
XsefPM Package Manifest Documentation

FreightTrust and Clearing

Oct 17, 2020

Contents:

1	Overview	1
1.1	Background	1
2	XSEF Specification [WIP]	3
2.1	Design Principles	3
2.2	Conventions	3
2.3	Document Format	4
2.4	Document Specification	4
2.5	Definitions	7
3	Use Cases	19
3.1	Keywords	19
3.2	Stand Alone Package with an Inheritable Transaction	20
4	Glossary	21
	Index	25

1.1 Background

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

eXtended Standard Exchange Format

> Specification for a modern SEF implementation

The eXtended Standard Exchange Format ('XSEF') is an open-standard file format - files ending with the extension .xsef.cfg - that defines the format and the format and implementation guideline for proprietary and standard EDI documents.

XSEF holds key advantages over other file formats in that it is designed for EDI

XSEF files are immediately useable by both users and computers They're small files and are easily transmitted easily via the web You can edit them with either a text editor or an XSEF manager XSEF is an open standard, so you can create and distribute XSEF files without special permissions or royalties under the Mozilla Public License 2.0.

The sections may appear in any order, with these exceptions: the .VER, if present, must be first. The .INI section must be the first section in the file if there is no .VER, or the second section if there is a .VER. Nothing may appear before these, including a comment. The .STD record, if used, must appear before .SETS. Example

```
` .VER 1.6 .INI KAVERPO,,004 010,X,X12-4010,Kaver Corp X12-4010 Purchase Order
.STD ,RE .SETS `
```

```
` [1] INVPO[2],,003[3]040,[4]X,X12-[5]3040,PO `
```

```
` `INVPO,,003 040,X,X12-3040,PO and INV for Slippers 'n Socks, Inc.` `
```

1. The standard or implementation name (INVPO in the example above), generally the same as the filename of the SEF file.

2. Reserved
3. The Functional Group Version, Release and Industry code which will identify the standard in any Functional Group Envelope Header Segment. Each code is separated by a space. In the example, there is no industry code. With an industry code, this field might contain: 003 030 UCS
4. The responsible agency code, which identifies the standards organization in the Functional Group Header: GC = GENDOD T = for T.D.C.C. (EDIA) TD = TRADACOMS UN = for UN/EDIFACT X = for ASC X12 (DISA)
5. The standard on which this implementation guidelines is based.
6. The description (title) of the implementation guideline.

The .VER section identifies the version and release of SEF, which is currently 1.6. It should be the first record in the file. If the .VER section is not present, SEF 1.0 is assumed.

These sections can occur anywhere after the .INI section. The .PRIVATE section provides a place for companies to place information that is useful to themselves but is of no interest to others. The .PUBLIC section marks the end of the private section.

.STD is only included for these standards: • Newer EDIFACT standards in which groups have position numbers • Newer EDIFACT and X12 standards that have repeating elements • Fixed-length standards like GENCOD.

The .SETS section defines the transaction set or message directory, including: • Each transaction set or message in the standard. • For each transaction set or message, it lists each segment reference in the order in which it appears. It also describes the requirement and quantity for each segment when it appears in a particular position in that set. • Loops, groups, and tables are also set up. In this section, all information about loops, groups, and segments is hierarchical: for example, the quantity for a PER segment only applies to that particular position in that particular set or message.

The .COMS section is the standard's composite data element dictionary: a list of all composites in the standard. It includes: • The composite name (C001, etc.). • A list of each subelement reference it contains, in order. • Each subelement's ID and requirement when used in this position of this composite. • Subelement relationals used within the composite. • Subelement repeat counts. • Masks for variations in the structure of a composite. If the standard has no composites, this section will be omitted.

The .ELMS section is the standard's data element dictionary: a list of each element, its type, and its minimum and maximum data value lengths.

The .CODES section is a list of each element that has code values, along with its code values. It also provides information about code sets.

SPDX-License-Identifier: MPL-2.0

XSEF Specification [WIP]

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

2.1 Design 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.
-

2.2 Conventions

2.2.1 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>

2.2.2 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`

2.3 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.
 - To ensure backwards compatibility, *manifest_version* is a forbidden top-level key.
-

2.4 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.

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

Jump To [Definitions](#)

2.4.1 XsefPM 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 `xsefpm/1`

2.4.2 Package Name: `name`

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

- Packages **should** include this field to be released on an XsefPM 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}$`

2.4.3 Package Version: `version`

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

- Packages **should** include this field to be released on an XsefPM registry.
- This value **should** conform to the [semver](#) version numbering specification.

Required If `name` is included.

Key `version`

Type String

2.4.4 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*

2.4.5 Sources: `sources`

The `sources` field defines a source tree that **should** comprise the full source tree necessary to recompile the transactions contained in this release.

Required No

Key `sources`

Type Object (String: *Sources Object*)

2.4.6 Transaction Types: `transactionTypes`

The `transactionTypes` field hosts the *Transaction Types* which have been included in this release.

- Packages **should** include transaction types that can be found in the source files for this package.
- Packages **should not** include transaction types from dependencies.
- Packages **should not** include abstract transactions in the transaction types section of a release.

Required No

Key `transactionTypes`

Type Object (String: *Transaction Type Object*)

Format

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

2.4.7 Transaction Groups: `transaction groups`

The `transaction groups` field holds the information about the transaction groups and their issuing agency (e.g. ASC or U.N). These determine the settings that can be used to generate the various `transactionTypes` included in this release.

Required Yes

Key `transaction groups`

Type Array (*the Transaction Group Information object*)

2.4.8 Deployments: `deployments`

The `deployments` field holds the information for the **Trading Channell** instance on which this release has *Transaction Instances* as well as the *Transaction Types* and other deployment details for those deployed transaction instances. The set of instance 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: *Transaction Instance Object*))

Format

- Keys **must** be a valid URI/BIP122 URI URL and/or chain definition.
 - Values **must be objects which conform to the following format.**
 - Keys **must** be valid *Transaction Instance Names*.
 - Values **must** be a valid *Transaction Instance Object*.
-

2.4.9 Build Dependencies: `buildDependencies`

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

Required No

Key `buildDependencies`

Type Object (String: String)

Format

- Keys **must** be valid *package-names*.
 - Values **must** be a *Content Addressable URI* which resolves to a valid package that conforms the same XsefPM manifest version as its parent.
-

2.5 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.

2.5.1 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 *Schema*.

Offsets: `offsets`

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

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 schema.

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.

2.5.2 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 schema where the value for this link value was written. These locations are 0-indexed from the beginning of the bytes representation of the corresponding schema.

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 unprefix hexadecimal representation of the corresponding schema.

Type: `type`

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

Required Yes

Type String

Allowed Values "literal" for schema literals

"reference" for named references to a particular *Transaction Instance*

Value: `value`

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

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 *Transaction Instance* from the current package the value should be the name of that transaction instance.

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

To reference a transaction 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>:<transaction-instance> where all of <p1>, <p2>, <pn> are valid package names and <transaction-instance> is a valid *Transaction Name*.
 - The <transaction-instance> value **must** be a valid *Transaction 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 <transaction-instance> value **must** be present in the keys of the matching chain.
-

2.5.3 The *Schema* Object

A schema object has the following key/value pairs.

Schema: *schema*

The *schema* field is a string containing the 0x prefixed hexadecimal representation of the schema.

Required Yes

Type String

Format 0x prefixed hexadecimal.

Link References: *linkReferences*

The *linkReferences* field defines the locations in the corresponding schema 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 schema 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 schema.

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*.
-

2.5.4 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 Array (String)

License: `license`

The `license` field declares the license associated with this package. This value **should** conform to the *SPDX* format. Packages **should** include this field. If a file *Source Object* defines its own license, that license takes precedence for that particular file over this package-scoped meta license.

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)

2.5.5 The *Sources* Object

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

Key A unique identifier for the source file. (string)

Value *SourceObject*

Source Object**Checksum: `checksum`**

Hash of the source file.

Required If there are no URLs present 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. **If** they are not, then either `content` or `checksum` **must** be included.
- `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 included.

Key `urls`

Value `Array(string)`

Content: `content`

Inlined transaction source.

Required If `urls` is not included.

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`

Type: `type`

The `type` field declares the type of the source file. The field **should** be one of the following values: `solidity`, `vyper`, `abi-json`, `solidity-ast-json`.

Required No

Key `type`

Type `String`

License: license

The `license` field declares the type of license associated with this source file. When defined, this license overrides the package-scoped *Meta License*.

Required No

Key `license`

Type String

2.5.6 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

2.5.7 The *Transaction Type* Object

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

Transaction Name: transactionName

The `transactionName` field defines the *Transaction Name* for this *Transaction Type*.

Required If the *Transaction Name* and *Transaction Alias* are not the same.

Type String

Format **Must** be a valid *Transaction Name*.

Source ID: sourceId

The global source identifier for the source file from which this transaction type was generated.

Required No

Type String

Value **Must** match a unique source ID included in the *Sources Object* for this package.

Deployment Schema: `deploymentSchema`

The `deploymentSchema` field defines the schema for this *Transaction Type*.

Required No

Type Object

Format **Must** conform to *the Schema Object* format.

Grammar Schema: `grammarSchema`

The `grammarSchema` field defines the unlinked 0x-prefixed grammar portion of *Schema* for this *Transaction Type*.

Required Yes

Type Object

Format **Must** conform to *the Schema Object* format.

ABI: `abi`

Required No

Type Array

Format **Must** conform to the *Ethereum Transaction ABI JSON* format.

UserDoc: `userdoc`

Required No

Type Object

Format **Must** conform to the *UserDoc* format.

DevDoc: `devdoc`

Required No

Type Object

Format **Must** conform to the *DevDoc* format.

2.5.8 The *Transaction Instance* Object

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

Transaction Type: `transactionType`

The `transactionType` field defines the *Transaction Type* for this *Transaction Instance*. This can reference any of the transaction types included in this *Package* or any of the transaction 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 transaction type from this Package, use the format `<transaction-alias>`.

- The `<transaction-alias>` value **must** be a valid *Transaction Alias*.
- The value **must** be present in the keys of the `transactionTypes` section of this Package.

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

- The `<package-name>` value **must** be present in the keys of the `buildDependencies` of this Package.
- The `<transaction-alias>` value **must** be a valid *Transaction Alias*.
- The resolved package for `<package-name>` must contain the `<transaction-alias>` value in the keys of the `transactionTypes` section.

Contract Address: `contract_address`

The `contract_address` field defines the **Contract Address** of the *Transaction 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 *Transaction 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 *transaction instance* was mined.

Required No

Type String

Format 0x prefixed hex encoded block hash.

grammar Schema: `grammarSchema`

The `grammarSchema` field defines the grammar portion of schema for this *Transaction Instance*. When present, the value from this field supersedes the `grammarSchema` from the *Transaction Type* for this *Transaction Instance*.

Required No

Type Object

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

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

2.5.9 The *Transaction Group Information* Object

The `transaction groups` field defines the various transaction groups and settings used during compilation of any *Transaction Types* or *Transaction Instance* included in this package.

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

Name `name`

The `name` field defines which transaction group was used in compilation.

Required Yes

Key `name`

Type String

Version: `version`

The `version` field defines the version of the transaction group. 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.

Required No

Key `settings`

Type Object

Transaction Types: `transactionTypes`

A list of the *Transaction Alias* in this package that used this transaction group to generate its outputs.

- All `transactionTypes` that locally declare `grammarSchema` **should** be attributed for by a transaction group object.
- A single `transactionTypes` **must** not be attributed to more than one transaction group.

Required No

Key `transactionTypes`

Type `Array(Transaction Alias)`

2.5.10 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).

The following use cases were considered during the creation of this specification.

HOSTNAME + PATH_TO_RESOURCE + “?” + CANONICAL_QUERY_STRING + “&X-Goog-Signature=” + REQUEST_SIGNATURE

Stand Alone Package with an Inheritable Transaction e.g. 810

See full description.

Dependent Package with an Inheritable Transaction e.g. transferable

See full description.

Stand Alone Package with a Reusable Transaction e.g. standard-token

See full description.

Each use case builds incrementally on the previous one.

3.1 Keywords

Stand Alone Package has no external dependencies (i.e. no `build_dependencies`), contains all transaction data needed without reaching into another package.

Dependent Package does not contain all necessary transaction data (i.e. has `build_dependencies`), **must** reach into a package dependency to retrieve data.

Inheritable Transaction doesn’t provide useful functionality on it’s own and is meant to serve as a base transaction for others to inherit from.

Reusable Transaction is useful on it’s own, meant to be used as-is.

Deployed Transaction/Library Refers to an instance of a transaction/library that has already been deployed to a specific address on a chain.

Package Dependency External dependency directly referenced via the `build_dependencies` of a package.

Deep Dependency External dependency referenced via the `build_dependencies` of a package dependency (or by reaching down dependency tree as far as necessary).

3.2 Stand Alone Package with an Inheritable Transaction

For the first example we'll look at a package which only contains transactions which serve as base transactions for other transactions to inherit from but do not provide any real useful functionality on their own. The common *edifact* pattern is an example for this use case.

For this example we will assume this file is located in the schema source file `./transactions/edifact.sol`

The *edifact* package contains a single schema source source file which is intended to be used as a base transaction for other transactions to be inherited from. The package does not define any pre-deployed addresses for the *edifact* transaction.

The smallest Package for this example looks like this:

```
{
  "manifest_version": "2",
  "version": "1.0.0",
  "package_name": "edifact",
  "sources": {
    "./transactions/edifact/D40.INVOC.json": " some string "
  }
}
```

A Package which includes more than the minimum information would look like this.

This fully fleshed out Package is meant to demonstrate various pieces of optional data that can be included. However, for the remainder of our *edifact* we will be using minimalistic Packages to keep our *edifact* as succinct as possible.

Terms utilized throughout the specification

API The JSON representation of the application programming interface. *see Open EDI*

Address A public identifier for an account on a particular chain or URI

Example: “ HOSTNAME + PATH_TO_RESOURCE + “?” + CANONICAL_QUERY_STRING + “&X-Goog-Signature=” + REQUEST_SIGNATURE “

Schema The XSD or JSON-Schema defining the Trading Channel

Schema can either be linked or unlinked. (see [Linking](#))

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

Example: “ <FIXME> “

Linked Schema A representation of a non-standard schema. The Schema utilizes a [Link References](#) to an Unlinked Schema, and can be replaced with the desired [Link Values](#).

Example: “ <FIXME> “

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

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

Example: “ HOSTNAME + PATH_TO_RESOURCE + “?” + CANONICAL_QUERY_STRING + “&X-Goog-Signature=” + REQUEST_SIGNATURE “

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

The transaction alias **must** use *one of* the following naming schemes:

- <transaction-name>
- <transaction-name><identifier>

The <transaction-name> portion **must** be the same as the *Transaction Name* for this transaction type.

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

Transaction Instance A transaction instance a specific deployed version of a *Transaction Type*.

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

Transaction Instance Name A name which refers to a specific *Transaction Instance* on a specific chain from the deployments of a single *Package*. This name **must** be unique across all other transaction 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 *Transaction Type*, package managers **should** use the *Transaction Alias* for that transaction type for this name.

In cases where there are multiple deployed instances of a given transaction 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.

Transaction Name The name found in the source code that defines a specific *Transaction Type*. These names **must** conform to the regular expression `^[a-zA-Z_$][a-zA-Z0-9_$]{0,255}$`.

There can be multiple transactions with the same transaction name in a projects source files.

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

Example:

```
transaction edifact.INVOC {  
    ...  
}
```

A deployed instance of the INVOC transaction would be of of type INVOC.

Identifier Refers generally to a named entity in the *Package*.

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

Link Reference A location within a transaction'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 *Schema*.

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 v2-package-specification for information about the format for package manifests.)

Prefixed *Schema* string with leading 0x.

Example 0xdeadb33f

Unprefixed Not *Prefixed*.

Example deadb33f

A

Address, [21](#)

API, [21](#)

C

Chain Definition, [21](#)

Content Addressable URI, [21](#)

I

Identifier, [22](#)

L

Link Reference, [22](#)

Link Value, [22](#)

Linking, [22](#)

P

Package, [22](#)

Package Manifest, [23](#)

Prefixed, [23](#)

S

Schema, [21](#)

T

Transaction Alias, [22](#)

Transaction Instance, [22](#)

Transaction Instance Name, [22](#)

Transaction Name, [22](#)

Transaction Type, [22](#)

U

Unprefixed, [23](#)