Specification

This document defines the specification for an EthPM package manifest. A package manifest provides metadata about a Package, and in most cases should provide sufficient information about the packaged contracts and its dependencies to do bytecode verification of its contracts.

Guiding Principles

This specification makes the following assumptions about the document lifecycle.

1. Package manifests are intended to be generated programatically by package management software as part of the release process.
2. Package manifests will be consumed by package managers during tasks like installing package dependencies or building and deploying new releases.
3. Package manifests 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.

Conventions

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

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`

Document Format

The canonical format is 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.

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.

See Also: Formalized (JSON-Schema) version of this specification: package.spec.json

Jump To: Definitions

EthPM Manifest Version: `manifest`

The `manifest` field defines the specification version that this document conforms to.

- Packages **must** include this field.

Required: Yes

Key: `manifest`

Type: String

Allowed Values: `ethpm/3`

Package Name: `name`

The `name` field defines a human readable name for this package.

- Packages **should** include this field to be released on an EthPM registry.
- 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 255 characters in length.

Required: If `version` is included.

Key: `name`

Type: String

Format: **must** match the regular expression `^[a-z][-a-z0-9]{0,255}$`

Package Version: `version`

The `version` field declares the version number of this release.

- Packages **should** include this field to be released on an EthPM registry.
- This value **should** conform to the semver version numbering specification.

Required: If `name` is included.

Key: `version`

Type: String

Package Metadata: `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: Package Meta Object

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.

Required: No

Key: `sources`

Type: Object (String: Sources Object)

Contract Types: `contractTypes`

The `contractTypes` field hosts 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.
- Packages **should not** include abstract contracts in the contract types section of a release.

Required: No

Key: `contractTypes`

Type: Object (String: Contract Type Object)

Format:

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

Compilers: `compilers`

The `compilers` field holds the information about the compilers and their settings that have been used to generate the various `contractTypes` included in this release.

Required: No

Key: `compilers`

Type: Array (the Compiler Information object)

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. There cannot be two different URI keys in a deployments field representing the same blockchain.

Required: No

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 following format.**
  - Keys **must** be valid Contract Instance Names.
  - Values **must** be a valid Contract Instance Object.

Build Dependencies: `buildDependencies`

The `buildDependencies` field defines a key/value mapping of Ethereum Packages that this project depends on.

Required: No

Key: `buildDependencies`

Type: Object (String: String)

Format:

- Keys **must** be valid package-names matching the regular expression `^[a-z][-a-z0-9]{0,255}$`.
- Values **must** be a Content Addressable URI which resolves to a valid package that conforms the same EthPM manifest version as its parent.

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.

The *Link Reference* Object

A Link Reference 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.

The *Link Value* Object

Describes a single Link Value.

A **Link Value 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: See Below.

Format

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

- 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`, see below.

Format

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

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.

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: `linkReferences`

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

Required:	No
Type:	Array
Format:	All values **must** be valid Link Reference objects. See also below.

Format

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: `linkDependencies`

The `linkDependencies` 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. See also below.

Format

Validation of this field includes the following:

- Two link value objects **must not** contain any of the same values for `offsets`.
- Each link value object **must** have a corresponding link reference object under the `linkReferences` field.
- The length of the resolved `value` **must** be equal to the `length` of the corresponding Link Reference.

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: Array (String)

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: Array(String)

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)

The *Sources* Object

A *Sources* object is defined to have the following fields.

Key: A global identifier for the source file. (string)

Value: SourceObject

Source Object

Checksum: `checksum`

Hash of the source file.

Required: **If** there are no URIs provided that contain a content hash.

Key: `checksum`

Value: ChecksumObject

URLs: `urls`

Array of urls that resolve to the same source file.

- Urls **should** be stored on a content-addressable filesystem.
- Urls **must** be prefixed with a scheme.
- 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.

Required: If `content` is not present.

Key: `urls`

Value: Array(string)

Content: `content`

Inlined contract source.

Required: If `urls` is not present.

Key: `content`

Value: string

Install Path: `installPath`

Filesystem path of source file.

- **Must** be a relative filesystem path that begins with a `./`.
- **Must** resolve to a path that is within the current virtual working directory.
- **Must** be unique across all included sources.

Required: This field **must** be included for the package to be writable to disk.

Key: `installPath`

Value: string

The *Checksum* Object

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

Algorithm: `algorithm`

The `algorithm` used to generate the corresponding hash.

Required: Yes

Type: String

Hash: `hash`

The `hash` of a source files contents generated with the corresponding algorithm.

Required: Yes

Type: String

The *Contract Type* Object

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

Contract Name: `contractName`

The `contractName` 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: `deploymentBytecode`

The `deploymentBytecode` field defines the bytecode for this Contract Type.

Required: No

Type: Object

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

Runtime Bytecode: `runtimeBytecode`

The `runtimeBytecode` 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: Array

Format: **must** conform to the Ethereum Contract ABI JSON format.

Natspec: `natspec`

Required: No

Type: Object

Format: The union of the UserDoc and DevDoc formats.

The *Contract Instance* Object

A **Contract Instance Object** represents a single deployed Contract Instance and is defined to have the following key/value pairs.

Contract Type: `contractType`

The `contractType` 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 `buildDependencies` section of the Package Manifest.

Required: Yes

Type: String

Format: See Below.

Format

Values for this field **must** conform to *one of* the two formats herein.

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 `contractTypes` 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 `buildDependencies` 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 `contractTypes` 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: `runtimeBytecode`

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

Required: No

Type: Object

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

Every entry in the `linkReferences` for this bytecode **must** have a corresponding entry in the `linkDependencies` section.

The *Compiler Information* Object

The `compilers` field defines the various compilers and settings used during compilation of any Contract Types or Contract Instance included in this pacakge.

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: `name`

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

Contract Types: `contractTypes`

A list of the Contract Alias in this package that used this compiler to generate its outputs.

- All `contractTypes` that locally declare `runtimeBytecode` **should** be attributed for by a compiler object.
- A single `contractTypes` **must** not be attributed to more than one compiler.

Required: No

Key: `contractTypes`

Type: Array(Contract Alias)

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).