs are used to define a blockchain via a subset of the BIP-122 spec.

```
blockchain://<genesis_hash>/block/<latest confirmed block hash>
```

The `<genesis hash>` represents the blockhash of the first block on the chain, and `<latest confirmed block hash>` represents the hash of the latest block that's been reliably confirmed (package managers should be free to choose their desired level of confirmations).

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index