1. Introduction

This file describes the data model of Alexandria. Definitions of the data are in the model package. The document consists of four parts:

- Data definition of the state data on the blockchain.
- Data definition of the transaction payload.
- Data definition of the client-side data, shared by the Major Tool and the Client.
- Data definition of events sent from the blockchain to the client side.

The remainder of this file motivates the chosen data storage and marshaling tools.

2. State Data on the Blockchain

We had to choose between JSON and Google Protocol Buffers. JSON is easier to debug, but it turns out that marshaling Golang variables into JSON is not deterministic as pointed out here: https://stackoverflow.com/questions/44755089/does-serialized-content-strictly-follow-the-order-in-definition-use-encoding-jso. Google Protocol Buffers is not deterministic too in theory, but in practice it is as long as no maps are applied in the GPB data definitions. This is explained here: https://havoc.io/post/deterministic-protobuf/. The choice for Google Protocol Buffers is based on these sources.

Using UUIDs, we can arrange that every Sawtooth address holds exactly one value. Each item mentioned in requirement AX-5050 has its id calculated as follows. When the item is created, the client doing so should first create a UUID. Then this UUID is hashed using SHA-512 and the hex representation of the hash is taken. The digits a-f should be small caps. Then the first 62 digits are taken. The final address becomes:

<6-digid transaction family> + <2-digid type code> + <62-digid remainder>

The exception to this scheme is the price list, which has a fixed address. This allows the blockchain to find its bootstrap information.

Timestamps are stored as integer values, the number of seconds since Unix Epoch. Timestamps are stored in 64-bit signed integers.

Optional fields do not need to be handled with omittd fields or null fields. The empty string is good enough, because an empty string is not a meaningful value itself.

The remainder of this section 2 gives detailed information per address type.

2.1. Settings

The Settings address is the only address that is not randomly chosen. It is the 6-digid transaction family appended with 0x00 repeated 32 times. The contents of this address is a marshaled Google Protocol Buffers message. The message has the following fields:

- priceMajorEditSettings int32.
- priceMajorCreatePerson int32.
- priceMajorChangePersonAuthorization int32.
- priceMajorChangeJournalAuthorization int32.
- pricePersonEdit int32.
- priceAuthorSubmitNewManuscript int32.
- priceAuthorSubmitNewVersion int32.
- priceAuthorAcceptAuthorship int32.
- priceReviewerSubmit int32.
- priceEditorAllowManuscriptReview int32.
- priceEditorRejectManuscript int32.
- priceEditorPublishManuscript int32.
- priceEditorAssignManuscript int32.
- priceEditorCreateJournal int32.
- priceEditorCreateVolume int32.
- priceEditorEditJournal int32.
- priceEditorAddColleague int32.
- priceEditorAcceptDuty int32.

There is no price for bootstrapping and for resigning as editor. Charging bootstrapping makes no sense because initially no one has credit. Charging resigning as editor is not logical. If an editor does not have credit, she can not do her job. The only sensible thing to do is resigning.

There is no need to save the bootstrap key in the settings address, because the bootstrap key is immediately embedded in person with isMajor = true.

2.2. Person

Person addresses have type code 0x01. The contents of a Person address is a marshaled Google Protocol Buffers message. The message has the following fields:

- id: string, should equal the address it appears in.
- createdOn: int64
- modifiedOn: int64
- publicKey: string, never blank.
- name: string, never blank
- email: string, not initialized as blank, but can be modified to blank.
- isMajor: bool, not null.
- isSigned: bool, not null.
- balance: int32, not null.
- biographyHash: string, empty string means there is no bibliography.
- biographyFormat: string, should be the empty string if there is no bibliography.

- organization: string, empty string means not set.
- telephone: string, empty string means not set.
- address: string, empty string means not set.
- postalCode: string, empty string means not set.
- country: string, empty string means not set.
- extraInfo: string, empty string means not set.

The createdOn and modifiedOn times are seconds since Epoch.

2.3. Manuscript and ManuscriptThread

Manuscript addresses have a type code of 0x10. The contents of a Manuscript address is a marshaled Google Protocol Buffers message. The message has the following fields:

- id: string, should equal the address it appears in.
- createdOn: int64.
- modifiedOn: int64.
- hash: string, not blank.
- manuscriptFormat: string, not blank.
- threadId: string, refers to a manuscript thread address.
- versionNumber: int32.
- commitMsg: string.
- title: string.
- author: Author repeated.
- status: ManuscriptStatus.
- journalId: string, not blank.
- volumeId: string, refers to a volume address.
- firstPage: string.
- lastPage: string.

The type Author refers to another Google Protocol Buffers message, which has the following fields:

- authorId: string, refers to a person address.
- didSign: bool.
- authorNumber: int32.

The type ManuscriptStatus is an enum with the following possible values:

- INIT
- NEW
- REVIEWABLE
- REJECTED
- PUBLISHED
- ASSIGNED

ManuscriptThread addresses have type code 0x18. The contents of a Manuscript-Thread address is a marshaled Google Protocol Buffers message. The message has the following fields:

- id: string, should equal the address it appears in.
- manuscriptId: string repeated, not null.
- isReviewable: bool, not null.

A manuscript thread does not have a createdOn or a modifiedOn field because that would duplicate the information in the referenced manuscripts. Logically, the creation date of a manuscript thread is the creation date of the first manuscript. And the modification date of a manuscript thread is the latest modification date comparing the modification dates of the manuscripts.

2.4. Journal and Volume

Journal addresses have type code 0x20. The contents of a Journal address is a marshaled Google Protocol Buffers message. The message has the following fields:

- id: string, should equal the address it appears in.
- createdOn: int64.
- modifiedOn: int64.
- title: string, not blank.
- isSigned: bool, not null.
- descriptionHash, the empty string means there is no description.
- descriptionFormat, should be the empty string if there is no description.
- editorInfo EditorInfo repeated.

The type EditorInfo refers here to another Google Protocol Buffers message. EditorInfo has the following fields:

- editorId: string, not null. The id of a person.
- editorState: EditorState, not null.

EditorState is an enum with the possible values EDITOR_PROPOSED and EDITOR_ACCEPTED.

Volume addresses have type code 0x28. The contents of a Volume address is a marshaled Google Protocol Buffers message. The message has the following fields:

- id: string, should equal the address it appears in.
- createdOn: int64.
- journalId: string. References the address of the journal it belongs to.
- issue: string.

There is no modifiedOn because volumes are not modified, apart from initially adding the manuscripts.

2.5. Review

Review addresses have type code 0x30. The contents of a Review address is a marshaled Google Protocol Buffers message. The message has the following fields:

- id: string, should equal the address it appears in.
- createdOn: int64.
- manuscriptId: string, references a manuscript address.
- reviewAuthorId: string, references a person address.
- hash: string, not blank.
- format: string, not blank.
- judgement: Judgement.
- isUsedByEditor: bool.

There is no modifiedOn because the only possible modification is setting isUsedByEditor. This field is only set when a manuscript is published, which is a status change of the manuscript.

Judgement is an enum with possible values REJECTED and ACCEPTED.

3. Transaction Payload

We chose Google Protocol Buffers because we did for state data. There are different kinds of transactions that have to fit in a common data structure. This could be achieved by combining a type value and a marshaled Google Protocol Buffers message into one byte array, but this is more difficult than including everything in one Google Protocol Buffers messages. Google Protocol Buffers allows fields to be combined into a OneOf-clause, allowing only one of the fields to be present. Using this approach, we combine a set of common header fields with one type-specific message.

Common fields of each transaction are:

- signer: string, the person address that identifies the person who signs.
- timestamp: int64. The client-generated timestamp.
- price: int32. The amount to be charged according to the transaction author. The transaction is invalid if the price is incorrect.

The remainder of section 3 specifies the type-specific messages.

3.1. Settings messages

There is a bootstrap message that has the following fields:

- prices: PriceList
- firstMajor: PersonCreate

PriceList is the message that holds the prices in the settings address. PersonCreate is the person create message given in section 3.2.1.

Prices are updated with a SettingsUpdate message. For each priceXXX field mentioned in section 2.1, it has a field priceXXXUpdate that is of type IntUpdate. Message IntUpdate has members OldValue and NewValue, both of type int32. It is omitted if there is no change. Otherwise, priceXXXUpdate.OldValue is the expected old price and priceXXXUpdate.NewValue is the new price to set.

3.2. Person messages

There is a person create message that is treated in subsection 3.2.1. There are three message types related to updating persons that are covered in subsections 3.2.2 - 3.2.4.

The person update messages cover all person fields mentioned in section 2.2 except the following:

- createdOn
- modifiedOn
- biographyFormat

The createdOn is the creation date and should not be updated. The modifiedOn field can not be updated independently, the transaction payload message header field timestamp is the new modifiedOn time of a modified person. The biographyFormat cannot be updated because it is fixed according to requirement AX-1090.

3.2.1. Person create

This message has the following fields:

- newPersonId: string
- publicKey: string
- name: string
- email: string

Not all fields mentioned in section 2.2. are filled on create. All fields can later be filled by the update messages defined in the next subsections.

3.2.2. Person property update

This message has the following fields:

- id: string, the subject being updated.
- publicKeyUpdate: StringUpdate.
- nameUpdate: StringUpdate.
- emailUpdate: StringUpdate.
- biographyHashUpdate: StringUpdate.
- organizationUpdate: StringUpdate.
- telephoneUpdate: StringUpdate.
- addressUpdate: StringUpdate.
- postalCodeUpdate: StringUpdate.

- countryUpdate: StringUpdate.
- extraInfoUpdate: StringUpdate.

StringUpdate is a Google Protocol Buffers similar to IntUpdate. A StringUpdate field is omitted if the value is not updated. If it is set, the OldValue and NewValue fields define the update.

3.2.3. Person authorization update

This message has the following fields:

- id: string, the subject being updated.
- makeMajor: bool, nullable.
- makeSigned: bool, nullable.

3.2.4. Person balance increment

This message has the following fields:

- id: string, the subject being updated.
- balanceIncrement: int32, not null.

3.3. Manuscript messages

This section present the manuscript-related messages, including the creation of reviews.

3.3.1. Create new manuscript (AX-1540)

This message has the following fields:

- manuscriptId: string.
- manuscriptThreadId: string.
- hash: string.
- manuscriptFormat: string.
- commitMsg: string, may be blank.
- title: string, not blank.
- authorId: string repeated. Each string is a person id.
- journalId: string, not blank.

The sequence of the author ids in their repeated field is significant. The index is the author number.

3.3.2. Create new manuscript version (AX-1550)

This message has the following fields.

- manuscriptId: string.
- previousManuscriptId: string.
- manuscriptThreadId: string.

- versionNumber: int32.
- hash: string.
- manuscriptFormat: string.
- commitMsg: string, not blank.
- title: string, not blank.
- authorId: string repeated. Each string is a person id.

3.3.3. Sign for being author (AX-1560)

This message has the following fields:

- manuscriptId: string, not blank.

3.3.4. Allow manuscript review (AX-1570)

This message has the following fields:

- manuscriptThreadId: string, not blank.
- manuscriptId: string repeated.

The manuscriptId repeated field lists all manuscripts in the thread. These addresses should be writable and they should be in the outputs of the transaction. Therefore it is clear to also require them in the message and check them with the blockchain state. The order of the maniscriptId items in the message is not important.

3.3.5. Write review (AX-1580)

This message has the following fields:

- reviewId: string
- manuscriptId: string
- hash: string, not blank.
- format: string, not blank.
- judgement: Judgement.

Reviews are always signed by their authors.

3.3.6. Judge manuscript (AX-1590)

This message has the following fields:

- manuscriptId: string
- reviewId: string repeated
- judgement: Judgement

The reviewId lists all the reviews the judgement is based on, requirement AX-1590. See section 2.5 for the definition of the Judgement enum.

3.3.7. Assign volume (AX-1600)

This message has the following fields:

- manuscriptId: string.
- volumeId: string.
- firstPage: string.
- lastPage: string.

3.4. Journal messages

This section lists journal and volume-related messages used as transaction payload.

3.4.1. Create journal (AX-2030)

The create journal message has the following fields:

- journalId: string, references a person.
- title: string, not blank.
- descriptionHash: string, is the empty string if not set.
- descriptionFormat: string, is the empty string if there is no description.

3.4.2. Update journal properties (AX-2040)

The update journal properties message has the following fields:

- journalId: string
- titleUpdate: StringUpdate
- descriptionHashUpdate: StringUpdate
- descriptionFormatUpdate: StringUpdate

See section 3.2.2. about StringUpdate.

3.4.3. Update journal authorization (AX-2050)

The update journal authorization message has the following fields:

- journalId: string, not blank.
- makeSigned: bool, not null.

3.4.4. Journal editor resign (AX-2060)

The journal editor resign message has the following fields:

- journalId: string.

3.4.5. Journal editor invite (AX-2070)

The journal editor invite message has the following fields:

- journalId: string.
- invitedEditorId: string.

3.4.6. Journal editor accept duty (AX-2080)

The journal editor accept duty message has the following fields:

- journalId: string.

3.4.7. Volume create (AX-2100)

The volume create message has the following fields:

- volumeId: string.
- journalId: string.
- issue: string.

3.5. Review messages

See section 3.3.5 for creating reviews.

4. Client-side data

On the client side, searching data is important. We hold the data in a SQLite 3 database. This way, no remote database is needed. The data resides in a local file and can be maintained with SQL statements.

Timestamps are stored as integer values, the number of seconds since Epoch. Timestamps are stored in 64-bit signed integers.

Optional fields do not need to be handled with null values. The empty string is good enough, because an empty string is not a meaningful value itself.

The following tables are created:

- Settings.
- Person.
- Manuscript.
- Author.
- Journal.
- Editor.
- Volume.
- Review.

Each of these tables is treated in its own subsection:

4.1. Settings

This table holds one row holding all the prices. The price names of section 2.1. are used as field names. In addition, there are createdOn and modifiedOn fields.

4.2. Person

The Person table has the following fields:

- id: string
- createdOn: int64
- modifiedOn: int64
- publicKey: string.
- name: string.
- email: string.
- isMajor: bool.
- isSigned: bool.
- balance: int32.
- biographyHash: string.
- biographyFormat: string.
- organization: string.
- telephone: string.
- address: string.
- postalCode: string.
- country: string.
- extraInfo.

4.3. Manuscript

The Manuscript table has the following fields.

- id: string.
- createdOn: int64.
- modifiedOn: int64.
- hash: string, not blank.
- manuscriptFormat: string, not blank.
- threadId: string.
- versionNumber: int32.
- commitMsg: string.
- title: string.
- status: ManuscriptStatus.
- journalId: string, not blank.
- volumeId: string.
- firstPage: string.
- lastPage: string.
- isReviewable: bool.

There is no table for manuscript threads. Therefore, we need the isRevieable field.

4.4. Author

This table has the following fields.

- manuscriptId: string
- personId: string.
- didSign: bool.
- authorNumber: int32.

4.5. Journal

The Journal table has the following fields:

- journalId: string, should equal the address it appears in.
- createdOn: int64.
- modifiedOn: int64.
- title: string, not blank.
- isSigned: bool, not null.
- descriptionHash, the empty string means there is no description.
- descriptionFormat, should be the empty string if there is no description.

4.6. Editor

The Editor table has the following fields.

- journalId.
- personId.
- editorState.

4.7. Volume

The Volume table has the following fields:

- volumeId: string.
- createdOn: int64.
- journalId: string.
- issue: string.

4.8. Review

The Review table has the following fields:

- id: string, should equal the address it appears in.
- createdOn: int64.
- manuscriptId: string, references a manuscript address.
- reviewAuthorId: string, references a person address.
- hash: string, not blank.
- format: string, not blank.

- judgement: Judgement.
- isUsedByEditor: bool.

5. Events

Sawtooth events have the following fields:

- The event type, a string.
- Attributes, which are name/value pairs.
- A byte array, which is opaque to Hyperledger Sawtooth.

The event type and the attributes can be used to filter events. Therefore, we give every event an empty byte array, putting all the meaning in the event type and the attributes. Integer or boolean values corresponding to attributes have to be encoded as strings.

Each event will have the properties “signerId”, “timestamp”, “transactionId” and “eventSeq”. The transaction id is the signature of the transaction header as explained in the Sawtooth documentation. The attributes “transactionId” and “eventSeq” are used to detect missed events as explained in section 5.9.

Hyperledger Sawtooth bundles multiple events in a sing Google Protocol Buffers message. Therefore, one transaction can easily create many events. We create an event type for each combination of a table and an action. We introduce separate event types xxxModificationTime to direct the client to update the modifiedOn field. This way, the modifiedOn field is updated only once for a transaction. xxxUpdate events thus do not use the timestamp. Create events do fill the createdOn field with the timestamp of the event.

All event types are listed in the following subsections.

5.1. Settings

5.1.1. Event type settingsCreate

This event has each price mentioned in section 2.1. as an attribute. The corresponding values are the prices.

5.1.2. Event type settingsUpdate

A settingsUpdate event holds the update of one price. It only has the updated price name as attribute, with the new price as value.

5.1.3. Event type settingsModificationTime

This event direct the client to update the modification time of the settings. We do not want to repeat this update for each field update.

5.2. Person

5.2.1. Event type personCreate

This event fills only the most important fields. The others have to be filled with personUpdate events. The following attributes have to be filled:

- personId
- publicKey
- name
- email

5.2.2. Event type personUpdate

This event updates one of the fields of the Person table. The attribute personId should always be set, and one of the following:

- publicKey.
- name.
- email.
- isMajor.
- isSigned.
- balance.
- biographyHash.
- biographyFormat.
- organization.
- telephone.
- address.
- postalCode.
- country.
- extraInfo.

5.2.3. Event type personModificationTime

In addition to the common attributes “signerId” and “timestamp”, only “personId” is needed. This event directs the client to update the modification time. This is not repeated for every field update.

5.3. Manuscript

5.3.1. Event type manuscriptCreate

The manuscript create event requires all of the following attributes to be set.

- manuscriptId.
- hash.
- manuscriptFormat.
- threadId.
- versionNumber.
- commitMsg.

- title.
- status.
- journalId.

5.3.2. Event type manuscriptUpdate

The “manuscriptId” attribute should always be available for this event. In addition, one of the following attributes should be available:

- status.
- volumeId.
- firstPage.
- lastPage.
- isReviewable.

5.3.3. Event type manuscriptModificationTime

This event is similar to ther xxxModificationTime events.

5.4. Author

5.4.1. Event type authorCreate

This event requires all of the following attributes to be available:

- manuscriptId.
- personId.
- authorNumber.

5.4.2. Event type authorUpdate

This event requires all of the following attributes:

- manuscriptId.
- didSign.

5.5. Journal

5.5.1. Event type journalCreate

This event requires the following attributes:

- journalId.
- title.

5.5.2. Event type journalUpdate

This event requires the attribute “journalId” and one of the following:

- isSigned.
- descriptionHash.

- descriptionFormat.

5.5.3. Event type journalUpdateModificationTime

Like similar events for other tables.

5.6. Editor

5.6.1. Event type editorCreate

This event has all of the following attributes:

- journalId.
- personId.
- editorState.

5.6.2. Event type editorUpdate

This event has all of the following attributes:

- journalId.
- personId.
- editorState.

5.6.3. Event type editorDelete

This event has all of the following attributes:

- journalId.
- personId.

5.7. Volume

5.7.1. Event type volumeCreate

This event has the following attributes:

- volumeId: string.
- journalId: string.
- issue: string.

5.8. Review

5.8.1. Event type reviewCreate

This event requires all of the following attributes to be available:

- reviewId.
- manuscriptId.
- reviewAuthorId.
- hash.

- format.
- judgement.

5.8.2. Event type reviewUpdate

This event requires all of the following attributes:

- reviewId.
- isUsedByEditor.

5.9. Event type transactionNumEvents

For each transaction, one event of type transactionNumEvents is generated. It can be used by the client to see whether all events of a transaction have been received. The event has requires the following attribute:

- numEvents.

For example, when a transactionNumEvents is generated with eventSeq = 0 (see the beginning of section 5) and numEvents = 5, then the client should expect four additional events for the transaction with eventSeq = 1, eventSeq = 2, eventSeq = 3 and eventSeq = 4.