# Valence Protocol Scope

## Valenodes

As described in the whitepaper Valenodes are registered with a collateral transaction and must perform network services. These are the backbone of the Valence Platform. Valenodes can be setup by anyone who meets the collateral and service level requirements. Services include;

* Minting blocks on the main UTXO chain.
* Creating and propagating Vapp lateral datachain definitions.
* Validating & serving Vapp’s lateral datachains.
* Routing encrypted messaging.
* Handling VQL queries from applications or users.
* Cross chain data oracle providers.

## Encrypted Messaging Protocol

Also described in the whitepaper, the EMP is a way to broadcast messages to pubkeyhashes, decoupling messages from IP addresses. This creates the first masternode/collateral protected encrypted onion routing network.

## Valence Data Schema

The overall data schema of Valence would consist of;

1. A UTXO blockchain which manages the VALE token.
2. A vapp\_registry datachain to manage Vapps state.
3. A valenode\_registry datachain to manage Valenodes state.
4. [n] datachain’s for each Vapp which has been registered.

## Lateral Datachains w. declarative conditional datastores

As with the UTXO blockchain, datachains consist of two leveldb data stores (index and chainstate) and the records are arranged into a merkle tree block structure for propagation.

<page 1, 2 & 3>

The vapp\_registry datachain (non UTXO blockchain) manages the initialization of Vapps. This contains a record which is linked to a payment on the UTXO chain, multiple signatures which are required to perform root changes to the Vapp, assigns a Vapp\_id and instructs the daemon to create a new datatachain for the newly registered Vapp with the generated Vapp\_id as it’s reference.

Isolating these records allow for the client to know which apps are available without needing to store every application’s data.

To give Vapps flexibility, rather than storing all the rules for the Vapp in a single record in the Vapp\_regirstry, the idea here is that the registration record is a pointer to an isolated data store that is built up of multiple user defined collections. Effectively creating a malleable, declarative structure and ruleset for consensus.

Each Vapp’s datachain would consist of at minimum 3 collections; schema, auth and data[n].

The schema contains records which define each data[n] collection’s structure and ruleset. Schema records can be of type “add”, “remove” or “redirect”, to link to or unlink or migrate collections in a way which means the data submitted before the change can still be validated. There should also be a record type which locks in the structure so it can no longer be modified if data structure permanence is critical to the function of the application e.g. it will be relied on by other Vapps.

<page 4>

The authchain contains records which specify whether data[n] is public, or if only a certain set of pubkeys can write records. Records can be of type “add” or “remove” to grant or revoke access in a nondestructive linear fashion. These records can only be added if they’re signed by the private key used to initialise the application. Potentially using the auth table, you could also grant permission for other keys to write to the auth table if we think it’s desirable.

<page 4>

Data[n] collections are where the bulk of an application’s data is stored and their structure and ruleset should be defined in way which can be described by a simple stack based OP\_CODE script so it can be safely peer validated during consensus. As well as field definition and data validation primitives like type, length, range etc. there should be ability to create conditions based on other blockchain states e.g.

<pseudocode>

“IF pubkey IN my\_app.auth AND my\_app.auth.collection=`my\_data`”

</pseudocode>

Which would only allow a new record to be written to the my\_data collection if it’s signed by a pubkey that has valid permission to write to that datachain.

Or

<pseudocode>

“IF pubkey IN other\_app.users AND other\_app.users.verification=true”

</pseudocode>

Which would only let a user write to the datachain if their pubkey was verified in another application which could be a Digital ID / KYC service for example.

There could be the ability to apply conditions not only to accept or reject the entire datawrite, but to each field however this would probably require some kind of basic ternary statements to be allowed in the query language and breaks from traditional SQL models.

<pseudocode>

“IF pubkey IN other\_app.users ? data.permissions=`ADMIN` : data.permissions=`GUEST`”

</pseudocode>

This statement would default the allowed permission to GUEST unless the pubkey appears in the other apps list, then the permission is set to ADMIN.

## Valence Query Language

This brings us to VQL which is a simple non turing complete scripting language which provides the structured way of maintaining and querying the leveldb datastores for Vapps.

VQL needs to translate to OP\_CODE scripts which can be used to set the consensus rules for Vapps and their datachains. It would also be the way which full nodes and masternodes query their local data stores. In the first instance VQL would be driven by the CLI but we could also create a management GUI which operates similarly to database management software for creating and maintaining Vapps, auth and datachains.

If we want to go down this route we need to spend a lot more time researching and writing a specification which we think is going to be efficient, does what we require and make sense when interacting with leveldb data. We would probably also need to get someone involved who has written this type of translation layer before or at least do a lot of research about how other people have safely created their own abstracted languages.

Beyond this, a stretch goal (or v2) of the VQL implementation would be to provide zero knowledge proofs so there is no disclosure necessary. This way the value could be encrypted on the other application’s datachain and you are not concerned what the value is, just if the condition is met.

<pseudocode>

“IF pubkey IN other\_app.users AND other\_app.user.age > 18”

</pseudocode>

If we could use a ZK proof for this and the data was encrypted in the other application, we would not reveal the users date of birth, just a boolean of whether they were over 18 or not.

## Prepaid Datachain Writes

This should give the ability for Vapps to define a VALE address which can have it’s balance deducted to pay for data writes. This creates the ability to fuel applications so users can interact with them without needing to own VALE. Stakers & Valenoders should be able specify an address to send their rewards to autonomously fuel their app. This could allows Vapps to run at a discount or a zero sum if the owner is engaged in infrastructure support. I think this would be a major step towards enabling mass adoption. A big barrier to entry for mainstream users to engage with Ethereum Dapps is that first they need to buy ETH to pay for the GAS. It would also need some serious thought on how to prevent malicious users from spamming data and depleting the prepaid VALE.

## Valenodes Write on Behalf w. Write Receipt

If prepaid datachain writes are possible, then it also creates a layer of privacy by disconnecting users from both the data submitted and the payment linked to the data. Users could broadcast data anonymously via the EMP to a randomly selected Valenode which commits it to the chain and the data write is paid for out of the prepaid balance assigned to that application. The user would need to get some receipt to validate the data was written and the data could not be modified since it is signed by a the sender.

## Valenodes as VQL Endpoints

This would allow apps to query Valenodes for Valence data without requiring to download the Valence blockchain. This would allow for Valence data to be easily accessed and used as fulfilment conditions on other blockchains. For example our Kauri wallet could create an atomic payment on the NAV blockchain which requires a secret to be revealed to be released. The secret is written to a Vapp after some action is performed and the wallet has a list of Valenodes which it can query and ask whether the secret has been revealed.

Users could configure “trusted” Valenodes which they accept the response from as truth, or potentially configure a pseudo-consensus threshold where they will query x of n random valenodes before accepting the response as accurate. This would also need some serious thought around its implementation to prevent network spam.

This function is why I’ve split the Valenode registry out from any other datachains, so it would be possible for thin clients and other applications to maintain a list of available Valenodes to query for Valence data without requiring to store the whole chain which provides redundancy and autonomy.

## Valenodes as Cross Chain Oracles

Valenode VQL endpoints are a way of making data available outside of the Valence blockchain. The other side of this is how to make data available in Valence from other blockchains. The idea here is that Valenodes could run nodes of other blockchain e.g. NAV or BTC. Vapps could be setup with conditions which relate to external events eg. You can only submit data to my application if you’ve paid the registration fee in NAV to my address. When a data submission is received which relies on such a condition the Valenode would query the external blockchain and write a record to the NAV oracle Vapp if the payment has been received. This record would only be written to the NAV oracle if it reaches a consensus amongst the Valenodes that the external event is accurate. This means we’re only ever recording relevant data to the valence chain when it affects the Valence consensus.

## Vapp Service Micropayments

The above Cross Chain Oracle feature has the potential to enable an economy of microservices where Vapp curators are paid to provide services. For example if an entity wants to provide document verification for KYC/AML as a part of a Valence Digital ID service, they could charge for the service by requiring an additional fee in any supported crypto to be paid for by the requester before the data will be written.

## Protocol Objectives / Summary

* Users can broadcast messages anonymously across the network.
* Vapp data schemas are modular and malleable with user defined conditional consensus.
* Vapp data is isolated from the UTXO chain but secured by the fee tx associated with it.
* Users can write to Vapps without needing to purchase VALE.
* Users can broadcast data to be written to Vapps anonymously and the data write isn’t linked to their VALE address if the application is prepaid.
* Valence data can be queried and used to fulfil conditions on external blockchains.
* Valence write conditions can be based on external blockchains.
* Vapp service providers can charge fees for providing premium services.