Verus Data eXchange Format (VDXF)

The Verus Data Exchange Format provides a fully interoperable system for defining data types that may consist of structured or or unstructured data and associated content or keys that may be used to retrieve such data from centralized or decentralized storage for use in and across centralized or decentralized applications.

The Verus Data eXchange Format (VDXF) object is a structured data representation of the Verus Data eXchange Format, designed to facilitate the exchange of information across different systems and programming languages. It encapsulates data in a serialized byte format, making it interoperable and easy to transmit or store. This document outlines the structure and functionality of a VDXF object, focusing on its serialized form rather than the methods used to manipulate it in any specific programming language.

Overview

The Verus Data Exchange Format enables application developers to define globally unique data types and publish references to the same, which may refer to structured or unstructured data that can be located unambiguously via an URL, which implicitly provides both location and decoding information, enabling applications to use such data, in whole or in part, if they know how, or even ignore parts of the data, while remaining compatible with those parts they understand. VDXF typee keys are globally unique identifiers, which are defined as human readable names along with a specification of how to define and convert unlimited length, human readable type names into collison-free 20 byte IDs, which can be used as type keys associated with content or location values in various forms of data records. These data records, which may have application specific structures or no structure at all, besides length form the basis of an interoperable data exchange format across decentralized applications.

Definition of VDXF types

VDXF is not a strongly opinionated or highly specified type description specification, and, instead, focuses on a model for recognizing an unlimited number of user defined data types, using a standard human readable format for definition and encoding of the type specifier, which is hashed, using the VDXF specification and standard methodology, to produce collision-free, 20 byte keys, which can be associated with retrieveable content hashes and location qualifiers that enable applications to locate, recognize types of, parse, and decode any form of application or system specific data. VDXF specifies some basic type formats, as necessary to enable initial applications, but leaves further specifications of applicaiton specific data formats, of which there may be an unlimited number, as an open-ended option for those needing new data type

definitions for efficient application development. It is recommended that new fundamental data types not be defined unless necessary, but adherence to such recommendation is not enforced at the consensus protocol layer.

Namespace for Type Definitions - VerusID

Namespaces for type definitions are equivalent to VerusIDs, a protocol first implemented on the Verus Blockchain, and also one that can support IDs registered on any blockchain or uniquely named system that becomes recognized via a consensus-based bridge on the Verus network. Currently, to be recognized as a unique namespace, the easiest way is to base it on a VerusID, registered on the Verus blockchain network.

Generally, one may think of two types of VerusIDs, those defined on the Verus network or on independent PBaaS (Public Blockchains as a Service) blockchains spawned originally from and registered on the Verus blockchain network, or VerusIDs, which may also exist on fully external systems that may have been created without any registration on the Verus network initially. In order for an externally created VerusID to be recognizable on the Verus blockchain network or by applications using the VDXF that are compatible with the Verus blockchain network that external system must provide a recognized bridge to the Verus blockchain.

First, it is important to understand the requirements of registered VerusID identity names, which will also inform how externally generated VerusIDs are recognized as well. For the purposes of the VDXF, we do not require compatibility of the internal structure of IDs across different systems, and only define compatibility requirements of the naming systems and how those names translate into recognisable IDs on the Verus network.

Implementation

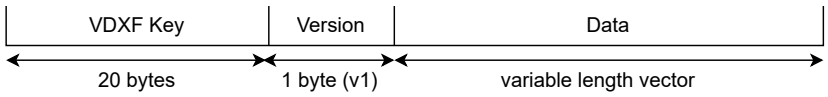
A VDXF object is fundamentally a serialized byte array that can be written to or interpreted by any system, given the appropriate libraries for handling its format. The serialization process transforms structured data into a byte stream, while deserialization reverses this process, reconstructing the original structured data from the byte stream.

Key Components

The VDXF object comprises several key components encoded into bytes:

- **VDXF Key**: A unique, 20 byte identifier for the VDXF object, that can be represented as a human readable string. This key is essential for identifying the type of data the VDXF object represents.
- **Version**: Indicates the version of the VDXF object.
- **Data**: The actual content stored within the VDXF object. This can be any form of data, structured or unstructured, that needs to be serialized.

Serialization Process



The serialization process involves converting the structured data within the VDXF object into a byte stream. This includes encoding the VDXF key, version, and actual data into bytes.

Components Encoding

- **VDXF Key**: 20 byte fixed length slice encoded using Base58Check
- **Version**: Serialized using variable integer encoding to optimize space. The version ensures that the serialized data can be correctly interpreted by systems aware of different versioning.
- **Data**: The actual data is serialized into a byte buffer. The format and encoding of this data can vary widely depending on the type of data being serialized and the intended use case.

Deserialization Process

Deserialization is the reverse of serialization, where the byte stream is converted back into structured data according to the VDXF format specification. This process involves reading the byte stream, extracting and decoding the VDXF key, version, and data components, and reconstructing the original structured data.

> **VDXF in Action**
>
> Learn how the CHIPS project (decentralized poker) uses VerusID and VDXF