RFC: Ecosystem Service Credit Module

PREAMBLE:
This document adheres to the following Structured process for RFCs. While the decision making process surrounding Regen Network's use of RFCs has yet to be formalized, this structure should give us a lightweight format to start working with.

Once the document has moved from DRAFT to IN REVIEW status, it is expected that community and team members provide feedback until the "To be reviewed by" date, at which point the RFC will be closed to comments, and a decision on whether or not the proposal is moved forward with will be made. Without an explicit decision making process noted in the RFC, it is assumed to use consensus decision making.

Authors:	Aaron Craelius, Cory Levinson
Created at:	02/27/2020
To be reviewed by:	4/21/2020
State:	ACCEPTED

Need

Core to Regen Network's value proposition is the ability for entities (individuals or organizations) to be able to design and issue credits for ecosystem services in the form of on-chain assets on the Regen Ledger.

Driving the initial technical specification for these credits is our work on Regen Registry, a public Ecosystem Service registry built on top of Regen Ledger. Regen Registry makes it possible to issue, trade and retire ecosystem service credits, including for instance soil health credits, carbon credits, biodiversity credits, etc.

Taking Regen Registry's pilot projects as our initial use case, we understand the needs of an ecosystem service credit module to be as follows (for a given Credit Class):

- Credits are represented as a fungible on-chain asset, where on-chain accounts can have a balance in the given credit
- Credits can be issued/minted at any time by a fixed set of "issuers"
- Credits are issued in batches (a batch of credits is hereafter referred to as a credit vintage)
- Upon issuance, a credit vintage points to a project, and geo-polygon, and mints all credits in that vintage to a set of accounts (typically the land steward / project owner)

    - Whether the project identifier, geo-polygon, and other metadata are stored on-chain or off-chain with a link, is to be specified in this RFC
    - There should be the ability for credits to be immediately retired on issuance (as set by the issuer)
- Credits for a given vintage can change state between “tradable” and “retired”
    - Tradable credits can be transferred between accounts, by the owner
    - Retired credits cannot be transferred, and cannot be unretired

Although the needs illustrated above are meant to primarily address our own internal use cases, we welcome further input & feedback from the larger crypto and ecosystem services communities. Our hope is that the circulation of this document enables us to broaden our understanding of what needs we should be considering when designing a system for on-chain credits. Feedback will either be addressed directly via revisions to this document, or by factoring feedback into future versions of this specification.

Approach

Previous Work

Prior to this RFC, initial specifications have been worked on in 2019 in the form of an initial Go specification on github.

Protobuf Definitions

The corresponding protocol buffer definitions for the approach outlined in this RFC can be found here.

Definitions

Credit Class

A credit class defines a type of credits that is maintained by a credit designer and issued by a credit issuer.

Credit Designer

A credit designer is the authority responsible for creating a credit class and updating its list of approved issuers as needed.

Credit Issuer

A credit issuer can issue credit vintages to project developers based on successful satisfaction of methodology constraints.

Credit Batch

A credit batch refers to a batch of credits issued at a single point in time (usually corresponding to some off or on-chain verification event, and corresponding to some project).

Credit

In this design credit vintages can be split up into any fractional amount (arbitrary precision decimal) as needed and thus credit vintages are the top-level thing issued, but they can be split up as needed. Credits is thus a loose term to describe some quantities of credits potentially of different vintages and classes. “One credit” would generally refer to 1.0 units of a given credit vintage.

Retirement

Retirement is the state in which a credit can no longer be transferred. In conventional blockchain terminology, this is practically equivalent to the burned state and the word burn may be used in the technical implementation. The main difference is that we still care to actively track the balance of retired credits. Conceptually retiring a credit implies that the holder of a credit is “consuming” the credit as an offset to satisfy voluntary or compliance-related offset commitments.

Credits that are retired cannot be un-retired by either the credit issuer, or credit designer.

Operations

Create Credit Class

The create credit class operation creates a new class of a credit class.

Arguments:

- The list of initial issuers of the new credit class
- arbitrary metadata bytes (optional)

The party signing this transaction is the credit designer. This operation will return a new credit class ID.

Update Credit Class

The update credit class operation will allow for the following to be changed:

- the list of approved issuers
- the credit designer
- arbitrary metadata bytes attached to the credit

Issue

The issue operation issues a credit vintage of a credit class. It must be signed by an approved issuer of the desired credit class and specify who the receiver of the issued credits will be as well as the number of units to issue in this vintage and metadata as described below.

In order to support use cases when credits are to be immediately retired upon issuance, for each account to be issued credits, both an amount of tradeable and retired credit units can be specified.

The arguments for the issue operation are thus:

- Credit Class ID
- Issuer
- Metadata
- List of:
    - Receiving account
    - Tradable units
    - Retired units

This operation will return a new credit vintage ID.

Metadata

Credit issuance requires a number of pieces of metadata to be provided. When a credit vintage is issued, the issuer can provide as an argument arbitrary metadata bytes. It is intended that this metadata field be used for the following types of information:

- Project Identifier
- Geography (in the form of a geo-polygon (eg: for agricultural classes)
- Dates (Start & End Dates that this vintage represents)

The metadata should be specified in the credit class. All credit vintage MUST provide metadata based on the respective credit class metadata specification.

The above fields may be included on-chain (serialized as a JSON string or protobuf), or linked to as off-chain data, in which case the metadata value would be a URI for resolving the off-chain data.

Overlap Behavior

It is generally illegal to issue a credit vintage of the same credit class for overlapping polygons and dates although the blockchain state machine will not explicitly enforce this as off-chain auditing and slashing is a more efficient and robust way to do this.

Send

The send operation transfers some units of a credit vintage from the current holder to another account. Retired/burned credits cannot be transferred.

Sends can be performed only by the account owner, or an authorized representative (via some external module like message authorization).

Retire/Burn

The retire or burn operation retires some units of a credit vintage that the signer of the transaction holds, making them non-transferrable.

Retiring can be done by either the credit owner, or the credit issuer.

Rationale

Fractional NFT Design

The design described above can be understood as a Fractional NFT (Non-Fungible Token) design where a credit vintage represents a non-fungible asset uniquely identified by its ID and some corresponding metadata. Each credit vintage then consists of a fixed total quantity of credits that are represented as fungible tokens.

If credits were represented as fully fungible tokens, then this would mean all credits of a given credit class would be completely interchangeable with each other regardless of the vintage they came from. This may indeed be the desired behavior in many cases, but representing a credit class in this way would remove the ability to be able to uniquely track credits from a given vintage, which is important for auditing purposes.

The other alternative, to preserve full auditability of all credits, would require a traditional NFT design, where each individual atomic credit unit (e.g. 1 ton of carbon) were represented by a unique ID as a non-fungible asset. While this would provide the most auditability, it drastically increases the technical complexity, and introduces usability issues (e.g. each individual credit would need to be traded as an individual transaction).

On-chain Metadata

This proposal focuses on minimizing on-chain metadata as much as possible. Benefits of this approach include:

- Reducing technical complexity, thus reducing development time

- Allowing for specific use cases to drive future functionality
- Reduce chain size and make it more responsive and more performant.

Specifically, the metadata fields in the credit class and credit vintage could be designed to be much more explicit, with several required on-chain metadata fields. Since we are not planning on implementing on-chain scraping of credit metadata, it seems more appropriate to allow for arbitrary bytes, and allow for standards for metadata best practices to evolve over a longer period of time once we can generalize across several use cases.

Complex State Machines

Another design decision made in this proposal was to only implement two possible states for credits: “`tradable`” and “`retired`”. These are the two known states that we need to be able to manage for our first use case with the registry. While it is assumed that there may be more complex state machines that we will need to support at a later date, we chose to start with satisfying the current use case for the same reasons illustrated above (reducing complexity, and allowing future use cases to drive more complex functionality or generalizations).