<repository>

LIPs

overview

repository: https://github.com/lukso-network/LIPs/
userName: lukso-network
repository: LIPs
branch: main
date: 2023-11-27T10:52:20+01:00 (1701078740)

<file>

path: /.prettierrc

url: https://github.com/lukso-network/LIPs/blob/main/.prettierrc

```
{
  "tabWidth": 2,
  "useTabs": false
}
```

</file>

<file>

path: /LIPs/lip-1.md

url: https://github.com/lukso-network/LIPs/blob/main/LIPs/lip-1.md

---

lip: 1
title: LIP Purpose and Guidelines
status: Active
type: Meta
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network), Martin Becze mb@ethereum.org (mailto:mb@ethereum.org), Hudson Jameson hudson@ethereum.org (mailto:hudson@ethereum.org), and others
created: 2015-10-27
updated: 2015-12-07, 2016-02-01, 2018-03-21, 2018-05-29, 2018-10-17, 2019-04-09

What is an LIP?

LIP stands for LUKSO Improvement Proposal. An LIP is a design document providing information to the LUKSO and Ethereum community, or describing a new feature for LUKSO or its processes or environment. The LIP should provide a concise technical specification of the feature and a rationale for the feature. The LIP author is responsible for building consensus within the community and documenting dissenting opinions.

We are also referencing EIPs (Ethereum Improvement Proposals) and ERCs (Ethereum Request for Comment), which can be found here at (https://github.com/ethereum/EIPs/tree/master/EIPS) [https://github.com/ethereum/EIPs/tree/master/EIPS]

LIP Rationale

We intend LIPs to be the primary mechanisms for proposing new features, for collecting community technical input on an issue, and for documenting the design decisions that have gone into LUKSO. Because the LIPs are maintained as text files in a versioned repository, their revision history is the historical record of the feature proposal.

For LUKSO implementers, LIPs are a convenient way to track the progress of their implementation. Ideally each implementation maintainer would list the LIPs that they have implemented. This will give end users a convenient way to know the current status of a given implementation or library.

LIP Types

There are three types of LIP:

- A Standard Track LIP describes any change that affects most or all LUKSO implementations, such as a change to the the network protocol, a change in block or transaction validity rules, proposed application standards/conventions, or any change or addition that affects the interoperability of applications using LUKSO. Furthermore Standard LIPs can be broken down into the following categories. Standards Track LIPs consist of three parts, a design document, implementation, and finally if warranted an update to the formal specification (https://github.com/ethereum/yellowpaper).
    - Core - improvements requiring a consensus fork, as well as changes that are not necessarily consensus critical but may be relevant.
    - Interface - includes improvements around client API/RPC (https://github.com/ethereum/wiki/wiki/JSON-RPC) specifications and standards, and also certain language-level standards like method names (EIP6 (https://github.com/ethereum/EIPs/blob/master/LIPS/eip-6.md)) and contract ABIs (https://github.com/ethereum/wiki/wiki/LUKSO-Contract-ABI). The label “interface” aligns with the interfaces repo (https://github.com/ethereum/interfaces) and discussion should primarily occur in that repository before an LIP is submitted to the LIPs repository.
    - LSP - LUKSO standard proposal. Application-level standards and conventions, including smart contract standards such as token standards (ERC20 (https://github.com/ethereum/EIPs/issues/20)), name registries (ERC26 (https://github.com/ethereum/EIPs/issues/26), ERC137 (https://github.com/ethereum/EIPs/issues/137)), URI schemes (ERC67 (https://github.com/ethereum/EIPs/issues/67)), library/package formats (EIP82 (https://github.com/ethereum/EIPs/issues/82)), and wallet formats (EIP75 (https://github.com/ethereum/EIPs/issues/75), EIP85 (https://github.com/ethereum/EIPs/issues/85)).

- An Informational LIP describes an LUKSO design issue, or provides general guidelines or information to the LUKSO community, but does not propose a new feature. Informational LIPs do not necessarily represent LUKSO community consensus or a recommendation, so users and implementers are free to ignore Informational LIPs or follow their advice.
- A Meta LIP describes a process surrounding LUKSO or proposes a change to (or an event in) a process. Process LIPs are like Standards Track LIPs but apply to areas other than the LUKSO protocol itself. They may propose an implementation, but not to LUKSO's codebase; they often

require community consensus; unlike Informational LIPs, they are more than recommendations, and users are typically not free to ignore them. Examples include procedures, guidelines, changes to the decision-making process, and changes to the tools or environment used in LUKSO development. Any meta-LIP is also considered a Process LIP.

It is highly recommended that a single LIP contain a single key proposal or new idea. The more focused the LIP, the more successful it tends to be. A change to one client doesn't require an LIP; a change that affects multiple clients, or defines a standard for multiple apps to use, does.

An LIP must meet certain minimum criteria. It must be a clear and complete description of the proposed enhancement. The enhancement must represent a net improvement. The proposed implementation, if applicable, must be solid and must not complicate the protocol unduly.

LIP Work Flow

Parties involved in the process are you, the champion or *LIP author*, the *LIP editors*.

:warning: Before you begin, vet your idea, this will save you time. Ask the LUKSO and Ethereum community first if an idea is original to avoid wasting time on something that will be be rejected based on prior research (searching the Internet does not always do the trick). It also helps to make sure the idea is applicable to the entire community and not just the author. Just because an idea sounds good to the author does not mean it will work for most people in most areas where LUKSO is used. Examples of appropriate public forums to gauge interest around your LIP include the LUKSO subreddit (https://www.reddit.com/r/lukso/), the Issues section of this repository (https://github.com/lukso-network/LIPs/issues), and one of the LUKSO Gitter chat rooms (https://gitter.im/lukso/). In particular, the Issues section of this repository (https://github.com/lukso-network/LIPs/issues) is an excellent place to discuss your proposal with the community and start creating more formalized language around your LIP.

Your role as the champion is to write the LIP using the style and format described below, shepherd the discussions in the appropriate forums, and build community consensus around the idea. Following is the process that a successful LIP will move along:

```
[ WIP ] -> [ DRAFT ] -> [ LAST CALL ] -> [ ACCEPTED ] -> [ FINAL ]
```

Each status change is requested by the LIP author and reviewed by the LIP editors. Use a pull request to update the status. Please include a link to where people should continue discussing your LIP. The LIP editors will process these requests as per the conditions below.

- Active -- Some Informational and Process LIPs may also have a status of “Active” if they are never meant to be completed. E.g. LIP 1 (this LIP).
- Work in progress (WIP) -- Once the champion has asked the LUKSO community whether an idea has any chance of support, they will write a draft LIP as a pull request (https://github.com/lukso-network/LIPs/pulls). Consider including an implementation if this will aid people in studying the LIP.
    - :arrow_right: Draft -- If agreeable, LIP editor will assign the LIP a number (generally the issue or PR number related to the LIP) and merge your pull request. The LIP editor will not unreasonably deny an LIP.
    - :x: Draft -- Reasons for denying draft status include being too unfocused, too broad, duplication of effort, being technically unsound, not providing proper motivation or addressing backwards compatibility, or not in keeping with the LUKSO philosophy (https://github.com/ethereum/wiki/wiki/White-Paper#philosophy).
- Draft -- Once the first draft has been merged, you may submit follow-up pull requests with further changes to your draft until such point as you believe the LIP to be mature and ready to proceed to

the next status. An LIP in draft status must be implemented to be considered for promotion to the next status (ignore this requirement for core LIPs).

    - :arrow_right: Last Call -- If agreeable, the LIP editor will assign Last Call status and set a review end date (review-period-end), normally 14 days later.
    - :x: Last Call -- A request for Last Call status will be denied if material changes are still expected to be made to the draft. We hope that LIPs only enter Last Call once, so as to avoid unnecessary noise on the RSS feed.

- Last Call -- This LIP will listed prominently on the https://eips.ethereum.org/ website (subscribe via RSS at last-call.xml (/last-call.xml)).

    - :x: -- A Last Call which results in material changes or substantial unaddressed technical complaints will cause the LIP to revert to Draft.
    - :arrow_right: Accepted (Core LIPs only) -- A successful Last Call without material changes or unaddressed technical complaints will become Accepted.
    - :arrow_right: Final (Not core LIPs) -- A successful Last Call without material changes or unaddressed technical complaints will become Final.

- Accepted (Core LIPs only) -- This LIP is in the hands of the LUKSO client developers. Their process for deciding whether to encode it into their clients as part of a hard fork is not part of the LIP process.

    - :arrow_right: Final -- Standards Track Core LIPs must be implemented in at least three viable LUKSO clients before it can be considered Final. When the implementation is complete and adopted by the community, the status will be changed to “Final”.

- Final -- This LIP represents the current state-of-the-art. A Final LIP should only be updated to correct errata.

Other exceptional statuses include:

- Deferred -- This is for core LIPs that have been put off for a future hard fork.
- Rejected -- An LIP that is fundamentally broken or a Core LIP that was rejected by the Core Devs and will not be implemented.
- Active -- This is similar to Final, but denotes an LIP which may be updated without changing its LIP number.
- Superseded -- An LIP which was previously final but is no longer considered state-of-the-art. Another LIP will be in Final status and reference the Superseded LIP.

What belongs in a successful LIP?

Each LIP should have the following parts:

- Preamble - RFC 822 style headers containing metadata about the LIP, including the LIP number, a short descriptive title (limited to a maximum of 44 characters), and the author details. See below (https://github.com/ethereum/LIPs/blob/master/LIPS/eip-1.md#eip-header-preamble) for details.
- Simple Summary - “If you can’t explain it simply, you don’t understand it well enough.” Provide a simplified and layman-accessible explanation of the LIP.
- Abstract - a short (~200 word) description of the technical issue being addressed.
- Motivation (*optional) - The motivation is critical for LIPs that want to change the LUKSO protocol. It should clearly explain why the existing protocol specification is inadequate to address the problem that the LIP solves. LIP submissions without sufficient motivation may be rejected outright.
- Specification - The technical specification should describe the syntax and semantics of any new

feature. The specification should be detailed enough to allow competing, interoperable implementations for any of the current LUKSO platforms (cpp-ethereum, go-ethereum, parity, ethereumJ, ethereumjs-lib, and others (https://github.com/ethereum/wiki/wiki/Clients).

- Rationale - The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.
- Backwards Compatibility - All LIPs that introduce backwards incompatibilities must include a section describing these incompatibilities and their severity. The LIP must explain how the author proposes to deal with these incompatibilities. LIP submissions without a sufficient backwards compatibility treatise may be rejected outright.
- Test Cases - Test cases for an implementation are mandatory for LIPs that are affecting consensus changes. Other LIPs can choose to include links to test cases if applicable.
- Implementations - The implementations must be completed before any LIP is given status “Final”, but it need not be completed before the LIP is merged as draft. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the principle of “rough consensus and running code” is still useful when it comes to resolving many discussions of API details.
- Copyright Waiver - All LIPs must be in the public domain. See the bottom of this LIP for an example copyright waiver.

LIP Formats and Templates

LIPs should be written in markdown (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) format.
Image files should be included in a subdirectory of the assets folder for that LIP as follow: assets/eip-X (for eip X). When linking to an image in the LIP, use relative links such as ../assets/eip-X/image.png.

LIP Header Preamble

Each LIP must begin with an RFC 822 style header preamble, preceded and followed by three hyphens (---). The headers must appear in the following order. Headers marked with "*" are optional and are described below. All other headers are required.

lip: <LIP number> (this is determined by the LIP editor)

title: <LIP title>

author: <a list of the author's or authors' name(s) and/or username(s), or name(s) and email(s). Details are below.>

* discussions-to: <a url pointing to the official discussion thread>

status: <Draft | Last Call | Accepted | Final | Active | Deferred | Rejected | Superseded>

* review-period-end: <date review period ends>

type: <Standards Track (Core, Interface, LSP) | Informational | Meta>

* category: <Core | Interface | LSP>

created: <date created on>

* updated: <comma separated list of dates>

* requires: <LIP number(s)>

* replaces: <LIP number(s)>

* superseded-by: <LIP number(s)>

* resolution: <a url pointing to the resolution of this LIP>

Headers that permit lists must separate elements with commas.

Headers requiring dates will always do so in the format of ISO 8601 (yyyy-mm-dd).

author header

The author header optionally lists the names, email addresses or usernames of the authors/owners of the LIP. Those who prefer anonymity may use a username only, or a first name and a username. The format of the author header value must be:

> Random J. User <address@dom.ain>

or

> Random J. User (@username)

if the email address or GitHub username is included, and

> Random J. User

if the email address is not given.

resolution header

The resolution header is required for Standards Track LIPs only. It contains a URL that should point to an email message or other web resource where the pronouncement about the LIP is made.

discussions-to header

While an LIP is a draft, a discussions-to header will indicate the mailing list or URL where the LIP is being discussed. As mentioned above, examples for places to discuss your LIP include LUKSO topics on Gitter (https://gitter.im/ethereum/topics), an issue in this repo or in a fork of this repo, LUKSO Magicians (https://ethereum-magicians.org/) (this is suitable for LIPs that may be contentious or have a strong governance aspect), and Reddit r/ethereum (https://www.reddit.com/r/ethereum/).

No discussions-to header is necessary if the LIP is being discussed privately with the author.

As a single exception, discussions-to cannot point to GitHub pull requests.

type header

The type header specifies the type of LIP: Standards Track, Meta, or Informational. If the track is Standards please include the subcategory (core, networking, interface, or ERC).

category header

The category header specifies the LIP's category. This is required for standards-track LIPs only.

created header

The created header records the date that the LIP was assigned a number. Both headers should be in yyyy-mm-dd format, e.g. 2001-08-14.

updated header

The updated header records the date(s) when the LIP was updated with "substantional" changes. This header is only valid for LIPs of Draft and Active status.

requires header

LIPs may have a requires header, indicating the LIP numbers that this LIP depends on.

superseded-by and replaces headers

LIPs may also have a superseded-by header indicating that an LIP has been rendered obsolete by a later document; the value is the number of the LIP that replaces the current document. The newer LIP must have a replaces header containing the number of the LIP that it rendered obsolete.

Auxiliary Files

LIPs may include auxiliary files such as diagrams. Such files must be named LIP-XXXX-Y.ext, where “XXXX” is the LIP number, “Y” is a serial number (starting at 1), and “ext” is replaced by the actual file extension (e.g. “png”).

Transferring LIP Ownership

It occasionally becomes necessary to transfer ownership of LIPs to a new champion. In general, we'd like to retain the original author as a co-author of the transferred LIP, but that's really up to the original author. A good reason to transfer ownership is because the original author no longer has the time or interest in updating it or following through with the LIP process, or has fallen off the face of the 'net (i.e. is unreachable or isn't responding to email). A bad reason to transfer ownership is because you don't agree with the direction of the LIP. We try to build consensus around an LIP, but if that's not possible, you can always submit a competing LIP.

If you are interested in assuming ownership of an LIP, send a message asking to take over, addressed to both the original author and the LIP editor. If the original author doesn't respond to email in a timely manner, the LIP editor will make a unilateral decision (it's not like such decisions can't be reversed :)).

LIP Editors

The current LIP editors are

* Fabian Vogelsteller (@frozeman)

* Leonard Schellenberg Detrio (@Leondroids)

LIP Editor Responsibilities

For each new LIP that comes in, an editor does the following:

- Read the LIP to check if it is ready: sound and complete. The ideas must make technical sense, even if they don't seem likely to get to final status.
- The title should accurately describe the content.
- Check the LIP for language (spelling, grammar, sentence structure, etc.), markup (Github flavored Markdown), code style

If the LIP isn't ready, the editor will send it back to the author for revision, with specific instructions.

Once the LIP is ready for the repository, the LIP editor will:

- Assign an LIP number (generally the PR number or, if preferred by the author, the Issue # if there was discussion in the Issues section of this repository about this LIP)
- Merge the corresponding pull request
- Send a message back to the LIP author with the next step.

Many LIPs are written and maintained by developers with write access to the LUKSO codebase. The LIP editors monitor LIP changes, and correct any structure, grammar, spelling, or markup mistakes we see.

The editors don't pass judgment on LIPs. We merely do the administrative & editorial part.

History

This document was derived heavily from [Ethereum's EIP-1] and Bitcoin's BIP-0001 (https://github.com/bitcoin/bips/) written by Amir Taaki which in turn was derived from Python's PEP-0001 (https://www.python.org/dev/peps/). In many places text was simply copied and modified. Although the PEP-0001 text was written by Barry Warsaw, Jeremy Hylton, and David Goodger, they are not responsible for its use in the LUKSO Improvement Process, and should not be bothered with technical questions specific to LUKSO or the LIP. Please direct all comments to the LIP editors.

See the revision history for further details (https://github.com/ethereum/LIPs/commits/master/LIPS/lip-1.md), which is also available by clicking on the History button in the top right of the LIP.

Bibliography

Copyright

</file>

<file>

path: /LSPs/LSP-0-ERC725Account.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-0-ERC725Account.md

---

lip: 0
title: ERC725 Account
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2021-09-21
requires: ERC165, ERC725X, ERC725Y, ERC1271, LSP1, LSP2, LSP14, LSP17, LSP20

Simple Summary

The ERC725Account standard outlines a blockchain smart contract account that integrates a suite of standards, including ERC725 (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md), to represent and manage digital identities.

Abstract

This standard defines a blockchain-based account system that humans, machines, organizations or other smart contracts can use.

Key functionalities of this account system include:

- Dynamic Information Attachment: leverages ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) to add generic information to the account post-deployment.
- Generic Execution: utilises ERC725X (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725x) to grant the account the capability to interact with other contracts, manage token transfers, and initiate contract creation with different operations.
- Signature Verification: uses ERC1271 (https://eips.ethereum.org/EIPS/eip-1271) to verify whether the signature on behalf of the account is valid.
- Notifications and Reactivity: with LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md), enable the account to be aware of assets or any other information and react accordingly (e.g., denying specific tokens).
- Secured Ownership Management: enforced by LSP14-Ownable2Step (./LSP-14-Ownable2Step.md), ensures transfer of ownership is secured through a 2-step process.
- Future-Proof Functionalitys: through LSP17-ContractExtension (./LSP-17-ContractExtension.md), allows the account to be extended and support new standardized functions and interfaceIds over time.
- Streamlined Account Interaction: via LSP20-CallVerification (./LSP-20-CallVerification.md), to make interactions with the account easier, enables direct function calls on the account from addresses other than the owner, with verification of the call occurring on the owner if it is a contract.

Motivation

Limitations of Externally Owned Accounts

Using Externally Owned Accounts (EOAs) as main accounts poses challenges in different aspects. The significant limitations include:

- No Information Attachment: EOAs can not store data readable by off-chain clients or other smart contracts.
- Weak Security:the account, represented by a public key is controlled by its corresponding private key. If the private key leaks or is lost, all associated assets, reputation, and control over other smart contracts are also lost.
- Exclusive Control: sharing the private key with other entities is not an option, as possession equals full control. This exclusivity prevents delegating certain access rights, social recovery, and other shared security measures.
- Information Tracking Limitation: EOAs do not allow for the internal tracking of asset transfers, followers, or other information. As a result, account holders must depend on external blockchain explorers to monitor their transaction history, which introduces inconvenience and slows down information access.

Lack of Standardization of Accounts

The absence of a standardized specification for blockchain accounts presents several challenges:

- Complex Development Landscape: The lack of a standard account template in blockchain leads to a diverse and complex array of custom smart contract accounts created by individual developers and organizations, which make it harder for other developers to build protocols on top of it.
- Adoption and Compatibility Hurdles: This diversity necessitates extra compatibility work, slowing the adoption and practical use of smart contract accounts across the blockchain ecosystem.

Advantages of Smart Contract Accounts

Adopting smart contract accounts over traditional EOAs streamlines blockchain infrastructure use. Smart contracts enable advanced features and enhanced control, such as:

- Dynamic Information Storage: Unlike EOAs, smart contract accounts can record static information, making them self-explanatory entities that can assist in identity verification for off-chain entities, and enable the decentralized tracking of any information stored such as assets, followers, etc. The information stored can also be standardized, for example LSP3-Profile-Metadata (./LSP-3-Profile-Metadata.md) for storing profile information, LSP5-ReceivedAssets (./LSP-5-ReceivedAssets.md) for storing the addresses of received assets, and LSP10-ReceivedVaults (./LSP-10-ReceivedVaults.md) for storing addresses of received vaults, etc ..
- Upgradeable Security: Owners can use a simple EOA for owning the account having basic level of security, or opt for an owner having a multisignature setup, or permissions access control contracts providing advanced layer of security in contorlling the account.
- Inclusive Control: Users can decide to own the account with a permission based access control contract (e.g: the LSP6-KeyManager (./LSP-6-KeyManager.md)) which allows owners to assign specific rights to different actors, like granting a party the right to update profile information without providing full transactional authority.
- Extended Execution Capabilities: Smart contract accounts can employ a broader set of operations compared to EOAs, including *delegatecall*, *staticcall*, and *create2*. This extension of capabilities allows for a wide range of actions and interactions that were previously not possible with standard

EOAs.

- Notifications and Automated Interaction Handling: The ability to be notified about different actions, and automate the responses to incoming information empowers smart contract accounts with a higher degree of functionality. When assets are received or certain data is detected, the account can autonomously initiate actions or updates, tailored to the event, enhancing the user experience and potential for automation within the blockchain ecosystem using (LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md)).

Specification

InterfaceId Calculation

LSP0-ERC725Account interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0x24871b3d.

This bytes4 interface id is calculated as the XOR of the following:

- The selector of the batchCalls(bytes[]) function signature.
- The ERC165 interface ID of the ERC725X (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725x) standard.
- The ERC165 interface ID of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) standard.
- The ERC165 interface ID of the ERC1271 (https://eips.ethereum.org/EIPS/eip-1271) standard (= isValidSignature(bytes32,bytes) function selector).
- The ERC165 interface ID of the LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) standard.
- The ERC165 interface ID of the LSP14-Ownable2Step (./LSP-14-Ownable2Step.md) standard.
- The ERC165 interface ID of the LSP17-ContractExtension (./LSP-17-ContractExtension.md) standard.
- The ERC165 interface ID of the LSP20-CallVerification (./LSP-20-CallVerification.md) standard.

Exclusive Called Functions Behavior

Owner Specific Functions

For smart contracts adhering to the LSP0-ERC725Account standard, certain functions are designated to be owner-exclusive and include:

- setData(bytes32,bytes)
- setDataBatch(bytes32[],bytes[])
- execute(uint256,address,uint256,bytes)
- executeBatch(uint256[],address[],uint256[],bytes[])
- transferOwnership(address)
- renounceOwnership()

These functions are designed to be initiated by the account owner to perform crucial operations like updating data, executing transactions, and transferring or renouncing ownership.

However, in alignment with the LSP20-CallVerification (./LSP-20-CallVerification.md) standard, these owner-exclusive functions can also be invoked by addresses allowed by the logic of the owner. This invocation triggers an internal verification process to the owner.

For example, if the caller address is not the owner, the function will call the lsp20VerifyCall(..) on the owner address function to validate the transaction before execution passing the following arguments:

- requestor: The address calling the function to be executed.
- target: The address of the account.
- caller: The address calling the function to be executed.
- value: The value sent by the caller to the function called on the account.
- receivedCalldata: The calldata sent by the caller to the account.

The expected outcome is for the bytes4 returned value to match the first bytes3 of the function selector of lsp20VerifyCall for the operation to proceed. Specifically, if the fourth byte of the call is 0x01, the function MUST call lsp20VerifyCallResult after execution of the function passing the following arguments:

- callHash: The keccak256 hash of the parameters of lsp20VerifyCall(address,address,address,uint256,bytes) parameters packed-encoded (concatened).

- callResult: the result of the function being called on the account.

    - if the function being called returns some data, the callResult MUST be the value returned by the function being called as abi-encoded bytes.
    - if the function being called does not return any data, the callResult MUST be an empty bytes.

This post verification allows for additional verification steps to be conducted within the transaction process.

Pending Owner Specific Functions

The function designated to be called by the pending owner is:

- acceptOwnership()

The same logic applies to acceptOwnership function, expcet that it should be either called by the pending owner or an address allowed by the pending owner. In case the caller is not the pending owner, the verification should happen according to the LSP20-CallVerification (./LSP-20-CallVerification.md) on the address of the pending owner.

> Read more about the Rational behind using LSP20-CallVerifiaction in the LSP0-ERC725Account standard.

Value Receive Notification and Reaction Specification

Owner Specific Payable Functions

In smart contracts compliant with the [LSP0-ERC725Account] standard, specific owner-exclusive functions are capable of handling transactions with attached value. These include:

- setData(bytes32,bytes)
- setDataBatch(bytes32[],bytes[])
- execute(uint256,address,uint256,bytes)
- executeBatch(uint256[],address[],uint256[],bytes[])

When these functions receive value (LYX or any other native token), the smart contract must emit a [UniversalReceiver] event to notify the contract of the value received. The event parameters are as follows:

- sender: The address that initiated the function call.
- value: The amount of value (in wei) sent with the call.
- typeId: A unique identifier indicating a value reception, keccak256("LSP0ValueReceived") > 0x9c4705229491d365fb5434052e12a386d6771d976bea61070a8c694e8affea3d.
- receivedData: The first four bytes of the calldata used to call the function, representing its signature.
- returnedData: Empty bytes, as these functions do not return data within this event context.

User-Callable Payable Functions

For functions that are payable and not restricted to the owner, such as:

- receive()
- fallback()
- universalReceiver(bytes32,bytes)

The contract should trigger the [universalReceiver] function logic before the called function execute if value is sent with the call. The arguments for the internal universalReceiver logic invocation are:

- typeId: The bytes32 hash obtained by keccak256 hashing of "LSP0ValueReceived".
- receivedData: Depending on the function called:
  - For receive(), it should be an empty bytes array.
  - For fallback(), it should include the entire calldata.
  - For universalReceiver(), it should contain all the calldata received.

After processing the internal logic, the UniversalReceiver event is emitted to signal the transaction and interaction with the smart contract account.

> When the value sent to the fallback function is forwarded to an extension, the universalReceiver function should not be invoked.
> Read more about the Rational behind using UniversalReceiver in the payable function of the standard.

Extending Functionalities and InterfaceIds Specification

Extending with Functions

To add a new function to the contract:

- Obtain the function's unique selector.
- Construct an LSP17Extension data key using the function selector.

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "(address, bytes1)",
  "valueContent": "(Address, bool)"
}
```

> <bytes4> is the functionSelector of the function to extend. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

- Develop a contract that implements the desired function and deploy it to the network.
- Store the deployed contract's address in the account contract under the previously constructed LSP17 data key.

For calls to the account that require forwarding value to the extension, append an additional 0x01 byte to the call. For calls without value transfer, record only the address.

Upon invocation, the extension contract is designed to receive the full call data. An extra 52 bytes are appended to this data, passing compacted the original caller's address (20 bytes) and the sent value (32 bytes). The extension contract may retrieve the caller address and sent value by extracting these details from the calldata's tail end, enabling it to discern the transaction's origin and associated value.

Extending InterfaceIds

To support a new InterfaceId:

- Obtain the ERC165 standardized function selector for supportsInterface(bytes4).
- Construct an LSP17Extension data key using the function selector.

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "(address, bytes1)",
  "valueContent": "(Address, bool)"
}
```

> <bytes4> is the functionSelector of supportsInterface(bytes4) function. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

- Develop and deploy a contract that includes the supportsInterface(bytes4) function, which is designed to support additional interfaceIds.
- Store the deployed contract's address in the account contract under the previously constructed LSP17 data key.

Methods

Smart contracts implementing the LSP0 standard MUST implement all of the functions listed below:

receive

```
receive() external payable;
```

The receive function allows for receiving native tokens.

Requirememnts:

- MUST adhere to the logic specified in Value Receive Notification and Reaction section.

fallback

```
fallback() external payable;
```

This function is part of the LSP17-ContractExtension (./LSP-17-ContractExtension.md) specification, with additional requirements as follows:

- MUST be payable.
- MUST adhere to the logic specified in Value Receive Notification and Reaction section.
- MUST return if the data sent to the contract is less than 4 bytes in length.
- MUST check for address of the extension and a boolean under the following ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) Data Key.
    - If there is no extension stored under the data key, the call should revert, except when the function selector is 0x00000000, if no extension is stored for this function selector, the call will pass.
    - If the data stored is strictly 20 bytes, call the extension and behave according to LSP17-ContractExtension (./LSP-17-ContractExtension.md) specification by appending the caller address and the value sent to the call.
    - If the data stored is 21 bytes, and the 21th byte is strictly 0x01 forward the value received to the extension. In any other case, the value should stay in the account.

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "(address, bytes1)",
  "valueContent": "(Address, bool)"
}
```

> <bytes4> is the functionSelector called on the account contract. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

> Read more about the [Rational behind using UniversalReceiver] in the payable function of the standard and [the use of 0x00000000 selector for Graffiti].

supportsInterface

```
function supportsInterface(bytes4 interfaceId) external view returns (bool);
```

This function is part of the ERC165 (https://eips.ethereum.org/EIPS/eip-165) specification, with additional requirements as follows:

- If the interfaceId being queried is not supported by the contract or inherited contracts, the data key attached below MUST be retrieved from the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) storage.
    - If there is an address stored under the data key, forward the supportsInterface(bytes4) call to the address and return the value.
    - If there is no address, execution ends normally.

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "(address, bytes1)",
  "valueContent": "(Address, bool)"
}
```

> <bytes4> is the functionSelector of supportsInterface(bytes4) function. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

owner

```
function owner() external view returns (address);
```

This function is part of the LSP14-Ownable2Step (./LSP-14-Ownable2Step.md) specification.

pendingOwner

```
function pendingOwner() external view returns (address);
```

This function is part of the LSP14-Ownable2Step (./LSP-14-Ownable2Step.md) specification.

transferOwnership

```
function transferOwnership(address newPendingOwner) external;
```

This function is part of the LSP14-Ownable2Step (./LSP-14-Ownable2Step.md) specification, with additional requirements as follows:

- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST override the LSP14 Type ID triggered by using transferOwnership(..) to the one below:
    - keccak256('LSP0OwnershipTransferStarted') > 0xe17117c9d2665d1dbeb479ed8058bbebde3c50ac50e2e65619f60006caac6926

acceptOwnership

```
function acceptOwnership() external;
```

This function is part of the LSP14-Ownable2Step (./LSP-14-Ownable2Step.md) specification, with additional requirements as follows:

- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST override the LSP14 Type IDs triggered by using accceptOwnership(..) to the ones below:
    - keccak256('LSP0OwnershipTransferred_SenderNotification') > 0xa4e59c931d14f7c8a7a35027f92ee40b5f2886b9fdcdb78f30bc5ecce5a2f814
    - keccak256('LSP0OwnershipTransferred_RecipientNotification') > 0xceca317f109c43507871523e82dc2a3cc64dfa18f12da0b6db14f6e23f995538

renounceOwnership

```
function renounceOwnership() external;
```

This function is part of the LSP14-Ownable2Step (./LSP-14-Ownable2Step.md) specification with additional requirements as follows:

- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST override the LSP14 Type IDs triggered by using renounceOwnership(..) to the ones below:
    - keccak256('LSP0OwnershipTransferred_SenderNotification') > 0xa4e59c931d14f7c8a7a35027f92ee40b5f2886b9fdcdb78f30bc5ecce5a2f814

batchCalls

```
function batchCalls(bytes[] calldata functionCalls) external returns (bytes[] memory results)
```

Enables the execution of a batch of encoded function calls on the current contract in a single transaction, provided as an array of bytes.

MUST use the DELEGATECALL (https://solidity-by-example.org/delegatecall/) opcode to execute each call in the same context of the current contract.

Parameters:

- functionCalls: an array of encoded function calls to be executed on the current contract.

The data field can be:

- an array of ABI-encoded function calls such as an array of ABI-encoded execute, setData, transferOwnership or any LSP0 functions.
- an array of bytes which will resolve to the fallback function to be checked for an extension.

Requirements:

- MUST NOT be payable.

Returns: results , an array of bytes containing the return values of each executed function call.

execute

```
function execute(uint256 operationType, address target, uint256 value, bytes memory data) external payable returns (bytes memory);
```

This function is part of the ERC725X (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725x) specification, with additional requirements as follows:

- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST adhere to the logic specified in Value Receive Notification and Reaction section.

executeBatch

```
function executeBatch(uint256[] memory operationsType, address[] memory targets, uint256[] memory values, bytes[] memory datas) external payable returns (bytes[] memory);
```

This function is part of the ERC725X (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725x) specification, with additional requirements as follows:

- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST adhere to the logic specified in Value Receive Notification and Reaction section.

getData

```
function getData(bytes32 dataKey) external view returns (bytes memory);
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification.

getDataBatch

```
function getDataBatch(bytes32[] memory dataKeys) external view returns (bytes[] memory);
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification.

setData

```
function setData(bytes32 dataKey, bytes memory dataValue) external payable;
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification, with additional requirements as follows:

- MUST be payable.
- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST adhere to the logic specified in Value Receive Notification and Reaction section.
- MUST emit only the first 256 bytes of the dataValue parameter in the DataChanged (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#datachanged) event.

setDataBatch

```
function setDataBatch(bytes32[] memory dataKeys, bytes[] memory dataValues) external payable;
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification, with additional requirements as follows:

- MUST be payable.
- MUST adhere to the logic specified in Exclusive Called Functions Behavior section.
- MUST adhere to the logic specified in Value Receive Notification and Reaction section.

- MUST emit only the first 256 bytes of the dataValue parameter in the DataChanged (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#datachanged) event.

isValidSignature

```
function isValidSignature(bytes32 hash, bytes memory signature) external view returns (bytes4);
```

This function is part of the ERC1271 (https://eips.ethereum.org/EIPS/eip-1271) specification, with additional requirements as follows:

- When the owner is an EOA, the function MUST return the success value (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1271.md#specification) if the address recovered from the hash and the signature via ecrecover (https://docs.soliditylang.org/en/v0.8.17/solidity-by-example.html?highlight=ecrecover#recovering-the-message-signer-in-solidity) is the owner of the contract. Otherwise, MUST return the failure value (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1271.md#specification).
- When the owner is a contract, it will call the isValidsignature(bytes32,bytes) function on the owner contract, and return the success value if the function returns the success value. In any other case such as non-standard return value or revert, it will return the failure value indicating an invalid signature.

universalReceiver

```
function universalReceiver(bytes32 typeId, bytes memory receivedData) external payable returns (bytes memory);
```

This function is part of the LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) specification, with additional requirements as follows:

- MUST adhere to the logic specified in Value Receive Notification and Reaction section.
- If an address is stored under the data key attached below and this address is a contract:
    - forwards the call to the universalReceiverDelegate(address,uint256,bytes32,bytes) (./LSP-1-UniversalReceiver.md#universalreceiverdelegate) function on the contract at this address ONLY IF this contract supports the LSP1-UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification).
    - if the contract at this address does not support the LSP1-UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification), execution continues normally.
- If there is no address stored under this data key, execution continues normally.

```
{
  "name": "LSP1UniversalReceiverDelegate",
  "key": "0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
  "keyType": "Singleton",
  "valueType": "address",
  "valueContent": "Address"
}
```

- If an address is stored under the data key attached below and this address is a contract:
    - forwards the call to the universalReceiverDelegate(address,uint256,bytes32,bytes) (./LSP-

1-UniversalReceiver.md#universalreceiverdelegate) function on the contract at this address ONLY IF this contract supports the LSP1-UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification).
    - if the contract at this address does not support the LSP1-UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification), execution continues normally.

- If there is no address stored under this data key, execution continues normally.

```
{
  "name": "LSP1UniversalReceiverDelegate:<bytes32>",
  "key": "0x0cfc51aec37c55a4d0b10000<bytes32>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

> <bytes32> is the typeId passed to the universalReceiver(..) function. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

- MUST return the returned value of the universalReceiverDelegate(address,uint256,bytes32,bytes) function on both retrieved contract abi-encoded as bytes. If there is no addresses stored under the data keys above or the call was not forwarded to them, the return value is the two empty bytes abi-encoded as bytes.

- MUST emit a UniversalReceiver (./LSP-1-UniversalReceiver.md#events) event if the function was successful with the call context, parameters passed to it and the function's return value.

ERC725Y Data Keys

LSP1UniversalReceiverDelegate

```
{
  "name": "LSP1UniversalReceiverDelegate",
  "key": "0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
  "keyType": "Singleton",
  "valueType": "address",
  "valueContent": "Address"
}
```

If the account delegates its universal receiver functionality to another smart contract, this smart contract address MUST be stored under the data key attached above. This call to this contract is performed when the universalReceiver(bytes32,bytes) function of the account is called and can react on the whole call regardless of the typeId.

Check LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

Mapped LSP1UniversalReceiverDelegate

```
{
  "name": "LSP1UniversalReceiverDelegate:<bytes32>",
  "key": "0x0cfc51aec37c55a4d0b10000<bytes32>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

The <bytes32\> in the data key name corresponds to the typeId passed to the universalReceiver(..) function.

> Warning
> When constructing this data key for a specific typeId, unique elements of the typeId SHOULD NOT be on the right side because of trimming rules.
>
> The <bytes32> is trimmed on the right side to keep only the first 20 bytes. Therefore, implementations SHOULD ensure that the first 20 bytes are unique to avoid clashes. For example, the bytes32 typeId below:
>
> `0x11112222333344445555666677778888999900000aaaabbbbccccddddeeeeffff`
>
> will be trimmed to 0x11112222333344445555666677778888999990000.
>
> See the section about the trimming rules for the key type Mapping (./LSP-2-ERC725YJSONSchema.md#mapping) in LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode this data key.

If the account delegates its universal receiver functionality to another smart contract, this smart contract address MUST be stored under the data key attached above. This call to this contract is performed when the universalReceiver(bytes32,bytes) function of the account is called with a specific typeId that it can react on.

Check the UniversalReceiver Delegation > Specification section in LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md#universalreceiver-delegation) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

LSP17Extension

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "(address, bytes1)",
  "valueContent": "(Address, bool)"
}
```

> <bytes4> is the functionSelector called on the account contract. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the data key.

If there is a function called on the account and the function does not exist, the fallback function lookup an address stored under the data key attached above and forwards the call to it with the value of the msg.sender and msg.value appended as extra calldata.

If the data stored is just 20 bytes, representing an address, or 21 bytes with the boolean set to false (anything other than 0x01), the extension will be called without sending the value received to the extension.

If the data stored is 21 bytes with the boolean set to true (strictly 0x01), the extension will be called with sending the value received to the extension. (Does not change that the value MUST be appended to the call)

Check the LSP17Extension Specification section in LSP17-ContractExtension (./LSP-17-ContractExtension.md#lsp17extendable-specification) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

Graffiti

Graffiti refers to the arbitrary messages or data sent to an LSP0-ERC725Account contract that do not match any existing function selectors, such as execute(..), setData(..), etc. These bytes, often carrying a message or additional information, are usually not intended to invoke specific functions within the contract.

When the account is called with specific bytes that do not match any function selector, it will first check its storage to see if there are any extensions set for these function selectors (bytes). If no extension is found, the call will typically revert. However, to emulate the behavior of calling an Externally Owned Account (EOA) with random bytes (which always passes), an exception has been made for the 0x00000000 selector.

When the account is called with data that starts with 0x00000000, it will first check for extensions. If none are found, the call will still pass, allowing it to match the behavior of calling an EOA and enabling the ability to send arbitrary messages to the account. For example, one might receive a message like "This is a gift" while sending native tokens.

Additionally, it is possible to set an extension for the 0x00000000 selector. With this custom extension, you can define specific logic that runs when someone sends graffiti to your account. For instance, you may choose to disallow sending graffiti by reverting the transaction, impose a fee for sending graffiti, or emit the graffiti on an external contract. This flexibility allows for various use cases and interactions with graffiti in the LSP0ERC725Account contracts.

Rationale

ER725Y Data Storage

The ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) standard is crucial for an account contract as it allows for flexible data storage using a key/value pair structure. This data can be used to store a wide variety of information, making it possible for the account to support numerous applications and services. It can also be used to help owner of the account to dictate a certain execution logic, for instance storing permissions of addresses allowed to interact with the account, can be read by an owner contract and allow execution based on it.

It's a future-proofing feature that enables the account to interact with any smart contract or address, adapting to new functionalities as the ecosystem evolves. The versatility of ERC725Y ensures that an account can remain relevant and functional as new use cases and requirements emerge.

Notification and Reaction

Notification and Reaction are essential features that facilitate user interaction and automation in the context of Web3, analogous to notifications in the Web2 ecosystem.

Importance of Notifications

Just as notifications are crucial for the functionality of Web2 accounts, providing real-time updates and alerts on various actions, Web3 accounts must also possess this capability. Notifications serve as the bedrock for creating reactive, user-oriented applications that inform users promptly about events affecting their accounts.

Web3 Adoption and User Experience

To encourage broader adoption of Web3 accounts, they must be equipped with a standardized notification system that users are already familiar with in other technological spheres. This standardization helps in bridging the user experience gap between traditional (Web2) and blockchain (Web3) platforms.
Adopting the LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) standard is critical for enabling accounts to be universally and reliably informed of interactions, such as the reception of tokens, follows, royalties, etc .. This allows for the development of more sophisticated and automated response mechanisms within smart contracts.

By allowing the account to react to notifications with customizable logic, developers can program an array of automated responses tailored to the needs of the account holder. This flexibility enhances the potential for automation and facilitates a more intuitive and user-friendly blockchain experience.

Value Receive Notification and Reaction

The notification and reaction mechanism provided on native token transfer in the LSP0-ERC725Account standard smart contracts is essential for several reasons.

Off-Chain Monitoring

It provides a standardized and efficient way for off-chain entities to monitor and record native token deposits in smart contracts by listening to a single event which is the UniversalReceiver with a specific typeId being keccak256("LSP0ValueReceived") > 0x9c4705229491d365fb5434052e12a386d6771d976bea61070a8c694e8affea3d. This standardization makes integration simpler and more reliable for external applications tracking native token flows.

Reactive Measures for Received Funds

It allows the smart contract to distinguish between transactions initiated by the owner and those from other users. For owner-initiated transactions, the system simply notifies receipt of value. However, for public transactions, the contract can execute pre-defined logic to accept, reject, or process the funds according to the owner's requirements, enhancing security and control over the contract's balance.

This ability to react to incoming transfers is crucial for managing funds under various conditions, including legal compliance or operational rules set by the contract owner.

The provision of this mechanism aims to balance the need for transaction transparency with the necessity for direct control over the smart contract's fund management processes.

LSP20-CallVerification Usage

The account stands as a versatile and secure solution in the realm of blockchain smart contracts, allowing for a variety of ownership configurations. Users can decide to own the account with an EOA or opt for a multi-signature owner for enhanced security or a permission access control contract like LSP6-KeyManager (./LSP-6-KeyManager.md).

Challenge of Diverse Ownership

This flexibility, however, brings forth a challenge for external entities like websites and protocols looking to interact with the account. The process requires identifying the owner of the account, understanding the ownership structure, and then tailoring the interaction accordingly (encoding the transaction according to the ABI of the owner contract). This could vary significantly depending on whether the owner is a simple address, a multi-signature wallet, or a more complex contract like the LSP6 KeyManager.

Consistent Interaction Regardless of Ownership

To streamline this interaction process and provide a uniform approach, the account has integrated the LSP20-CallVerification standard. This standard allows any external entity to directly call functions on the account. The account, in turn, internally validates these calls by consulting with its owner, ensuring that the owner's logic and rules are followed.

This results in a consistent and straightforward interaction model for external entities. They can interact with the account in the same way, regardless of the ownership structure in place.

Example Scenario with LSP6-KeyManager

Let's take a practical example to illustrate this. Suppose the account is owned by an LSP6-KeyManager (./LSP-6-KeyManager.md), which operates based on permissions. An external controller (an address) wishes to update some data on the account using the setData function. Since LSP20 Standard is applied, the controller can directly call setData on the account. The setData function then forwards this call, along with the caller's information, to the LSP6-KeyManager (./LSP-6-KeyManager.md) contract which evaluate whether the controller has the necessary permissions to perform this action. If yes, the call is executed; if not, it is denied.

This streamlined process facilitated by the LSP20-CallVerification (./LSP-20-CallVerification.md) standard ensures a uniform and user-friendly way of interacting with the account, making it more accessible and easier to integrate into various applications and services.

Account Unbiasedness

The rationale for emphasizing account unbiasedness within a smart contract framework stems from the recognition of diverse user needs and the value of reputational continuity.

Diversity of User Preferences

Smart contract users come with different security needs and preferences which can change over time. A user might prefer different control mechanisms like social recovery, access control lists, or multisig setups at different stages. To cater to this variability, the underlying account structure should be feature-agnostic, providing only the core functionalities without pre-set features that presume the needs of all users.

Avoiding Feature Presumption

By designing accounts to be unbiased in terms of features, it allows for a neutral starting point where any specialized functionality can be layered on top by the user's choice. This approach acknowledges that no one-size-fits-all solution can effectively serve the varying and evolving requirements of all users.

Reputation and Continuity

Users build reputation and trust on their account addresses over time, through transaction history, balance accumulation, and interactions with other contracts. Having to migrate to a new account because of a change in preference can disrupt this continuity. Users face the inconvenience of transferring tokens and

data and lose the intangible value of their established reputation, which is not transferable between accounts.

By separating core account functionalities from optional features, the system supports a modular approach to security and control customization. Users can retain their base account while changing the feature sets applied to it, maintaining their reputational capital and avoiding the disruption of account migrations. Thus, the unbiased basic account acts as a persistent, adaptable foundation for controlling the user identity on the blockchain, accommodating different needs.

Extending Functionalities and InterfaceIds

The rationale for the capability to extend functionalities and support new InterfaceIds post-deployment is anchored in adaptability and future-proofing of smart contract accounts.

Ongoing Standardization

The blockchain ecosystem is continuously evolving, with new standards and best practices emerging regularly. Post-deployment adaptability ensures that an account can remain relevant and compliant with new standards as they arise. Extension mechanisms allow accounts to adopt new functionalities without needing to be redeployed. This approach supports the seamless adoption of innovations and standards, making the contract resilient to obsolescence.

Functionality Accessibility

By allowing the invocation of functions that are not natively present in the contract but provided through extensions, the account can maintain compatibility with new standards and interfaces. This prevents the contract from failing when interacting with newly defined functions that it did not originally include, thereby maintaining the contract's utility and interactions within the evolving ecosystem.

Disclosure of Supported InterfaceIds

Extending the ERC165 interfaceIds that the account supports is critical for transparency and interoperability. It allows other contracts and services to detect supported interfaces and interact with the account accordingly, facilitating broader compatibility across the ecosystem.

With the capacity to evolve, accounts can have a longer operational life, reducing the need for users to migrate to newer accounts with updated features. This stability is crucial for maintaining the trust and reputation associated with a blockchain account over time.

Usage

The versatile nature of the unbiased account, with its capability to integrate various functions and ownership protocols, enables it to serve a wide range of purposes. From acting as a comprehensive Blockchain Profile for individuals in the Web3 space to functioning as an autonomous decentralized organization (DAO), the adaptability of the account caters to multiple use cases.

Blockchain Profile

For individual users navigating the complexities of Web3, the account can be customized to serve as a Blockchain Profile—a singular digital identity encapsulating user reputation, assets, and transaction history. It simplifies interactions within the ecosystem by abstracting the underlying technical processes and providing

a user-friendly interface for managing digital assets and identities, enhancing the user experience in decentralized applications (DApps), finance, and beyond.

Organization

An unbiased account can be structured to embody an Organization, enabling entities to govern operations, manage funds, and maintain compliance with legal frameworks that prohibit unsolicited transfers. The account's data keys can be standardized for organizational use, providing a clear and efficient way to manage access to contracts, execute transactions, and ensure that operations are fully auditable and transparent.

Reference Implementation

An implementation can be found in the [lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol)](https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol) repository.

Security Considerations

Delegatecall

The use of delegatecall in smart contracts is a sensitive operation with significant security implications. As it allows one contract to execute code in the context of another, it can potentially lead to the compromise or complete destruction of the contract's state if not handled correctly. Therefore, for accounts where delegatecall functionality is deemed unnecessary or where its risks outweigh its benefits, it is advisable to eliminate paths that permit its use in the owner contract, thus closing off a vector for attacks and vulnerabilities while being able to use it later with an owner upgrade.

Signature Replay

In the case of protocols interacting with the account, it is paramount to incorporate security measures that specifically address signature-based threats. To guard against replay attacks, where a signature is used maliciously on different contracts, the protocol must ensure that the account address is part of the signed data. This inclusion uniquely binds a signature to an account and context, preventing the misuse of signed messages and enhancing the overall security of the transaction verification process.

Interface Cheat Sheet

```
interface ILSP0  /* is ERC165 */ {


    // ERC725X

    event Executed(uint256 indexed operation, address indexed to, uint256 indexed value, bytes4 selector);

    event ContractCreated(uint256 indexed operation, address indexed contractAddress, uint256 indexed value, bytes32 salt);


    function execute(uint256 operationType, address to, uint256 value, bytes memory data) external payable returns (bytes memory);
```

```
    function executeBatch(uint256[] memory operationsType, address[] memory targets, uint256[] memory values, bytes[] memory datas) external payable returns(bytes[] memory);

    // ERC725Y

    event DataChanged(bytes32 indexed dataKey, bytes dataValue);

    function getData(bytes32 dataKey) external view returns (bytes memory dataValue);

    function setData(bytes32 dataKey, bytes memory dataValue) external payable;

    function getDataBatch(bytes32[] memory dataKeys) external view returns (bytes[] memory dataValues);

    function setDataBatch(bytes32[] memory dataKeys, bytes[] memory dataValues) external payable;

    // ERC1271

    function isValidSignature(bytes32 hash, bytes memory signature) external view returns (bytes4 returnedStatus);

    // LSP0 (ERC725Account)

    receive() external payable;

    fallback() external payable;

    function batchCalls(bytes[] calldata data) external returns (bytes[] memory results);

    // LSP1

    event UniversalReceiver(address indexed from, uint256 indexed value, bytes32 indexed typeId, bytes receivedData, bytes returnedValue);

    function universalReceiver(bytes32 typeId, bytes memory data) external payable returns (bytes memory);

    // LSP14

    event OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner);

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    event RenounceOwnershipInitiated();

    event OwnershipRenounced();
```

```
    function owner() external view returns (address);

    function pendingOwner() external view returns (address);

    function transferOwnership(address newOwner) external;

    function acceptOwnership() external;

    function renounceOwnership() external;

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-1-UniversalReceiver.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-1-UniversalReceiver.md

---

lip: 1
title: Universal Receiver
author: JG Carvalho (@jgcarv), Fabian Vogelsteller <@frozeman>
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2019-09-01
requires: ERC165

Simple Summary

<!--"If you can't explain it simply, you don't understand it well enough." Provide a simplified and layman-accessible explanation of the LIP.-->

An entry function enabling a contract to receive arbitrary information.

Abstract

<!--A short (~200 word) description of the technical issue being addressed.-->

Similar to a smart contract's fallback function, which allows a contract to be notified of an incoming transaction with a value, the universalReceiver(bytes32,bytes) function allows for any contract to receive information about any interaction.

This allows receiving contracts to react on incoming transfers or other interactions.

Motivation

There is often the need to inform other smart contracts about actions another smart contract did perform.

A good example are token transfers, where the token smart contract should inform receiving contracts about the transfer.

By creating a universal function that many smart contracts implement, receiving of asset and information can be unified.

In cases where smart contracts function as a profile or wallet over a long time, an upgradable universal receiver can allow for future assets to be received, without the need for the interface to be changed.

Specification

LSP1-UniversalReceiver interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0x6bb56a14.

Smart contracts implementing the LSP1-UniversalReceiver standard MUST implement the ERC165 (https://eips.ethereum.org/EIPS/eip-165) supportsInterface(..) function and MUST support ERC165 (https://eips.ethereum.org/EIPS/eip-165) and LSP1 interface ids.

Every contract that complies with the LSP1-UniversalReceiver standard MUST implement:

Methods

universalReceiver

```
function universalReceiver(bytes32 typeId, bytes memory data) external payable returns (bytes memory)
```

Allows to be called by any external contract to inform the contract about any incoming transfers, interactions or simple information.

The universalReceiver(...) function can be customized to react on a different aspect of the call such as the typeId, the data sent, the caller or the value sent to the function (*e.g, reacting on a token or a vault transfer*).

Parameters:

- typeId is the hash of a standard, or the type relative to the data received.
- data is a byteArray of arbitrary data. Receiving contracts SHOULD take the typeId in consideration to properly decode the data.

Returns: bytes which can be used to encode response values.

> Note: The universalReceiver(...) function COULD be allowed to return no data (no return as the equivalent of the opcode instruction return(memory_pointer, 0)).
> If any bytes data is returned, bytes not conforming to the default ABI encoding will result in a revert. See the specification for the abi-encoding of bytes (https://docs.soliditylang.org/en/v0.8.19/abi-spec.html#formal-specification-of-the-encoding) for more details.

Events

UniversalReceiver

```
event UniversalReceiver(address indexed from, uint256 indexed value, bytes32 indexed typeId, bytes receivedData, bytes returnedValue);
```

This event MUST be emitted when the universalReceiver function is succesfully executed.

Values:

- from is the address calling the universalReceiver(..) function.
- value is the amount of value sent to the universalReceiver(..) function.
- typeId is the hash of a standard, or the type relative to the data received.
- receivedData is a byteArray of arbitrary data received.
- returnedValue is the data returned by the universalReceiver(..) function.

UniversalReceiver Delegation

UniversalReceiver delegation allows to forward the universalReceiver(..) call on one contract to another external contract, allowing for upgradeability and changing behaviour of the initial universalReceiver(..) call.

Motivation

The ability to react to upcoming actions with a logic hardcoded within the universalReceiver(..) function comes with limitations, as only a fixed functionality can be coded or the UniversalReceiver event be fired.

This section explains a way to forward the call to the universalReceiver(..) function to an external smart contract to extend and change funcitonality over time.

The delegation works by simply forwarding a call to the universalReceiver(..) function to a delegated smart contract calling the universalReceiverDelegate(..) function on the external smart contract.
As the external smart contract doesn't know about the inital msg.sender and the msg.value, this specification proposes to add these values as arguments. This allows the external contract to know the full context of the initial universalReceiver call and react accordingly.

Specification

LSP1-UniversalReceiverDelegate interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0xa245bbda.

The UniversalReceiverDelegate is an optional extension. It allows the universalReceiver(..) function to delegate its functionality to an external contract that can be customized to react differently based on the typeId and the data received.

Methods

universalReceiverDelegate

```
function universalReceiverDelegate(address caller, uint256 value, bytes32 typeId, bytes memory data) external returns (bytes memory);
```

Allows to be called by any external contract when an address wants to delegate its universalReceiver functionality to another smart contract.

Parameters:

- caller is the address calling the universalReceiver function.
- value is the amount of value sent to the universalReceiver function.
- typeId is the hash of a standard, or the type relative to the data received.
- data is a byteArray of arbitrary data. Receiving contracts should take the typeId in consideration to properly decode the data.

Returns: bytes, which can be used to encode response values.

If the contract implementing the LSP1 standard is an ERC725Y, the address of the UniversalReceiverDelegate contract COULD be stored under the following ERC725Y data key:

```
{
  "name": "LSP1UniversalReceiverDelegate",
  "key": "0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
  "keyType": "Singleton",
  "valueType": "address",
  "valueContent": "Address"
}
```

Additionally, some specific UniversalReceiverDelegate contracts COULD be mapped to react on specific typeId. The address of each of these contracts for each specific typeId COULD be stored under the following ERC725Y data key:

```
{
  "name": "LSP1UniversalReceiverDelegate:<bytes32>",
  "key": "0x0cfc51aec37c55a4d0b10000<bytes32>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

The <bytes32\> in the data key name corresponds to the typeId passed to the universalReceiver(..) function. For example, for the typeId 0xcafecafecafecafecafecafecafecafecafecafebeefbeefbeefbeefbeefbeef, the data key above will be constructed as follow:

```
0x0cfc51aec37c55a4d0b10000cafecafecafecafecafecafecafecafecafecafe
```

Rationale

This is an abstraction of the ideas behind ERC223 (https://github.com/ethereum/EIPs/issues/223) and ERC777 (https://eips.ethereum.org/EIPS/eip-777), that contracts are called when they are receiving tokens or other assets.

With this proposal, we can allow contracts to receive any information in a generic manner over a standardised interface.

As this function is generic and only the sent typeId changes, smart contract accounts that can upgrade its behaviour using the UniversalReceiverDelegate technique can be created.

The UniversalReceiverDelegate functionality COULD be implemented using call, or delegatecall, both of which have different security properties.

Implementation

An implementation can be found in the lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol) repository.

UniversalReceiver Example:

After transfering token from TokenABC to MyWallet, the owner of MyWallet contract can know, by looking at the emitted UniversalReceiver event, that the typeId is _TOKEN_RECEIVING_HASH.

Enabling the owner to know the token sender address and the amount sent by looking into the data.

```
// SPDX-License-Identifier: CC0-1.0
pragma solidity ^0.8.0;


contract TokenABC {

    // Hash of the word `_TOKEN_RECEIVING_HASH`
    bytes32 constant public _TOKEN_RECEIVING_HASH =
0x7901c95ea4b5fe1fba45bb8c10d7ddabf715dea547785f933d8be283925c4883;

    bytes4 _INTERFACE_ID_LSP1 = 0x6bb56a14;

    function sendToken(address to, uint256 amount) public {
        balance[msg.sender] -= amount;
        balance[to] += amount;
        _informTheReceiver(to, amount);
    }

    function _informTheReceiver(address receiver, uint256 amount) internal {
        // If the contract receiving the tokens supports LSP1 InterfaceID then call the unviersalReceiver function
        if(ERC165Checker.supportsInterface(receiver,_INTERFACE_ID_LSP1)){
            ILSP1(receiver).universalReceiver(_TOKEN_RECEIVING_HASH, abi.encodePacked(msg.sender,
amount));
        }
    }
}

contract MyWallet is ERC165, ILSP1 {

    bytes4 _INTERFACE_ID_LSP1 = 0x6bb56a14;

    constructor() public {
        _registerInterface(_INTERFACE_ID_LSP1);
    }

    function universalReceiver(bytes32 typeId, bytes memory data) public payable returns (bytes memory) {
        emit UniversalReceiver(msg.sender, msg.value, typeId, data, 0x);
        return 0x0;
    }
}
```

UniversalReceiverDelegate Example:

This example is the same example written above except that MyWallet contract now delegates the universalReceiver functionality to a UniversalReceiverDelegate contract.

The TokenABC contract will inform the MyWallet contract about the transfer by calling the universalReceiver(..) function. This function will then call the universalReceiver(..) function on the UniversalReceiverDelegate address set by the owner, to react on the transfer accordingly.

```
// SPDX-License-Identifier: CC0-1.0
pragma solidity ^0.8.0;
```

```
contract TokenABC {

    // Hash of the word `_TOKEN_RECEIVING_HASH`
    bytes32 constant public _TOKEN_RECEIVING_HASH =
0x7901c95ea4b5fe1fba45bb8c10d7ddabf715dea547785f933d8be283925c4883;

    bytes4 _INTERFACE_ID_LSP1 = 0x6bb56a14;

    function sendToken(address to, uint256 amount) public onlyOwner {
        balance[to] += amount;
        _informTheReceiver(to, amount);
    }

    function _informTheReceiver(address receiver, uint256 amount) internal {
        // If the contract receiving the tokens supports LSP1 InterfaceID then call the unviersalReceiver function
        if(ERC165Checker.supportsInterface(receiver,_INTERFACE_ID_LSP1)){
            ILSP1(receiver).universalReceiver(_TOKEN_RECEIVING_HASH,
abi.encodePacked(address(this),amount));
        }
    }
}

contract MyWallet is ERC165, ILSP1 {

    bytes4 _INTERFACE_ID_LSP1 = 0x6bb56a14;
    bytes4 _INTERFACE_ID_LSP1_DELEGATE = 0xa245bbda;

    address public universalReceiverDelegate;

    constructor() public {
        _registerInterface(_INTERFACE_ID_LSP1);
    }

    function setUniversalReceiverDelegate(address _newUniversalReceiverDelegate) public onlyOwner {
        // The address set SHOULD support LSP1Delegate InterfaceID
        universalReceiverDelegate = _newUniversalReceiverDelegate;
    }

    function universalReceiver(bytes32 typeId, bytes memory data) public payable returns (bytes memory) {

        // if the address set as universalReceiverDelegate supports LSP1Delegate then call the
universalReceiverDelegate function

if(ERC165Checker.supportsInterface(universalReceiverDelegate,_INTERFACE_ID_LSP1_DELEGATE)){

            // Call the universalReceiverDelegate function on universalReceiverDelegate address
            returnedData = ILSP1(universalReceiverDelegate).universalReceiverDelegate(msg.sender,
msg.value, typeId, data);
```

```
        emit UniversalReceiver(msg.sender, msg.value, typeId, data, returnedData);
        return returnedData;
    }
}



contract UniversalReceiverDelegate is ERC165, ILSP1 {

    bytes4 _INTERFACE_ID_LSP1_DELEGATE = 0xa245bbda;

    constructor() public {
        _registerInterface(_INTERFACE_ID_LSP1_DELEGATE);
    }

    function universalReceiverDelegate(address caller, uint256 value, bytes32 typeId, bytes memory data)
public returns (bytes memory) {
        // Any logic could be written here:
        // - Interfact with DeFi protocol contract to sell the new tokens received automatically.
        // - Register the token received on other registery contract.
        // - Allow only tokens with `_TOKEN_RECEIVING_HASH` hash and reject the others.
        // - revert; so in this way the wallet will have the option to reject any token.
    }
}
```

Interface Cheat Sheet

```
interface ILSP1  /* is ERC165 */ {

    event UniversalReceiver(address indexed from, uint256 value, bytes32 indexed typeId, bytes
receivedData, bytes returnedValue);


    function universalReceiver(bytes32 typeId, bytes memory data) external payable returns (bytes memory);

}

interface ILSP1Delegate  /* is ERC165 */ {


    function universalReceiverDelegate(address caller, uint256 value, bytes32 typeId, bytes memory data)
external returns (bytes memory);

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-10-ReceivedVaults.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-10-ReceivedVaults.md

---

lip: 10
title: Received Vaults
author:
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2021-12-1
requires: LSP2

Simple Summary

This standard describes a set of ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) data key-value pairs that can be used to store addresses of received vaults in a ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract.

Abstract

The following two data keys (including their ERC725Y JSON schema) are proposed to represent vaults owned by a smart contract:

- LSP10Vaults[] to hold an array of vault addresses
- LSP10VaultsMap to hold a mapping of the index in the former array and the interface ID of the standard used by the vault. This enables to quickly differentiate vaults standards apart without the need to query each vault smart contract separately.

The data key LSP10VaultsMap also helps to prevent adding duplications to the array, when automatically added via smart contract (e.g. a LSP1-UniversalReceiverDelegate (./LSP-1-UniversalReceiver.md)).

Motivation

To be able to display received vaults in a profile we need to keep track of all received vaults contract addresses. This is important for LSP0-ERC725Account (./LSP-0-ERC725Account.md).

Specification

Every contract that supports the LSP9Vault standard SHOULD have the following data keys:

ERC725Y Data Keys

LSP10Vaults[]

References issued smart contract vaults.

```
{
  "name": "LSP10Vaults[]",
  "key": "0x55482936e01da86729a45d2b87a6b1d3bc582bea0ec00e38bdb340e3af6f9f06",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
}
```

LSP10VaultsMap

References owned LSP9Vaults (./LSP-9-Vault.md). This data key exists so that smart contracts can detect whether the address of a vault is present in the LSP10Vaults[] array without looping all over it on-chain. Moreover, it helps to identify at which index in the LSP10Vaults[] the vault address is located for easy access and to change or remove this specific vault from the array. Finally, it also allows the detection of the interface supported by the vault.

The data value MUST be constructed as follows: bytes4(standardInterfaceId) + uint128(indexNumber). Where:

- standardInterfaceId = the ERC165 interface ID (https://eips.ethereum.org/EIPS/eip-165) of a LSP9Vaults (./LSP-9-Vault.md): 0xfd4d5c50.
- indexNumber = the index in the LSP10Vaults[] Array

Value example: 0xfd4d5c50000000000000000c (interfaceId: 0xfd4d5c50, index position 0x000000000000000c = 12).

```
{
  "name": "LSP10VaultsMap:<address>",
  "key": "0x192448c3c0f88c7f238c0000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
}
```

Rationale

Implementation

An implementation can be found in the lukso-network/standards-scenarios (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/tree/develop/contracts/LSP1UniversalReceiver/LSP1UniversalReceiverDelegateVault);
Below is the ERC725Y JSON interface of the LSP10ReceivedVaults.

ERC725Y JSON Schema LSP10ReceivedVaults:

```
[
  {
    "name": "LSP10Vaults[]",
    "key": "0x55482936e01da86729a45d2b87a6b1d3bc582bea0ec00e38bdb340e3af6f9f06",
    "keyType": "Array",
    "valueType": "address",
    "valueContent": "Address"
  },
  {
    "name": "LSP10VaultsMap:<address>",
    "key": "0x192448c3c0f88c7f238c0000<address>",
    "keyType": "Mapping",
    "valueType": "(bytes4,uint128)",
    "valueContent": "(Bytes4,Number)"
  }
]
```


Copyright

</file>

<file>

path: /LSPs/LSP-11-BasicSocialRecovery.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-11-BasicSocialRecovery.md

---

lip: 11
title: Basic Social Recovery
author:
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2021-2-14
requires: ERC165, ERC725, LSP2, LSP6

Simple Summary

This standard describes a basic social recovery contract that can recover access to ERC725 (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md) contracts through a LSP6-KeyManager (./LSP-6-KeyManager.md).

Abstract

This standard provides a mechanism for recovering access to ERC725 contracts such as tokens, NFTs, and Universal Profiles by adding a new controller address through the Key Manager after a recovery process.

The social recovery contract should ensure a flexible and secure process where guardians, nominated by the owner, can select one address. The address after reaching the guardiansThreshold and after successfully providing the secret word which produces the same hash as the secret hash set by the owner, will be granted the owner permissions in the recovered target and a new hash will be set and a new recovery counter will be created.

Motivation

Any Key could be lost or leaked due to a certain accident, so it is not advised to rely on a singular key to control ERC725 contracts through the Key Manager and a social recovery contract is needed in this case.

In the case above, the user can reach out to his guardians and ask them to vote for a specific address. There are many possible options for whom to select as a guardian. The three most common options are:

- EOAs controlled by the wallet holder themselves (via paper mnemonics or cold storage devices)
- Friends and family members (EOAs or Universal Profiles)
- Institutions, which could vote for a provided address if they get a valid confirmation via phone number, email or video call

Specification

LSP11 interface id according to [ERC165 (https://eips.ethereum.org/EIPS/eip-165)](https://eips.ethereum.org/EIPS/eip-165): 0x049a28f1.

Methods

Smart contracts implementing the LSP11 standard MUST implement all of the functions listed below:

target

```
function target() external view returns (address)
```

Returns the address of the linked ERC725 contract to recover.

getRecoveryCounter

```
function getRecoveryCounter() external view returns (uint256)
```

Returns the number of finished successful recovery processes.

getGuardians

```
function getGuardians() external view returns (address[] memory)
```

Returns the array of guardian addresses set.

isGuardian

```
function isGuardian(address _address) external view returns (bool)
```

Returns *true* if the provided address is a guardian, *false* otherwise.

Parameters:

- _address: the address to query.

getGuardiansThreshold

```
function getGuardiansThreshold() external view returns (uint256)
```

Returns the minimum number of guardians selection required by an address to start a recovery process.

getRecoverySecretHash

```
function getRecoverySecretHash() external view returns (bytes32)
```

Returns the recovery secret hash set by the owner.

getGuardianChoice

```
function getGuardianChoice(address guardian) external view returns (address)
```

Returns the address that a guardian selected for target recovery.

Parameters:

- guardian: the address that guardian has selected.

addGuardian

```
function addGuardian(address newGuardian) external
```

Adds a guardian of the target. MUST fire the AddedGuardian event.

Parameters:

- newGuardian: the address of the guardian to set.

Requirements:

- MUST be called only by the owner.

removeGuardian

```
function removeGuardian(address currentGuardian) external
```

Removes an existing guardian of the target. MUST fire the RemovedGuardian event.

Parameters:

- currentGuardian: the address of the guardian to remove.

Requirements:

- MUST be called only by the owner.

setGuardiansThreshold

```
function setGuardiansThreshold(uint256 newThreshold) external
```

Sets the minimum number of selection by the guardians required so that an address can recover ownership to the linked target contract. MUST fire the GuardianThresholdChanged event.

If the GuardiansThreshold is equal to 0, the social recovery contract will act as a password recovery contract.

Parameters:

- newThreshold: the threshold to set.

Requirements:

- MUST be called only by the owner.

setRecoverySecretHash

```
function setRecoverySecretHash(bytes32 newSecretHash) external
```

Sets the hash of the plainSecret needed to recover the target after reaching the guardians threshold. MUST fire the SecretHashChanged event.

Parameters:

- newHash: the hash of the plainSecret.

Requirements:

- MUST be called only by the owner.
- MUST not be bytes32(0).

selectNewController

```
function selectNewController(address addressSelected) external
```

Select an address to be a potentiel controller address if he reaches the guardian threshold and provide the correct plainSecret. MUST fire the SelectedNewController event.

Parameters:

- addressSelected: The address selected by the guardian.

Requirements:

- MUST be called only by the guardians.

recoverOwnership

```
function recoverOwnership(address recoverer, string memory plainSecret, bytes32 newHash) external
```

Increment the recovery counter and recovers the ownership permissions in the linked target for the recoverer if he has reached the guardiansThreshold and given the right plainSecret that produce the secretHash. MUST fire the RecoveryProcessSuccessful event and the SecretHashChanged event.

Parameters:

- recoverer: the address of the recoverer.

- plainSecret: the plain secret that should produce the secretHash with *keccak256* function.
- newHash: the new secret hash to set for the future recovery process.

Requirements:

- MUST have provided the right plainSecret that produces the secretHash originally set by the owner.

Events

GuardianAdded

```
event GuardianAdded(address indexed newGuardian);
```

MUST be emitted when setting a new guardian for the target.

GuardianRemoved

```
event GuardianRemoved(address indexed removedGuardian);
```

MUST be emitted when removing an existing guardian for the target.

GuardiansThresholdChanged

```
event GuardiansThresholdChanged(uint256 indexed guardianThreshold);
```

MUST be emitted when changing the guardian threshold.

SecretHashChanged

```
event SecretHashChanged(bytes32 indexed secretHash);
```

MUST be emitted when changing the secret hash.

SelectedNewController

```
event SelectedNewController(uint256 indexed currentRecoveryCounter, address indexed guardian, address indexed addressSelected);
```

MUST be emitted when a guardian select a new potentiel controller address for the linked target.

RecoveryProcessSuccessful

```
event RecoveryProcessSuccessful(uint256 indexed recoveryCounter, address indexed newController, bytes32 indexed newSecretHash, address[] guardians);
```

MUST be emitted when the recovery process is finished by the controller who reached the guardian threshold and submitted the string that produce the secretHash

Setup

In order to allow the social recovery contract to recover the linked target and add new permissions, the linked target should have an [LSP6-KeyManager (./LSP-6-KeyManager.md)](./LSP-6-KeyManager.md) as owner and the social recovery contract should have ADDCONTROLLER and EDITPERMISSIONS permissions set inside the linked target

under this ERC725Y data key.

```
{
  "name": "AddressPermissions:Permissions:<address>",
  "key": "0x4b80742de2bf82acb3630000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "bytes32",
  "valueContent": "BitArray"
}
```

Rationale

This standard was inspired by the current recovery process in some crypto wallets with a balance between relying on the guardians and a secret hash.

In this case, it is ensured that guardians can't act maliciously and would need a secret word to recover. The same goes for the secret word if it is exposed, only addresses who reached the guardiansThreshold can use it to recover the target.

Implementation

An implementation can be found in the [lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/pull/114)](https://github.com/lukso-network/lsp-smart-contracts/pull/114) repository.

Interface Cheat Sheet

```
interface ILSP11  /* is ERC165 */ {

    event GuardianAdded(address indexed newGuardian);

    event GuardianRemoved(address indexed removedGuardian);

    event GuardiansThresholdChanged(uint256 indexed guardianThreshold);

    event SecretHashChanged(bytes32 indexed secretHash);

    event SelectedNewController(
        uint256 indexed recoveryCounter,
        address indexed guardian,
        address indexed controllerSelected
    );

    event RecoveryProcessSuccessful(
        uint256 indexed recoveryCounter,
        address indexed newController,
        bytes32 indexed newSecretHash,
        address[] guardians
    );


    function target() external view returns (address);

    function getRecoveryCounter() external view returns (uint256);

    function getGuardians() external view returns (address[] memory);

    function isGuardian(address _address) external view returns (bool);

    function getGuardiansThreshold() external view returns (uint256);

    function getRecoverySecretHash() external view returns (bytes32);

    function getGuardianChoice(address guardian) external view returns (address);

    function addGuardian(address newGuardian) external;

    function removeGuardian(address currentGuardian) external;

    function setRecoverySecretHash(bytes32 newRecoverSecretHash) external;

    function setGuardiansThreshold(uint256 guardiansThreshold) external;

    function selectNewController(address addressSelected) external;

    function recoverOwnership(address recoverer, string memory plainSecret, bytes32 newHash) external;

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-12-IssuedAssets.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-12-IssuedAssets.md

---

lip: 12
title: Issued Assets
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2022-05-24
requires: LSP2

Simple Summary

This standard describes a set of ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) data key values to store addresses of issued assets by an ERC725X (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract.

Abstract

This data key value standard describes a set of data keys that can be added to an ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract to describe issued assets:

- LSP12IssuedAssets[] is an LSP2 array (./LSP-2-ERC725YJSONSchema.md) of addresses.
- LSP12IssuedAssetsMap is a dynamic address mapping, which contains:
    - an ERC165 interface ID (https://eips.ethereum.org/EIPS/eip-165) to easily identify the standard used by the mapped asset smart contract
    - and the index in the LSP12IssuedAssets[] array.

The data key LSP12IssuedAssetsMap exists so that smart contracts can detect if an address is present in the array (e.g. as done in the LSP1-UniversalReceiverDelegate (./LSP-1-UniversalReceiver.md)).

Motivation

This standard allows any smart contract to state that it issued a certain asset. The asset itself MUST reference the issuer smart contract as well, for it be verifable issued. This allows other smart contracts to link the authenticity of an asset to a specific issuer. See also LSP4 - DigitalAsset Metadata (./LSP-4-DigitalAsset-Metadata.md) for the owner and LSP4Creators[].

A full verification flow for an asset should contain a check on the asset and the issuer smart contract. If we use an asset using LSP4 - DigitalAsset Metadata (./LSP-4-DigitalAsset-Metadata.md) and a LSP0 - ERC725Account (./LSP-0-ERC725Account.md) as the issuer. The flow should looks as follows:

1. Smart contract that receives asset, should check for the owner or the LSP4Creators[] array and retrieve an issuer address.
2. Then check on the issuer that a LSP12IssuedAssetsMap is set for this asset address

Specification

Every contract that supports the ERC725Account SHOULD have the following data keys:

ERC725Y Data Keys

LSP12IssuedAssets[]

An array of issued smart contract assets, like tokens (*e.g.: LSP7 Digital Assets (./LSP-7-DigitalAsset)*) and NFTs (*e.g.: LSP8 Identifiable Digital Assets (./LSP-8-IdentifiableDigitalAsset)*).

```
{
  "name": "LSP12IssuedAssets[]",
  "key": "0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
}
```

For more info about how to access each index of the LSP12IssuedAssets[] array, see ERC725Y JSON Schema > keyType: Array (https://github.com/lukso-network/LIPs/blob/master/LSPs/LSP-2-ERC725YJSONSchema.md#array)

LSP12IssuedAssetsMap

References issued smart contract assets, like tokens (*e.g.: LSP7 Digital Assets (./LSP-7-DigitalAsset)*) and NFTs (*e.g.: LSP8 Identifiable Digital Assets (./LSP-8-IdentifiableDigitalAsset)*). This data key exists so that smart contracts can detect whether the address of an asset is present in the LSP12IssuedAssets[] array without looping all over it on-chain. Moreover, it helps to identify at which index in the LSP12IssuedAssets[] the asset address is located for easy access and to change or remove this specific asset from the array. Finally, it also allows the detection of the interface supported by the asset.

The data value MUST be constructed as follows: bytes4(standardInterfaceId) + uint128(indexNumber). Where:

- standardInterfaceId = the ERC165 interface ID (https://eips.ethereum.org/EIPS/eip-165) of the standard that the token or asset smart contract implements (if the ERC165 interface ID is unknown, standardInterfaceId = 0xffffffff).
- indexNumber = the index in the LSP12IssuedAssets[] Array

Value example: 0x5fcaac27000000000000000c (interfaceId: 0x5fcaac27 for a LSP7 (./LSP-7-DigitalAsset.md) token, index position 0x000000000000000c = 12).

```
{
  "name": "LSP12IssuedAssetsMap:<address>",
  "key": "0x74ac2555c10b9349e78f0000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
}
```

Rationale

Implementation

ERC725Y JSON Schema LSP12IssuedAssets:

```
[
  {
    "name": "LSP12IssuedAssets[]",
    "key": "0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd",
    "keyType": "Array",
    "valueType": "address",
    "valueContent": "Address"
  },
  {
    "name": "LSP12IssuedAssetsMap:<address>",
    "key": "0x74ac2555c10b9349e78f0000<address>",
    "keyType": "Mapping",
    "valueType": "(bytes4,uint128)",
    "valueContent": "(Bytes4,Number)"
  }
]
```

Copyright

</file>

<file>

path: /LSPs/LSP-14-Ownable2Step.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-14-Ownable2Step.md

---

lip: 14
title: Ownable 2 Step
author:
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2022-09-23
requires: ERC173, LSP1

Simple Summary

<!--"If you can't explain it simply, you don't understand it well enough." Provide a simplified and layman-accessible explanation of the LIP.-->

This standard describes an extended version of EIP173 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-173.md) that uses a 2-step process to transfer or renounce ownership of a contract, instead of instant execution.

In addition, this standard defines hooks that call the universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function of the current owner and new owner, if these addresses are contracts that implement LSP1. This aims to:

- notify when the new owner of the contract should accept ownership.
- notify the previous and new owner when ownership of the contract has been fully transferred.

Abstract

<!--A short (~200 word) description of the technical issue being addressed.-->

Because owning the contract allows access to sensitive methods, transferring to the wrong address or renouncing ownership of the contract by accident in a single transaction can be highly dangerous. Having those two processes work in 2 steps substantially reduces the probability of transferring or renouncing ownership of the contract by accident.

Motivation

<!--The motivation is critical for LIPs that want to change the Ethereum protocol. It should clearly explain why the existing protocol specification is inadequate to address the problem that the LIP solves. LIP submissions without sufficient motivation may be rejected outright.-->

The particular issue that the LSP14 standard solves is the irreversible nature of transferring or renouncing ownership of a contract.

Transferring ownership of the contract in a single transaction does not guarantee that the address behind the new owner (EOA or contract) is able to control the Ownable contract. For instance, if the new owner lost its private key or if the new owner is a contract that does not have any generic execution function.

Letting the new owner accept ownership of the contract guarantees that the contract is owned by an address (EOA or contract) that can be controlled, and that control over the contract implementing LSP14 will not be lost.

Finally, transferring ownership of the contract in two-steps enables the new owner to decide if he wants to become the new owner or not.

Specification

ERC165 (https://eips.ethereum.org/EIPS/eip-165) interface id: 0x94be5999

This interface id can be used to detect Ownable2Step contracts.

Methods

The methods are based on the methods from ERC173 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-173.md#specification) (Ownable), with additional changes. *See below for details*

owner

```
function owner() external view returns (address);
```

Returns the address of the current contract owner.

pendingOwner

```
function pendingOwner() external view returns (address);
```

Returns the address of the upcoming new owner that was initiated by the current owner via transferOwnership(address).

Requirements:

- MUST be address(0) if no ownership transfer is in progress.
- MUST be set to a new address when transferring ownership of the contract via transferOwnership(address).
- SHOULD be cleared once the pendingOwner() has accepted ownership of the contract.

transferOwnership

```
function transferOwnership(address newOwner) external;
```

Sets the newOwner as pendingOwner(). To transfer ownership fully of the contract, the pending owner MUST accept ownership via the function acceptOwnership().

MUST emit a OwnershipTransferredStarted event once the new owner was set as pendingOwner().

Requirements:

- MUST only be called by the current owner() of the contract.

LSP1 Hooks:

- If the new owner address supports LSP1 (./LSP-1-UniversalReceiver.md) interface, SHOULD call the new owner's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the default parameters below:
    - typeId: keccak256('LSP14OwnershipTransferStarted') > 0xee9a7c0924f740a2ca33d59b7f0c2929821ea9837ce043ce91c1823e9c4e52c0
    - data: The data sent SHOULD be abi encoded and contain the current owner (address) and the pending owner (address) respectively.

The Type ID associated with this hook COULD be altered in a contract that inherits from LSP14. This allows for more straightforward identification of the contract whose ownership is being transferred. Example where the LSP14 type ID is overriden can be found in LSP0 (LSP-0-ERC725Account.md#transferownership) and LSP9 (LSP-9-Vault.md#transferownership) standards.

acceptOwnership

```
function acceptOwnership() external;
```

Allows the pendingOwner() to accept ownership of the contract.

This function MUST be called as the second step (after transferOwnership(address)) by the current pendingOwner() to finalize the ownership transfer.

MUST emit a OwnershipTransferred (https://eips.ethereum.org/EIPS/eip-173#specification) event once the new owner has claimed ownership of the contract.

Requirements:

- MUST only be called by the pendingOwner().

LSP1 Hooks:

- If the previous owner is a contract that supports LSP1 (./LSP-1-UniversalReceiver.md) interface, SHOULD call the previous owner's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the default parameters below:
    - typeId: keccak256('LSP14OwnershipTransferred_SenderNotification') > 0xa124442e1cc7b52d8e2ede2787d43527dc1f3ae0de87f50dd03e27a71834f74c
    - data: The data sent SHOULD be abi encoded and contain the previous owner (address) and the new owner (address) respectively.
- If the new owner is a contract that supports LSP1 (./LSP-1-UniversalReceiver.md) interface, SHOULD call the new owner's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the default parameters below:
    - typeId: keccak256('LSP14OwnershipTransferred_RecipientNotification') > 0xe32c7debcb817925ba4883fdbfc52797187f28f73f860641dab1a68d9b32902c
    - data: The data sent SHOULD be abi encoded and contain the previous owner (address) and the new owner (address) respectively.

The Type IDs associated with these hooks can be altered in a contract that inherits from LSP14. This allows for more straightforward identification of the contract whose ownership is being transferred. Examples where the LSP14 type IDs are overriden can be found in the LSP0 (LSP-0-ERC725Account.md#acceptownerhsip) and LSP9 (LSP-9-Vault.md#acceptownership) standards.

renounceOwnership

```
function renounceOwnership() external;
```

Leaves the contract without an owner. Once ownership of the contract is renounced, it MUST NOT be possible to call functions restricted to the owner only.

Since renouncing ownership is a sensitive operation, it SHOULD be done as a two step process by calling renounceOwnership(..) twice. First to initiate the process, second as a confirmation.

MUST emit a RenounceOwnershipInitiated event on the first renounceOwnership(..) call.
MUST emit OwnershipTransferred (https://eips.ethereum.org/EIPS/eip-173#specification) event after successfully renouncing the ownership.

Requirements:

- MUST be called only by the owner() only.

- The second call MUST happen AFTER the delay of 100 blocks and within the next 100 blocks from the first renounceOwnership(..) call.
- If 200 blocks have passed, the renounceOwnership(..) call phase SHOULD reset the process, and a new one will be initated.

LSP1 Hooks:

- If the previous owner is a contract that supports LSP1 (./LSP-1-UniversalReceiver.md) interface, SHOULD call the previous owner's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the default parameters below on the second renounce ownership call:
    - typeId: keccak256('LSP14OwnershipTransferred_SenderNotification') > 0xa124442e1cc7b52d8e2ede2787d43527dc1f3ae0de87f50dd03e27a71834f74c
    - data: The data sent SHOULD be abi encoded and contain the previous owner (address) and the new owner (address) respectively.

Events

OwnershipTransferStarted

```
event OwnershipTransferStarted(address indexed currentOwner, address indexed newOwner);
```

MUST be emitted when the process of transferring owhership of the contract is initiated.

Values:

- currentOwner Address of the current owner of the contract that implements LSP14.
- newOwner Address that will receive ownership of the contract that implements LSP14.

OwnershipTransferred

```
event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
```

MUST be emitted when ownership of the contract has been transferred.

RenounceOwnershipStarted

```
event RenounceOwnershipStarted();
```

MUST be emitted when the process of renouncing ownership of the contract is initiated.

OwnershipRenounced

```
event OwnershipRenounced();
```

MUST be emitted when ownership of the contract has been renounced.

Interface Cheat Sheet

```
interface ILSP14  /* is ERC173 */ {

    event OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner);

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    event RenounceOwnershipInitiated();

    event OwnershipRenounced();


    function owner() external view returns (address);

    function pendingOwner() external view returns (address);

    function transferOwnership(address newOwner) external; // onlyOwner

    function acceptOwnership() external; // only pendingOwner()

    function renounceOwnership() external; // onlyOwner

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-15-TransactionRelayServiceAPI.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-15-TransactionRelayServiceAPI.md

---

lip: 15
title: Transaction Relay Service API
author: Hugo Masclet git@hugom.xyz (mailto:git@hugom.xyz), Callum Grindle callumgrindle@gmail.com (mailto:callumgrindle@gmail.com)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2022-10-05

Simple Summary

A Transaction Relay Service API for consistency across all Transaction Relay Service providers.

Abstract

The LSP-6-KeyManager (./LSP-6-KeyManager.md) proposes an executeRelayCall() (./LSP-6-KeyManager.md#executerelaycall) function. It allows anybody to execute _calldata payload on a set ERC725 X or Y smart contract, given they have a signed message from a valid executor. This opens the way to Transaction Relay Services which send transactions on behalf of a user to cover their gas costs.

This document describes the API for a Transaction Relay Service.

Motivation

Standardizing the Transaction Relay Service API enables applications to be compatible with all Transaction Relay Services which may be built, and avoids a situation where specific applications are only compatible with specific Transaction Relay Services. This is essential for an open marketplace of Transaction Relay Services where a user can select the service which best fits their needs.

Specification

API

POST /execute

Executes a signed transaction on behalf of a Universal Profile using executeRelayCall.

- address - The address of the Universal Profile which is executing the transaction.
- transaction - An object containing the transaction parameters which will be executed with executeRelayCall.
    - abi - The abi-encoded transaction data (*e.g: a function call on the Universal Profile smart contract*) which will be passed as the payload parameter to the executeRelayCall function.
    - signature - The signed message according to LSP6 specification.
    - nonce - The nonce of the KeyManager fetched by calling getNonce(address address, uint128 channelId) on the LSP6 KeyManager contract.
    - validityTimestamps (optional) - Two concatenated uint128 timestamps which indicate a time duration for which the transaction will be considered valid. If no validityTimestamps parameter is passed the relayer should assume that validityTimestamps is 0 and the transaction will be valid indefinitely until it is executed.

Request body

```
{
  "address": "0xBB645D97B0c7D101ca0d73131e521fe89B463BFD", // Address of the UP
  "transaction": {
    "abi":
"0x7f23690c5ef83ad9559033e6e941db7d7c495acdce616347d28e90c7ce47cbfcfcad3bc5000000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000596f357c6aa5a21984a83b7eef4cb0720ac1fcf5a45e9d84c653d97b71bbe89b7a728c386a697066733a2f2f516d624b43744b4d75733767415244706177446 87a32506a4e36616f64346b69794e436851726d3451437858454b00000000000000",
    "signature":
"0x43c958b1729586749169599d7e776f18afc6223c7da21107161477d291d497973b4fc50a724b1b2ab98f3f8cf1d5cdbbbdf3512e4fbfbdc39732229a15beb14a1b",
    "nonce": 1, // KeyManager nonce
    "validityTimestamps": "0x00000000000000000000000006420f3f0000000000000000000000000065ec82d0"
  }
}
```

Response

```
{
  "transactionHash": "0xBB645D97B0c7D101ca0d73131e521fe89B463BFD"
}
```

POST /quota

Returns the available quota left for a registered Universal Profile.

- signature - The result of signing a hash calculated as an EIP-712 hash where the message is keccak256(address, timestamp).
- address - The controller address with permissions on the Universal Profile used to create the signature value.
- timestamp - Represents the time the signature was created. Must be +/- 300 seconds from current time to be considered a valid request. Value should be int, int256, uint or uint256.

Request body

```
{
  "address": "0xBB645D97B0c7D101ca0d73131e521fe89B463BFD",
  "timestamp": 1656408193,
  "signature":
"0xf480c87a352d42e49112257cc6afab0ff8365bb769424bb42e79e78cd11debf24fd5665b03407d8c2ce994cf5d718031a51a657d4308f146740e17e15b9747ef1b"
}
```

Response

```
{
  "quota": 1543091, // You have YYY left
  "unit": "gas", // could be "lyx", "transactionCount"
  "totalQuota": 5000000, // total gas for the month
  "resetDate": 1656408193
}
```

- quota shows available balance left in units defined by unit
- unit could be gas, lyx or transactionCount depending on the business model
- totalQuota reflects total limit. i.e. available + used quota since reset
- resetDate gives date that available quota will reset, e.g. a monthly allowance

> Quota systems could also use a Pay As You Go model, in which case totalQuota and resetData can be omitted

Copyright

</file>

<file>

path: /LSPs/LSP-16-UniversalFactory.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-16-UniversalFactory.md

---

lip: 16
title: Universal Factory
author:
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2022-11-19
requires: EIP104, EIP155, EIP1167

Simple Summary

This standard defines a universal factory smart contract, that will allow to deploy different types of smart contracts using CREATE2 (https://eips.ethereum.org/EIPS/eip-1014) opcode after being deployed with Nick Factory (https://github.com/Arachnid/deterministic-deployment-proxy) in order to produce the same address on different chains.

Abstract

This standard defines several functions to be used to deploy different types of contracts using CREATE2 (https://eips.ethereum.org/EIPS/eip-1014) such as normal and initializable contracts. These functions take into consideration the need to initialize in the same transaction for initializable contracts. The initialize data is included in the salt to avoid squatting addresses on different chains.

The same bytecode and salt will produce the same contract address on different chains, if and only if, the UniversalFactory contract was deployed on the same address on each chain, using Nick Factory (https://github.com/Arachnid/deterministic-deployment-proxy) in our case.

Motivation

Possessing a private key allows for the control of the corresponding address across multiple chains. Consequently, it enables users to verify an address on a particular chain and send assets on other chains, under the presumption that the same entity controls the associated addresses.

Through the use of smart contracts, having a contract at a certain address on one chain does not inherently mean that an identical contract exists at the same address, under the same entity's control, on another chain. If a user sends assets, presuming they control the same address across different chains, those assets may become inaccessible. Thus, the capability to duplicate the same contract address on other chains is beneficial, guaranteeing access to the other smart contract in case there was a mistake sending assets to another chain.

Similarly, deploying a contract across various chains with an identical address can establish a kind of multi-chain identity. This approach can prove advantageous, particularly with factory, registry, and account-based contracts.

Specification

UniversalFactory Setup

Before the deployment of the UniversalFactory on any network, people should make sure that the [Nick Factory (https://github.com/Arachnid/deterministic-deployment-proxy)](https://github.com/Arachnid/deterministic-deployment-proxy) is deployed on the same network.

Nick Factory Deployment

The Nick Factory should be located at this address 0x4e59b44847b379578588920ca78fbf26c0b4956c on the network. If there is no code on this address, it means that the contract is not deployed yet.

To deploy, the following raw transaction should be broadcasted to the network 0xf8a58085174876e800830186a08080b853604580600e600039806000f350fe7ffffffffffffffffffffffffffffffffffffffffffffff after funding the deployer address: 0x3fab184622dc19b6109349b94811493bf2a45362 with gasPrice (100 gwei) * gasLimit (100000).

Check [Nick's Factory repository (https://github.com/Arachnid/deterministic-deployment-proxy/tree/master)](https://github.com/Arachnid/deterministic-deployment-proxy/tree/master) for more information.

UniversalFactory Deployment

After the deployment of Nick Factory on the network, the UniversalFactory can be reproduced at the standardized address given sending the same salt and bytecode.

In order to create the UniversalFactory contract, one should send a transaction to the [Nick Factory (https://github.com/Arachnid/deterministic-deployment-proxy)](https://github.com/Arachnid/deterministic-deployment-proxy) address with data field equal to salt + bytecode.

The address produced should be equal to 0x160000700D62B8dDC65FaeD5AC5Add2d2e30A803.

UniversalFactory Configuration

Standardized Address

0x160000700D62B8dDC65FaeD5AC5Add2d2e30A803

Standardized Salt

0x2ae1d025b6ff376f9a7934f37e22ad50dcde9c3e7dbdf3397e9df80e224e9e40

Standardized Bytecode

0x6080604052348015610010576000080fd5b50610ede80610020600039600f3fe6080604052600436106100

UniversalFactory Source Code

This is an exact copy of the code of the [LSP16 UniversalFactory smart contract (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP16UniversalFactory/LSP16UniversalFactory.sol)](https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP16UniversalFactory/LSP16UniversalFactory.sol).

- The source code is generated with 0.8.17 compiler version and with 9999999 optimization runs.
- The imported contracts are part of the 4.8.2 version of the @openzeppelin/contracts package.
- Navigate to [lsp-smart-contract (https://github.com/lukso-network/lsp-smart-contracts)](https://github.com/lukso-network/lsp-smart-contracts) repo and checkout to 94c8ae1122c4e405eaa6eb37e35e4917caec2a6a commit to obtain the exact copy of the code, change the compiler settings in hardhat.config.ts and compile to produce the same bytecode.

<details>
<summary>Click to Expand</summary>
<pre>

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

// libraries
import {Create2} from "@openzeppelin/contracts/utils/Create2.sol";
import {Clones} from "@openzeppelin/contracts/proxy/Clones.sol";

// errors

/**
 * @dev Reverts when there is no revert reason bubbled up by the contract created when initializing
 */
error ContractInitializationFailed();

/**
 * @dev Reverts when msg.value sent to {deployCreate2AndInitialize} function is not equal to the sum of the
 * `initializeCalldataMsgValue` and `constructorMsgValue`
 */
error InvalidValueSum();

/**
 * @title LSP16 Universal Factory
 * @dev Factory contract to deploy different types of contracts using the CREATE2 opcode
 * standardized as LSP16 - UniversalFactory:
 * https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-16-UniversalFactory.md
 *
 * The UniversalFactory will be deployed using Nick's Factory
(0x4e59b44847b379578588920ca78fbf26c0b4956c)
 *
 * The deployed address can be found in the LSP16 specification.
```

```
 * Please refer to the LSP16 Specification to obtain the exact bytecode and salt that
 * should be used to produce the address of the UniversalFactory on different chains.
 *
 * This factory contract is designed to deploy contracts at the same address on multiple chains.
 *
 * The UniversalFactory can deploy 2 types of contracts:
 * - non-initializable (normal deployment)
 * - initializable (external call after deployment, e.g: proxy contracts)
 *
 * The `providedSalt` parameter given by the deployer is not used directly as the salt by the CREATE2
opcode.
 * Instead, it is used along with these parameters:
 *  - `initializable` boolean
 *  - `initializeCalldata` (when the contract is initializable and `initializable` is set to `true`).
 * These three parameters are concatenated together and hashed to generate the final salt for CREATE2.
 *
 * See {generateSalt} function for more details.
 *
 * The constructor and `initializeCalldata` SHOULD NOT include any network-specific parameters (e.g:
chain-id,
 * a local token contract address), otherwise the deployed contract will not be recreated at the same address
 * across different networks, thus defeating the purpose of the UniversalFactory.
 *
 * One way to solve this problem is to set an EOA owner in the `initializeCalldata`/constructor
 * that can later call functions that set these parameters as variables in the contract.
 *
 * The UniversalFactory must be deployed at the same address on different chains to successfully deploy
 * contracts at the same address across different chains.
 */
contract LSP16UniversalFactory {
    /**
     * @dev placeholder for the `initializeCallData` param when the `initializable` boolean is set to `false`.
     */
    bytes private constant _EMPTY_BYTE = "";

    /**
     * @dev Emitted whenever a contract is created
     * @param contractCreated The address of the contract created
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     * @param generatedSalt The salt used by the `CREATE2` opcode for contract deployment
     * @param initialized The Boolean that specifies if the contract must be initialized or not
     * @param initializeCalldata The bytes provided as initializeCalldata (Empty string when `initialized` is set
to false)
     */
    event ContractCreated(
        address indexed contractCreated,
        bytes32 indexed providedSalt,
        bytes32 generatedSalt,
        bool indexed initialized,
        bytes initializeCalldata
    );
```

```
    /**
     * @dev Deploys a contract using the CREATE2 opcode. The address where the contract will be deployed
     * can be known in advance via the {computeAddress} function.
     *
     * This function deploys contracts without initialization (external call after deployment).
     *
     * The `providedSalt` parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed with
     * keccak256: `keccak256(abi.encodePacked(false, providedSalt))`. See {generateSalt} function for more details.
     *
     * Using the same `byteCode` and `providedSalt` multiple times will revert, as the contract cannot be deployed
     * twice at the same address.
     *
     * If the constructor of the contract to deploy is payable, value can be sent to this function to fund
     * the created contract. However, sending value to this function while the constructor is not payable will
     * result in a revert.
     *
     * @param byteCode The bytecode of the contract to be deployed
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     *
     * @return The address of the deployed contract
     */
    function deployCreate2(bytes calldata byteCode, bytes32 providedSalt)
        public
        payable
        virtual
        returns (address)
    {
        bytes32 generatedSalt = generateSalt(providedSalt, false, _EMPTY_BYTE);
        address contractCreated = Create2.deploy(msg.value, generatedSalt, byteCode);
        emit ContractCreated(contractCreated, providedSalt, generatedSalt, false, _EMPTY_BYTE);

        return contractCreated;
    }

    /**
     * @dev Deploys a contract using the CREATE2 opcode. The address where the contract will be deployed
     * can be known in advance via the {computeAddress} function.
     *
     * This function deploys contracts with initialization (external call after deployment).
     *
     * The `providedSalt` parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed with
     * keccak256: `keccak256(abi.encodePacked(true, initializeCalldata, providedSalt))`.
     * See {generateSalt} function for more details.
     *
```

```
    * Using the same `byteCode`, `providedSalt` and `initializeCalldata` multiple times will revert, as the
    * contract cannot be deployed twice at the same address.
    *
    * If the constructor or the initialize function of the contract to deploy is payable, value can be sent along
    * with the deployment/initialization to fund the created contract. However, sending value to this function
while
    * the constructor/initialize function is not payable will result in a revert.
    *
    * Will revert if the `msg.value` sent to the function is not equal to the sum of `constructorMsgValue` and
    * `initializeCalldataMsgValue`.
    *
    * @param byteCode The bytecode of the contract to be deployed
    * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
    * that will be used by the `CREATE2` opcode for contract deployment
    * @param initializeCalldata The calldata to be executed on the created contract
    * @param constructorMsgValue The value sent to the contract during deployment
    * @param initializeCalldataMsgValue The value sent to the contract during initialization
    *
    * @return The address of the deployed contract
    */
   function deployCreate2AndInitialize(
       bytes calldata byteCode,
       bytes32 providedSalt,
       bytes calldata initializeCalldata,
       uint256 constructorMsgValue,
       uint256 initializeCalldataMsgValue
   ) public payable virtual returns (address) {
       if (constructorMsgValue + initializeCalldataMsgValue != msg.value) revert InvalidValueSum();

       bytes32 generatedSalt = generateSalt(providedSalt, true, initializeCalldata);
       address contractCreated = Create2.deploy(constructorMsgValue, generatedSalt, byteCode);
       emit ContractCreated(
           contractCreated,
           providedSalt,
           generatedSalt,
           true,
           initializeCalldata
       );

       (bool success, bytes memory returndata) = contractCreated.call{
           value: initializeCalldataMsgValue
       }(initializeCalldata);
       _verifyCallResult(success, returndata);

       return contractCreated;
   }

   /**
    * @dev Deploys an ERC1167 minimal proxy contract using the CREATE2 opcode. The address where
the contract will be deployed
    * can be known in advance via the {computeERC1167Address} function.
    *
```

```
     * This function deploys contracts without initialization (external call after deployment).
     *
     * The `providedSalt` parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed with
     * keccak256: `keccak256(abi.encodePacked(false, providedSalt))`. See {generateSalt} function for more details.
     *
     * See {generateSalt} function for more details.
     *
     * Using the same `implementationContract` and `providedSalt` multiple times will revert, as the contract cannot be deployed
     * twice at the same address.
     *
     * Sending value to the contract created is not possible since the constructor of the ERC1167 minimal proxy is not payable.
     *
     * @param implementationContract The contract address to use as the base implementation behind the proxy that will be deployed
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     *
     * @return The address of the minimal proxy deployed
     */
    function deployERC1167Proxy(address implementationContract, bytes32 providedSalt)
        public
        virtual
        returns (address)
    {
        bytes32 generatedSalt = generateSalt(providedSalt, false, _EMPTY_BYTE);

        address proxy = Clones.cloneDeterministic(implementationContract, generatedSalt);
        emit ContractCreated(proxy, providedSalt, generatedSalt, false, _EMPTY_BYTE);

        return proxy;
    }

    /**
     * @dev Deploys an ERC1167 minimal proxy contract using the CREATE2 opcode. The address where the contract will be deployed
     * can be known in advance via the {computeERC1167Address} function.
     *
     * This function deploys contracts with initialization (external call after deployment).
     *
     * The `providedSalt` parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed with
     * keccak256: `keccak256(abi.encodePacked(true, initializeCalldata, providedSalt))`.
     * See {generateSalt} function for more details.
     *
     * Using the same `implementationContract`, `providedSalt` and `initializeCalldata` multiple times will revert, as the
     * contract cannot be deployed twice at the same address.
     *
```

```
     * If the initialize function of the contract to deploy is payable, value can be sent along to fund the created
     * contract while initializing. However, sending value to this function while the initialize function is not
     * payable will result in a revert.
     *
     * @param implementationContract The contract address to use as the base implementation behind the
proxy that will be deployed
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     * @param initializeCalldata The calldata to be executed on the created contract
     *
     * @return The address of the minimal proxy deployed
     */
    function deployERC1167ProxyAndInitialize(
        address implementationContract,
        bytes32 providedSalt,
        bytes calldata initializeCalldata
    ) public payable virtual returns (address) {
        bytes32 generatedSalt = generateSalt(providedSalt, true, initializeCalldata);

        address proxy = Clones.cloneDeterministic(implementationContract, generatedSalt);
        emit ContractCreated(proxy, providedSalt, generatedSalt, true, initializeCalldata);

        (bool success, bytes memory returndata) = proxy.call{value: msg.value}(initializeCalldata);
        _verifyCallResult(success, returndata);

        return proxy;
    }

    /**
     * @dev Computes the address of a contract to be deployed using CREATE2, based on the input
parameters.
     * Any change in one of these parameters will result in a different address. When the `initializable`
     * boolean is set to `false`, `initializeCalldata` will not affect the function output.
     *
     * @param byteCodeHash The keccak256 hash of the bytecode to be deployed
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     * @param initializable A boolean that indicates whether an external call should be made to initialize the
     * contract after deployment
     * @param initializeCalldata The calldata to be executed on the created contract if `initializable` is set to
`true`
     *
     * @return The address where the contract will be deployed
     */
    function computeAddress(
        bytes32 byteCodeHash,
        bytes32 providedSalt,
        bool initializable,
        bytes calldata initializeCalldata
    ) public view virtual returns (address) {
        bytes32 generatedSalt = generateSalt(providedSalt, initializable, initializeCalldata);
        return Create2.computeAddress(generatedSalt, byteCodeHash);
```

```
    }

    /**
     * @dev Computes the address of an ERC1167 proxy contract based on the input parameters.
     * Any change in one of these parameters will result in a different address. When the `initializable`
     * boolean is set to `false`, `initializeCalldata` will not affect the function output.
     *
     * @param implementationContract The contract to create a clone of according to ERC1167
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     * @param initializable A boolean that indicates whether an external call should be made to initialize the
     * proxy contract after deployment
     * @param initializeCalldata The calldata to be executed on the created contract if `initializable` is set to
`true`
     *
     * @return The address where the ERC1167 proxy contract will be deployed
     */
    function computeERC1167Address(
        address implementationContract,
        bytes32 providedSalt,
        bool initializable,
        bytes calldata initializeCalldata
    ) public view virtual returns (address) {
        bytes32 generatedSalt = generateSalt(providedSalt, initializable, initializeCalldata);
        return Clones.predictDeterministicAddress(implementationContract, generatedSalt);
    }

    /**
     * @dev Generates the salt used to deploy the contract by hashing the following parameters
(concatenated
     * together) with keccak256:
     *
     * - the `providedSalt`
     * - the `initializable` boolean
     * - the `initializeCalldata`, only if the contract is initializable (the `initializable` boolean is set to `true`)
     *
     * The `providedSalt` parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is
     * used along with these parameters:
     *  - `initializable` boolean
     *  - `initializeCalldata` (when the contract is initializable and `initializable` is set to `true`).
     * These three parameters are concatenated together and hashed to generate the final salt for CREATE2.
     *
     * This approach ensures that in order to reproduce an initializable contract at the same address on
another chain,
     * not only the `providedSalt` is required to be the same, but also the initialize parameters within the
`initializeCalldata`
     * must also be the same.
     *
     * This maintains consistent deployment behaviour. Users are required to initialize contracts with the same
parameters across
     * different chains to ensure contracts are deployed at the same address across different chains.
     *
```

```
    * ----------
    * Example (for initializable contracts)
    *
    * For an existing contract A on chain 1 owned by X, to replicate the same contract at the same address with
    * the same owner X on chain 2, the salt used to generate the address should include the initializeCalldata
    * that assigns X as the owner of contract A.
    *
    * For instance, if another user, Y, tries to deploy the contract at the same address
    * on chain 2 using the same providedSalt, but with a different initializeCalldata to make Y the owner instead of X,
    * the generated address would be different, preventing Y from deploying the contract with different ownership
    * at the same address.
    * ----------
    *
    * However, for non-initializable contracts, if the constructor has arguments that specify the deployment behavior, they
    * will be included in the bytecode. Any change in the constructor arguments will lead to a different contract's bytecode
    * which will result in a different address on other chains.
    *
    * ----------
    * Example (for non-initializable contracts)
    *
    * If a contract is deployed with specific constructor arguments on chain 1, these arguments are embedded within the bytecode.
    * For instance, if contract B is deployed with a specific `tokenName` and `tokenSymbol` on chain 1, and a user wants to deploy
    * the same contract with the same `tokenName` and `tokenSymbol` on chain 2, they must use the same constructor arguments to
    * produce the same bytecode. This ensures that the same deployment behaviour is maintained across different chains,
    * as long as the same bytecode is used.
    *
    * If another user Z, tries to deploy the same contract B at the same address on chain 2 using the same `providedSalt`
    * but different constructor arguments (a different `tokenName` and/or `tokenSymbol`), the generated address will be different.
    * This prevents user Z from deploying the contract with different constructor arguments at the same address on chain 2.
    * ----------
    *
    * The providedSalt was hashed to produce the salt used by CREATE2 opcode to prevent users from deploying initializable contracts
    * using non-initializable functions such as {deployCreate2} without having the initialization call.
    *
    * In other words, if the providedSalt was not hashed and was used as it is as the salt by the CREATE2 opcode, malicious users
    * can check the generated salt used for the already deployed initializable contract on chain 1, and deploy the contract
    * from {deployCreate2} function on chain 2, with passing the generated salt of the deployed contract as
```

```
providedSalt
     * that will produce the same address but without the initialization, where the malicious user can initialize after.
     *
     * @param initializable The Boolean that specifies if the contract must be initialized or not
     * @param initializeCalldata The calldata to be executed on the created contract if `initializable` is set to `true`
     * @param providedSalt The salt provided by the deployer, which will be used to generate the final salt
     * that will be used by the `CREATE2` opcode for contract deployment
     *
     * @return The generated salt which will be used for CREATE2 deployment
     */
    function generateSalt(
        bytes32 providedSalt,
        bool initializable,
        bytes memory initializeCalldata
    ) public pure virtual returns (bytes32) {
        if (initializable) {
            return keccak256(abi.encodePacked(true, initializeCalldata, providedSalt));
        } else {
            return keccak256(abi.encodePacked(false, providedSalt));
        }
    }

    /**
     * @dev Verifies that the contract created was initialized correctly.
     * Bubble the revert reason if present, revert with `ContractInitializationFailed` otherwise.
     */
    function _verifyCallResult(bool success, bytes memory returndata) internal pure virtual {
        if (!success) {
            // Look for revert reason and bubble it up if present
            if (returndata.length != 0) {
                // The easiest way to bubble the revert reason is using memory via assembly
                // solhint-disable no-inline-assembly
                /// @solidity memory-safe-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert ContractInitializationFailed();
            }
        }
    }
}
```

</pre>
</details>

Methods

deployCreate2

```
function deployCreate2(bytes calldata byteCode, bytes32 providedSalt) public payable returns (address);
```

Deploys a contract using the CREATE2 opcode without initialization (external call after deployment).

The address where the contract will be deployed can be known in advance via the computeAddress function.

The providedSalt parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed via keccak256 with prepending a false boolean. See generateSalt function for more details.

```
keccak256(abi.encodePacked(false, providedSalt))
```

MUST emit the ContractCreated (./LSP-16-UniversalFactory.md#contractcreated) event after deploying the contract.

Parameters:

- byteCode: The bytecode of the contract to deploy
- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment

Return:

- contractCreated: The address of the contract created.

Requirements:

- If value is associated with the contract creation, the constructor of the contract to deploy MUST be payable, otherwise the call will revert.
- MUST NOT use the same bytecode and providedSalt twice, otherwise the call will revert.

deployCreate2AndInitialize

```
function deployCreate2AndInitialize(bytes calldata byteCode, bytes32 providedSalt, bytes calldata initializeCalldata, uint256 constructorMsgValue, uint256 initializeCalldataMsgValue) public payable returns (address);
```

Deploys a contract using the CREATE2 opcode with initialization (external call after deployment).

The address where the contract will be deployed can be known in advance via the computeAddress function.

The providedSalt parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed via keccak256 with prepending a true boolean and the initializeCalldata parameter. See generateSalt function for more details.

```
keccak256(abi.encodePacked(true, initializeCalldata, providedSalt))
```

MUST emit the ContractCreated (./LSP-16-UniversalFactory.md#contractcreated) event after deploying the contract.

Parameters:

- byteCode: The bytecode of the contract to deploy
- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment
- initializeCalldata: The calldata to be executed on the created contract
- constructorMsgValue: The value sent to the contract during deploymentcontract
- initializeCalldataMsgValue: The value sent to the contract during initialization

Return:

- contractCreated: The address of the contract created.

Requirements:

- If some value is transferred during the contract creation, the constructor of the contract to deploy MUST be payable, otherwise the call will revert.
- If some value is transferred during the initialization call, the initialize function called on the contract to deploy MUST be payable, otherwise the call will revert.
- The sum of constructorMsgValue and initializeCalldataMsgValue MUST be equal to the value associated with the function call.
- MUST NOT use the same bytecode, providedSalt and initializeCalldata twice, otherwise the call will revert.

deployERC1167Proxy

```
function deployERC1167Proxy(address implementationContract, bytes32 providedSalt) public returns (address);
```

Deploys an ERC1167 minimal proxy contract using the CREATE2 opcode.

The address where the contract will be deployed can be known in advance via the computeERC1167Address function.

The providedSalt parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed via keccak256 with prepending a false boolean. See generateSalt function for more details.

```
keccak256(abi.encodePacked(false, providedSalt))
```

MUST emit the ContractCreated (./LSP-16-UniversalFactory.md#contractcreated) event after deploying the contract.

Parameters:

- implementationContract: The contract to create a clone of according to ERC1167
- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment

Return:

- proxy: The address of the contract created

Requirements:

- MUST NOT use the same implementationContract and providedSalt twice, otherwise the call will revert.

deployERC1167ProxyAndInitialize

```
function deployERC1167Proxy(address implementationContract, bytes32 providedSalt, bytes calldata initializeCalldata) public returns (address);
```

Deploys an ERC1167 minimal proxy contract using the CREATE2 opcode with initialization (external call after deployment).

The address where the contract will be deployed can be known in advance via the computeERC1167Address function.

The providedSalt parameter is not used directly as the salt by the CREATE2 opcode. Instead, it is hashed via keccak256 with prepending a true boolean and the initializeCalldata parameter. See generateSalt function for more details.

```
keccak256(abi.encodePacked(true, initializeCalldata, providedSalt))
```

MUST emit the ContractCreated (./LSP-16-UniversalFactory.md#contractcreated) event after deploying the contract.

Parameters:

- implementationContract: The contract to create a clone of according to ERC1167
- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment
- initializeCalldata: The calldata to be executed on the created contract

Return:

- proxy: The address of the contract created

Requirements:

- MUST NOT use the same implementationContract and providedSalt twice, otherwise the call will revert.
- If value is associated with the initialization call, the initialize function called on the contract to deploy MUST be payable, otherwise the call will revert.
- MUST NOT use the same bytecode, providedSalt and initializeCalldata twice, otherwise the call will revert.

computeAddress

```
function computeAddress(bytes32 bytecodeHash, bytes32 providedSalt, bool initializable, bytes calldata initializeCalldata) public view virtual returns (address)
```

Computes the address of a contract to be deployed using CREATE2, based on the input parameters. Any change in one of these parameters will result in a different address.

When the initializable boolean is set to false, initializeCalldata will not affect the function output.

Parameters:

- byteCodeHash: The keccak256 hash of the bytecode to be deployed
- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment
- initializable: A boolean that indicates whether an external call should be made to initialize the contract after deployment
- initializeCalldata: The calldata to be executed on the created contract if initializable is set to true

Return:

- contractToCreate: The address where the contract will be deployed.

computeERC1167Address

```
function computeERC1167Address(address implementationContract, bytes32 providedSalt, bool initializable, bytes calldata initializeCalldata) public view virtual returns (address)
```

Computes the address of a contract to be deployed using CREATE2, based on the input parameters. Any change in one of these parameters will result in a different address.

When the initializable boolean is set to false, initializeCalldata will not affect the function output.

Parameters:

- implementationContract: The contract to create a clone of according to ERC1167
- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment
- initializable: A boolean that indicates whether an external call should be made to initialize the contract after deployment
- initializeCalldata: The calldata to be executed on the created contract if initializable is set to true

Return:

- proxyToCreate: The address where the contract will be deployed.

generateSalt

```
function generateSalt(bytes32 providedSalt, bool initializable, bytes memory initializeCalldata) public view virtual returns (bytes32)
```

Generates the salt used to deploy the contract by hashing the following parameters (concatenated together) with keccak256:

- the providedSalt
- the initializable boolean
- the initializeCalldata, only if the contract is initializable (the initializable boolean is set to true)

This approach ensures that in order to reproduce an initializable contract at the same address on another chain, not only the providedSalt is required to be the same, but also the initialize parameters within the initializeCalldata must also be the same.

This maintains consistent deployment behaviour. Users are required to initialize contracts with the same parameters across different chains to ensure contracts are deployed at the same address across different chains.

Parameters:

- providedSalt: The salt provided by the deployer, which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment
- initializable: A boolean that indicates whether an external call should be made to initialize the contract after deployment
- initializeCalldata: The calldata to be executed on the created contract if initializable is set to true

Return:

- generatedSalt: The generated salt which will be used for CREATE2 deployment

Events

ContractCreated

```
event ContractCreated(address indexed contractCreated, bytes32 indexed providedSalt, bytes32 generatedSalt, bool indexed initializable, bytes initializeCalldata);
```

MUST be emitted when a contract is created using the UniversalFactory contract.

Parameters:

- contractCreated: The address of the contract created
- providedSalt: The salt provided by the deployer which will be used to generate the final salt that will be used by the CREATE2 opcode for contract deployment
- generatedSalt: The salt used by the CREATE2 opcode for contract deployment
- initialized: The Boolean that specifies if the contract must be initialized or not
- initializeCalldata: The bytes provided as initializeCalldata (Empty string when initialized is set to false)

Rationale

The Nick Factory is utilized to deploy the UniversalFactory, taking into consideration that we want to avoid the dependency on a single entity for regular contract deployment. By leveraging the Nick Factory, any user can deploy it, given the same parameters.

Moreover, the Nick Factory (https://github.com/Arachnid/deterministic-deployment-proxy) is chosen due to its widespread deployment and compatibility, even on chains that don't support pre-EIP-155 (./eip-155.md) transactions (the method by which Nick Factory was deployed). This compatibility aligns with the standard's objective, enabling the replication of contract addresses on a broad array of chains.

Minimal proxy (https://eips.ethereum.org/EIPS/eip-1167) contracts can be deployed with the deployCreate2 and deployCreate2AndInitialize functions. However, new functions have been introduced to reduce potential errors during user interaction and to optimize gas costs. The clone library undertakes the responsibility of deploying and generating the minimal proxy bytecode by accepting the base contract's address.

The salt provided during function calls isn't used directly as the salt for the CREATE2 opcode. This is because this contract is designed to tie the creation of contracts to their initial deployment/initialization parameters. This methodology prevents address squatting and ensures that a contract A, owned by EOA A on chain 1, cannot be replicated at the same address where contract A is owned by EOA B on chain 2. Including the initialization data in the salt ensures that contracts maintain consistent deployment behavior and parameters across various chains.

Furthermore, if a contract is not initializable, the provided salt is hashed to prevent it from being used as it is, as a generated salt from a salt and initialization data from another chain. This avoids the scenario where generated salts are used to reproduce the same address of proxies without the initialization/external call effect.

Implementation

An implementation can be found in the [lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP16UniversalFactory/LSP16UniversalFactory.sol)](https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP16UniversalFactory/LSP16UniversalFactory.sol) repository.

Security Consideration

Knowing that deploying a contract using the UniversalFactory will allow to deploy the same contract on other chains with the same address, people should be aware and watch out to use contracts that don't have a logic that protects against replay-attacks.

The constructor parameters or/and initialization data SHOULD NOT include any network-specific parameters (e.g: chain-id, a local token contract address), otherwise the deployed contract will not be recreated at the same address across different networks, thus defeating the purpose of the UniversalFactory.

Copyright

</file>

<file>

path: /LSPs/LSP-17-ContractExtension.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-17-ContractExtension.md

---

lip: 17
title: Contract Extension
author:
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2021-11-19
requires: ERC165

Simple Summary

This standard describes a mechanism for adding additional functionality to a contract after it has been deployed, through the use of extensions.

Abstract

This proposal defines two types of contracts:

- an extendable contract which functionalities are extended.
- an extension contract which contains the functionality that will be added to the extendable contract.

When the extendable contract is called with a function that is not part of its public interface, it can forward this call to an extension contract, through the use of its fallback function.

The extension contract function is able to access the original msg.sender and msg.value of the extendable contract by appending them to the calldata of the call to the extension.

The extendable contract SHOULD map function selectors to extension contract addresses that handle those functions.

Motivation

After deploying a smart contract to the network, it is not possible to add new functionalities to it. This limitation is significant for smart contracts that aim to support a wider range of functionalities, especially those that may be standardized in the future.

Implementing a mechanism for attaching extensions to a specific contract not only makes the contract more extendable and able to support a wider range of functionalities over time, but it also enables the reusability of extensions across multiple contracts. This reduces the need for deploying multiple contracts with the same logic to the blockchain network, thus potentially decreasing chain congestion and reducing gas costs.

LSP17Extendable Specification

ERC165 Interface ID

LSP17-Extendable interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0xa918fa6b.

This bytes4 interface id is calculated as the first 4 bytes of the keccak256 of the word "LSP17Extendable" since there is no public functions available.

Smart contracts that adhere to the LSP17Extendable standard MUST include the supportsInterface(..) function, as specified in the ERC165 (https://eips.ethereum.org/EIPS/eip-165) standard, and MUST support the LSP17Extendable interfaceId.

They SHOULD also check whether the interface being queried is supported within the supportsInterface(..) extension, if it exists.

Behavior

Whenever a function is called on an extendable contract and the function does not exist, the fallback function of the extendable contract MUST call the function on the extension mapped using the CALL opcode.

The calldata MUST be appended with 52 extra bytes as follows:

- The msg.sender calling the extendable contract without any pad, MUST be 20 bytes.
- The msg.value received to the extendable contract, MUST be 32 bytes.

The standard does not enforce a specific method for mapping function selectors to the addresses of the extension contracts, nor does it require specific methods for setting and querying the addresses of the extensions.

As an example, a mapping of function selectors to extension contracts can be used, such as a mapping(bytes4 => address). However, any other data structure can also be used to map function selectors to extension contract addresses.

> Note: the contract implementing the LSP17 Extendable sub-standard does not enforce forwarding the amount of native tokens received (msg.value) to the extension. It is up to the implementation to define this behavior and whether the native tokens received while calling a function that does not exist should be forwarded or not to the extension.

ERC725Y Data Key

If the contract implementing the LSP17 standard is an ERC725Y contract, the extension contracts COULD be stored under the following ERC725Y data key:

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "(address, bool)",
  "valueContent": "(Address, Boolean)"
}
```

The <bytes4> is the functionSelector called on the account contract. For instance, for the selector 0xaabbccdd, the data key above would be constructed as:

```
0xcee78b4094da860110960000aabbccdd0000000000000000000000000000
```

The boolean at the end can be used to specify whether the value sent to the extendable contract should be sent to the extension or stay in the extendable contract.

For example, having this boolean set to true (as a 0x01 hex byte) means that the value received should be forwarded to the extension.

```
                    address
  vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
0xcafecafecafecafecafecafecafecafecafecafe01
                                  ^^
                                  true
```

> See LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the data key, and the Mapping (./LSP-2-ERC725YJSONSchema.md#mapping) section to learn the padding rules.

LSP17Extension Specification

LSP17-Extension interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0xcee78b40.

This bytes4 interface id is calculated as the first 4 bytes of the keccak256 of the word "LSP17Extension" since there is no public functions available.

Smart contracts that adhere to the LSP17Extension standard MUST include the supportsInterface(..) function, as specified in the ERC165 (https://eips.ethereum.org/EIPS/eip-165) standard, and MUST support the LSP17Extension interfaceId.

Overview

Normally, contract functions use msg.sender and msg.value global variables for validation in Solidity. However, when an extendable contract calls an extension using the CALL opcode, the msg.sender on the extension contract will always be the address of the extendable contract.

To access the original msg.sender and msg.value sent to the extendable contract, the extendable contract will append them as extra calldata and extension contract can use the following functions to retrieve them within the extension:

```
function _extendableMsgSender() internal view virtual returns (address) {
    return address(bytes20(msg.data[msg.data.length - 52:msg.data.length - 32]));
}
```

```
function _extendableMsgValue() internal view virtual returns (uint256) {
    return uint256(bytes32(msg.data[msg.data.length - 32:]));
}
```

The original calldata sent to the extendable contract can be retrieved using this function:

```
function _extendableMsgData() internal view virtual returns (bytes memory) {
    return msg.data[:msg.data.length - 52];
}
```

If a validation mechanism exists in the extension contract, it should depend on the _extendableMsgSender(), _extendableMsgValue() or _extendableMsgData() functions instead of the msg.sender and msg.value global variables, as anyone can call the extendable contract and trigger a call to the extensions.

Security Considerations

A function selector clash can occur when two different function signatures hash to the same four-byte hash. Users need to take extra care to avoid adding functions that map to a function selector already existing.

Rationale

The design of this standard was inspired by EIP-2535 Diamonds, Multi-Facet Proxy (https://eips.ethereum.org/EIPS/eip-2535), which also proposes a way to add functionality to a smart contract through the use of facets (extensions). However, EIP-2535 uses the DELEGATECALL opcode to execute the function on the extension contract, allowing it to access the storage of the extendable contract.

This proposal, on the other hand, uses the CALL opcode to forward the function call to the extension contract. This design decision was made to enhance security by mitigating the risk of malicious actors exploiting the selfdestruct functionality or altering the storage of the extendable contract.

By appending the msg.sender and msg.value to the calldata when forwarding the function call from the extendable to the extension contract, this proposal allows the extension contract to know the address of the caller and the value associated with the call.

This design decision ensures that the extension contract can maintain context of the original call and allows the extension contract to make proper decisions based on the call context.

Implementation

An implementation can be found in the lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP17ContractExtension/) repository.

Copyright

</file>

<file>

path: /LSPs/LSP-2-ERC725YJSONSchema.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md

---

lip: 2
title: ERC725Y JSON Schema
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2020-07-01
requires: ERC725Y

Table of Content

Simple Summary

This schema defines how a single ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) data key-value pair can be described. It can be used as an abstract structure over the storage of an ERC725Y smart contract.

Abstract

ERC725Y enables storing any data in a smart contract as bytes32 => bytes data key-value pairs.

Although this improves interaction with the data stored, it remains difficult to understand the layout of the contract storage. This is because both the data key and the value are addressed in raw bytes.

This schema allows to standardize those data keys and values so that they can be more easily accessed and interpreted. It can be used to create ERC725Y sub-standards, made of pre-defined sets of ERC725Y data keys.

Motivation

A schema defines a blueprint for how a data store is constructed.

In the context of smart contracts, it can offer a better view of how the data is organised and structured within the contract storage.

Using a standardised schema over ERC725Y enables those data keys and values to be easily readable and automatically parsable. Contracts and interfaces can know how to read and interact with the storage of an ERC725Y smart contract.

The advantage of such schema is to allow interfaces or smart contracts to better decode (read, parse and interpret) the data stored in an ERC725Y contract. It is less error-prone due to knowing data types upfront. On the other hand, it also enables interfaces and contracts to know how to correctly encode data, before being set on an ERC725Y contract.

This schema is for example used in ERC725 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) based smart contracts like LSP3-Profile-Metadata (./LSP-3-Profile-Metadata.md#implementation) and LSP4-DigitalAsset-Metadata (./LSP-4-DigitalAsset-Metadata.md#implementation).

Specification

> Note: described sets might not yet be complete, as they could be extended over time.

To make ERC725Y data keys readable, we describe a data key-value pair as a JSON object containing the following entries:

```
{
  "name": "...",
  "key": "...",
  "keyType": "...",
  "valueType": "...",
  "valueContent": "..."
}
```

The table below describes each entries with their available options.

Title	Description
name	the name of the data key
key	the unique identifier of the data key
keyType	*How* the data key must be treated <hr> Singleton Array Mapping MappingWithGrouping
valueType	*How* a value MUST be decoded <hr> bool string address uintN intN bytesN bytes uintN[] intN[] string[] address[] bytes[] bytes[CompactBytesArray] bytesN[CompactBytesArray] Tuple: (valueType1,valueType2,...)
valueContent	*How* a value SHOULD be interpreted <hr> Boolean String Address Number BytesN Bytes Keccak256 BitArray URL AssetURL JSONURL Markdown Literal (*e.g.:* 0x1345ABCD...)

Data Key Name

The name is the human-readable format of an ERC725Y data key. It's the basis which is used to generate the 32 bytes key hash. Names can be arbitrarily chosen, but SHOULD highlight the meaning of content behind the data value.

In scenarios where an ERC725Y data key is part of an LSP Standard, the data key name SHOULD be comprised of the following: LSP{N}{KeyName}, where

- LSP: abbreviation for LUKSO Standards Proposal.
- N: the Standard Number this data key refers to.
- KeyName: base of the data key name. Should represent the meaning of a value stored behind the data key.

e.g.: MyCustomKeyName or LSP4TokenName

Data Key Hash

The key is a bytes32 value that acts as the unique identifier for the data key, and is what is used to retrive the data value from a ERC725Y smart contract via ERC725Y.getData(bytes32 dataKey) or ERC725Y.getData(bytes32[] dataKeys).

Usually keccak256 hashing algorithm is used to generate the bytes32 data key. However, *how* the data key is constructed varies, depending on the keyType.

Key Type

The keyType determines the format of the data key(s).

keyType	Description	Example
Singleton	A simple data key	bytes32(keccak256("MyKeyName")) --- MyKeyName --> 0x35e6950bc8d21a1699e58328a3c4066df5803bb0b570d0150cb3819288
Array	An array spanning multiple ERC725Y data keys	bytes32(keccak256("MyKeyName[]")) --- MyKeyName[] --> 0x24f6297f3abd5a8b82f1a48cee167cdecef40aa98fbf14534ea3539f66ca8
Mapping	A data key that consist of 2 sections, where the last section can also be a dynamic value	bytes10(keccak256("MyKeyName")) + bytes2(0) + bytes20(keccak256("MyMapName") or <mixed type>) --- MyKeyName:MyMapName --> 0x35e6950bc8d21a1699e5000075060e3cd7d40450e94d415fb5992ced9a
MappingWithGrouping	A data key that consist of 3 sections, where the last two sections can also be dynamic values	bytes6(keccak256("MyKeyName")) + bytes4(keccak256("MyMapNam <mixed type>) + bytes2(0) + bytes20(keccak256("MySubMapNam <mixed type>) --- MyKeyName:MyMapName:<address> --> 0x35e6950bc8d275060e3c0000cafecafecafecafecafecafecafecafecaf

Value Type

Describes the underlying data type(s) of a value stored under a specific ERC725Y data key. It refers to the type for the smart contract language like Solidity (https://docs.soliditylang.org).

The valueType is relevant for interfaces to know how a value MUST be encoded/decoded. This include:

- how to decode a value fetched via ERC725Y.getData(...)
- how to encode a value that needs to be set via ERC725Y.setData(...).

The valueType can also be useful for typecasting. It enables contracts or interfaces to know how to manipulate the data and the limitations behind its type. To illustrate, an interface could know that it cannot set the value to 300 if its valueType is uint8 (max uint8 allowed = 255).

valueType	Description
bool	a value as either true or false
string	an UTF8 encoded string
address	a 20 bytes long address

uintN	an unsigned integer (= only positive number) of size N
bytesN	a bytes value of fixed-size N, from bytes1 up to bytes32
bytes	a bytes value of dynamic-size
uintN[]	an array of signed integers
string[]	an array of UTF8 encoded strings
address[]	an array of addresses
bytes[]	an array of dynamic size bytes
bytesN[]	an array of fixed size bytes
bytes[CompactBytesArray]	a compacted bytes array of dynamic size bytes
bytesN[CompactBytesArray]	a compacted bytes array of fixed size bytes
Tuple: (valueType1,valueType2,...)	a tuple of valueTypes

Value Content

The valueContent of a LSP2 Schema describes how to interpret the content of the returned *decoded* value.

Knowing how to interpret the data retrieved under a data key is is the first step in understanding how to handle it. Interfaces can use the valueContent to adapt their behaviour or know how to display data fetched from an ERC725Y smart contract.

As an example, a string could be interpreted in multiple ways, such as:

- a single word, or a sequence of words (*e.g.: "My Custom Token Name"*)
- an URL (*e.g.: "ipfs://QmW4nUNy3vtvr3DxZHuLfSLnhzKMe2WmgsUsEGPPFh8Ztp"*)

Using the following two LSP2 schemas as examples:

```
{
  "name": "MyProfileDescription",
  "key": "0xd0f1819a38d741fce6a6b74406251c521768033029cd254f0f5cd29ca58f3390",
  "keyType": "Singleton",
  "valueType": "string",
  "valueContent": "String"
},
{
  "name": "MyWebsite",
  "key": "0x449560072375b299bab5a695ea268c32c52d4820e4458e5f02f308c588e6715a",
  "keyType": "Singleton",
  "valueType": "string",
  "valueContent": "URL"
}
```

An interface could decode both values retrieved under these data keys as string, but:

- display the profile description as plain text.
- display the website URL as an external link.

Valid valueContent are:

valueContent	Description
Boolean	a boolean value (true or false)
String	an UTF8 encoded string

Address	an address
Number	a Number (positive or negative, depending on the keyType)
BytesN	a bytes value of fixed-size N, from bytes1 up to bytes32
Bytes	a bytes value of dynamic-size
Keccak256	a 32 bytes long hash digest, obtained from the keccak256 hashing algorithm
BitArray	an array of single 1 or 0 bits
URL	an URL encoded as an UTF8 string
AssetURL	The content contains the hash function, hash and link to the asset file
JSONURL	hash function, hash and link to the JSON file
Markdown	a structured Markdown mostly encoded as UTF8 string
0x1345ABCD...	a literal value, when the returned value is expected to equal some specific bytes

The valueContent field can also define a tuple of value contents (for instance, when the valueType is a tuple of types, as described above). In this case, each value content MUST be defined between parentheses. For instance: (Bytes4,Number).

This is useful for decoding tools, to know how to interpret each value type in the tuple.

valueContent when valueType or keyType is an array

In the case where:

a) the keyType is an Array.
b) *or* the valueType is an array [] (compacted or not).

the valueContent describes how to interpret each entry in the array, not the whole array itself. Therefore the valueContent field MUST NOT include [].

We can use the LSP2 Schema below as an example to better understand. This LSP2 Schema below defines a data key that represents a list of social media profiles related to a user.

Reading the ERC725Y storage using this data key will return an array of abi-encoded string[]. Therefore the interface should use the valueType to decode the retrieved value. The valueContent however defines that each string in the array must be interpreted as a social media URL.

```
{
  "name": "MySocialMediaProfiles",
  "key": "0x161761c54f6b013a4b4cbb1247f703c94ae5dfe32081554ad861781f48d47513",
  "keyType": "Singleton",
  "valueType": "string[]",
  "valueContent": "URL"
}
```

keyType

Singleton

A Singleton data key refers to a simple data key. It is constructed using bytes32(keccak256("KeyName")),

Below is an example of a Singleton data key type:

```
{
  "name": "MyKeyName",
  "key": "0x35e6950bc8d21a1699e58328a3c4066df5803bb0b570d0150cb3819288e764b2",
  "keyType": "Singleton",
  "valueType": "...",
  "valueContent": "..."
}
```

keccak256("MyKeyName") =
0x35e6950bc8d21a1699e58328a3c4066df5803bb0b570d0150cb3819288e764b2

Array

An array of elements, where each element has the same valueType.

> *The advantage of the keyType Array over a standard array of elements like address[], is that the amount of elements that can be stored is unlimited.*
> *Storing an encoded array as a value, will require a set amount of gas, which can exceed the block gas limit.*

Requirements:

A data key of Array type MUST have the following requirements:

- The name of the data key MUST have a [] (square brackets) at the end.
- The key itself MUST be the keccak256 hash digest of the full data key name, including the square brackets []
- The value stored under the full data key hash MUST contain the total number of elements (= array length). It MUST be updated every time a new element is added or removed to/from the array.
- The value stored under the full data key hash MUST be stored as uint128 (16 bytes long, padded left with leading zeros).

Construction:

For the Array keyType, the initial key contains the total number of elements stored in the Array (= array length). It is constructed using bytes32(keccak256(KeyName)).

Each Array element can be accessed through its own key. The key of an Array element consists of bytes16(keccak256(KeyName)) + bytes16(uint128(ArrayElementIndex)), where:

- bytes16(keccak256(KeyName)) = The first 16 bytes are the keccak256 hash of the full Array data key name (including the []) (e.g.: LSP12IssuedAssets[])
- bytes16(uint128(ArrayElementIndex)) = the position (= index) of the element in the array (NB: elements index access start at 0)

> Note: an ERC725Y data key of keyType Array can contain up to max(uint128) elements. This is because:
>
> - the value stored under the Array length data key,
> - the index part of an Array index data key,
>
> are both 16 bytes long, which is equivalent to a uint128.

example:

Below is an example for the Array data key named LSP12IssuedAssets[].

- total number of elements:
    - key: 0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd,
    - value: 0x00000000000000000000000000000002 (2 elements)
- element 1: key: 0x7c8c3416d6cda87cd42c71ea1843df2800000000000000000000000000000000, value: 0x123... (index 0)
- element 2: key: 0x7c8c3416d6cda87cd42c71ea1843df2800000000000000000000000000000001, value: 0x321... (index 1)
  ...

```
{
  "name": "LSP12IssuedAssets[]",
  "key": "0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd",
  "keyType": "Array",
  "valueType": "address", // describes the type of each element
  "valueContent": "Address" // describes the value of each element
}
```

```
key: keccak256('LSP12IssuedAssets[]') =
0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd
value: uint128 (array length) e.g. 0x00000000000000000000000000000002

// array items

// 1st element (index 0)
key: 0x7c8c3416d6cda87cd42c71ea1843df2800000000000000000000000000000000
value: 0xcafecafecafecafecafecafecafecafecafecafe

// 2nd element (index 1)
key: 0x7c8c3416d6cda87cd42c71ea1843df2800000000000000000000000000000001
value: 0xcafecafecafecafecafecafecafecafecafecafe
```

Mapping

A Mapping data key is constructed using:

bytes10(keccak256("MyKeyName")) + bytes2(0) + bytes20(keccak256("MyMapName") or <mixed type>).

<mixed type> can be one of uint<M>, address, bool or bytes<M> types.

- uint<M>, bool will be left padded and left-cut, if larger than 20 bytes.
- bytes<M> and address and static word hashes (bytes32) will be left padded, but right-cut, if larger than 20 bytes.

example:

```
// Examples:
MyKeyName:MyMapName //
0x35e6950bc8d21a1699e58328a3c4066df5803bb0b570d0150cb3819288e764b2 +
0x75060e3cd7d40450e94d415fb5992ced9ad8f058649e805951f558364152f9ed
"0x35e6950bc8d21a1699e5000075060e3cd7d40450e94d415fb5992ced9ad8f058"

MyKeyName:<address> // 0xcafecafecafecafecafecafecafecafecafecafe
"0x35e6950bc8d21a1699e50000cafecafecafecafecafecafecafecafecafe"

MyKeyName:<uint32> // 4081242941
"0x35e6950bc8d21a1699e50000000000000000000000000000000f342d33d"


MyKeyName:<bytes4> // 0xabcd1234
"0x35e6950bc8d21a1699e50000000000000000000000000000000abcd1234"

MyKeyName:<bytes32> // 0xaaaabbbbccccddddeeeeffff11112222333344445555666677778888999aaaa
"0x35e6950bc8d21a1699e50000aaaabbbbccccddddeeeeffff1111222233334444"

MyKeyName:<bool> // true
"0x35e6950bc8d21a1699e50000000000000000000000000000000000000000001"


// ERC725Y JSON schema
{
  "name": "FirstWord:<bytes4>",
  "key": "0xf49648de3734d6c545820000<bytes4>",
  "keyType": "Mapping",
  "valueType": "...",
  "valueContent": "..."
}
```

MappingWithGrouping

A MappingWithGrouping data key is constructed using:

bytes6(keccak256("MyKeyName")) + bytes4(keccak256("MyMapName") or <mixed type>) + bytes2(0) + bytes20(keccak256("MySubMapName") or <mixed type>).

<mixed type> can be one of uint<M>, address, bool or bytes<M> types.

- uint<M>, bool will be left padded and left-cut, if it's larger than the max bytes of that section.
- bytes<M> and address and static word hashes (bytes32) will be left padded, but right-cut, if it's larger than the max bytes of that section.

e.g. AddressPermissions:Permissions:<address> > 0x4b80742de2bf 82acb363 0000 cafecafecafecafecafecafecafecafecafecafe.

example:

```
// Examples:
MyKeyName:MyMapName:MySubMapName //
0x35e6950bc8d21a1699e58328a3c4066df5803bb0b570d0150cb3819288e764b2 +
0x75060e3cd7d40450e94d415fb5992ced9ad8f058649e805951f558364152f9ed +
0x221cba00b07da22c3775601ffea5d3406df100dbb7b1c86cb2fe3739f0fe79a1
"0x35e6950bc8d275060e3c0000221cba00b07da22c3775601ffea5d3406df100db"

MyKeyName:MyMapName:<address>
"0x35e6950bc8d275060e3c0000cafecafecafecafecafecafecafecafecafecafe"

// For more examples static examples see the "Mapping" examples

MyKeyName:<bytes2>:<uint32> // ffff 4081242941
"0x35e6950bc8d20000ffff000000000000000000000000000000000f342d33d"

MyKeyName:<address>:<address> // 0xabcdef11abcdef11abcdef11abcdef11ffffffff,
0xcafecafecafecafecafecafecafecafecafecafe
"0x35e6950bc8d2abcdef110000cafecafecafecafecafecafecafecafecafecafe"

MyKeyName:MyMapName:<bytes32> //
0xaaaabbbbccccddddeeeeffff11112222333344445555666677778888999aaaa
"0x35e6950bc8d275060e3c0000aaaabbbbccccddddeeeeffff1111222233334444"

MyKeyName:<bytes32>:<bool> //
0xaaaabbbbccccddddeeeeffff11112222333344445555666677778888999aaaa
"0x35e6950bc8d2aaaabbbb0000000000000000000000000000000000000000000001"


// ERC725Y JSON schema
{
  "name": "AddressPermissions:Permissions:<address>",
  "key": "0x4b80742de2bf82acb3630000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "...",
  "valueContent": "..."
}
```

ValueType

bytes[CompactBytesArray]

A bytes[CompactBytesArray] represents an array of bytes values *encoded in a compact way*. The elements contained in the array are bytes values with different dynamic lengths.

In a compact bytes array of bytes, each element is prefixed with 2 bytes to specify its length.

For instance, 0xaabbccdd in a bytes[CompactBytesArray] is encoded as 0x0004aabbccdd, where:

- 0x0004 = 4 represents the total number of bytes in 0xaabbccdd.
- 0xaabbccdd is the actual value of the element.

> Note: the maximum length of each element is 65535, because two bytes (equivalent to a uint16) are used to store the length of each element and the maximum value of a uint16 is 65535.

example

If we want to have the following bytes as elements in the compacted bytes array:

```
[
  0xaabbccdd,                  // element 1 length is 4 in hex:   0x04
  0xcafecafecafecafecafecafecafe, // element 2 length is 14 in hex:  0x0E
  0xff                         // element 3 length is 1 in hex:   0x01
]
```

The representation of these dynamic elements in a compacted bytes array would be:

0x0004 aabbccdd 000e cafecafecafecafecafecafecafe 0001 ff >
0x0004aabbccdd000ecafecafecafecafecafecafecafe0001ff

bytesN[CompactBytesArray]

Like a bytes[CompactBytesArray] a bytesN[CompactBytesArray] represents an array of bytesN values *encoded in a compact way*. The difference is that all the elements contained in the array have the same length N.

In a compact bytes array of bytesN, each element is prefixed with 1 byte that specify the length N.

For instance, in a bytes8[CompactBytesArray] an entry like 0x1122334455667788 is encoded as 0x00081122334455667788, where:

- 0x0008 = 8 to represent that 0x1122334455667788 contains 8 bytes.
- 0x1122334455667788 is the actual value of the element.

> Note: because two bytes are used to store the length of each element, the maximum N length allowed is 65535 (two bytes are equivalent to the maximum value of a uint16 is 65535)

example:

If we want to have the following bytes8 elements encoded as a bytes8[CompactBytesArray]:

```
[
  0x1122334455667788,
  0xcafecafecafecafe,
  0xbeefbeefbeefbeef
]
```

We will obtain the following:

0x0008 1122334455667788 0008 cafecafecafecafe 0008 beefbeefbeefbeef >
0x0008112233445566778800 08cafecafecafecafe0008beefbeefbeefbeef.

Where each byte 0x0008 in the final encoded value represents the length N of each element.

```
 vvvv            vvvv            vvvv
0x00081122334455667788000 8cafecafecafecafe0008beefbeefbeefbeef
```

Tuples of valueType

The valueType can also be a tuple of types. In this case, the value stored under the ERC725Y data key is a mixture of multiple values concatenated together (the values are just *"glued together"*).

Tuples of valueTypes offer a convenient way to store more than one information under a single ERC725Y data key. In the example below, the value below can be represented as a tuple to store the address of a smart contract (0xcafecafecafecafecafecafecafecafecafecafe) + its interfaceID (0xbeefbeef), all under one single data key.

```
(address,bytes4)
0xcafecafecafecafecafecafecafecafecafecafebeefbeef
```

The main purpose why tuples of valueTypes exist in LSP2 is because it offers a way to store more than one information under a data key. For instance (address,bytes4) is a useful tuple to

LSP2 tuples are different than Solidity tuples. In Solidity, values defined in the tuple are padded. In the case of LSP2 they are not.

In the case of tuple of valueTypes, the types MUST be defined between parentheses, comma separated without parentheses.

```
(valueType1,valueType2,valueType3,...)
```

example 1:

For a schema that includes the following tuple as valueType.

```
{
  "name": "...",
  "key": "...",
  "keyType": "...",
  "valueType": "(bytes4,bytes8)",
  "valueContent": "..."
}
```

And the following values:

- bytes4 value = 0xcafecafe
- bytes8 value = 0xbeefbeefbeefbeef

The tuple of valueType MUST be encoded as:

```
0xcafecafebeefbeefbeefbeef
```

example 2:

For a schema that includes the following tuple as valueType.

```
{
  "name": "...",
  "key": "...",
  "keyType": "...",
  "valueType": "(bytes8,address)",
  "valueContent": "..."
}
```

And the following values:

- bytes4 value = 0xca11ab1eca11ab1e
- bytes8 value = 0x95222290DD7278Aa3Ddd389Cc1E1d165CC4BAfe5

The tuple of valueType MUST be encoded as:

```
0xca11ab1eca11ab1e95222290DD7278Aa3Ddd389Cc1E1d165CC4BAfe5
```

example 3:

For a schema that includes the following tuple as valueType.

```
{
  "name": "...",
  "key": "...",
  "keyType": "...",
  "valueType": "(address,uint128,bytes4,bool,bytes)",
  "valueContent": "..."
}
```

And the following values:

- address value = 0x388C818CA8B9251b393131C08a736A67ccB19297
- uint128 value = the number 5,918 (0x0000000000000000000000000000171E in hex)
- bytes4 value = 0xf00df00d
- bool value = true (0x01 in hex)
- bytes value = 0xcafecafecafecafecafecafecafe

```
0x388C818CA8B9251b393131C08a736A67ccB19297000000000000000000000000000171Ef00df00d01cafecafecafecafecafecafecafe
```

ValueContent

BitArray

A BitArray describes an array that contains a sequence of bits (1s and 0s).

Each bit can be either set (1) or not (0). The point of the BitArray valueContent is that there are only two possible values, so they can be stored in one bit.

A BitArray can be used as a mapping of values to states (on/off, allowed/disallowed, locked/unlocked, valid/invalid), where the max number of available values that can be mapped is *n* bits.

example:

The example shows how a BitArray value can be read and interpreted.

```
{
  "name": "MyPermissions",
  "key": "0xaacedf1d8b2cc85524a881760315208fb03c6c26538760922d6b9dee915fd66a",
  "keyType": "Singleton",
  "valueType": "bytes1",
  "valueContent": "BitArray"
}
```

As the data key name suggests, it defines a list of (user-defined) permissions, where each permission maps to a single bit at position n.

- When a bit at position n is set (1), the permission defined at position n will be set.
- When a bit at position n is not set (0), the permission defined at position n will not be set.

Since the valueType is of type bytes1, this data key can hold 8 user-defined permissions.

For instance, for the following permissions:

SIGN	TRANSFER VALUE	DEPLOY	DELEGATE	CALL	STATIC CALL	CALL SET DATA	CHANGE OWNER
0 / 1	0 / 1	0 / 1	0 / 1	0 / 1	0 / 1	0 / 1	0 / 1

Setting only the permission SET DATA will result in the following bytes1 value (and its binary representation)

```
> Permission SET DATA = permissions set to 0000 0010
`0x02` (4 in decimal)
```

SIGN	TRANSFER VALUE	DEPLOY	DELEGATE	CALL	STATIC CALL	CALL SET DATA	CHANGE OWNER
0	0	0	0	0	0	1	0

Setting multiple permissions like TRANSFER VALUE + CALL + SET DATA will result in the following bytes1 value (and its binary representation)

```
> Permissions set to 0100 0110
`0x46` (70 in decimal)
```

SIGN	TRANSFER VALUE	DEPLOY	DELEGATE	CALL	STATIC CALL	CALL SET DATA	CHANGE OWNER
0	1	0	0	0	1	1	0

The idea is to always read the value of a BitArray data key as binary digits, while its content is always written as a bytes1 (in hex) in the ERC725Y contract storage.

AssetURL

The content is bytes containing the following format:
bytes4(keccack256('hashFunction')) + bytes32(keccack256(assetBytes)) + utf8ToHex('AssetURL')

Known hash functions:

- 0x8019f9b1: keccak256('keccak256(bytes)')

example:

The following shows an example of how to encode an AssetURL:

```
const hashFunction = web3.utils.keccak256('keccak256(bytes)').substr(0, 10)
> '0x8019f9b1'

// Local file read
let hash = web3.utils.keccak256(fs.readFileSync('./file.png'))
> '0xd47cf10786205bb08ce508e91c424d413d0f6c48e24dbfde2920d16a9561a723'

// or browser fetch
const assetBuffer = await
fetch('https://ipfs.lukso.network/ipfs/QmW4nUNy3vtvr3DxZHuLfSLnhzKMe2WmgsUsEGPPFh8Ztp').then(as
ync (response) => {
    return response.arrayBuffer().then((buffer) => new Uint8Array(buffer));
  });

hash = web3.utils.keccak256(assetBuffer)
> '0xd47cf10786205bb08ce508e91c424d413d0f6c48e24dbfde2920d16a9561a723'

// store the asset file anywhere and encode the URL
const url = web3.utils.utf8ToHex('ipfs://QmW4nUNy3vtvr3DxZHuLfSLnhzKMe2WmgsUsEGPPFh8Ztp')
>
'0x697066733a2f2f516d57346e554e7933767476723344785a48754c66534c6e687a4b4d6532576d6773557
3454750504668385a7470'

// final result (to be stored on chain)
const AssetURL = hashFunction + hash.substring(2) + url.substring(2)
                 ^           ^               ^
                 0x8019f9b1   + d47cf10786205bb0... + 697066733a2f2...

// structure of the AssetURL
0x8019f9b1 +      d47cf10786205bb08ce508e91c424d413d0f6c48e24dbfde2920d16a9561a723 +
697066733a2f2f516d57346e554e7933767476723344785a48754c66534c6e687a4b4d6532576d677355734
54750504668385a7470
^               ^                                                  ^
keccak256(utf8)    hash                                                encoded URL

// example value
0x8019f9b1d47cf10786205bb08ce508e91c424d413d0f6c48e24dbfde2920d16a9561a723697066733a2f2f5
16d57346e554e7933767476723344785a48754c66534c6e687a4b4d6532576d6773557345475050466838
5a7470
```

JSONURL

The content is bytes containing the following format:
bytes4(keccak256('hashFunction')) + bytes32(keccak256(JSON.stringify(JSON))) + utf8ToHex('JSONURL')

Known hash functions:

- 0x6f357c6a: keccak256('keccak256(utf8)')

example:

The following shows an example of how to encode a JSON object:

```
const json = JSON.stringify({
    myProperty: 'is a string',
    anotherProperty: {
        sdfsdf: 123456
    }
})

const hashFunction = web3.utils.keccak256('keccak256(utf8)').substr(0, 10)
> '0x6f357c6a'

const hash = web3.utils.keccak256(json)
> '0x820464ddfac1bec070cc14a8daf04129871d458f2ca94368aae8391311af6361'

// store the JSON anywhere and encode the URL
const url = web3.utils.utf8ToHex('ifps://QmYr1VJLwerg6pEoscdhVGugo39pa6rycEZLjtRPDfW84UAx')
>
'0x696670733a2f2f516d597231564a4c776572673670456f73636468564775676f3339706136727963455a4c
6a7452504466573834554178'

// final result (to be stored on chain)
const JSONURL = hashFunction + hash.substring(2) + url.substring(2)
                ^            ^                 ^
                0x6f357c6a   + 820464ddfac1be... + 696670733a2f2...

// structure of the JSONURL
0x6f357c6a +       820464ddfac1bec070cc14a8daf04129871d458f2ca94368aae8391311af6361 +
696670733a2f2f516d597231564a4c776572673670456f73636468564775676f3339706136727963455a4c6a
74525044665738345541 78
^                ^                                                  ^
keccak256(utf8)    hash                                                    encoded URL

// example value
0x6f357c6a820464ddfac1bec070cc14a8daf04129871d458f2ca94368aae8391311af6361696670733a2f2f51
6d597231564a4c776572673670456f73636468564775676f3339706136727963455a4c6a745250446657383
4554178
```

To decode, reverse the process:

```
const data = myContract.methods.getData('0xsomeKey..').call()
>
'0x6f357c6a820464ddfac1bec070cc14a8daf04129871d458f2ca94368aae8391311af6361696670733a2f2f516d597231564a4c776572673670456f736364685647756 76f33397061367279634 55a4c6a745250446657383 4554178'

// slice the bytes to get its pieces
const hashFunction = data.slice(0, 10)
const hash = '0x' + data.slice(0, 74)
const url = '0x' + data.slice(74)

// check if it uses keccak256
if(hashFunction === '0x6f357c6a') {
    // download the json file
    const json = await ipfsMini.catJSON(
        web3.utils.hexToUtf8(url).replace('ipfs://','')
    );

    // compare hashes
    if(web3.utils.keccak256(JSON.stringify(json)) === hash)
        return
            ? json
            : false
}
```

Rationale

The structure of the data key value layout as JSON allows interfaces to auto decode these data key values as they will know how to decode them.

Implementation

Below is an example of an ERC725Y JSON Schema containing 3 x ERC725Y data keys.

Using such schema allows interfaces to auto decode and interpret the values retrieved from the ERC725Y data key-value store.

```
[
 {
  "name": "SupportedStandards:LSP3Profile",
  "key": "0xeafec4d89fa9619884b600005ef83ad9559033e6e941db7d7c495acdce616347",
  "keyType": "Mapping",
  "valueType": "bytes4",
  "valueContent": "0xabe425d6"
 },
 {
  "name": "LSP3Profile",
  "key": "0x5ef83ad9559033e6e941db7d7c495acdce616347d28e90c7ce47cbfcfcad3bc5",
  "keyType": "Singleton",
  "valueType": "bytes",
  "valueContent": "JSONURL"
 },
 {
  "name": "LSP12IssuedAssets[]",
  "key": "0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
 }
]
```


Copyright

</file>

<file>

path: /LSPs/LSP-20-CallVerification.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-20-CallVerification.md

---

lip: 20
title: Call Verification
author: skimaHarvey
discussions-to:
status: Draft
type: LSP
created: 2023-03-15
requires:

Simple Summary

This standard introduces a mechanism for delegating the verification of a function call to another contract.

Abstract

The Call Verification standard introduces a way for a smart contract to delegate the conditions or requirements needed to call a specific function to another smart contract.

This approach offers increased flexibility, where the call requirements can be checked before or/and after the execution of the function being called on another contract.

Motivation

In certain situations, a smart contract may need to modify the conditions or requirements for calling a specific function. These requirements might be complex or subject to change, making them difficult to manage within the same contract.

Delegating the function call requirements to another smart contract enables a more dynamic and adaptable approach. This makes it easier to update, modify, or enhance the requirements without affecting the primary contract's functionality. The Call Verification standard aims to provide a solution that allows contracts to be more versatile and adaptable in response to changing conditions or requirements.

Specification

There are two distinct contracts involved in the LSP20 Call Verification standard, each playing a different role in achieving the desired functionality:

- the contract that receives the initial function call (LSP20-CallVerification) and asks the verifier contract for verification.
- the verifier contract (LSP20-CallVerifier) that handles the actual verification process.

The contract receiving the function call and calling a verifier contract MUST support the LSP20-CallVerification interfaceId: 0x1a0eb6a5, calculated as the first 4 bytes of the keccak256 hash of the string "LSP20CallVerification". This interface ensures that the required behavior is available for performing the necessary verifications.

The verifier contract receiving the verification call MUST support the LSP20-CallVerifier interfaceId: 0x0d6ecac7, calculated as the XOR of the functions mentioned below.

Methods

Smart contracts implementing the LSP20-CallVerifier interfaceId SHOULD implement both of the functions listed below:

lsp20VerifyCall

```
function lsp20VerifyCall(address requestor, address target, address caller, uint256 value, bytes memory callData) external returns (bytes4 returnedStatus);
```

This function is the pre-verification function.

It can be used to run any form of verification mechanism prior to running the actual function being called.

Parameters:

- requestor: The address that requested to make the call to target.
- target: The address of the contract that implements the LSP20CallVerification interface.
- caller: The address who called the function on the contract delegating the verification mechanism.
- value: The value sent by the caller to the function called on the contract delegating the verification

mechanism.
- receivedCalldata: The calldata sent by the caller to the contract delegating the verification mechanism.

Returns:

- returnedStatus: the status determining if the verification succeeded or not.

Requirements

- the bytes4 success value returned MUST be of the following format:
    - the first 3 bytes MUST be the lsp20VerifyCall(address,address,address,uint256,bytes) function selector. This determines if the call to the function is allowed or not.
    - any value for the last 4th byte is accepted.
    - if the 4th byte is 0x01, this determines if the lsp20VerifyCallResult(bytes32,bytes) function should be called after the original function call (The byte that invokes the lsp20VerifyCallResult(bytes32,bytes) function is strictly 0x01).

lsp20VerifyCallResult

```
function lsp20VerifyCallResult(bytes32 callHash, bytes memory callResult) external returns (bytes4 returnedStatus);
```

This function is the post-verification function.

It can be used to run any form of verification mechanism after having run the actual function that was initially called.

Parameters:

- callHash: The keccak256 hash of the parameters of lsp20VerifyCall(address,address,address,uint256,bytes) parameters packed-encoded (concatened).
- callResult: the result of the function being called on the contract delegating the verification mechanism.
    - if the function being called returns some data, the callResult MUST be the value returned by the function being called as abi-encoded bytes.
    - if the function being called does not return any data, the callResult MUST be an empty bytes.

Returns:

- returnedStatus: the status determining if the verification succeeded or not.

Requirements

- MUST return the lsp20VerifyCallResult(bytes32,bytes) function selector if the call to the function is allowed.

Handling Verification Result

When calling the functions lsp20VerifyCall(...) and lsp20VerifyCallResult(...), the LSP20 standard does not differentiate between:

- if a revert occurred when calling one of these functions.
- if these functions executed successfully but returned a non-success value.

Both of these scenarios mean that the verification failed when calling the contract that implements the LSP20 interface.

Rationale

The lsp20VerifyCall(..) and lsp20VerifyCallResult(..) functions work in tandem to verify certain conditions before and after the execution of a function within the same transaction. To optimize gas costs and improve efficiency, the callHash parameter is introduced in the lsp20VerifyCallResult(..) function.

lsp20VerifyCall(..) takes the caller, value, and data as parameters and is invoked before the execution of the targeted function. Based on the return value of this function, it is determined whether lsp20VerifyCallResult(..) will run.

Instead of passing the same parameters (caller, value, data) along with the result of the executed function to lsp20VerifyCallResult(..), the callHash parameter is used. The callHash is the keccak256 hash of the concatenated lsp20VerifyCall(..) parameters. Since both functions are invoked in the same transaction, a user can hash these parameters in lsp20VerifyCall(..) and store them under the hash. Later, the stored values can be retrieved using the callHash provided in lsp20VerifyCallResult(..).

This approach has been adopted because passing the same parameters again would be expensive in terms of gas costs, and it's not always necessary for the user to access these parameters in lsp20VerifyCallResult(..). If a user needs to use these parameters, they should store them in the contract storage during the lsp20VerifyCall(..) invocation.

Example: Reentrancy Check

In a case where the parameters are not relevant for lsp20VerifyCallResult(..), such as checking for reentrancy, the first lsp20VerifyCall(..) function will be checking for reentrancy and will set the reentrancy flag to true. Then, lsp20VerifyCallResult(..) can simply set the reentrancy flag back to false without needing access to the original parameters.

Implementation

An implementation can be found in the [lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol)](https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol) repository.

Interface Cheat Sheet

```
interface ILSP20  /* is ERC165 */ {

  function lsp20VerifyCall(address requestor, address target, address caller, uint256 value, bytes memory receivedCalldata) external returns (bytes4 returnedStatus);

  function lsp20VerifyCallResult(bytes32 callHash, bytes memory callResult) external returns (bytes4);

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-23-LinkedContractsFactory.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-23-LinkedContractsFactory.md

---

lip: 23
title: LSP23 Linked Contracts Factory
author: skimaharvey
discussions-to: <URL>
status: Draft
type: LSP
created: 2023-08-16
requires: ERC1167

Table of Content

Simple Summary

Deploying smart contracts that need to interact with each other can be complicated. This is especially true when these contracts need to know each other's details at the time they are created. The LSP23 standard simplifies this process by providing a unified way to deploy such interdependent contracts. It also allows for additional actions to be taken automatically after the contracts are deployed, making the whole process more streamlined and less error-prone.

Abstract

The LSP23 Linked Contracts Factory standard introduces a unified interface for deploying interdependent smart contracts, also known as linked contracts. These contracts often require each other's addresses at the time of deployment, creating a circular dependency. LSP23 addresses this by allowing for the deployment of a primary contract and one or more secondary contracts, linking them together. The standard also supports

the deployment of contracts as ERC1167 minimal proxies and enables optional post-deployment modules for executing additional logic. This serves to simplify and standardize the deployment of complex contract systems.

Motivation

While there are various methods for deploying smart contracts in the current ecosystem, the challenge of deploying interdependent contracts in a streamlined and secure manner remains. Custom solutions often exist, but they can be cumbersome and prone to errors. LSP23 aims to standardize this process, reducing the complexity and potential for mistakes. By generating the salt for contract deployment within the function, the standard also ensures a fully decentralized deployment process. This not only guarantees consistent contract addresses across multiple chains but also serves as a preventive measure against address squatting, enhancing the overall security and integrity of the deployment process.

Specification

Linked Contracts Factory Setup

Deployment

Before the deployment of the LSP23LinkedContractsFactory on any network, people should make sure that the Nick Factory is deployed on the same network.

Nick Factory Deployment

The Nick Factory should be located at this address 0x4e59b44847b379578588920ca78fbf26c0b4956c on the network. If there is no code on this address, it means that the contract is not deployed yet.

To deploy, the following raw transaction should be broadcasted to the network 0xf8a58085174876e800830186a08080b853604580600e600039806000f350fe7ffffffffffffffffffffffffffffffffffffffffffffff after funding the deployer address: 0x3fab184622dc19b6109349b94811493bf2a45362 with gasPrice (100 gwei) * gasLimit (100000).

Check [Nick's Factory repository (https://github.com/Arachnid/deterministic-deployment-proxy/tree/master)] for more information.

LSP23LinkedContractsFactory Deployment

After the deployment of Nick Factory on the network, the LSP23LinkedContractsFactory can be reproduced at the standardized address given sending the same salt and bytecode.

In order to create the UniversalFactory contract, one should send a transaction to the [Nick Factory (https://github.com/Arachnid/deterministic-deployment-proxy)] address with data field equal to [salt] + [bytecode].

The address produced should be equal to 0x2300000A84D25dF63081feAa37ba6b62C4c89a30.

UniversalFactory Configuration

Standardized Address

0x2300000A84D25dF63081feAa37ba6b62C4c89a30

Standardized Salt

0x12a6712f113536d8b01d99f72ce168c7e1090124db54cd16f03c20000022178c

Standardized Bytecode

0x6080604052348015610010576000080fd5b506112478061002060003960003f3fe6080604052600436106100

UniversalFactory Source Code

This is an exact copy of the code of the [LSP23 LinkedContractsFactory smart contract].

- The source code is generated with 0.8.17 compiler version and with 9999999 optimization runs.
- The imported contracts are part of the 4.9.2 version of the @openzeppelin/contracts package.
- Navigate to [lsp-smart-contract (https://github.com/lukso-network/lsp-smart-contracts)] repo and checkout to b8eca3c5696acf85239130ef67edec9e8c134bfa commit to obtain the exact copy of the code, change the compiler settings in hardhat.config.ts and compile to produce the same bytecode.

<details>
<summary>Click to Expand</summary>
<pre>

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import { Create2 } from "@openzeppelin/contracts/utils/Create2.sol";
import { Clones } from "@openzeppelin/contracts/proxy/Clones.sol";
import { IPostDeploymentModule } from "./IPostDeploymentModule.sol";
import { ILSP23LinkedContractsFactory } from "./ILSP23LinkedContractsFactory.sol";
import { InvalidValueSum, PrimaryContractProxyInitFailureError, SecondaryContractProxyInitFailureError }
from "./LSP23Errors.sol";

contract LSP23LinkedContractsFactory is ILSP23LinkedContractsFactory {
  /**
   * @inheritdoc ILSP23LinkedContractsFactory
   */
  function deployContracts(
    PrimaryContractDeployment calldata primaryContractDeployment,
    SecondaryContractDeployment calldata secondaryContractDeployment,
    address postDeploymentModule,
    bytes calldata postDeploymentModuleCalldata
  )
    public
    payable
    returns (address primaryContractAddress, address secondaryContractAddress)
  {
    /* check that the msg.value is equal to the sum of the values of the primary and secondary contracts */
    if (
      msg.value !=
      primaryContractDeployment.fundingAmount +
        secondaryContractDeployment.fundingAmount
    ) {
      revert InvalidValueSum();
```

```
    }

    primaryContractAddress = _deployPrimaryContract(
      primaryContractDeployment,
      secondaryContractDeployment,
      postDeploymentModule,
      postDeploymentModuleCalldata
    );

    secondaryContractAddress = _deploySecondaryContract(
      secondaryContractDeployment,
      primaryContractAddress
    );

    emit DeployedContracts(
      primaryContractAddress,
      secondaryContractAddress,
      primaryContractDeployment,
      secondaryContractDeployment,
      postDeploymentModule,
      postDeploymentModuleCalldata
    );

    /* execute the post deployment logic in the postDeploymentModule if postDeploymentModule is not
address(0) */
    if (postDeploymentModule != address(0)) {
      /* execute the post deployment module logic in the postDeploymentModule */
      IPostDeploymentModule(postDeploymentModule).executePostDeployment(
        primaryContractAddress,
        secondaryContractAddress,
        postDeploymentModuleCalldata
      );
    }
  }

  /**
   * @inheritdoc ILSP23LinkedContractsFactory
   */
  function deployERC1167Proxies(
    PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
    SecondaryContractDeploymentInit calldata secondaryContractDeploymentInit,
    address postDeploymentModule,
    bytes calldata postDeploymentModuleCalldata
  )
    public
    payable
    returns (address primaryContractAddress, address secondaryContractAddress)
  {
    /* check that the msg.value is equal to the sum of the values of the primary and secondary contracts */
    if (
      msg.value !=
      primaryContractDeploymentInit.fundingAmount +
```

```
      secondaryContractDeploymentInit.fundingAmount
    ) {
      revert InvalidValueSum();
    }

    /* deploy the primary contract proxy with the primaryContractGeneratedSalt */
    primaryContractAddress = _deployAndInitializePrimaryContractProxy(
      primaryContractDeploymentInit,
      secondaryContractDeploymentInit,
      postDeploymentModule,
      postDeploymentModuleCalldata
    );

    /* deploy the secondary contract proxy */
    secondaryContractAddress = _deployAndInitializeSecondaryContractProxy(
      secondaryContractDeploymentInit,
      primaryContractAddress
    );

    emit DeployedERC1167Proxies(
      primaryContractAddress,
      secondaryContractAddress,
      primaryContractDeploymentInit,
      secondaryContractDeploymentInit,
      postDeploymentModule,
      postDeploymentModuleCalldata
    );

    /* execute the post deployment logic in the postDeploymentModule if postDeploymentModule is not
address(0) */
    if (postDeploymentModule != address(0)) {
      /* execute the post deployment logic in the postDeploymentModule */
      IPostDeploymentModule(postDeploymentModule).executePostDeployment(
        primaryContractAddress,
        secondaryContractAddress,
        postDeploymentModuleCalldata
      );
    }
  }

  /**
   * @inheritdoc ILSP23LinkedContractsFactory
   */
  function computeAddresses(
    PrimaryContractDeployment calldata primaryContractDeployment,
    SecondaryContractDeployment calldata secondaryContractDeployment,
    address postDeploymentModule,
    bytes calldata postDeploymentModuleCalldata
  )
    public
    view
    returns (address primaryContractAddress, address secondaryContractAddress)
```

```
{
  bytes32 primaryContractGeneratedSalt = _generatePrimaryContractSalt(
    primaryContractDeployment,
    secondaryContractDeployment,
    postDeploymentModule,
    postDeploymentModuleCalldata
  );

  primaryContractAddress = Create2.computeAddress(
    primaryContractGeneratedSalt,
    keccak256(primaryContractDeployment.creationBytecode)
  );

  bytes memory secondaryContractByteCodeWithAllParams;
  if (secondaryContractDeployment.addPrimaryContractAddress) {
    secondaryContractByteCodeWithAllParams = abi.encodePacked(
      secondaryContractDeployment.creationBytecode,
      abi.encode(primaryContractAddress),
      secondaryContractDeployment.extraConstructorParams
    );
  } else {
    secondaryContractByteCodeWithAllParams = secondaryContractDeployment
      .creationBytecode;
  }

  secondaryContractAddress = Create2.computeAddress(
    keccak256(abi.encodePacked(primaryContractAddress)),
    keccak256(secondaryContractByteCodeWithAllParams)
  );
}

/**
 * @inheritdoc ILSP23LinkedContractsFactory
 */
function computeERC1167Addresses(
  PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
  SecondaryContractDeploymentInit calldata secondaryContractDeploymentInit,
  address postDeploymentModule,
  bytes calldata postDeploymentModuleCalldata
)
  public
  view
  returns (address primaryContractAddress, address secondaryContractAddress)
{
  bytes32 primaryContractGeneratedSalt = _generatePrimaryContractProxySalt(
    primaryContractDeploymentInit,
    secondaryContractDeploymentInit,
    postDeploymentModule,
    postDeploymentModuleCalldata
  );

  primaryContractAddress = Clones.predictDeterministicAddress(
```

```
    primaryContractDeploymentInit.implementationContract,
    primaryContractGeneratedSalt
  );

  secondaryContractAddress = Clones.predictDeterministicAddress(
    secondaryContractDeploymentInit.implementationContract,
    keccak256(abi.encodePacked(primaryContractAddress))
  );
}

function _deployPrimaryContract(
  PrimaryContractDeployment calldata primaryContractDeployment,
  SecondaryContractDeployment calldata secondaryContractDeployment,
  address postDeploymentModule,
  bytes calldata postDeploymentModuleCalldata
) internal returns (address primaryContractAddress) {
  bytes32 primaryContractGeneratedSalt = _generatePrimaryContractSalt(
    primaryContractDeployment,
    secondaryContractDeployment,
    postDeploymentModule,
    postDeploymentModuleCalldata
  );

  /* deploy the primary contract */
  primaryContractAddress = Create2.deploy(
    primaryContractDeployment.fundingAmount,
    primaryContractGeneratedSalt,
    primaryContractDeployment.creationBytecode
  );
}

function _deploySecondaryContract(
  SecondaryContractDeployment calldata secondaryContractDeployment,
  address primaryContractAddress
) internal returns (address secondaryContractAddress) {
  /**
   * If `addPrimaryContractAddress` is `true`, the following will be appended to the constructor params:
   * - The primary contract address
   * - `extraConstructorParams`
   */
  bytes memory secondaryContractByteCode = secondaryContractDeployment
    .creationBytecode;

  if (secondaryContractDeployment.addPrimaryContractAddress) {
    secondaryContractByteCode = abi.encodePacked(
      secondaryContractByteCode,
      abi.encode(primaryContractAddress),
      secondaryContractDeployment.extraConstructorParams
    );
  }

  secondaryContractAddress = Create2.deploy(
```

```
    secondaryContractDeployment.fundingAmount,
    keccak256(abi.encodePacked(primaryContractAddress)),
    secondaryContractByteCode
  );
}

function _deployAndInitializePrimaryContractProxy(
  PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
  SecondaryContractDeploymentInit calldata secondaryContractDeploymentInit,
  address postDeploymentModule,
  bytes calldata postDeploymentModuleCalldata
) internal returns (address primaryContractAddress) {
  bytes32 primaryContractGeneratedSalt = _generatePrimaryContractProxySalt(
    primaryContractDeploymentInit,
    secondaryContractDeploymentInit,
    postDeploymentModule,
    postDeploymentModuleCalldata
  );

  /* deploy the primary contract proxy with the primaryContractGeneratedSalt */
  primaryContractAddress = Clones.cloneDeterministic(
    primaryContractDeploymentInit.implementationContract,
    primaryContractGeneratedSalt
  );

  /* initialize the primary contract proxy */
  (bool success, bytes memory returnedData) = primaryContractAddress.call{
    value: primaryContractDeploymentInit.fundingAmount
  }(primaryContractDeploymentInit.initializationCalldata);
  if (!success) {
    revert PrimaryContractProxyInitFailureError(returnedData);
  }
}

function _deployAndInitializeSecondaryContractProxy(
  SecondaryContractDeploymentInit calldata secondaryContractDeploymentInit,
  address primaryContractAddress
) internal returns (address secondaryContractAddress) {
  /* deploy the secondary contract proxy with the primaryContractGeneratedSalt */
  secondaryContractAddress = Clones.cloneDeterministic(
    secondaryContractDeploymentInit.implementationContract,
    keccak256(abi.encodePacked(primaryContractAddress))
  );

  /**
   * If `addPrimaryContractAddress` is `true`, the following will be appended to the `initializationCalldata`:
   * - The primary contract address
   * - `extraInitializationBytes`
   */
  bytes memory secondaryInitializationBytes = secondaryContractDeploymentInit
    .initializationCalldata;
```

```
  if (secondaryContractDeploymentInit.addPrimaryContractAddress) {
    secondaryInitializationBytes = abi.encodePacked(
      secondaryInitializationBytes,
      abi.encode(primaryContractAddress),
      secondaryContractDeploymentInit.extraInitializationParams
    );
  }

  /* initialize the primary contract proxy */
  (bool success, bytes memory returnedData) = secondaryContractAddress.call{
    value: secondaryContractDeploymentInit.fundingAmount
  }(secondaryInitializationBytes);
  if (!success) {
    revert SecondaryContractProxyInitFailureError(returnedData);
  }
}

function _generatePrimaryContractSalt(
  PrimaryContractDeployment calldata primaryContractDeployment,
  SecondaryContractDeployment calldata secondaryContractDeployment,
  address postDeploymentModule,
  bytes calldata postDeploymentModuleCalldata
) internal pure virtual returns (bytes32 primaryContractGeneratedSalt) {
  /* generate salt for the primary contract
   *  the salt is generated by hashing the following elements:
   *   - the salt
   *   - the secondary contract bytecode
   *   - the secondary addPrimaryContractAddress boolean
   *   - the secondary extraConstructorParams
   *   - the postDeploymentModule address
   *   - the postDeploymentModuleCalldata
   *
   */
  primaryContractGeneratedSalt = keccak256(
    abi.encode(
      primaryContractDeployment.salt,
      secondaryContractDeployment.creationBytecode,
      secondaryContractDeployment.addPrimaryContractAddress,
      secondaryContractDeployment.extraConstructorParams,
      postDeploymentModule,
      postDeploymentModuleCalldata
    )
  );
}

function _generatePrimaryContractProxySalt(
  PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
  SecondaryContractDeploymentInit calldata secondaryContractDeploymentInit,
  address postDeploymentModule,
  bytes calldata postDeploymentModuleCalldata
) internal pure virtual returns (bytes32 primaryContractProxyGeneratedSalt) {
  /**
```

```
    * Generate the salt for the primary contract
    * The salt is generated by hashing the following elements:
    *  - the salt
    *  - the secondary implementation contract address
    *  - the secondary contract initialization calldata
    *  - the secondary contract addPrimaryContractAddress boolean
    *  - the secondary contract extra initialization params (if any)
    *  - the postDeploymentModule address
    *  - the callda to the post deployment module
    *
    */
   primaryContractProxyGeneratedSalt = keccak256(
     abi.encode(
       primaryContractDeploymentInit.salt,
       secondaryContractDeploymentInit.implementationContract,
       secondaryContractDeploymentInit.initializationCalldata,
       secondaryContractDeploymentInit.addPrimaryContractAddress,
       secondaryContractDeploymentInit.extraInitializationParams,
       postDeploymentModule,
       postDeploymentModuleCalldata
     )
   );
  }
}
```

</pre>
</details>

Methods

LSP23LinkedContractsFactory

The LSP23LinkedContractsFactory contract provides two main functions for deploying linked contracts: deployContracts and deployERC1167Proxies. Both functions allow the deployment of primary and secondary contracts, with optional post-deployment modules.

deployContracts

```
function deployContracts(
    PrimaryContractDeployment calldata primaryContractDeployment,
    SecondaryContractDeployment calldata secondaryContractDeployment,
    address postDeploymentModule,
    bytes calldata postDeploymentModuleCalldata
)
    public
    payable
    returns (
        address primaryContractAddress,
        address secondaryContractAddress
    );
```

Deploys primary and secondary contracts and links them together. Optionally executes a post-deployment module.

Parameters:

- primaryContractDeployment: Deployment parameters for the primary contract.
- secondaryContractDeployment: Deployment parameters for the secondary contract.
- postDeploymentModule: Address of the post-deployment module to execute.
- postDeploymentModuleCalldata: Calldata for the post-deployment module.

Returns: address, address , the addresses of the deployed primary and secondary contracts.

deployERC1167Proxies

```
function deployERC1167Proxies(
    PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
    SecondaryContractDeploymentInit calldata secondaryContractDeploymentInit,
    address postDeploymentModule,
    bytes calldata postDeploymentModuleCalldata
)
    public
    payable
    returns (
        address primaryContractAddress,
        address secondaryContractAddress
    );
```

Deploys primary and secondary contracts as ERC1167 proxies and links them together. Optionally executes a post-deployment module.

Parameters:

- primaryContractDeploymentInit: Deployment parameters for the primary contract proxy.
- secondaryContractDeploymentInit: Deployment parameters for the secondary contract proxy.
- postDeploymentModule: Address of the post-deployment module to execute.
- postDeploymentModuleCalldata: Calldata for the post-deployment module.

Returns: address, address , the addresses of the deployed primary and secondary contract proxies.

PostDeploymentModule

Post-deployment modules are optionals and executed after the primary and secondary contracts are deployed. The executePostDeployment function is called with the addresses of the deployed primary and secondary contracts as well as the calldata for the post-deployment module.

```
    function executePostDeployment(
        address primaryContract,
        address secondaryContract,
        bytes calldata calldataToPostDeploymentModule
    ) public;
```

Executes post-deployment logic.

Parameters:

- primaryContract: Address of the primary contract.
- secondaryContract: Address of the secondary contract.
- calldataToPostDeploymentModule: Calldata for the post-deployment module.

Implementation

An implementation can be found in the [lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/develop/contracts/LSP23LinkedContractsDeployment/LSP23LinkedContractsFactory.sol)]

Interface Cheat Sheet

ILSP23LinkedContractsFactory

```
interface ILSP23LinkedContractsFactory {
    /**
     * @dev Emitted when a primary and secondary contract are deployed.
     * @param primaryContract Address of the deployed primary contract.
     * @param secondaryContract Address of the deployed secondary contract.
     * @param primaryContractDeployment Parameters used for the primary contract deployment.
     * @param secondaryContractDeployment Parameters used for the secondary contract deployment.
     * @param postDeploymentModule Address of the post-deployment module.
     * @param postDeploymentModuleCalldata Calldata passed to the post-deployment module.
     */
    event DeployedContracts(
        address indexed primaryContract,
        address indexed secondaryContract,
        PrimaryContractDeployment primaryContractDeployment,
        SecondaryContractDeployment secondaryContractDeployment,
        address postDeploymentModule,
        bytes postDeploymentModuleCalldata
    );

    /**
     * @dev Emitted when proxies of a primary and secondary contract are deployed.
     * @param primaryContract Address of the deployed primary contract proxy.
     * @param secondaryContract Address of the deployed secondary contract proxy.
     * @param primaryContractDeploymentInit Parameters used for the primary contract proxy deployment.
     * @param secondaryContractDeploymentInit Parameters used for the secondary contract proxy
deployment.
     * @param postDeploymentModule Address of the post-deployment module.
     * @param postDeploymentModuleCalldata Calldata passed to the post-deployment module.
     */
    event DeployedERC1167Proxies(
        address indexed primaryContract,
        address indexed secondaryContract,
        PrimaryContractDeploymentInit primaryContractDeploymentInit,
        SecondaryContractDeploymentInit secondaryContractDeploymentInit,
        address postDeploymentModule,
        bytes postDeploymentModuleCalldata
    );
```

```
    /**
     * @param salt A unique value used to ensure each created proxies are unique. (Can be used to deploy the contract at a desired address.)
     * @param fundingAmount The value to be sent with the deployment transaction.
     * @param creationBytecode The bytecode of the contract with the constructor params.
     */
    struct PrimaryContractDeployment {
        bytes32 salt;
        uint256 fundingAmount;
        bytes creationBytecode;
    }

    /**
     * @param fundingAmount The value to be sent with the deployment transaction.
     * @param creationBytecode The constructor + runtime bytecode (without the primary contract's address as param)
     * @param addPrimaryContractAddress If set to `true`, this will append the primary contract's address + the `extraConstructorParams` to the `creationBytecode`.
     * @param extraConstructorParams Params to be appended to the `creationBytecode` (after the primary contract address) if `addPrimaryContractAddress` is set to `true`.
     */
    struct SecondaryContractDeployment {
        uint256 fundingAmount;
        bytes creationBytecode;
        bool addPrimaryContractAddress;
        bytes extraConstructorParams;
    }

    /**
     * @param salt A unique value used to ensure each created proxies are unique. (Can be used to deploy the contract at a desired address.)
     * @param fundingAmount The value to be sent with the deployment transaction.
     * @param implementationContract The address of the contract that will be used as a base contract for the proxy.
     * @param initializationCalldata The calldata used to initialise the contract. (initialization should be similar to a constructor in a normal contract.)
     */
    struct PrimaryContractDeploymentInit {
        bytes32 salt;
        uint256 fundingAmount;
        address implementationContract;
        bytes initializationCalldata;
    }

    /**
     * @param fundingAmount The value to be sent with the deployment transaction.
     * @param implementationContract The address of the contract that will be used as a base contract for the proxy.
     * @param initializationCalldata The first part of the initialisation calldata, everything before the primary contract address.
```

```
    * @param addPrimaryContractAddress If set to `true`, this will append the primary contract's address +
the `extraInitializationParams` to the `initializationCalldata`.
    * @param extraInitializationParams Params to be appended to the `initializationCalldata` (after the
primary contract address) if `addPrimaryContractAddress` is set to `true`
    */
    struct SecondaryContractDeploymentInit {
        uint256 fundingAmount;
        address implementationContract;
        bytes initializationCalldata;
        bool addPrimaryContractAddress;
        bytes extraInitializationParams;
    }

    /**
    * @dev Deploys a primary and a secondary linked contract.
    * @notice Contracts deployed. Contract Address: `primaryContractAddress`. Primary Contract Address:
`primaryContractAddress`
    *
    * @param primaryContractDeployment Contains the needed parameter to deploy a contract. (`salt`,
`fundingAmount`, `creationBytecode`)
    * @param secondaryContractDeployment Contains the needed parameter to deploy the secondary
contract. (`fundingAmount`, `creationBytecode`, `addPrimaryContractAddress`, `extraConstructorParams`)
    * @param postDeploymentModule The module to be executed after deployment
    * @param postDeploymentModuleCalldata The data to be passed to the post deployment module
    *
    * @return primaryContractAddress The address of the primary contract.
    * @return secondaryContractAddress The address of the secondary contract.
    */
    function deployContracts(
        PrimaryContractDeployment calldata primaryContractDeployment,
        SecondaryContractDeployment calldata secondaryContractDeployment,
        address postDeploymentModule,
        bytes calldata postDeploymentModuleCalldata
    )
        external
        payable
        returns (
            address primaryContractAddress,
            address secondaryContractAddress
        );

    /**
    * @dev Deploys proxies of a primary contract and a secondary linked contract
    * @notice Contract proxies deployed. Primary Proxy Address: `primaryContractAddress`. Secondary
Contract Proxy Address: `secondaryContractAddress`
    *
    * @param primaryContractDeploymentInit Contains the needed parameters to deploy a proxy contract.
(`salt`, `fundingAmount`, `implementationContract`, `initializationCalldata`)
    * @param secondaryContractDeploymentInit Contains the needed parameters to deploy the secondary
proxy contract. (`fundingAmount`, `implementationContract`, `addPrimaryContractAddress`,
`initializationCalldata`, `extraInitializationParams`)
    * @param postDeploymentModule The module to be executed after deployment.
```

```
    * @param postDeploymentModuleCalldata The data to be passed to the post deployment module.
    *
    * @return primaryContractAddress The address of the deployed primary contract proxy
    * @return secondaryContractAddress The address of the deployed secondary contract proxy
    */
    function deployERC1167Proxies(
        PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
        SecondaryContractDeploymentInit
            calldata secondaryContractDeploymentInit,
        address postDeploymentModule,
        bytes calldata postDeploymentModuleCalldata
    )
        external
        payable
        returns (
            address primaryContractAddress,
            address secondaryContractAddress
        );

    /**
    * @dev Computes the addresses of a primary contract and a secondary linked contract
    *
    * @param primaryContractDeployment Contains the needed parameter to deploy the primary contract.
(`salt`, `fundingAmount`, `creationBytecode`)
    * @param secondaryContractDeployment Contains the needed parameter to deploy the secondary
contract. (`fundingAmount`, `creationBytecode`, `addPrimaryContractAddress`, `extraConstructorParams`)
    * @param postDeploymentModule The module to be executed after deployment
    * @param postDeploymentModuleCalldata The data to be passed to the post deployment module
    *
    * @return primaryContractAddress The address of the deployed primary contract.
    * @return secondaryContractAddress The address of the deployed secondary contract.
    */
    function computeAddresses(
        PrimaryContractDeployment calldata primaryContractDeployment,
        SecondaryContractDeployment calldata secondaryContractDeployment,
        address postDeploymentModule,
        bytes calldata postDeploymentModuleCalldata
    )
        external
        view
        returns (
            address primaryContractAddress,
            address secondaryContractAddress
        );

    /**
    * @dev Computes the addresses of a primary and a secondary linked contracts proxies to be created
    *
    * @param primaryContractDeploymentInit Contains the needed parameters to deploy a primary proxy
contract. (`salt`, `fundingAmount`, `implementationContract`, `initializationCalldata`)
    * @param secondaryContractDeploymentInit Contains the needed parameters to deploy the secondary
proxy contract. (`fundingAmount`, `implementationContract`, `addPrimaryContractAddress`,
```

```
`initializationCalldata`, `extraInitializationParams`)
     * @param postDeploymentModule The module to be executed after deployment.
     * @param postDeploymentModuleCalldata The data to be passed to the post deployment module.
     *
     * @return primaryContractAddress The address of the deployed primary contract proxy
     * @return secondaryContractAddress The address of the deployed secondary contract proxy
     */
    function computeERC1167Addresses(
        PrimaryContractDeploymentInit calldata primaryContractDeploymentInit,
        SecondaryContractDeploymentInit
            calldata secondaryContractDeploymentInit,
        address postDeploymentModule,
        bytes calldata postDeploymentModuleCalldata
    )
        external
        view
        returns (
            address primaryContractAddress,
            address secondaryContractAddress
        );
}
```

IPostDeploymentModule

```
interface IPostDeploymentModule {
    /**
     * @dev Executes post-deployment logic.
     * @param primaryContractAddress Address of the deployed primary contract or proxy.
     * @param secondaryContractAddress Address of the deployed secondary contract or proxy.
     * @param postDeploymentModuleCalldata Calldata for the post-deployment module.
     */
    function executePostDeployment(
        address primaryContractAddress,
        address secondaryContractAddress,
        bytes calldata postDeploymentModuleCalldata
    ) external;
}
```

Copyright

</file>

<file>

path: /LSPs/LSP-25-ExecuteRelayCall.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-25-ExecuteRelayCall.md

---

lip: 25
title: Execute Relay Call
author: Jean Cavallera (@CJ42)
discussions-to: <URL>
status: Draft
type: LSP
created: 2023-08-09
requires: ERC165

Table of Content

Simple Summary

This standard describes an interface and a signature scheme that can be used to integrate meta-transactions (transactions where users do not pay for the gas) inside a smart contract.

Abstract

Meta transactions in web3 are used in many different context. From interacting with Universal Profiles to transferring tokens and NFTs to voting systems, where delegates can submit transactions on behalf of other users and pay for the gas cost.

Motivation

By having a common smart contract interface, applications and protocols can implement the LSP25 Execute Relay Call standard to start implementing gas-less transaction. This can be beneficial for protocols and applications like Digital Marketplaces or Universal Profile that want to onboard new users without requiring

users to hold native tokens.

Specification

LSP25-ExecuteRelayCall interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0x5ac79908

Smart contracts implementing the LSP25 standard MUST implement the ERC165 (https://eips.ethereum.org/EIPS/eip-165) supportsInterface(..) function and MUST support the ERC165 (https://eips.ethereum.org/EIPS/eip-165) and LSP25 interface ids.

Methods

getNonce

```
function getNonce(address signer, uint128 channel) external view returns (uint256)
```

Returns the latest nonce for a signer on a specific channel. A signer can choose a channel number arbitrarily and use this nonce to sign a calldata payload that can be executed as a meta-transaction by any address via executeRelayCall function.

Parameters:

- signer: the address of the signer of the transaction.
- channel : the channel which the signer wants to use for executing the transaction.

Returns: uint256 , the current nonce.

Calldata payloads signed with incremental nonces on the same channel for the same signer are executed in order. e.g, in channel X, the second calldata payload signed with the second nonce will not be successfully executed until the first calldata payload signed with the first nonce has been executed.

Calldata payloads signed with nonces on different channels are executed independently from each other, regardless of when they got executed or if they got executed successfully or not. e.g, the calldata payload signed with the fourth nonce on channel X can be successfully executed even if the calldata payload signed with the first nonce of channel Y:

- was executed before.
- was executed and reverted.

> X and Y can be any arbitrary number between 0 and 2^128.

Read what are multi-channel nonces.

executeRelayCall

```
function executeRelayCall(
    bytes memory signature,
    uint256 nonce,
    uint256 validityTimestamps,
    bytes memory payload
)
    external
    payable
    returns (bytes memory)
```

Allows anybody to execute a calldata payload given they have a valid signature from a signer. The signature MUST be formed according to the LSP25 Signature specification format.

Parameters:

- signature: A 65 bytes long ethereum signature.
- nonce: MUST be the nonce of the address that signed the message. This can be obtained via the getNonce(address address, uint256 channel) function.
- validityTimestamps: Two uint128 timestamps concatenated together. The first timestamp determines from when the calldata payload can be executed, and the second timestamp determines a deadline after which the payload is no longer valid. If validityTimestamps is 0, the payload is valid indefinitely at any point in time and the checks for the timestamps are skipped.
  payload can be executed, the second timestamp determines a deadlines after which the calldata payload is not valid anymore. If validityTimestamps is 0, the calldata payload is valid at indefinitely at any point in time and the checks for the timestamps are skipped.
- payload: The abi-encoded function call to be executed. This could be a function to be called on the current contract implementing LSP25 or an external target contract.

Returns: bytes. If the call succeeded, these bytes MUST be the returned data as abi-decoded bytes of the function call defined by the payload parameter. Otherwise revert with a reason-string.

Requirements:

- The address recovered from the signature and the digest signed MUST have permission(s) for the action(s) being executed. Check Permissions to know more.
- The nonce passed to the function MUST be a valid nonce according to the multi-channel nonce section.
- MUST send the value passed by the caller to the call on the linked target contract.

See the section Signature Format below to learn how to sign LSP25 Execute Relay Call meta-transactions.

> Note: Non payable functions will revert in case of calling them and passing value along the call.

executeRelayCallBatch

```
function executeRelayCallBatch(
    bytes[] memory signatures,
    uint256[] memory nonces,
    uint256[] memory validityTimestamps,
    uint256[] memory values,
    bytes[] memory payloads
)
    external
    payable
    returns (bytes[] memory)
```

Allows anybody to execute a batch of calldata payloads given they have valid signatures from signers. Each signature MUST be formed according to the LSP25 Signature specification format.

Parameters:

- signatures: An array of bytes65 ethereum signature.
- nonce: An array of nonces from the address/es that signed the digests. This can be obtained via the getNonce(address address, uint256 channel) function.
- values: An array of native token amounts to transfer to the linked target contract alongside the call on each iteration.
- validityTimestamps: An array of uint256 formed of Two uint128 timestamps concatenated, the first timestamp determines from when the calldata payload can be executed, the second timestamp delimits the end of the validity of the calldata payload. If validityTimestamps is 0, the checks of the timestamps are skipped.
- payloads: An array of calldata payloads to be executed on the linked target contract on each iteration.

Returns: bytes[] , an array of returned as abi-decoded array of bytes[] of the linked target contract, if the calls succeeded, otherwise revert with a reason-string.

Requirements:

- MUST comply to the requirements of the executeRelayCall(bytes,uint256,bytes) function.
- Each array parameters provided to the function MUST have the same length.
- The sum of each element of the values array MUST be equal to the total value sent to the function.
- Each nonces passed to the function MUST be a valid nonce according to the multi-channel nonce section.
- MUST send the value passed by the caller to the call on the linked target contract.

Signature Format

In order to submit relay calls successfully, users MUST sign the relay calls to be executed according to the specification below.

The hash digest that MUST be signed MUST be constructed according to the version 0 of EIP-191 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-191.md#version-0x00) with the following format:

```
0x19 <0x00> <Implementation address> <LSP25_VERSION> <chainId> <nonce> <validityTimestamps> <value> <calldata>
```

The table below breakdown each parameters in details:

Value	Type	Description

0x19	bytes1	byte intended to ensure that the signed_data is not valid RLP.
0x00	bytes1	version 0 of EIP191 (https://eips.ethereum.org/EIPS/eip-191).
Implementation address	address	The address of the contract implementing LSP25 that will execute the calldata.
LSP25_VERSION	uint256	Version relative to the LSP25ExecuteRelayCall standard defined equal to 25.
chainId	uint256	The chainId of the blockchain where the Key Manager is deployed.
nonce	uint256	The nonce to sign the calldata with.
validityTimestamps	uint256	Two uint128 timestamps concatenated, the first timestamp determines from when the calldata payload can be executed, the second timestamp delimits the end of the validity of the calldata payload. If validityTimestamps is 0, the checks of the timestamps are skipped.
value	uint256	The amount of native token to transfer to the linked target contract alongside the call.
calldata	bytes	The abi-encoded function call to be executed.

These parameters MUST be packed encoded (not zero padded, leading 0s are removed), then hashed with keccak256 to produce the hash digest.

For signing, users should apply the same steps and sign the final hash obtained at the end.

Multi-Channel Nonces

What are multi-channel nonces?

This concept was taken from https://github.com/amxx/permit#out-of-order-execution (https://github.com/amxx/permit#out-of-order-execution).

Using nonces prevent old signed transactions from being replayed again (replay attacks). A nonce is an arbitrary number that can be used just once in a transaction.

Problem of Sequential Nonces

With native transactions, nonces are strictly sequential. This means that messages with sequential nonces must be executed in order. For instance, in order for message number 4 to be executed, it must wait for message number 3 to complete.

However, sequential nonces come with the following limitation:

Some users may want to sign multiple message, allowing the transfer of different assets to different recipients. In that case, the recipient want to be able to use / transfer their assets whenever they want, and will certainly not want to wait on anyone before signing another transaction.

This is where out-of-order execution comes in.

Benefits of multi-channel nonces

Out-of-order execution is achieved by using multiple independent channels. Each channel's nonce behaves as expected, but different channels are independent. This means that messages 2, 3, and 4 of channel 0 must be executed sequentially, but message 3 of channel 1 is independent, and only depends on message 2 of channel 1.

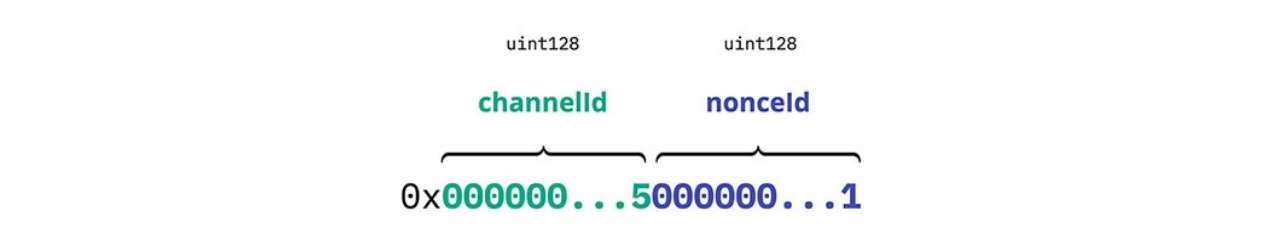
The benefit is that the signer key can determine for which channel to sign the nonces. Relay services will have to understand the channel the signer choose and execute the transactions of each channel in the right order, to prevent failing transactions.

How nonces are represented across channels?

The LSP25 standard allows out-of-order execution of messages by using nonces through multiple channels.

Nonces are represented as uint256 from the concatenation of two uint128 : the channelId and the nonceId.

- left most 128 bits : channelId
- right most 128 bits: nonceId



 Example of multi channel nonce, where channelId = 5 and nonceId = 1

The current nonce can be queried using the function:

```
function getNonce(address _address, uint256 _channel) public view returns (uint256)
```

Since the channelId represents the left-most 128 bits, using a minimal value like 1 will return a huge nonce number: 2**128 equal to 340282366920938463463374607431768211456.

After the signed transaction is executed the nonceId will be incremented by +1, this will increment the nonce by 1 as well because the nonceId represents the first 128 bits of the nonce so it will be 340282366920938463463374607431768211457.

```
_nonces[signer][nonce >> 128]++
```

nonce >> 128 represents the channel which the signer chose for executing the transaction. After looking up the nonce of the signer at that specific channel it will be incremented by 1 ++.

For sequential messages, users could use channel 0 and for out-of-order messages they could use channel n.

Important: It's up to the user to choose the channel that he wants to sign multiple sequential orders on it, not necessary 0.

Rationale

There are several factors that motivated the design of LSP25 Execute Relay Call to be developed as its own standard.

Firstly, the idea of meta transaction aims to simplify user transactions on the LUKSO network. Users can send payments and transact between each other without having to worry about gas fees. This simplifies the process of using the LUKSO network, which in turns increase and promote user adoption.

Secondly, gasless Meta-Transactions can assist in minimizing the cost of utilizing the LUKSO network, which is especially essential for low-budget users. This enables protocols and dApps building on LUKSO to offer a better experience for their users or customers.

Finally, by adopting a generic standard for meta transactions, any protocol or dApp can implement gas-less transactions by simply adopting the standard, without relying on custom built solution. This also improve inter-operability between contracts and protocols, since different contracts and different applications rely on the same standard and contract API/ABI to send execute relay calls (= meta transactions) between each others.

Design decision

Several implementation of meta-transactions add the gas parameter in the data of the call to be signed. This is aimed to prevent gas griefing attacks, where a malicious relayer can manipulate the behaviour of the called contract by controlling the gas provided for the transaction.

In the LSP25 standard, the signature format does not require to include the supplied gas. The reasons for this design are the following:

- this add more complexity on the user or dApp interface side, requiring the user/interface to add more logic to provide the gas parameter and sign it.
- when using meta transactions, the user must trust the relayer in the first place, to allow the relayer (an address of a peer or a service) to execute signed transactions on its behalf.

However, not adding the gas parameter implies some considerations that applications that implement the LSP25 standard should be aware of:

- Not including the gas parameter allows the relayer to deternine the gas amount for the transaction.
- If the provided gas is insufficient, the entire transaction could revert, which is the expected behaviour.
- Implementations should be aware of the impact if the contract being called behaves differently based on the gas supplied. A malicious relayer could effectively control that behaviour by adjusting the specified gas when submitting the transaction to the network.

Implementations of the LSP25 standard can add the gas parameter in the data parameter of the signature format to mitigate these risks listed above.

Implementation

An implementation can be found in the [lukso-network/lsp-smart-contracts]

Interface Cheat Sheet

```
interface ILSP25 /* is ERC165 */ {

  function getNonce(address from, uint128 channelId) external view returns (uint256);

  function executeRelayCall(
    bytes calldata signature,
    uint256 nonce,
    uint256 validityTimestamps,
    bytes calldata payload
  )
    external
    payable
    returns (bytes memory);

  function executeRelayCallBatch(
    bytes[] calldata signatures,
    uint256[] calldata nonces,
    uint256[] calldata validityTimestamps,
    uint256[] calldata values,
    bytes[] calldata payloads
  )
    external
    payable
    returns (bytes[] memory);
}
```

Copyright

</file>

<file>

path: /LSPs/LSP-3-Profile-Metadata.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-3-Profile-Metadata.md

---

lip: 3
title: Profile Metadata
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2019-07-12
requires: ERC165, ERC725Y, LSP1, LSP2, LSP5, LSP12

Simple Summary

This standard describes a set of ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) key value stores that are useful to describe a smart contract based profile.

Abstract

This standard, defines a set of data key-value pairs that are useful to create a public on-chain profile, based on an ERC725Account (./LSP-0-ERC725Account.md).

Motivation

This standard describes meta data that can be added to an ERC725Account (https://github.com/lukso-network/LIPs/blob/master/LSPs/LSP-0-ERC725Account.md), to give it a profile like character.

Specification

Every contract that supports the Universal Profile standard SHOULD add the following ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) data keys:

ERC725Y Data Keys

SupportedStandards:LSP3Profile

The supported standard SHOULD be LSP3Profile

```
{
  "name": "SupportedStandards:LSP3Profile",
  "key": "0xeafec4d89fa9619884b600005ef83ad9559033e6e941db7d7c495acdce616347",
  "keyType": "Mapping",
  "valueType": "bytes4",
  "valueContent": "0x5ef83ad9"
}
```

LSP3Profile

A JSON file that describes the profile information, including profile image, background image, description and related links.

```
{
  "name": "LSP3Profile",
  "key": "0x5ef83ad9559033e6e941db7d7c495acdce616347d28e90c7ce47cbfcfcad3bc5",
  "keyType": "Singleton",
  "valueType": "bytes",
  "valueContent": "JSONURL"
}
```

For construction of the JSONURL value see: ERC725Y JSON Schema (./LSP-2-ERC725YJSONSchema.md#JSONURL)

The linked JSON file SHOULD have the following format:

```
{
   "LSP3Profile": {
```

```
    "name": "string", // a self chosen username (will likely be replaced by an ENS name)
    "description": "string" // A description, describing the person, company, organisation or creator of the profile.
    "links": [ // links related to the profile
      {
        "title": "string", // a title for the link.
        "url": "string" // the link itself
      },
      ...
    ],
    "tags": [ "string", "string", ... ], // tags related to the profile
    "avatar": [ // a 3D file avatar mostly in FBX and/or OBJ format, multiple file formats of an avatar can be added
      { // example of a hash based avatar verification
        "verification": {
          "method": 'keccak256(bytes)',
          "data": 'string', // bytes32 hash of the file
        },
        "url": 'string',
        "fileType": 'string'
      },
      { // example of a signature based avatar verification
        "verification": {
          "method": 'ecdsa',
          "data": 'string', // signer that signed the bytes of the file
          "source": 'string' // e.g url returning the signature of the signed file
        },
        "url": 'string',
        "fileType": 'string'
      },
      { // example of a NFT/smart contract based avatar
        "address": Address, // the address of an LSP7 or LSP8
        "tokenId": 32bytes  // (optional) if token contract is an LSP7
      }
    ]
    // below each image type SHOULD have different size of the same image, so that interfaces can choose which one to load for better loading performance
    "profileImage": [ // One image in different sizes, representing the profile.
      { // example of a hash based image verification
        "width": Number,
        "height": Number,
        "url": 'string',
        "verification": {
          "method": 'keccak256(bytes)',
          "data": 'string', // bytes32 hash of the image
        }
      },
      { // example of a signature based image verification
        "width": Number,
        "height": Number,
        "url": 'string',
        "verification": {
```

```
                "method": 'ecdsa',
                "data": 'string', // signer that signed the bytes of the image
                "source": 'string' // e.g url returning the signature of the signed image
            }
        },
        { // example of a NFT/smart contract based image
            "address": Address, // the address of an LSP7 or LSP8
            "tokenId": 32bytes  // (optional) if token contract is an LSP7
        }
    ],
    "backgroundImage": [ // Image in different sizes, that can be used in conjunction with profile image to give a more personal look to a profile.
        { // example of a hash based image verification
            "width": Number,
            "height": Number,
            "url": 'string',
            "verification": {
                "method": 'keccak256(bytes)',
                "data": 'string', // bytes32 hash of the image
            }
        },
        { // example of a signature based image verification
            "width": Number,
            "height": Number,
            "url": 'string',
            "verification": {
                "method": 'ecdsa',
                "data": 'string', // signer that signed the bytes of the image
                "source": 'string' // e.g url returning the signature of the signed image
            }
        },
        { // example of a NFT/smart contract based image
            "address": Address, // the address of an LSP7 or LSP8
            "tokenId": 32bytes  // (optional) if token contract is an LSP7
        }
    ]
  }
}
```

Example:

```
{
  LSP3Profile: {
    name: 'frozeman',
    description: 'The inventor of ERC725 and ERC20...',
    links: [
      { title: 'Twitter', url: 'https://twitter.com/feindura' },
      { title: 'lukso.network', url: 'https://lukso.network' }
    ],
    tags: [ 'brand', 'public profile' ],
    avatar: [
      {
```

```
          verification: {
            method: 'keccak256(bytes)',
            data: '0x98fe032f81c43426fbcfb21c780c879667a08e2a65e8ae38027d4d61cdfe6f55',
          },
          url: 'ifps://QmPJESHbVkPtSaHntNVY5F6JDLW8v69M2d6khXEYGUMn7N',
          fileType: 'fbx'
        }
      ],
      profileImage: [
        {
          width: 1800,
          height: 1013,
          url: 'ifps://QmPJESHbVkPtSaHntNVY5F6JDLW8v69M2d6khXEYGUMn7N',
          verification: {
            method: 'keccak256(bytes)',
            data: '0x98fe032f81c43426fbcfb21c780c879667a08e2a65e8ae38027d4d61cdfe6f55',
          }
        },
        // OR use an NFT as profile image
        {
          "address": 0x1231c7436a77a009a97e48e4e10c92e89fd95fe15, // the address of an LSP7 or
LSP8
          "tokenId": 0xdDe1c7436a77a009a97e48e4e10c92e89fd95fe1556fc5c62ecef57cea51aa37  //
(optional) if token contract is an LSP7
        },
      ],
      backgroundImage: [
        {
          width: 1800,
          height: 1013,
          url: 'ifps://QmPJESHbVkPtSaHntNVY5F6JDLW8v69M2d6khXEYGUMn7N',
          verification: {
            method: 'keccak256(bytes)',
            data: '0x98fe032f81c43426fbcfb21c780c879667a08e2a65e8ae38027d4d61cdfe6f55',
          }
        },
        {
          width: 1024,
          height: 576,
          url: 'ifps://QmZc9uMJxyUeUpuowJ7AD6MKoNTaWdVNcBj72iisRyM9Su',
          verification: {
            method: 'keccak256(bytes)',
            data: '0xfce1c7436a77a009a97e48e4e10c92e89fd95fe1556fc5c62ecef57cea51aa37',
          }
        }
      ]
    }
}
```

Rationale

Profile's metadata is important for creating a verifiable public account that is the source of asset issuance, or a verifiable public appearance. This metadata does not need to belong to a real world person, but gives the account a "recognisable face".

Implementation

A implementation can be found in the [lukso-network/universalprofile-smart-contracts (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/UniversalProfile.sol)](https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/UniversalProfile.sol);
The below defines the JSON interface of the LSP3Profile.

ERC725Y JSON Schema LSP3Profile:

```
[
 {
  "name": "SupportedStandards:LSP3Profile",
  "key": "0xeafec4d89fa9619884b600005ef83ad9559033e6e941db7d7c495acdce616347",
  "keyType": "Mapping",
  "valueType": "bytes4",
  "valueContent": "0x5ef83ad9"
 },
 {
  "name": "LSP3Profile",
  "key": "0x5ef83ad9559033e6e941db7d7c495acdce616347d28e90c7ce47cbfcfcad3bc5",
  "keyType": "Singleton",
  "valueType": "bytes",
  "valueContent": "JSONURL"
 },
 // from LSP12 IssuedAssets
 {
  "name": "LSP12IssuedAssets[]",
  "key": "0x7c8c3416d6cda87cd42c71ea1843df28ac4850354f988d55ee2eaa47b6dc05cd",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
 },
 {
  "name": "LSP12IssuedAssetsMap:<address>",
  "key": "0x74ac2555c10b9349e78f0000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
 },

 // from LSP5 ReceivedAssets
 {
  "name": "LSP5ReceivedAssets[]",
  "key": "0x6460ee3c0aac563ccbf76d6e1d07bada78e3a9514e6382b736ed3f478ab7b90b",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
 },
```

```
{
  "name": "LSP5ReceivedAssetsMap:<address>",
  "key": "0x812c4334633eb816c80d0000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
},

// from ERC725Account
{
  "name": "LSP1UniversalReceiverDelegate",
  "key": "0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
  "keyType": "Singleton",
  "valueType": "address",
  "valueContent": "Address"
}
]
```

Copyright

</file>

<file>

path: /LSPs/LSP-4-DigitalAsset-Metadata.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-4-DigitalAsset-Metadata.md

---

lip: 4
title: Digital Asset Metadata
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2020-07-21
requires: ERC725Y, LSP2

Simple Summary

This standard describes a set of ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) data key-value pairs that describe a digital asset.

Abstract

This standard, defines a set of data key-value pairs that are useful to describe a digital asset.

Motivation

This standard aims to create a better version of tokens and NFTs to allow more functionality to be attached to those assets, and the ability for those assets to change over time.
As NFTs mostly have a creator, those creators should be able to improve the assets (link better 3D files, as 3D file standards improve), or change attributes.
One could even think of a smart contract system that can increase attributes based on certain inputs automatically.

An LSP4 Digital Asset is controlled by a single owner, expected to be a ERC725 (https://github.com/ERC725Alliance/ERC725/blob/main/docs/ERC-725.md) smart contract. This owner is able to setData(...) (https://github.com/ERC725Alliance/ERC725/blob/main/docs/ERC-725.md#setdata), and therefore change values of data keys, and can potentially mint new items.

Specification

ERC725Y Data Keys

SupportedStandards:LSP4DigitalAsset

The supported standard SHOULD be LSP4DigitalAsset


```
{
  "name": "SupportedStandards:LSP4DigitalAsset",
  "key": "0xeafec4d89fa9619884b60000a4d96624a38f7ac2d8d9a604ecf07c12c77e480c",
  "keyType": "Mapping",
  "valueType": "bytes4",
  "valueContent": "0xa4d96624"
}
```


LSP4TokenName

A string representing the name of the token.

The LSP4TokenName data key is OPTIONAL. If this data key is present and used, the following requirements apply:

Requirements

- the value of the LSP4TokenName MUST NOT be changeable and set only on deployment or during initialization of the token.


```
{
  "name": "LSP4TokenName",
  "key": "0xdeba1e292f8ba88238e10ab3c7f88bd4be4fac56cad5194b6ecceaf653468af1",
  "keyType": "Singleton",
  "valueType": "string",
  "valueContent": "String"
}
```


This MUST NOT be changeable, and set only during initialization of the token.

LSP4TokenSymbol

A string representing the symbol for the token.

The LSP4TokenSymbol data key is OPTIONAL. If this data key is present and used, the following requirements and recommendations apply:

Requirements

- the value of the LSP4TokenSymbol MUST NOT be changeable and set only on deployment or during initialization of the token.

Recommendations

- Symbols SHOULD be UPPERCASE.
- Symbols SHOULD NOT contain any white spaces.
- Symbols SHOULD contain only ASCII characters.


```
{
  "name": "LSP4TokenSymbol",
  "key": "0x2f0a68ab07768e01943a599e73362a0e17a63a72e94dd2e384d2c1d4db932756",
  "keyType": "Singleton",
  "valueType": "string",
  "valueContent": "String"
}
```


LSP4TokenType

A string representing the type of token that this contract represents.
NOTE: More token types COULD be added later.

Requirements

This MUST NOT be changeable, and set only during initialization of the token contract.

Value	Type	Description
0	Token	Only valid for LSP7, meaning its a generic token, where the LSP4Metadata represents the token information.
1	NFT	If the contract is LSP7 or LSP8, then the LSP4Metadata represents the information of a single NFT, that has multiple ownable amounts or IDs. If its an LSP8 each individual token ID COULD have its own custom metadata specific for the token ID, but MUST NOT be a different NFT, just different metadata per item in the NFT. Those COULD be set using LSP8TokenIdType and LSP8MetadataTokenURI. See LSP8 for details (./LSP-8-IdentifiableDigitalAsset.md). If its an LSP7 contract, the decimals function MUST return 0.
2	Collection	Only valid for LSP8. The LSP4Metadata represents the information of a the collection, and each individual token ID represents its own NFT and MUST have its own metadata set using LSP8TokenIdType and LSP8MetadataTokenURI. See LSP8 for details (./LSP-8-IdentifiableDigitalAsset.md).

<!-- - LSP26NFT -->


```
{
  "name": "LSP4TokenType",
  "key": "0xe0261fa95db2eb3b5439bd033cda66d56b96f92f243a8228fd87550ed7bdfdb3",
  "keyType": "Singleton",
  "valueType": "uint256",
  "valueContent": "Number"
}
```

LSP4Metadata

The description of the asset.

```
{
  "name": "LSP4Metadata",
  "key": "0x9afb95cacc9f95858ec44aa8c3b685511002e30ae54415823f406128b85b238e",
  "keyType": "Singleton",
  "valueType": "bytes",
  "valueContent": "JSONURL"
}
```

For more informations on how to construct the JSONURL, see: [ERC725Y JSON Schema > valueContent > JSONURL (./LSP-2-ERC725YJSONSchema.md#JSONURL)]

The linked JSON file SHOULD have the following format:

> Note: the "attributes" field is OPTIONAL.

```
{
  "LSP4Metadata": {
    "name": "string", // name of the DigitalAsset if not defined in LSP4TokenName
    "description": "string",
    "links": [ // links related to DigitalAsset
      {
        "title": "string", // a title for the link.
        "url": "string" // the link itself
      },
      ...
    ],
    "icon": [ // SHOULD be used for LSP7 icons
      // multiple sizes of the same icon
      { // example of a verificationData based image verification
        "width": Number,
        "height": Number,
        "url": 'string',
        "verification": {
          "method": 'keccak256(bytes)',
          "data": 'string', // bytes32 hash of the image
        }
      },
      { // example of a signature based image verification
        "width": Number,
        "height": Number,
        "url": 'string',
        "verification": {
          "method": 'ecdsa',
          "data": 'string', // signer that signed the bytes of the image
          "source": 'string' // e.g url returning the signature of the signed image
        }
      },
```

```
        { // example of a NFT/smart contract based image
            "address": Address, // the address of an LSP7 or LSP8
            "tokenId": 32bytes  // (optional) if token contract is an LSP7
        }
        ...
    ],
    "images": [ // COULD be used for LSP8 NFT art
        // multiple images in different sizes, related to the DigitalAsset, image 0, should be the main image
        // array of different sizes of the same image
        [
            { // example of a verificationData based image verification
                "width": Number,
                "height": Number,
                "url": 'string',
                "verification": {
                    "method": 'keccak256(bytes)',
                    "data": 'string', // bytes32 hash of the image
                }
            },
            { // example of a signature based image verification
                "width": Number,
                "height": Number,
                "url": 'string',
                "verification": {
                    "method": 'ecdsa',
                    "data": 'string', // signer that signed the bytes of the image
                    "source": 'string' // e.g url returning the signature of the signed image
                }
            },
            { // example of a NFT/smart contract based image
                "address": Address, // the address of an LSP7 or LSP8
                "tokenId": 32bytes  // (optional) if token contract is an LSP7
            }
            ...
        ],
        [...]
    ],
    "assets": [ // SHOULD be used for any assets of the token (e.g. 3d assets, high res pictures or music,
etc)
        {
            "url": 'string',
            "fileType": 'string',
            "verification": {
                "method": 'keccak256(bytes)',
                "data": 'string', // bytes32 hash of the asset
            }
        },
        { // example of a NFT/smart contract based asset
            "address": Address, // the address of an LSP7 or LSP8
            "tokenId": 32bytes  // (optional) if token contract is an LSP7
        }
    ],
```

```
    "attributes": [
      {
        "key": "string",    // name of the attribute
        "value": "string", // value assigned to the attribute
        "type": "string | number | boolean",   // for encoding/decoding purposes
      },
    ]
    ...
  }
}
```

Example:

```
{
  LSP4Metadata: {
    description: 'The first digial golden pig.',
    links: [
      { title: 'Twitter', url: 'https://twitter.com/goldenpig123' },
      { title: 'goldenpig.org', url: 'https://goldenpig.org' }
    ],
    icon: [
      {
        width: 256,
        height: 256,
        url: 'ifps://QmW5cF4r9yWeY1gUCtt7c6v3ve7Fzdg8CKvTS96NU9Uiwr',
        verification: {
          method: 'keccak256(bytes)',
          data: '0x01299df007997de92a820c6c2ec1cb2d3f5aa5fc1adf294157de563eba39bb6f',
        }
      }
    ],
    images: [ // COULD be used for LSP8 NFT art
      [
        {
          width: 1024,
          height: 974,
          url: 'ifps://QmW4wM4r9yWeY1gUCtt7c6v3ve7Fzdg8CKvTS96NU9Uiwr',
          verification: {
            method: 'keccak256(bytes)',
            data: '0xa9399df007997de92a820c6c2ec1cb2d3f5aa5fc1adf294157de563eba39bb6e',
          }
        },
        ... // more image sizes
      ],
      ... // more images
    ],
    assets: [{
      verification: {
        method: 'keccak256(bytes)',
        data: '0x98fe032f81c43426fbcfb21c780c879667a08e2a65e8ae38027d4d61cdfe6f55',
      },
      url: 'ifps://QmPJESHbVkPtSaHntNVY5F6JDLW8v69M2d6khXEYGUMn7N',
```

```
        fileType: 'fbx'
      }],
      attributes: [
        {
          key: 'Standard type',
          value: 'LSP',
          type: "string"
        },
        {
          key: 'Standard number',
          value: 4,
          type: "number"
        },
        {
          key: '     ',
          value: true,
          type: "boolean"
        }
      ]
    }
}
```

LSP4Creators[]

An array of (ERC725Account (./LSP-0-ERC725Account.md)) addresses that defines the creators of the digital asset.

```
{
  "name": "LSP4Creators[]",
  "key": "0x114bd03b3a46d48759680d81ebb2b414fda7d030a7105a851867accf1c2352e7",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
}
```

For more informations about how to access each index of the LSP4Creators[] array, see ERC725Y JSON Schema > keyType: Array (./LSP-2-ERC725YJSONSchema.md#Array)

LSP4CreatorsMap

References the creator addresses for this asset. This data key exists so that smart contracts can detect whether the address of a creator is present in the LSP4Creators[] array without looping all over it on-chain. Moreover, it helps to identify at which index in the LSP4Creators[] the creator address is located for easy access and to change or remove this specific creator from the array. Finally, it also allows the detection of the interface supported by the creator.

The valueContent MUST be constructed as follows: bytes4(standardInterfaceId) + uint128(indexNumber). Where:

- standardInterfaceId = if the creator address is a smart contract, the ERC165 interface ID (https://eips.ethereum.org/EIPS/eip-165) of the standard that the smart contract implements. Otherwise 0xffffffff in the case where the creator address is:

    - an Externally Owned Account, or
    - a contract implementing no ERC165 interface ID.

- indexNumber = the index in the LSP4Creators[] Array

```
{
  "name": "LSP4CreatorsMap:<address>",
  "key": "0x6de85eaf5d982b4e5da00000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
}
```

Rationale

There can be many token implementations, and this standard fills a need for common metadata describing issuers, creators and the token itself.

Implementation

An implementation can be found in the lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/LSP4DigitalAssetMetadata/LSP4DigitalAssetMetadata.sol) repository.
The below defines the JSON interface of the LSP4DigitalAssetMetadata.

ERC725Y JSON Schema LSP4DigitalAssetMetadata:

```
[
 {
  "name": "SupportedStandards:LSP4DigitalAsset",
  "key": "0xeafec4d89fa9619884b60000a4d96624a38f7ac2d8d9a604ecf07c12c77e480c",
  "keyType": "Mapping",
  "valueType": "bytes4",
  "valueContent": "0xa4d96624"
 },
 {
  "name": "LSP4TokenName",
  "key": "0xdeba1e292f8ba88238e10ab3c7f88bd4be4fac56cad5194b6ecceaf653468af1",
  "keyType": "Singleton",
  "valueType": "string",
  "valueContent": "String"
 },
 {
  "name": "LSP4TokenSymbol",
  "key": "0x2f0a68ab07768e01943a599e73362a0e17a63a72e94dd2e384d2c1d4db932756",
  "keyType": "Singleton",
  "valueType": "string",
  "valueContent": "String"
 },
 {
  "name": "LSP4Metadata",
  "key": "0x9afb95cacc9f95858ec44aa8c3b685511002e30ae54415823f406128b85b238e",
  "keyType": "Singleton",
  "valueType": "bytes",
  "valueContent": "JSONURL"
 },
 {
  "name": "LSP4Creators[]",
  "key": "0x114bd03b3a46d48759680d81ebb2b414fda7d030a7105a851867accf1c2352e7",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
 },
 {
  "name": "LSP4CreatorsMap:<address>",
  "key": "0x6de85eaf5d982b4e5da00000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
 }
]
```


Copyright

</file>

<file>

path: /LSPs/LSP-5-ReceivedAssets.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-5-ReceivedAssets.md

---

lip: 5
title: Received Assets
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network)
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2021-09-21
requires: LSP2

Simple Summary

This standard describes a set of ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) data key values to store addresses of received assets in a ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract.

Abstract

This data key value standard describes a set of data keys that can be added to an ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract to describe received assets:

- LSP5ReceivedAssets[] is an LSP2 array (./LSP-2-ERC725YJSONSchema.md) of addresses.
- LSP5ReceivedAssetsMap is a dynamic address mapping, which contains:
    - an ERC165 interface ID (https://eips.ethereum.org/EIPS/eip-165) to easily identify the standard used by the mapped asset smart contract
    - and the index in the LSP5ReceivedAssets[] array.

The data key LSP5ReceivedAssetsMap exists so that smart contracts can detect if an address is present in the array (e.g. as done in the LSP1-UniversalReceiverDelegate (./LSP-1-UniversalReceiver.md)).

Motivation

This standard allows to create a decentralised portfolio of owned assets by a smart contract. See LSP3 - Profile Metadata (./LSP-3-Profile-Metadata.md), or LSP9 Vault (./LSP-9-Vault.md).

Specification

Every contract that supports the ERC725Account SHOULD have the following data keys:

ERC725Y Data Keys

LSP5ReceivedAssets[]

An array of received smart contract assets, like tokens (*e.g.: LSP7 Digital Assets (./LSP-7-DigitalAsset)*) and NFTs (*e.g.: LSP8 Identifiable Digital Assets (./LSP-8-IdentifiableDigitalAsset)*).

```
{
  "name": "LSP5ReceivedAssets[]",
  "key": "0x6460ee3c0aac563ccbf76d6e1d07bada78e3a9514e6382b736ed3f478ab7b90b",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
}
```

For more info about how to access each index of the LSP5ReceivedAssets[] array, see ERC725Y JSON Schema > keyType: Array (https://github.com/lukso-network/LIPs/blob/master/LSPs/LSP-2-ERC725YJSONSchema.md#array)

LSP5ReceivedAssetsMap

References received smart contract assets, like tokens (*e.g.: LSP7 Digital Assets (./LSP-7-DigitalAsset)*) and NFTs (*e.g.: LSP8 Identifiable Digital Assets (./LSP-8-IdentifiableDigitalAsset)*). This data key exists so that smart contracts can detect whether the address of an asset is present in the LSP5ReceivedAssets[] array without looping all over it on-chain. Moreover, it helps to identify at which index in the LSP5ReceivedAssets[] the asset address is located for easy access and to change or remove this specific asset from the array. Finally, it also allows the detection of the interface supported by the asset.

The data value MUST be constructed as follows: bytes4(standardInterfaceId) + uint128(indexNumber). Where:

- standardInterfaceId = the ERC165 interface ID (https://eips.ethereum.org/EIPS/eip-165) of the standard that the token or asset smart contract implements (if the ERC165 interface ID is unknown, standardInterfaceId = 0xffffffff).
- indexNumber = the index in the LSP5ReceivedAssets[] Array

Value example: 0x5fcaac27000000000000000c (interfaceId: 0x5fcaac27 for a LSP7 (./LSP-7-DigitalAsset.md) token, index position 0x000000000000000c = 12).

```
{
  "name": "LSP5ReceivedAssetsMap:<address>",
  "key": "0x812c4334633eb816c80d0000<address>",
  "keyType": "Mapping",
  "valueType": "(bytes4,uint128)",
  "valueContent": "(Bytes4,Number)"
}
```

Rationale

Implementation

An implementation of setting received assets from a smart contract can be found in the LSP1UniversalReceiverDelegate (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/LSP1UniversalReceiver/LSP1UniversalReceiverDelegateUP/LSP1UniversalRe smart contract.

ERC725Y JSON Schema LSP5ReceivedAssets:

```
[
  {
    "name": "LSP5ReceivedAssets[]",
    "key": "0x6460ee3c0aac563ccbf76d6e1d07bada78e3a9514e6382b736ed3f478ab7b90b",
    "keyType": "Array",
    "valueType": "address",
    "valueContent": "Address"
  },
  {
    "name": "LSP5ReceivedAssetsMap:<address>",
    "key": "0x812c4334633eb816c80d0000<address>",
    "keyType": "Mapping",
    "valueType": "(bytes4,uint128)",
    "valueContent": "(Bytes4,Number)"
  },
]
```


Copyright

</file>

<file>

path: /LSPs/LSP-6-KeyManager.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-6-KeyManager.md

---

lip: 6
title: Key Manager
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network), Jean Cavallera contact.cj42@protonmail.com (mailto:contact.cj42@protonmail.com)
discussions-to:
status: Draft
type: LSP
created: 2021-08-03
requires: ERC165, ERC1271, LSP2, LSP20, LSP25

Simple Summary

This standard describes a KeyManager contract with a set of pre-defined permissions for addresses. A KeyManager contract can control an ERC725Account (./LSP-0-ERC725Account.md) like account, or any other ERC725 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract.

Abstract

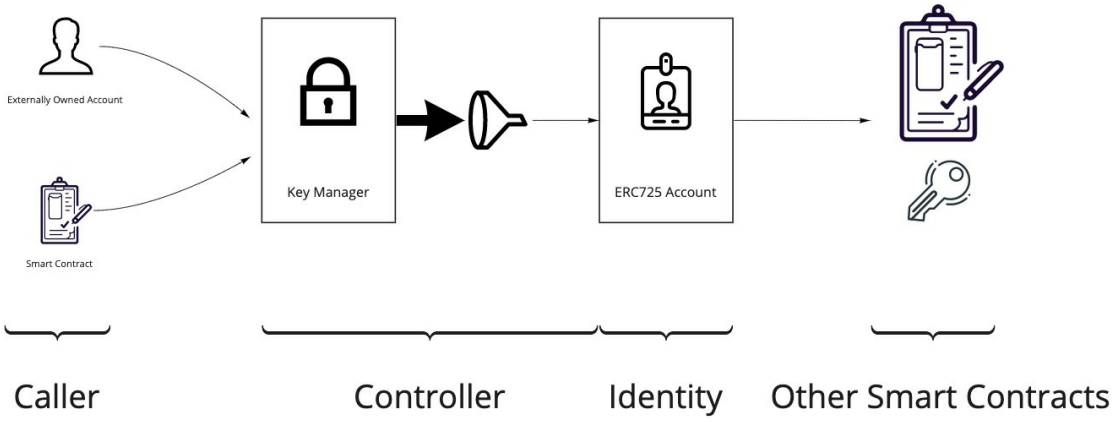
This standard allows for controlling addresses to be restricted through multiple permissions, to act on and through this KeyManager on a controlled smart contract (for example an ERC725Account (./LSP-0-ERC725Account.md)).

The KeyManager functions as a gateway for the ERC725Account (./LSP-0-ERC725Account.md) restricting an address actions based on set permissions.

Permissions are described in the Permissions section. Furthermore addresses can be restricted to only talk to certain other smart contracts or address, specific functions or smart contracts supporting only specifc standard interfaces.

The Permissions are stored under the ERC725Y data key-value store of the linked ERC725Account (./LSP-0-ERC725Account.md), and can therefore survive an upgrade to a new KeyManager contract.

The flow of a transactions is as follows:



Motivation

The benefit of a KeyManager is to externalise the permission logic from ERC725Y and X (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) contracts (such as an ERC725Account (./LSP-0-ERC725Account.md)). This allows for such logic to be upgraded without needing to change the core ERC725Account (./LSP-0-ERC725Account.md) contract.

Storing the permissions at the core ERC725Account (./LSP-0-ERC725Account.md) itself, allows it to survive KeyManager upgrades and opens the door to add additional KeyManager logic in the future, without loosing already set address permissions.

Specification

LSP6-KeyManager interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0x23f34c62.

Smart contracts implementing the LSP6 standard MUST implement and support the following standard and their interfaces:

- ERC165 (https://eips.ethereum.org/EIPS/eip-165)
- ERC1271 (https://eips.ethereum.org/EIPS/eip-1271)
- LSP20-CallVerification (./LSP-20-CallVerification.md)
- LSP25-ExecuteRelayCall (./LSP-25-ExecuteRelayCall.md)
- the LSP6 interface functions defined below.

Every contract that supports the LSP6 standard MUST implement:

Methods

target

```
function target() external view returns (address)
```

Returns the address of the target smart contract controlled by this Key Manager. The controlled smart contract can be one of the following:

- ERC725X contract
- ERC725Y contract
- an ERC725 contract, implementing both ERC725X and ERC725Y (e.g: an ERC725Account (./LSP-0-ERC725Account.md)).

isValidSignature

```
function isValidSignature(bytes32 hash, bytes memory signature) external view returns (bytes4);
```

This function is part of the ERC1271 (https://eips.ethereum.org/EIPS/eip-1271) specification, with additional requirements as follows:

- MUST recover the address from the hash and the signature and return the ERC1721 success value (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1271.md#specification) if the address recovered have the SIGN Permission, if not, MUST return the ERC1271 fail value (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1271.md#specification).

lsp20VerifyCall

```
function lsp20VerifyCall(address caller, uint256 value, bytes memory receivedCalldata) external returns (bytes4 returnedStatus);
```

This function is part of the LSP20-CallVerification (./LSP-20-CallVerification.md) specification, with additional requirements as follows:

- MUST be called only by the linked target.
- MUST verify the permission of the caller based on the receivedCalldata as defined in the permission section.
- MUST emit the PermissionsVerified event after verifying permissions.
- MUST set the reentrancy guard to true if the first 4 bytes of receivedCalldata are any function other than setData(..)/setDataBatch(..) and check for reentrancy permission if the call was reentrant.
- MUST return the success value with the 0x01 bytes that indicates that lsp20VerifyCallResult(..) MUST be invoked.

lsp20VerifyCallResult

```
function lsp20VerifyCallResult(bytes32 callHash, bytes memory callResult) external returns (bytes4 returnedStatus);
```

This function is part of the LSP20-CallVerification (./LSP-20-CallVerification.md) specification, with additional requirements as follows:

- MUST be called only by the linked target.
- MUST set the reentrancy guard to false if it's set to true.
- MUST return the success value.

executeRelayCall

```
function executeRelayCall(
    bytes memory signature,
    uint256 nonce,
    uint256 validityTimestamps,
    bytes memory payload
)
    external
    payable
    returns (bytes memory)
```

This function is part of the LSP25-ExecuteRelayCall (./LSP-25-ExecuteRelayCall.md) specification, with additional requirements as follows.

- The Key Manager's address MUST be used for the <Implementation address> in the LSP25 Signature Format (./LSP-25-ExecuteRelayCall.md#signature-format) when generating signature for execute relay calls.
- The address of the signer recovered from the signature MUST have the right permissions to execute the calldata payload.
- The event PermissionsVerified MUST be emitted after verifying the permissions of the signer address recovered.

executeRelayCallBatch

```
function executeRelayCallBatch(
    bytes[] memory signatures,
    uint256[] memory nonces,
    uint256[] memory validityTimestamps,
    uint256[] memory values,
    bytes[] memory payloads
)
    external
    payable
    returns (bytes[] memory)
```

This function is part of the LSP25-ExecuteRelayCall (./LSP-25-ExecuteRelayCall.md) specification, with additional requirements as follows.

- The Key Manager's address MUST be used for the <Implementation address> in the LSP25 Signature Format (./LSP-25-ExecuteRelayCall.md#signature-format) to generate each signatures in the batch.
- The address of the signer recovered from each signature in the signatures[] array MUST have the right permissions to execute the the associated calldata payload in the payloads[] array.
- The event PermissionsVerified MUST be emitted after verifying the permissions of each signer address recovered.

execute

```
function execute(bytes memory payload) external payable returns (bytes memory)
```

Execute a payload on the linked target contract.

MUST fire the PermissionsVerified event.

Parameters:

- payload: The abi-encoded function call to be executed on the linked target contract.

Returns: bytes , the returned data as abi-decoded bytes of the call on ERC725 smart contract, if the call succeeded, otherwise revert with a reason-string.

Requirements:

- The first 4 bytes of the payload payload MUST correspond to one of the function selector on the ERC725 smart contract such as:
    - setData(bytes32,bytes) (./LSP-0-ERC725Account.md#setdata)
    - setDataBatch(bytes32[],bytes[]) (./LSP-0-ERC725Account.md#setdatabatch)
    - execute(uint256,address,uint256,bytes) (./LSP-0-ERC725Account.md#execute)
    - transferOwnership(address) (./LSP-0-ERC725Account.md#transferownership)
    - acceptOwnership() (./LSP-0-ERC725Account.md#acceptownership)
- MUST send the value passed by the caller to the call on the linked target.

    > Non payable functions will revert in case of calling them and passing value along the call.

- The caller MUST have permission for the action being executed. Check Permissions to know more.

executeBatch

```
function executeBatch(uint256[] memory values, bytes memory payloads[]) external payable returns (bytes[] memory)
```

Execute a batch of payloads on the linked target contract.

MUST fire the PermissionsVerified event on each iteration.

Parameters:

- values: The array of values to be sent to the target contract along the call on each iteration.
- payloads: The array of calldata payloads to be executed on the target contract on each iteration.

Returns: bytes[] , an array of returned data as abi-decoded array of bytes[] of the call on ERC725 smart contract, if the calls succeeded, otherwise revert with a reason-string.

Requirements:

- The parameters length MUST be equal.
- The sum of each element of the values array MUST be equal to the value sent to the function.
- MUST comply to the requirements of the execute(bytes) function.

Events

PermissionsVerified

```
event PermissionsVerified(address indexed signer, uint256 indexed value, bytes4 indexed selector);
```

MUST be fired when the permissions of a call was successfully verified.

Permissions

The permissions MUST be checked against the following address, depending on the function/method being called:

- against the caller parameter in the cases of lsp20VerifyCall(address,uint256,bytes).
- against the caller in the cases of execute(bytes) and executeBatch(uint256[],bytes[]).
- against the signer address, recovered from the signature and the digest, in the cases of executeRelayCall(bytes,uint256,bytes) and executeRelayCallBatch(bytes[],uint256[],uint256[],bytes[]).

The permissions MUST be stored as BitArray (./LSP-2-ERC725YJSONSchema.md#bitarray) under the AddressPermissions:Permissions:<address> data key on the target.

> In the descriptions of each permissions below, the term *controller address* refers to an address that has some permissions set under the target contract linked to this Key Manager.

CHANGEOWNER

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000001

- Allows changing the owner of the target contract by calling transferOwnership(address) (./LSP-0-ERC725Account.md#transferownership) and acceptOwnership() (./LSP-0-ERC725Account.md#acceptownership).

ADDCONTROLLER

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000002

- Allows incrementing the length of the AddressPermissions[] data key and adding an address at a new index of the array.
- Allows adding permissions for a new controller address under the AddressPermissions:Permissions:<address> data key.
- Allows adding the restrictions for the call operations such as CALL, STATICCALL, and DELEGATECALL and SETDATA permissions stored respectively under the AddressPermissions:AllowedCalls:<address> and AddressPermissions:AllowedERC725YDataKeys:<address>.

The value of these data keys SHOULD be validated before being set to avoid edge cases.

> All the actions above are done using the setData function

EDITPERMISSIONS

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000004

- Allows decrementing the length of the AddressPermissions[] data key and changing the address at an existing index of the array.
- Allows changing permissions for an existing controller address under the AddressPermissions:Permissions:<address> data key.
- Allows changing existing restrictions for the call operations such as CALL, STATICCALL, and DELEGATECALL and SETDATA permissions stored respectively under the AddressPermissions:AllowedCalls:<address> and AddressPermissions:AllowedERC725YDataKeys:<address>.

The value of these data keys SHOULD be validated before being set to avoid edge cases.

> All the actions above are done using the setData function

ADDEXTENSIONS

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000008

- Allows adding new extension address/es for new function selectors stored under LSP17Extension (./LSP-0-ERC725Account.md#lsp17extension) data key.

CHANGEEXTENSIONS

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000010

- Allows changing existing extension address/es for function selectors stored under LSP17Extension (./LSP-0-ERC725Account.md#lsp17extension) data key.

ADDUNIVERSALRECEIVERDELEGATE

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000020

- Allows adding new UniversalReceiverDelegate address/es stored under new LSP1UniversalReceiverDelegate (./LSP-0-ERC725Account.md#lsp1universalreceiverdelegate) and Mapped LSP1UniversalReceiverDelegate (./LSP-0-ERC725Account.md#mapped-lsp1universalreceiverdelegate) data keys.

CHANGEUNIVERSALRECEIVERDELEGATE

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000040

- Allows changing existing UniversalReceiverDelegate address/es stored under LSP1UniversalReceiverDelegate (./LSP-0-ERC725Account.md#lsp1universalreceiverdelegate) and Mapped LSP1UniversalReceiverDelegate (./LSP-0-ERC725Account.md#mapped-

lsp1universalreceiverdelegate) data keys.

REENTRANCY

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000080

- Allows reentering the public execute(bytes), executeBatch(uint256[],bytes[]), executeRelayCall(bytes,uint256,bytes) and executeRelayCallBatch(bytes[],uint256[],uint256[],bytes[]) functions.

SUPER_TRANSFERVALUE

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000100

- Allows transferring value from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target without any restrictions.

TRANSFERVALUE

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000200

- Allows transferring value from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target with restricting to specific standards, addresses or functions.

> Check AddressPermissions:AllowedCalls:<address> for more information about the restrictions.

SUPER_CALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000400

- Allows executing a payload with CALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) operation from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target without any restrictions.

CALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000000800

- Allows executing a payload with CALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) operation from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target with restricting to specific standards, addresses or functions.

> Check AddressPermissions:AllowedCalls:<address> to know more about the restrictions.

SUPER_STATICCALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000001000

- Allows executing a payload with STATICCALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) operation from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target without any restrictions.

STATICCALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000002000

- Allows executing a payload with STATICCALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) operation from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target with restricting to specific standards, addresses or functions.

> Check AddressPermissions:AllowedCalls:<address> for more information about the restrictions.

SUPER_DELEGATECALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000004000

- Allows executing a payload with DELEGATECALL (https://solidity-by-example.org/delegatecall/) operation from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target without any restrictions.

DELEGATECALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000008000

- Allows executing a payload with DELEGATECALL (https://solidity-by-example.org/delegatecall/) operation from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target with restricting to specific standards, addresses or functions.

> Check AddressPermissions:AllowedCalls:<address> for more information about the restrictions.

DEPLOY

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000010000

- Allows creating a contract with CREATE (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) and CREATE2 (https://eips.ethereum.org/EIPS/eip-1014) operations from the target contract through execute(..) (./LSP-0-ERC725Account.md#execute) function of the target.

The permission SUPER_TRANSFERVALUE is REQUIRED to fund the contract with some native tokens while deploying it.

SUPER_SETDATA

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000020000

- Allows setting data keys on the target contract through setData(bytes32,bytes) (./LSP-0-ERC725Account.md#execute) and setData(bytes32[],bytes[]) functions without any restrictions on the data keys to set.

The data keys related to permissions, extensions, UniversalReceiverDelegate MUST be checked with their own permission.

SETDATA

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000040000

- Allows setting data keys on the target contract through setData(bytes32,bytes) (./LSP-0-ERC725Account.md#execute) and setData(bytes32[],bytes[]) functions with restricting to specific data keys to set.

The data keys related to permissions, extensions, UniversalReceiverDelegate MUST be checked with their own permission.

> Check AddressPermissions:AllowedERC725YDataKeys:<address> for more information about the restrictions.

ENCRYPT

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000080000

- Allows encrypting data to be used for on/off-chain purposes.

DECRYPT

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000100000

- Allows decrypting data to be used for on/off-chain purposes.

SIGN

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000200000

- Validates the signed messages by the target contract to be used for on/off-chain purposes.

EXECUTE_RELAY_CALL

BitArray representation: 0x0000000000000000000000000000000000000000000000000000000000400000

- Allows a controller's signed relay calls to be executable. This permission MUST be checked against the controller that signed the relay call.

ERC725Y Data Keys

The permissions that the KeyManager reads, are stored on the controlled-contracts ERC725Y data key value store (for example an ERC725Account (./LSP-0-ERC725Account.md))

The following ERC725Y data keys are used to read permissions and restrictions of certain addresses.

These data keys are based on the LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) standard, and use the key type MappingWithGrouping (./LSP-2-ERC725YJSONSchema.md#mappingwithgrouping)

AddressPermissions[]

```
{
  "name": "AddressPermissions[]",
  "key": "0xdf30dba06db6a30e65354d9a64c609861f089545ca58c6b4dbe31a5f338cb0e3",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
}
```

Contains an array of addresses, that have some permission set. This is mainly useful for interfaces to know which address holds which permissions.

Note that this data key is OPTIONAL. It is not necessary to add a controller under this data key if setting a new controller under AddressPermissions:Permissions:<address>

For more information about how to access each index of the AddressPermissions[] array, see: ERC725Y JSON Schema > keyType Array (./LSP-2-ERC725YJSONSchema.md#array)

AddressPermissions:Permissions:<address>

```
{
  "name": "AddressPermissions:Permissions:<address>",
  "key": "0x4b80742de2bf82acb3630000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "bytes32",
  "valueContent": "BitArray"
}
```

Contains a set of permissions for an address. Permissions defines what an address can do on the target contract (*eg: edit the data key-value store via SETDATA*), or can perform on behalf of the target.

Since the valueType of this data key is bytes32, up to 255 different permissions can be defined. This includes the default permissions defined. Custom permissions can be defined on top of the default one.

AddressPermissions:AllowedCalls:<address>

```
{
  "name": "AddressPermissions:AllowedCalls:<address>",
  "key": "0x4b80742de2bf393a64c70000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "(bytes4,address,bytes4,bytes4)[CompactBytesArray]",
  "valueContent": "(BitArray,Address,Bytes4,Bytes4)"
}
```

Contains a compact bytes array of call restrictions, addresses, interfaceIds and function selectors a controller address is allowed to execute and interact with.

Each allowed call is made of four elements concatenated together as a tuple that forms a final bytes32 long value.

The full list of allowed calls MUST be constructed as a CompactBytesArray (./LSP-2-ERC725YJSONSchema.md#bytescompactbytesarray) according to LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) as follow:

```
<0020> <bytes4 restrictionOperations> <bytes20 allowedAddress> <bytes4 allowedInterfaceId> <bytes4 allowedFunction> <0020> ... <0020> ...
```

> NB: the three dots ... are placeholders for <bytes4 callRestrictions> <bytes20 allowedAddress> <bytes4 allowedInterfaceId> <bytes4 allowedFunction> and used for brevity.

- 0020: 0020 in decimals is 32, the total number of bytes of the four elements below concatenated together.
- callRestrictions: A bitArray that represents the list of restricted call types applied for this allowed call (address - interfaceId - function).

  The call restrictions are defined with specific bits, starting from the least significant bit (the rightmost bit):

  - transferValue 00000001
  - call 00000010
  - staticcall 00000100
  - delegatecall 00001000

  Custom implementations of the LSP6 Standard can add more operations as needed.
- allowedAddress: The address called by the target contract.
- allowedInterfaceId: The ERC165 interface id being supported by the contract called from the target.
- allowedFunction: The function selector being called on the contract called by the target contract.
- If the value of the data key is empty, execution is disallowed.
- restrictionOperations MUST NOT discarded.
- Check is discarded for an element if the value is full ff bytes. e.g, 0xffffffff for interfaceIds and function selectors and 0xffffffffffffffffffffffffffffffffffffffff for addresses. There MUST be at most 2 discarded checks, meaning 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff data key is disallowed.

Example 1:

If controller A has CALL permission, and have the following value for AllowedCalls:

```
0x002000000002cafecafecafecafecafecafecafecafecafecafe11223344bb11bb11
```

The restrictionOperations is 0x00000002, meaning that the restrictions above only applies when the operation CALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) is being executed.

The controller A is allowed to interact via the CALL operation with the function selector 0xbb11bb11 on the 0xcafecafecafecafecafecafecafecafecafecafe address as long as the address supports 0x11223344 interfaceId through ERC165 (https://eips.ethereum.org/EIPS/eip-165).

Example 2:

If controller B has DELEGATECALL permission, and have the following value for AllowedCalls:

```
0x002000000004cafecafecafecafecafecafecafecafecafefffffffffffff002000000004ffffffffffffffffffffffffffffffffffffffff68686868ffffffff
```

The restrictionOperations on both elements is 0x00000004, it means that the restrictions above only applies when the operation DELEGATECALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) is being executed.

The controller B is allowed to interact with:

- the address 0xcafecafecafecafecafecafecafecafecafecafe only via DELEGATECALL without any restriction on the interfaceId or the function selector.
- any address supporting the 0x68686868 interfaceId without any restriction on the function.

Example 3:

If controller B has DELEGATECALL permission, and have the following value for AllowedCalls:

```
0x002000000002cafecafecafecafecafecafecafecafecafecafe11223344bb11bb11
```

The restrictionOperations is 0x00000002, it means that the restrictions above only applies when the operation CALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) is being executed.

Given that the controller B have the DELEGATECALL Permission and the restrictions only applies for the CALL operation, any execution of DELEGATECALL operation will fail even to the function 0xbb11bb11 on 0xcafecafecafecafecafecafecafecafecafecafe address.

Example 4:

If controller B has TRANSFERVALUE and CALL permissions, and have the following value for AllowedCalls:

```
0x002000000003cafecafecafecafecafecafecafecafecafecafe11223344bb11bb11
```

The callRestrictions is 0x00000003, it means that the restrictions above applies when the operation CALL (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) or TRANSFERVALUE (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#execute) is being executed.

The controller B is allowed to CALL or TransferValue to:

- the contract deployed at address 0xcafecafecafecafecafecafecafecafecafecafe as long as this contract supports the interfaceId 0x11223344 through ERC165 (https://eips.ethereum.org/EIPS/eip-165).

- only the function with selector 0xbb11bb11 on this contract.
- this function on this contract with or without transferring value at the same time (because of the additional TransferValue in the callRestrictions).

Example 5:

If controller B has TRANSFERVALUE and CALL permissions, and have the following value for AllowedCalls:

```
0x002000000001cafecafecafecafecafecafecafecafecafe11223344bb11bb11002000000002ffffffffffffffffffffffffffffffffffffffff68686868ffffffff
```

Each element of the compact bytes array have a different restrictionOperations:

- 0x000000001 for the first element allowing the controller B to only TransferValue to the function 0xbb11bb11 on 0xcafecafecafecafecafecafecafecafecafecafe address as long as the address supports 0x11223344 interfaceId through ERC165 (https://eips.ethereum.org/EIPS/eip-165).
- 0x000000002 for the second element allowing the controller B to only Call any function on any contract that support the interface ID 0x68686868 through ERC165 (https://eips.ethereum.org/EIPS/eip-165).

AddressPermissions:AllowedERC725YDataKeys:<address>

```
{
  "name": "AddressPermissions:AllowedERC725YDataKeys:<address>",
  "key": "0x4b80742de2bf866c29110000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "bytes[CompactBytesArray]",
  "valueContent": "Bytes"
}
```

Contains a compact bytes array of dynamic ERC725Y data keys that the address is restricted to modify in case of setting normal data with SETDATA permission.

- If the value of the data key is empty, setting data is disallowed.

The compact bytes array MUST be constructed in this format according to LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md):

```
<length of the data key prefix> <data key prefix>
```

- length of the data key prefix: The length of the prefix of the data key which the rest is dynamic. MUST be a number between 1 and 32.
- data key prefix: The prefix of the data key to be checked against the data keys being set.

Below is an example based on a LSP2 Mapping (./LSP-2-ERC725YJSONSchema.md#Mapping) key type, where first word = MyCoolGroupName, and second word = MySuperItem.

```
name: "MyCoolGroupName:MySuperItem";
key: 0x49b3e05bd43c5ac82f1000000a0b207005afb968993d50cd35b2b56d5531a7e1;
```


Example 1:

- If address A has SETDATA permission, and have the following value for AllowedERC725YDataKeys:

```
> 0x 0020 49b3e05bd43c5ac82f1000000a0b207005afb968993d50cd35b2b56d5531a7e1
> 0x002049b3e05bd43c5ac82f1000000a0b207005afb968993d50cd35b2b56d5531a7e1
```

> 0020 (32 in decimals) is the length of the data key to be set.

Resolve to:

Address A is only allowed to set the value for the data key attached above.

Example 2:

- If address B has SETDATA permission, and have the following value for AllowedERC725YDataKeys:

```
> 0x 000a 49b3e05bd43c5ac82f10 0020
beefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef
>
0x000a49b3e05bd43c5ac82f100020beefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef
```

> 000a (10 in decimals) is the length of the 49b3e05bd43c5ac82f10 prefix

Resolve to:

Address B is only allowed to set the value for the data 0xbeefbeef..beef data key and any data key that starts with 0x49b3e05bd43c5ac82f10.

By setting the value to 0x49b3e05bd43c5ac82f10 in the list of allowed ERC725Y data keys, one address can set any data key starting with the first word MyCoolGroupName:.... This means that this address can set any sub-mapping data keys of MyCoolGroupName (not only the sub-mapping MySuperItem from the example above).

Rationale

<!--The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.-->

This standard was inspired by how files permissions are designed in UNIX based file systems.

Files are assigned permissions as a 3 digit numbers, where each of the 3 digits is an octal value representing a set of permissions.
The octal value is calculated as the sum of permissions, where *read* = 4, *write* = 2, and *execute* = 1

To illustrate, for a file set with permission 755, the group permission (second digit) would be *read* and *execute* (See figure below). Each number is simply a three binary placeholder, each one holding the number that correspond to the access level in r, w, x order.

Implementation

<!--The implementations must be completed before any LIP is given status "Final", but it need not be completed before the LIP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the principle of "rough consensus and running code" is still useful when it comes to resolving many discussions of API details.-->

A implementation can be found in the [lukso-network/universalprofile-smart-contracts (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/LSP6KeyManager/LSP6KeyManager.sol)](https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/LSP6KeyManager/LSP6KeyManager.sol);
The below defines the JSON interface of the target(#target) contract.

ERC725Y JSON Schema LSP6KeyManager, set at the target(#target) contract:


```
[
 {
  "name": "AddressPermissions[]",
  "key": "0xdf30dba06db6a30e65354d9a64c609861f089545ca58c6b4dbe31a5f338cb0e3",
  "keyType": "Array",
  "valueType": "address",
  "valueContent": "Address"
 },
 {
  "name": "AddressPermissions:Permissions:<address>",
  "key": "0x4b80742de2bf82acb3630000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "bytes32",
  "valueContent": "BitArray"
 },
 {
  "name": "AddressPermissions:AllowedCalls:<address>",
  "key": "0x4b80742de2bf393a64c70000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "(bytes4,address,bytes4,bytes4)[CompactBytesArray]",
  "valueContent": "(BitArray,Address,Bytes4,Bytes4)"
 },
 {
  "name": "AddressPermissions:AllowedERC725YDataKeys:<address>",
  "key": "0x4b80742de2bf866c29110000<address>",
  "keyType": "MappingWithGrouping",
  "valueType": "bytes[CompactBytesArray]",
  "valueContent": "Bytes"
 }
]
```


Interface Cheat Sheet

```
interface ILSP6  /* is ERC165 */ {

    // ERC1271

    function isValidSignature(bytes32 hash, bytes memory signature) external view returns (bytes4 returnedStatus);


    // LSP6

    event PermissionsVerified(address indexed signer, uint256 indexed value, bytes4 indexed selector);


    function target() external view returns (address);


    function execute(bytes calldata payload) external payable returns (bytes memory);

    function executeBatch(uint256[] calldata values, bytes[] calldata payloads) external payable returns (bytes[] memory);

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-7-DigitalAsset.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-7-DigitalAsset.md

---

lip: 7
title: Digital Asset
author: Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network), Claudio Weck claudio@fanzone.media (mailto:claudio@fanzone.media), Matthew Stevens <@mattgstevens>, Ankit Kumar <@ankitkumar9018>
discussions-to: https://discord.gg/E2rJPP4 (LUKSO), https://discord.gg/PQvJQtCV (FANZONE)
status: Draft
type: LSP
created: 2021-09-02
requires: ERC165, ERC173, ERC725Y, LSP1, LSP2, LSP4, LSP17

<!--You can leave these HTML comments in your merged LIP and delete the visible duplicate text guides, they will not appear and may be helpful to refer to if you edit it again. This is the suggested template for new LIPs. Note that an LIP number will be assigned by an editor. When opening a pull request to submit your LIP,

please use an abbreviated title in the filename, lip-draft_title_abbrev.md. The title should be 44 characters or less.-->

Simple Summary

<!--"If you can't explain it simply, you don't understand it well enough." Provide a simplified and layman-accessible explanation of the LIP.-->

A standard interface for digital assets, for either fungible or non-fungible tokens.

Abstract

<!--A short (~200 word) description of the technical issue being addressed.-->

This standard defines an interface for tokens where minting and transfering is specified with an amount of tokens. It is based on ERC20 (https://github.com/ethereum/EIPs/issues/20) with some ideas from ERC777 (https://eips.ethereum.org/EIPS/eip-777).

Motivation

<!--The motivation is critical for LIPs that want to change the Lukso protocol. It should clearly explain why the existing protocol specification is inadequate to address the problem that the LIP solves. LIP submissions without sufficient motivation may be rejected outright.-->

This standard aims to support many token use cases, both fungible and non-fungible, to be used as the base implementation that is defined with other LSP standards in mind.

A commonality with LSP8 IdentifiableDigitalAsset (./LSP-8-IdentifiableDigitalAsset.md) is desired so that the two token implementations use similar naming for functions, events, and using hooks to notify token senders and receivers using LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md).

Specification

ERC165 (https://eips.ethereum.org/EIPS/eip-165) interface id: 0xdaa746b7

The LSP7 interface ID is calculated as the XOR of the LSP7 interface (see interface cheat-sheet below) and the LSP17 Extendable interface ID (./LSP-17-ContractExtension.md#erc165-interface-id).

ERC725Y Data Keys

This standard also expects data keys from LSP4 DigitalAsset-Metadata (./LSP-4-DigitalAsset-Metadata.md#erc725ykeys).

Methods

decimals

```
function decimals() external view returns (uint8);
```

Returns the number of decimals used to get its user representation.

If the token is non-divisible then 0 SHOULD be used, otherwise 18 is the common value.

Returns: uint8 the number of decimals to tranfrom a token value when displaying.

totalSupply

```
function totalSupply() external view returns (uint256);
```

Returns the number of existing tokens.

Returns: uint256 the number of existing tokens.

balanceOf

```
function balanceOf(address tokenOwner) external view returns (uint256);
```

Returns the number of tokens owned by tokenOwner.

Parameters:

- tokenOwner the address to query.

Returns: uint256 the number of tokens owned by this address.

authorizeOperator

```
function authorizeOperator(address operator, uint256 amount, bytes memory operatorNotificationData)
external;
```

Sets amount as the amount of tokens operator address has access to from callers tokens.

To increase or decrease the authorized amount of an operator, it's advised to call revokeOperator(..) function first, and then call authorizeOperator(..) with the new amount to authorize, to avoid front-running through an allowance double-spend exploit.
Check more information [in this document (https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/)](https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/).

MUST emit an [AuthorizedOperator event](#).

Parameters:

- operator the address to authorize as an operator.
- amount the amount of tokens operator has access to.
- operatorNotificationData the data to send when notifying the operator via LSP1.

Requirements:

- operator cannot be calling address.
- operator cannot be the zero address.

LSP1 Hooks:

- If the operator is a contract that supports LSP1 interface, it SHOULD call operator's [universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver)](#) function with the parameters below:
    - typeId: keccak256('LSP7Tokens_OperatorNotification') > 0x386072cc5a58e61263b434c722725f21031cd06e7c552cfaa06db5de8a320dbc

    - data: The data sent SHOULD be abi encoded and contain the tokenOwner (address), amount (uint256) , and the operatorNotificationData (bytes) respectively.

revokeOperator

```
function revokeOperator(address operator, bool notify, bytes memory operatorNotificationData) external;
```

Removes operator address as an operator of callers tokens.

MUST emit a RevokedOperator event.

Parameters:

- operator the address to revoke as an operator.
- notify the boolean indicating whether to notify the operator via LSP1 or not.
- operatorNotificationData the data to send when notifying the operator via LSP1.

Requirements:

- operator cannot be calling address.
- operator cannot be the zero address.

LSP1 Hooks:

- If the notify parameter is set to true, and the operator is a contract that supports LSP1 interface, it SHOULD call operator's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP7Tokens_OperatorNotification') > 0x386072cc5a58e61263b434c722725f21031cd06e7c552cfaa06db5de8a320dbc
    - data: The data sent SHOULD be abi encoded and contain the tokenOwner (address), amount (uint256) (0 in case of revoke), and the operatorNotificationData (bytes) respectively.

increaseAllowance

```
function increaseAllowance(address operator, uint256 addedAmount, bytes memory operatorNotificationData) external;
```

Increase the allowance of operator by addedAmount. This is an alternative approach to {authorizeOperator} that can be used as a mitigation for the double spending allowance problem. Notify the operator based on the LSP1-UniversalReceiver standard.

Parameters:

- operator the address to increase allowance as an operator.
- addedAmount the amount to add to the existing allowance of tokens operator has access to.
- operatorNotificationData the data to send when notifying the operator via LSP1.

Requirements:

- operator's original allowance cannot be zero.

LSP1 Hooks:

- If the operator is a contract that supports LSP1 interface, it SHOULD call operator's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP7Tokens_OperatorNotification') > 0x386072cc5a58e61263b434c722725f21031cd06e7c552cfaa06db5de8a320dbc
    - data: The data sent SHOULD be abi encoded and contain the tokenOwner (address), amount (uint256) (new allowance) , and the operatorNotificationData (bytes) respectively.

<be>

decreaseAllowance

```
function decreaseAllowance(address operator, uint256 subtractedAmount, bytes memory operatorNotificationData) external;
```

Decrease the allowance of operator by subtractedAmount. This is an alternative approach to {authorizeOperator} that can be used as a mitigation for the double spending allowance problem. Notify the operator based on the LSP1-UniversalReceiver standard.

Parameters:

- operator the address to decrease allowance as an operator.
- subtractedAmount the amount to substract to the existing allowance of tokens operator has access to.
- operatorNotificationData the data to send when notifying the operator via LSP1.

LSP1 Hooks:

- If the operator is a contract that supports LSP1 interface, it SHOULD call operator's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP7Tokens_OperatorNotification') > 0x386072cc5a58e61263b434c722725f21031cd06e7c552cfaa06db5de8a320dbc
    - data: The data sent SHOULD be abi encoded and contain the tokenOwner (address), amount (uint256) (new allowance) , and the operatorNotificationData (bytes) respectively.

authorizedAmountFor

```
function authorizedAmountFor(address operator, address tokenOwner) external view returns (uint256);
```

Returns amount of tokens operator address is authorized to spent from tokenOwner.
Operators can send and burn tokens on behalf of their owners. The tokenOwner is their own operator.

Parameters:

- operator the address to query operator status for.

Returns: uint256, the amount of tokens operator has access to from tokenOwner.

getOperatorsOf

```
function getOperatorsOf(address tokenOwner) external view returns (address[] memory);
```

Returns a list of operators allowed to transfer tokens on behalf of a tokenOwner from its balance. Their balance can be queried via authorizedAmountFor(address,address)

Parameters:

- tokenOwner the address to query the list of operators for.

Returns: address[], a list of token operators for tokenOwner.

transfer

```
function transfer(address from, address to, uint256 amount, bool force, bytes memory data) external;
```

Transfers amount of tokens from from to to. The force parameter will be used when notifying the token sender and receiver and revert.

MUST emit a Transfer event when transfer was successful.

Parameters:

- from the sending address.
- to the receiving address.
- amount the amount of tokens to transfer.
- force when set to TRUE, to may be any address; when set to FALSE to must be a contract that supports LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) and not revert.
- data additional data the caller wants included in the emitted event, and sent in the hooks to from and to addresses.

Requirements:

- from cannot be the zero address.
- to cannot be the zero address.
- amount tokens must be owned by from.
- If the caller is not from, it must be an operator for from with access to at least amount tokens.

LSP1 Hooks:

- If the token sender is a contract that supports LSP1 interface, it SHOULD call the token sender's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP7Tokens_SenderNotification') > 0x429ac7a06903dbc9c13dfcb3c9d11df8194581fa047c96d7a4171fc7402958ea
    - data: The data sent SHOULD be packed encoded and contain the sender (address), receiver (address), amount (uint256) and the data (bytes) respectively.

- If the token recipient is a contract that supports LSP1 interface, it SHOULD call the token recipient's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:

    - typeId: keccak256('LSP7Tokens_RecipientNotification') >0x20804611b3e2ea21c480dc465142210acf4a2485947541770ec1fb87dee4a55c
    - data: The data sent SHOULD be packed encoded and contain the sender (address), receiver (address), amount (uint256) and the data (bytes) respectively.

Note: LSP1 Hooks MUST be implemented in any type of token transfer (mint, transfer, burn, transferBatch).

transferBatch

```
function transferBatch(address[] memory from, address[] memory to, uint256[] memory amount, bool force, bytes[] memory data) external;
```

Transfers many tokens based on the list from, to, amount. If any transfer fails, the call will revert.

MUST emit a Transfer event for each transfered token.

Parameters:

- from the list of sending addresses.
- to the list of receiving addresses.
- amount the amount of tokens to transfer.
- force when set to TRUE, to may be any address; when set to FALSE to must be a contract that supports LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) and not revert.
- data the list of additional data the caller wants included in the emitted event, and sent in the hooks to from and to addresses.

Requirements:

- from, to, amount lists are the same length.
- no values in from can be the zero address.
- no values in to can be the zero address.
- each amount tokens must be owned by from.
- If the caller is not from, it must be an operator for from with access to at least amount tokens.

Events

Transfer

```
event Transfer(address indexed operator, address indexed from, address indexed to, uint256 amount, bool force, bytes data);
```

MUST be emitted when amount tokens is transferred from from to to.

AuthorizedOperator

```
event AuthorizedOperator(address indexed operator, address indexed tokenOwner, uint256 amount, bytes operatorNotificationData);
```

MUST be emitted when tokenOwner enables operator for amount tokens.

RevokedOperator

```
event RevokedOperator(address indexed operator, address indexed tokenOwner, bool notify, bytes memory operatorNotificationData);
```

MUST be emitted when tokenOwner disables operator.

Rationale

<!--The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.-->

There should be a base token standard for the LSP ecosystem of contracts, which will allow common tooling and clients to be built. Existing tools and clients that expect ERC20 (https://github.com/ethereum/EIPs/issues/20) & ERC777 (https://eips.ethereum.org/EIPS/eip-777) can be made to work with this standard by using "compatability" contract extensions that match the desired interface.

Operators

To clarify the ability of an address to access tokens from another address, operator was chosen as the name for functions, events and variables in all cases. This is originally from ERC777 (https://eips.ethereum.org/EIPS/eip-777) standard and replaces the allowance functionality from ERC20 (https://github.com/ethereum/EIPs/issues/20).

Token Transfers

There is only one transfer function, which is aware of operators. This deviates from ERC20 (https://github.com/ethereum/EIPs/issues/20) and ERC777 (https://eips.ethereum.org/EIPS/eip-777) which added functions specifically for the token owner to use, and for those with access to tokens. By having a single function to call this makes it simple to move tokens, and the caller will be exposed in the Transfer event as an indexed value.

Usage of hooks

When a token is changing owners (minting, transfering, burning) an attempt is made to notify the token sender and receiver using LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) interface. The implementation uses _notifyTokenSender and _notifyTokenReceiver as the internal functions to process this.

The force parameter sent during function transfer SHOULD be used when notifying the token receiver, to determine if it must support LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md). This is used to prevent accidental token transfers, which may results in lost tokens: non-contract addresses could be a copy paste issue, contracts not supporting LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) might not be able to move tokens.

Implementation

<!--The implementations must be completed before any LIP is given status "Final", but it need not be completed before the LIP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the principle of "rough consensus and running code" is still useful when it comes to resolving many discussions of API details.-->

A implementation can be found in the lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/main/contracts/LSP7DigitalAsset/LSP7DigitalAsset.sol);

Interface Cheat Sheet

```
interface ILSP7 is /* IERC165 */ {

    // ERC173

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);


    function owner() external view returns (address);

    function transferOwnership(address newOwner) external; // onlyOwner

    function renounceOwnership() external; // onlyOwner


    // ERC725Y

    event DataChanged(bytes32 indexed dataKey, bytes dataValue);


    function getData(bytes32 dataKey) external view returns (bytes memory value);

    function setData(bytes32 dataKey, bytes memory value) external; // onlyOwner

    function getDataBatch(bytes32[] memory dataKeys) external view returns (bytes[] memory values);

    function setDataBatch(bytes32[] memory dataKeys, bytes[] memory values) external; // onlyOwner


    // LSP7

    event Transfer(address indexed operator, address indexed from, address indexed to, uint256 amount,
bool force, bytes data);

    event AuthorizedOperator(address indexed operator, address indexed tokenOwner, uint256 indexed
amount, bytes operatorNotificationData);

    event RevokedOperator(address indexed operator, address indexed tokenOwner, bool notify, bytes
operatorNotificationData);


    function decimals() external view returns (uint8);

    function totalSupply() external view returns (uint256);

    function balanceOf(address tokenOwner) external view returns (uint256);

    function authorizeOperator(address operator, uint256 amount, bytes memory operatorNotificationData)
external;
```

```
    function revokeOperator(address to, bool notify, bytes memory operatorNotificationData) external;

    function increaseAllowance(address operator, uint256 addedAmount, bytes memory
operatorNotificationData) external;

    function decreaseAllowance(address operator, uint256 subtractedAmount, bytes memory
operatorNotificationData) external;

    function authorizedAmountFor(address operator, address tokenOwner) external view returns (uint256);

    function getOperatorsOf(address tokenOwner) external view returns (address[] memory);

    function transfer(address from, address to, uint256 amount, bool force, bytes memory data) external;

    function transferBatch(address[] memory from, address[] memory to, uint256[] memory amount, bool
force, bytes[] memory data) external;

}
```

Copyright

</file>

<file>

path: /LSPs/LSP-8-IdentifiableDigitalAsset.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-8-IdentifiableDigitalAsset.md

---

lip: 8
title: Identifiable Digital Asset
author: Claudio Weck claudio@fanzone.media (mailto:claudio@fanzone.media), Fabian Vogelsteller fabian@lukso.network (mailto:fabian@lukso.network), Matthew Stevens <@mattgstevens>, Ankit Kumar <@ankitkumar9018>
discussions-to: https://discord.gg/E2rJPP4 (LUKSO), https://discord.gg/PQvJQtCV (FANZONE)
status: Draft
type: LSP
created: 2021-09-02
requires: ERC165, ERC725Y, LSP1, LSP2, LSP4, LSP17

<!--You can leave these HTML comments in your merged LIP and delete the visible duplicate text guides, they will not appear and may be helpful to refer to if you edit it again. This is the suggested template for new LIPs. Note that an LIP number will be assigned by an editor. When opening a pull request to submit your LIP, please use an abbreviated title in the filename, lip-draft_title_abbrev.md. The title should be 44 characters or less.-->

Simple Summary

<!--"If you can't explain it simply, you don't understand it well enough." Provide a simplified and layman-accessible explanation of the LIP.-->

The LSP8 Identifiable Digital Asset Standard defines a standard interface for uniquely identifiable digital assets. It allows tokens to be uniquely traded and given with metadata using ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md) and LSP4 (./LSP-4-DigitalAsset-Metadata.md#lsp4metadata).

Abstract

<!--A short (~200 word) description of the technical issue being addressed.-->

This standard defines an interface for tokens that are identified with a tokenId, based on ERC721 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md). A bytes32 value is used for tokenId to allow many uses of token identification including numbers, contract addresses, and any other unique identifiers (*e.g:* serial numbers, NFTs with unique names, hash values, etc...).

This standard defines a set of data-key value pairs that are useful to know what the tokenId represents, and the associated metadata for each tokenId.

Motivation

<!--The motivation is critical for LIPs that want to change the Lukso protocol. It should clearly explain why the existing protocol specification is inadequate to address the problem that the LIP solves. LIP submissions without sufficient motivation may be rejected outright.-->

This standard aims to support use cases not covered by LSP7 DigitalAsset (./LSP-7-DigitalAsset.md), by using a tokenId instead of an amount of tokens to mint, burn, and transfer tokens. Each tokenId may have metadata (either as a on-chain ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md) contract or off-chain JSON) in addition to the LSP4 DigitalAsset-Metadata (./LSP-4-DigitalAsset-Metadata.md#erc725ykeys) metadata of the smart contract that mints the tokens. In this way a minted token benefits from the flexibility & upgradability of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md) standard, and transfering a token carries the history of ownership and metadata updates. This is beneficial for a new generation of NFTs.

A commonality with LSP7 DigitalAsset (./LSP-7-DigitalAsset.md) is desired so that the two token implementations use similar naming for functions, events, and using hooks to notify token senders and receivers using LSP1.

Specification

ERC165 (https://eips.ethereum.org/EIPS/eip-165) interface id: 0x30dc5278

The LSP8 interface ID is calculated as the XOR of the LSP8 interface (see interface cheat-sheet below) and the LSP17 Extendable interface ID (./LSP-17-ContractExtension.md#erc165-interface-id).

ERC725Y Data Keys - LSP8 Contract

These are the expected data keys for an LSP8 contract that can mints identifiable tokens (NFTs).

This standard can also be combined with the data keys from LSP4 DigitalAsset-Metadata. (./LSP-4-DigitalAsset-Metadata.md#erc725ykeys).

LSP8TokenIdType

This data key describes the type of the tokenId and can take one of the following enum values described in the table below.

In the context of LSP8, a contract implementing the LSP8 standard represents a collection of unique non-fungible tokens (NFT). The LSP8 collection contract is responsible for minting these tokens.

Each token part of the collection is identifiable through its unique tokenId.

However, these NFTs can be represented differently depending on the use case. This is referred to as the type of the tokenId.

The LSP8TokenIdType metadata key provides this information and describes how to treat the NFTs parts of the LSP8 collection.

This MUST NOT be changeable, and set only during initialization of the LSP8 token contract.

Value	Type	Description
0	uint256	each NFT is represented with a unique number. This number is an incrementing count, where each minted token is assigned the next number.
1	string	each NFT is represented using a unique name (as a short utf8 encoded string, no more than 32 characters long)
2	bytes32	each NFT is represented using a 32 bytes long unique identifier.
3	bytes32	each NFT is represented using a 32 bytes hash digest.
4	address	each NFT is represented as its own smart contract that can hold its own metadata (*e.g* *ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) compatible*).


```
{
  "name": "LSP8TokenIdType",
  "key": "0x715f248956de7ce65e94d9d836bfead479f7e70d69b718d47bfe7b00e05b4fe4",
  "keyType": "Singleton",
  "valueType": "uint256",
  "valueContent": "Number"
}
```


LSP8MetadataTokenURI:<tokenId>

This data key stores the URI of the metadata for a specific tokenId.

It consists of a bytes4 identifier of the hash function and the URI string.

When metadata JSON is created for a tokenId, the URI COULD be stored in the storage of the LSP8 contract.

The value stored under this data key is a tuple (bytes4,string) that contains the following elements:

- bytes4 = the 4 bytes identifier of the hash function used to generate the URI:
    - if the tokenId is a hash (LSP8TokenIdType is 4): *see details below*.
    - if the tokenId is any other LSP8TokenIdType: MUST be 0x00000000.
- string = the URI where the metadata for the tokenId can be retrieved.

```
{
  "name": "LSP8MetadataTokenURI:<address|uint256|bytes32|string>",
  "key": "0x1339e76a390b7b9ec9010000<address|uint256|bytes32|string>",
  "keyType": "Mapping",
  "valueType": "(bytes4,string)",
  "valueContent": "(Bytes4,URI)"
}
```

> For construction of the Mapping data key see: LSP2 ERC725Y JSON Schema > keyType = Mapping (./LSP-2-ERC725YJSONSchema.md#mapping)

When bytes4 = 0x00000000

The URI of some NFTs could be alterable, for example in the case of NFTs that need their metadata to change overtime.

In this case, the first bytes4 in the tuple MUST be set to 0x00000000 (4 x zero bytes), which describes that the URI can be changed over the lifetime of the NFTs.

When bytes4 = some 4 bytes value (Example)

To represent the hash function keccak256:

- bytes4 value in the tuple to represent the hash function keccak256 = 0x6f357c6a

This can be obtained as follow:

keccak256('keccak256(utf8)') =
0x6f357c6a956bf6b8a917ccf88cc1d3388ff8d646810d0393fe69ae7ee228004f

LSP8TokenMetadataBaseURI

This data key defines the base URI for the metadata of each tokenIds present in the LSP8 contract.

The complete URI that points to the metadata of a specific tokenId MUST be formed by concatenating this base URI with the tokenId.
As {LSP8TokenMetadataBaseURI}{tokenId}.

⚠ TokenIds MUST be in lowercase, even for the tokenId type address (= address not checksumed).

- LSP8TokenIdType 0 (= uint256)
  e.g. http://mybase.uri/1234
- LSP8TokenIdType 1 (= string)
  e.g. http://mybase.uri/name-of-the-nft
- LSP8TokenIdType 2 or 3 (= bytes32)
  e.g.
  http://mybase.uri/e5fe3851d597a3aa8bbdf8d8289eb9789ca2c34da7a7c3d0a7c442a87b81d5c2
- LSP8TokenIdType 4 (= address)
  e.g. http://mybase.uri/0x43fb7ab43a3a32f1e2d5326b651bbae713b02429

Some Base URIs could be alterable, for example in the case of NFTs that need their metadata to change overtime.

In this case, the first bytes4 in the tuple (for the valueType/valueContent) MUST be set to 0x00000000 (4 x zero bytes), which describes that the URI can be changed over the lifetime of the NFTs.

If the tokenId type is a hash (LSP8TokenIdType 4), the first bytes4 in the tuple represents the hash function.

Example:

To represent the hash function keccak256:

- bytes4 value in the tuple to represent the hash function keccak256 = 0x6f357c6a

This can be obtained as follow:

keccak256('keccak256(utf8)') =
0x6f357c6a956bf6b8a917ccf88cc1d3388ff8d646810d0393fe69ae7ee228004f

```
{
  "name": "LSP8TokenMetadataBaseURI",
  "key": "0x1a7628600c3bac7101f53697f48df381ddc36b9015e7d7c9c5633d1252aa2843",
  "keyType": "Singleton",
  "valueType": "(bytes4,string)",
  "valueContent": "(Bytes4,URI)"
}
```

ERC725Y Data Keys of external contract for tokenID type 4 (address)

When the LSP8 contract uses the tokenId type 4 (= address), each tokenId minted is an ERC725Y smart contract that can have its own metadata.
We refer to this contract as the tokenId metadata contract.

In this case, each tokenId present in the LSP8 contract references an other ERC725Y contract.

The tokenId metadata contract SHOULD contain the following ERC725Y data key in its storage.

LSP8ReferenceContract

This data key stores the address of the LSP8 contract that minted this specific tokenId (defined by the address of the tokenId metadata contract).

It is a reference back to the LSP8 Collection it comes from.

If the LSP8ReferenceContract data key is set, it MUST NOT be changeable.

```
{
  "name": "LSP8ReferenceContract",
  "key": "0x708e7b881795f2e6b6c2752108c177ec89248458de3bf69d0d43480b3e5034e6",
  "keyType": "Singleton",
  "valueType": "(address,bytes32)",
  "valueContent": "(Address,bytes32)"
}
```

LSP8 TokenId Metadata

The metadata for a specific of a uniquely identifiable digital asset (when this tokenId is represented by its own ERC725Y contract) can follow the JSON format of the LSP4Metadata (./LSP-4-DigitalAsset-Metadata.md#lsp4metadata) data key.

This JSON format includes an "attributes" field to describe unique properties of the tokenId.

Methods

totalSupply

```
function totalSupply() external view returns (uint256);
```

Returns the number of existing tokens.

Returns: uint256 the number of existing tokens.

balanceOf

```
function balanceOf(address tokenOwner) external view returns (uint256);
```

Returns the number of tokens owned by tokenOwner.

Parameters:

- tokenOwner the address to query.

Returns: uint256 the number of tokens owned by this address.

tokenOwnerOf

```
function tokenOwnerOf(bytes32 tokenId) external view returns (address);
```

Returns the tokenOwner address of the tokenId token.

Parameters:

- tokenId the token to query.

Requirements:

- tokenId must exist

Returns: address the token owner.

tokenIdsOf

```
function tokenIdsOf(address tokenOwner) external view returns (bytes32[] memory);
```

Returns the list of tokenIds for the tokenOwner address.

Parameters:

- tokenOwner the address to query.

Returns: bytes32[] the list of owned token ids.

authorizeOperator

```
function authorizeOperator(address operator, bytes32 tokenId, bytes memory operatorNotificationData) external;
```

Makes operator address an operator of tokenId.

MUST emit an AuthorizedOperator event.

Parameters:

- operator the address to authorize as an operator.
- tokenId the token to enable operator status to.
- operatorNotificationData the data to send when notifying the operator via LSP1.

Requirements:

- tokenId must exist
- caller must be current tokenOwner of tokenId.
- operator cannot be calling address.
- operator cannot be the zero address.

LSP1 Hooks:

- If the operator is a contract that supports LSP1 interface, it SHOULD call operator's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP8Tokens_OperatorNotification') > 0x8a1c15a8799f71b547e08e2bcb2e85257e81b0a07eee2ce6712549eef1f00970
    - data: The data sent SHOULD be abi encoded and contain the tokenOwner (address), tokenId (bytes32), isAuthorized (boolean), and the operatorNotificationData (bytes) respectively.

revokeOperator

```
function revokeOperator(address operator, bytes32 tokenId, bool notify, bytes memory operatorNotificationData) external;
```

Removes operator address as an operator of tokenId.

MUST emit a RevokedOperator event.

Parameters:

- operator the address to revoke as an operator.
- tokenId the token to disable operator status to.
- operatorNotificationData the data to send when notifying the operator via LSP1.

Requirements:

- tokenId must exist
- caller must be current tokenOwner of tokenId.
- operator cannot be calling address.

- operator cannot be the zero address.

LSP1 Hooks:

- If the notify boolean is set to true and the operator is a contract that supports LSP1 interface, it SHOULD call operator's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP8Tokens_OperatorNotification') > 0x8a1c15a8799f71b547e08e2bcb2e85257e81b0a07eee2ce6712549eef1f00970
    - data: The data sent SHOULD be abi encoded and contain the tokenOwner (address), tokenId (bytes32), isAuthorized (boolean), and the operatorNotificationData (bytes) respectively.

isOperatorFor

```
function isOperatorFor(address operator, bytes32 tokenId) external view returns (bool);
```

Returns whether operator address is an operator of tokenId.
Operators can send and burn tokens on behalf of their owners. The tokenOwner is their own operator.

Parameters:

- operator the address to query operator status for.
- tokenId the token to query.

Requirements:

- tokenId must exist
- caller must be current tokenOwner of tokenId.

Returns: bool, TRUE if operator address is an operator of tokenId, FALSE otherwise.

getOperatorsOf

```
function getOperatorsOf(bytes32 tokenId) external view returns (address[] memory);
```

Returns all operator addresses of tokenId.

Parameters:

- tokenId the token to query.

Requirements:

- tokenId must exist
- caller must be current tokenOwner of tokenId.
- operator cannot be calling address.

Returns: address[] the list of operators.

transfer

```
function transfer(address from, address to, bytes32 tokenId, bool force, bytes memory data) external;
```

Transfers tokenId token from from to to. The force parameter will be used when notifying the token sender and receiver.

MUST emit a Transfer event when transfer was successful.

Parameters:

- from the sending address.
- to the receiving address.
- from and to cannot be the same address.
- tokenId the token to transfer.
- force when set to TRUE, to may be any address; when set to FALSE to must be a contract that supports LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) and successfully processes a call to universalReceiver(bytes32 typeId, bytes memory data).
- data additional data the caller wants included in the emitted event, and sent in the hooks to from and to addresses.

Requirements:

- from cannot be the zero address.
- to cannot be the zero address.
- tokenId token must be owned by from.
- If the caller is not from, it must be an operator of tokenId.

LSP1 Hooks:

- If the token sender is a contract that supports LSP1 interface, it SHOULD call the token sender's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP8Tokens_SenderNotification') = 0xb23eae7e6d1564b295b4c3e3be402d9a2f0776c57bdf365903496f6fa481ab00
    - data: The data sent SHOULD be ABI encoded and contain the sender (address), receiver (address), tokenId (bytes32) and the data (bytes) respectively.

- If the token recipient is a contract that supports LSP1 interface, it SHOULD call the token recipient's universalReceiver(...) (./LSP-1-UniversalReceiver.md#universalreceiver) function with the parameters below:
    - typeId: keccak256('LSP8Tokens_RecipientNotification') = 0x0b084a55ebf70fd3c06fd755269dac2212c4d3f0f4d09079780bfa50c1b2984d
    - data: The data sent SHOULD be ABI encoded and contain the sender (address), receiver (address), tokenId (bytes32) and the data (bytes) respectively.

Note: LSP1 Hooks MUST be implemented in any type of token transfer (mint, transfer, burn, transferBatch).

transferBatch

```
function transferBatch(address[] memory from, address[] memory to, bytes32[] memory tokenId, bool force, bytes[] memory data) external;
```

Transfers many tokens based on the list from, to, tokenId. If any transfer fails, the call will revert.

MUST emit a Transfer event for each transfered token.

Parameters:

- from the list of sending addresses.
- to the list of receiving addresses.
- tokenId the list of tokens to transfer.
- force when set to TRUE, to may be any address; when set to FALSE to must be a contract that supports LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) and successfully processes a call to universalReceiver(bytes32 typeId, bytes memory data).
- data the list of additional data the caller wants included in the emitted event, and sent in the hooks to from and to addresses.

Requirements:

- from, to, tokenId lists are the same length.
- no values in from can be the zero address.
- no values in to can be the zero address.
- from and to cannot be the same address at the same.
- each tokenId token must be owned by from.
- If the caller is not from, it must be an operator of each tokenId.

Events

Transfer

```
event Transfer(address operator, address indexed from, address indexed to, bytes32 indexed tokenId, bool force, bytes data);
```

MUST be emitted when tokenId token is transferred from from to to.

AuthorizedOperator

```
event AuthorizedOperator(address indexed operator, address indexed tokenOwner, bytes32 indexed tokenId, bytes operatorNotificationData);
```

MUST be emitted when tokenOwner enables operator for tokenId.

RevokedOperator

```
event RevokedOperator(address indexed operator, address indexed tokenOwner, bytes32 indexed tokenId, bool notified, bytes operatorNotificationData);
```

MUST be emitted when tokenOwner disables operator for tokenId.

Rationale

<!--The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.-->

There should be a base token standard that allows tracking unique assets for the LSP ecosystem of contracts, which will allow common tooling and clients to be built. Existing tools and clients that expect ERC721 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md) can be made to work with this standard by using "compatability" contract extensions that match the desired interface.

Token Identifier

Every token is identified by a unique bytes32 tokenId which SHALL NOT change for the life of the contract. The pair (contract address, uint256 tokenId) is globally unique and a fully-qualified identifier for a specific asset on-chain. While some implementations may find it convenient to use the tokenId as an uint256 that is incremented for each minted token, callers SHALL NOT assume that tokenIds have any specific pattern to them, and MUST treat the tokenId as a "black box". Also note that a tokenId MAY become invalid (when burned).

The choice of bytes32 tokenId allows a wide variety of applications including numbers, contract addresses, and hashed values (ie. serial numbers).

Operators

To clarify the ability of an address to access tokens from another address, operator was chosen as the name for functions, events and variables in all cases. This is originally from ERC777 (https://eips.ethereum.org/EIPS/eip-777) standard and replaces the approve functionality from ERC721 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md).

Token Transfers

There is only one transfer function, which is aware of operators. This deviates from ERC721 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md) and ERC777 (https://eips.ethereum.org/EIPS/eip-777) which added functions specifically for the token owner to use, and for those with access to tokens. By having a single function to call this makes it simple to move tokens, and the caller will be exposed in the Transfer event as an indexed value.

Usage of hooks

When a token is changing owners (minting, transfering, burning) an attempt is made to notify the token sender and receiver using LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) interface. The implementation uses _notifyTokenSender and _notifyTokenReceiver as the internal functions to process this.

The force parameter sent during function transfer SHOULD be used when notifying the token receiver, to determine if it must support LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md). This is used to prevent accidental token transfers, which may results in lost tokens: non-contract addresses could be a copy paste issue, contracts not supporting LSP1 UniversalReceiver (./LSP-1-UniversalReceiver.md) might not be able to move tokens.

Implementation

<!--The implementations must be completed before any LIP is given status "Final", but it need not be completed before the LIP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the principle of "rough consensus and running code" is still useful when it comes to resolving many discussions of API details.-->

A implementation can be found in the [lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/develop/contracts/LSP8IdentifiableDigitalAsset/LSP8IdentifiableDigitalAsset.sol)](https://github.com/lukso-network/lsp-universalprofile-smart-contracts/blob/develop/contracts/LSP8IdentifiableDigitalAsset/LSP8IdentifiableDigitalAsset.sol).

ERC725Y JSON Schema LSP8IdentifiableDigitalAsset:


```
[
 {
  "name": "LSP8TokenIdType",
  "key": "0x715f248956de7ce65e94d9d836bfead479f7e70d69b718d47bfe7b00e05b4fe4",
  "keyType": "Singleton",
  "valueType": "uint256",
  "valueContent": "Number"
 },
 {
  "name": "LSP8MetadataTokenURI:<address|uint256|bytes32|string>",
  "key": "0x1339e76a390b7b9ec9010000<address|uint256|bytes32|string>",
  "keyType": "Mapping",
  "valueType": "(bytes4,string)",
  "valueContent": "(Bytes4,URI)"
 },
 {
  "name": "LSP8TokenMetadataBaseURI",
  "key": "0x1a7628600c3bac7101f53697f48df381ddc36b9015e7d7c9c5633d1252aa2843",
  "keyType": "Singleton",
  "valueType": "(bytes4,string)",
  "valueContent": "(Bytes4,URI)"
 },
 {
  "name": "LSP8ReferenceContract",
  "key": "0x708e7b881795f2e6b6c2752108c177ec89248458de3bf69d0d43480b3e5034e6",
  "keyType": "Singleton",
  "valueType": "(address,bytes32)",
  "valueContent": "(Address,bytes32)"
 }
]
```


Interface Cheat Sheet

```
interface ILSP8 is /* IERC165 */ {

  // ERC173

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);


  function owner() external view returns (address);

  function transferOwnership(address newOwner) external override; // onlyOwner

  function renounceOwnership() external virtual; // onlyOwner
```

```
    // ERC725Y

    event DataChanged(bytes32 indexed dataKey, bytes dataValue);


    function getData(bytes32 dataKey) external view returns (bytes memory value);

    function setData(bytes32 dataKey, bytes memory value) external; // onlyOwner

    function getDataBatch(bytes32[] memory dataKeys) external view returns (bytes[] memory values);

    function setDataBatch(bytes32[] memory dataKeys, bytes[] memory values) external; // onlyOwner


    // LSP8

    event Transfer(address operator, address indexed from, address indexed to, bytes32 indexed tokenId,
bool force, bytes data);

    event AuthorizedOperator(address indexed operator, address indexed tokenOwner, bytes32 indexed
tokenId, bytes operatorNotificationData);

    event RevokedOperator(address indexed operator, address indexed tokenOwner, bytes32 indexed
tokenId, bool notified, bytes operatorNotificationData);


    function totalSupply() external view returns (uint256);

    function balanceOf(address tokenOwner) external view returns (uint256);

    function tokenOwnerOf(bytes32 tokenId) external view returns (address);

    function tokenIdsOf(address tokenOwner) external view returns (bytes32[] memory);

    function authorizeOperator(address operator, bytes32 tokenId, bytes memory operatorNotificationData)
external;

    function revokeOperator(address operator, bytes32 tokenId, bool notify, bytes memory
operatorNotificationData) external;

    function isOperatorFor(address operator, bytes32 tokenId) external view returns (bool);

    function getOperatorsOf(bytes32 tokenId) external view returns (address[] memory);

    function transfer(address from, address to, bytes32 tokenId, bool force, bytes memory data) external;

    function transferBatch(address[] memory from, address[] memory to, bytes32[] memory tokenId, bool
force, bytes[] memory data) external;
}
```

Copyright

</file>

<file>

path: /LSPs/LSP-9-Vault.md

url: https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-9-Vault.md

---

lip: 9
title: Vault
author:
discussions-to: https://discord.gg/E2rJPP4
status: Draft
type: LSP
created: 2021-09-21
requires: ERC165, ERC725X, ERC725Y, LSP1, LSP2, LSP14, LSP17

Simple Summary

This standard describes a version of an ERC725 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md) smart contract, that represents a blockchain vault.

Abstract

This standard defines a vault that can hold assets and interact with other contracts. It has the ability to attach information via ERC725Y (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md#erc725y) to itself, execute, deploy or transfer value to any other smart contract or EOA via ERC725X (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-725.md#erc725x). It can be notified of incoming assets via the LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) function.

Motivation

Specification

LSP9-Vault interface id according to ERC165 (https://eips.ethereum.org/EIPS/eip-165): 0x28af17e6.

This bytes4 interface id is calculated as the XOR of the selector of batchCalls function and the following standards: ERC725Y, ERC725X, LSP1-UniversalReceiver, LSP14Ownable2Step and LSP17Extendable.

Smart contracts implementing the LSP9 standard MUST implement the ERC165 (https://eips.ethereum.org/EIPS/eip-165) supportsInterface(..) function and MUST support the LSP9, ERC725X, ERC725Y, LSP1, LSP14 and LSP17Extendable interface ids.

Methods

Smart contracts implementing the LSP9 standard MUST implement all of the functions listed below:

receive

```
receive() external payable;
```

The receive function allows for receiving native tokens.

MUST emit a ValueReceived event when receiving native token.

fallback

```
fallback() external payable;
```

This function is part of the LSP17 (./LSP-17-ContractExtension.md) specification, with additional requirements as follows:

- MUST be payable.
- MUST emit a ValueReceived event if value was sent alongside some calldata.
- MUST return if the data sent to the contract is less than 4 bytes in length.
- MUST check for address of the extension under the following ERC725Y Data Key, and call the extension.

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

> <bytes4> is the functionSelector called on the vault contract. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

- MUST NOT revert when there is no extension set for 0x00000000.

owner

```
function owner() external view returns (address);
```

This function is part of the LSP14 (./LSP-14-Ownable2Step.md) specification.

pendingOwner

```
function pendingOwner() external view returns (address);
```

This function is part of the LSP14 (./LSP-14-Ownable2Step.md) specification.

transferOwnership

```
function transferOwnership(address newPendingOwner) external;
```

This function is part of the LSP14 (./LSP-14-Ownable2Step.md) specification, with additional requirements as follows:

- MUST override the LSP14 Type ID triggered by using transferOwnership(..) to the one below:
    - keccak256('LSP0OwnershipTransferStarted') > 0xe17117c9d2665d1dbeb479ed8058bbebde3c50ac50e2e65619f60006caac6926

acceptOwnership

```
function acceptOwnership() external;
```

This function is part of the LSP14 (./LSP-14-Ownable2Step.md) specification, with additional requirements as follows:

- MUST override the LSP14 Type IDs triggered by using accceptOwnership(..) to the ones below:
    - keccak256('LSP0OwnershipTransferred_SenderNotification') > 0xa4e59c931d14f7c8a7a35027f92ee40b5f2886b9fdcdb78f30bc5ecce5a2f814
    - keccak256('LSP0OwnershipTransferred_RecipientNotification') > 0xceca317f109c43507871523e82dc2a3cc64dfa18f12da0b6db14f6e23f995538

renounceOwnership

```
function renounceOwnership() external;
```

This function is part of the LSP14 (./LSP-14-Ownable2Step.md) specification.

batchCalls

```
function batchCalls(bytes[] calldata functionCalls) external returns (bytes[] memory results)
```

Enables the execution of a batch of encoded function calls on the current contract in a single transaction, provided as an array of bytes.

MUST use the delegatecall opcode to execute each call in the same context of the current contract.

Parameters:

- functionCalls: an array of encoded function calls to be executed on the current contract.

The data field can be:

- an array of ABI-encoded function calls such as an array of ABI-encoded execute, setData, transferOwnership or any LSP9 functions.
- an array of bytes which will resolve to the fallback function to be checked for an extension.

Requirements:

- MUST NOT be payable.

Returns: results , an array of bytes containing the return values of each executed function call.

execute

```
function execute(uint256 operationType, address target, uint256 value, bytes memory data) external payable returns (bytes memory);
```

This function is part of the ERC725X (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725x) specification, with additional requirements as follows:

- MUST revert when the operation type is DELEGATECALL.
- MUST emit a ValueReceived event before external calls or contract creation if the function receives native tokens.

executeBatch

```
function executeBatch(uint256[] memory operationsType, address[] memory targets, uint256[] memory values, bytes[] memory datas) external payable returns (bytes[] memory);
```

This function is part of the ERC725X (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725x) specification, with additional requirements as follows:

- MUST revert when one of the operations type is DELEGATECALL.
- MUST emit a ValueReceived event before external calls or contract creation if the function receives native tokens.

getData

```
function getData(bytes32 dataKey) external view returns (bytes memory);
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification.

getDataBatch

```
function getDataBatch(bytes32[] memory dataKeys) external view returns (bytes[] memory);
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification.

setData

```
function setData(bytes32 dataKey, bytes memory dataValue) external;
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification, with additional requirements as follows:

- MUST allow the owner to setData.
- MUST allow the Universal Receiver Delegate contracts to setData only in reentrant calls of the universalReceiver(..) function of the LSP9Vault.
- MUST emit only the first 256 bytes of the dataValue parameter in the DataChanged (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#datachanged) event.

setDataBatch

```
function setDataBatch(bytes32[] memory dataKeys, bytes[] memory dataValues) external;
```

This function is part of the ERC725Y (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#erc725y) specification, with additional requirements as follows:

- MUST allow the owner to setData.
- MUST allow the Universal Receiver Delegate contracts to setData only in reentrant calls of the universalReceiver(..) function of the LSP9Vault.
- MUST emit only the first 256 bytes of the dataValue parameter in the DataChanged (https://github.com/ERC725Alliance/ERC725/blob/develop/docs/ERC-725.md#datachanged) event.

universalReceiver

```
function universalReceiver(bytes32 typeId, bytes memory receivedData) external payable returns (bytes memory);
```

This function is part of the LSP1 (./LSP-1-UniversalReceiver.md) specification, with additional requirements as follows:

- MUST emit a ValueReceived event before external calls if the function receives native tokens.
- If an address is stored under the data key attached below and this address is a contract:
    - forwards the call to the universalReceiverDelegate(address,uint256,bytes32,bytes) (./LSP-1-UniversalReceiver.md#universalreceiverdelegate) function on the contract at this address ONLY IF this contract supports the LSP1UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification).
    - if the contract at this address does not supports the LSP1UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification), execution continues normally.
- If there is no address stored under this data key, execution continues normally.

```
{
  "name": "LSP1UniversalReceiverDelegate",
  "key": "0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
  "keyType": "Singleton",
  "valueType": "address",
  "valueContent": "Address"
}
```

- If an address is stored under the data key attached below and this address is a contract:
    - forwards the call to the universalReceiverDelegate(address,uint256,bytes32,bytes) (./LSP-1-UniversalReceiver.md#universalreceiverdelegate) function on the contract at this address ONLY IF this contract supports the LSP1UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification).
    - if the contract at this address does not supports the LSP1UniversalReceiverDelegate interface id (./LSP-1-UniversalReceiver.md#specification), execution continues normally.
- If there is no address stored under this data key, execution continues normally.

```
{
  "name": "LSP1UniversalReceiverDelegate:<bytes32>",
  "key": "0x0cfc51aec37c55a4d0b10000<bytes32>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

The <bytes32\> in the data key name corresponds to the typeId passed to the universalReceiver(..) function.

> Warning
> When constructing this data key for a specific typeId, unique elements of the typeId SHOULD NOT be on the right side because of trimming rules.
>
> The <bytes32> is trimmed on the right side to keep only the first 20 bytes. Therefore, implementations SHOULD ensure that the first 20 bytes are unique to avoid clashes. For example, the bytes32 typeId below:
>
> `0x11112222333344445555666677778888999990000aaaabbbbccccddddeeeeffff`
>
> will be trimmed to 0x11112222333344445555666677778888999990000.
>
> See the section about the trimming rules for the key type Mapping (./LSP-2-ERC725YJSONSchema.md#mapping) in LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode this data key.

- MUST return the returned value of the universalReceiverDelegate(address,uint256,bytes32,bytes) function on both retreived contract abi-encoded as bytes. If there is no addresses stored under the data keys above or the call was not forwarded to them, the return value is the two empty bytes abi-encoded as bytes.
- MUST emit a UniversalReceiver (./LSP-1-UniversalReceiver.md#events) event if the function was successful.

Check the UniversalReceiver Delegation > Specification section in LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md#universalreceiver-delegation) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

Events

ValueReceived

```
event ValueReceived(address indexed sender, uint256 indexed value);
```

MUST be emitted when a native token transfer was received.

ERC725Y Data Keys

LSP1UniversalReceiverDelegate

```
{
  "name": "LSP1UniversalReceiverDelegate",
  "key": "0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
  "keyType": "Singleton",
  "valueType": "address",
  "valueContent": "Address"
}
```

If the vault delegates its universal receiver functionality to another smart contract, this smart contract address MUST be stored under the data key attached above. This call to this contract is performed when the universalReceiver(bytes32,bytes) function of the vault is called and can react on the whole call regardless of typeId.

Check LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

Mapped LSP1UniversalReceiverDelegate

```
{
  "name": "LSP1UniversalReceiverDelegate:<bytes32>",
  "key": "0x0cfc51aec37c55a4d0b10000<bytes32>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

> <bytes32> is the typeId passed to the universalReceiver(..) function. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

If the vault delegates its universal receiver functionality to another smart contract, this smart contract address MUST be stored under the data key attached above. This call to this contract is performed when the universalReceiver(bytes32,bytes) function of the vault is called with a specific typeId that it can react on.

Check LSP1-UniversalReceiver (./LSP-1-UniversalReceiver.md) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

LSP17Extension

```
{
  "name": "LSP17Extension:<bytes4>",
  "key": "0xcee78b4094da860110960000<bytes4>",
  "keyType": "Mapping",
  "valueType": "address",
  "valueContent": "Address"
}
```

> <bytes4> is the functionSelector called on the vault contract. Check LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) to learn how to encode the key.

If there is a function called on the vault and the function does not exist, the fallback function lookup an address stored under the data key attached above and forwards the call to it with the value of the msg.sender and msg.value appended as extra calldata.

Check LSP17-ContractExtension (./LSP-17-ContractExtension.md) and LSP2-ERC725YJSONSchema (./LSP-2-ERC725YJSONSchema.md) for more information.

Graffiti

Graffiti refers to the arbitrary messages or data sent to an LSP9-Vault contract that do not match any existing function selectors, such as execute(..), setData(..), etc. These bytes, often carrying a message or additional information, are usually not intended to invoke specific functions within the contract.

When the vault is called with specific bytes that do not match any function selector, it will first check its storage to see if there are any extensions set for these function selectors (bytes). If no extension is found, the call will typically revert. However, to emulate the behavior of calling an Externally Owned Account (EOA) with random bytes (which always passes), an exception has been made for the 0x00000000 selector.

When the vault is called with data that starts with 0x00000000, it will first check for extensions. If none are found, the call will still pass, allowing it to match the behavior of calling an EOA and enabling the ability to send arbitrary messages to the vault. For example, one might receive a message like "This is a gift" while sending native tokens.

Additionally, it is possible to set an extension for the 0x00000000 selector. With this custom extension, you can define specific logic that runs when someone sends graffiti to your vault. For instance, you may choose to disallow sending graffiti by reverting the transaction, impose a fee for sending graffiti, or emit the graffiti on an external contract. This flexibility allows for various use cases and interactions with graffiti in the LSP9Vaults contracts.

Rationale

The ERC725Y general data key value store allows for the ability to add any kind of information to the contract, which allows future use cases. The general execution allows full interactability with any smart contract or address. And the universal receiver allows the reaction to any future asset.

Implementation

An implementation can be found in the lukso-network/lsp-smart-contracts (https://github.com/lukso-network/lsp-smart-contracts/blob/main/contracts/LSP9Vault/LSP9VaultCore.sol) repository.
The below defines the JSON interface of the LSP9Vault.

ERC725Y JSON Schema LSP9Vault:

```
[
  // From LSP1
  {
    "name":"LSP1UniversalReceiverDelegate",
    "key":"0x0cfc51aec37c55a4d0b1a65c6255c4bf2fbdf6277f3cc0730c45b828b6db8b47",
    "keyType":"Singleton",
    "valueType":"address",
    "valueContent":"Address"
  },
  {
    "name":"LSP1UniversalReceiverDelegate:<bytes32>",
    "key":"0x0cfc51aec37c55a4d0b10000<bytes32>",
    "keyType":"Mapping",
    "valueType":"address",
    "valueContent":"Address"
  },
  // From LSP17
  {
    "name":"LSP17Extension:<bytes4>",
    "key":"0xcee78b4094da860110960000<bytes4>",
    "keyType":"Mapping",
    "valueType":"address",
    "valueContent":"Address"
  }
]
```

Interface Cheat Sheet

```
interface ILSP9  /* is ERC165 */ {


    // ERC725X

    event Executed(uint256 indexed operation, address indexed to, uint256 indexed  value, bytes4 selector);\

    event ContractCreated(uint256 indexed operation, address indexed contractAddress, uint256 indexed
value);


    function execute(uint256 operationType, address to, uint256 value, bytes memory data) external payable
returns (bytes memory); // onlyOwner

    function executeBatch(uint256[] memory operationsType, address[] memory targets, uint256[] memory
values, bytes[] memory datas) external payable returns(bytes[] memory); // onlyOwner


    // ERC725Y

    event DataChanged(bytes32 indexed dataKey, bytes dataValue);
```

```
    function getData(bytes32 dataKey) external view returns (bytes memory dataValue);

    function setData(bytes32 dataKey, bytes memory dataValue) external; // onlyOwner

    function getDataBatch(bytes32[] memory dataKeys) external view returns (bytes[] memory dataValues);

    function setDataBatch(bytes32[] memory dataKeys, bytes[] memory dataValues) external; // onlyOwner


    // LSP9 (LSP9Vault)

    event ValueReceived(address indexed sender, uint256 indexed value);


    receive() external payable;

    fallback() external payable;

    function batchCalls(bytes[] calldata data) external returns (bytes[] memory results);


    // LSP1

    event UniversalReceiver(address indexed from, uint256 indexed value, bytes32 indexed typeId, bytes
receivedData, bytes returnedValue);


    function universalReceiver(bytes32 typeId, bytes memory data) external payable returns (bytes memory);


    // LSP14

    event OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner);

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    event RenounceOwnershipInitiated();

    event OwnershipRenounced();


    function owner() external view returns (address);

    function pendingOwner() external view returns (address);

    function transferOwnership(address newOwner) external; // onlyOwner

    function acceptOwnership() external;

    function renounceOwnership() external; // onlyOwner
```

```
}
```

Copyright

Copyright and related rights waived via CC0 (https://creativecommons.org/publicdomain/zero/1.0/).

</file>

<file>

path: /README.md

url: https://github.com/lukso-network/LIPs/blob/main/README.md

LIPs

LUKSO Improvement Proposals (LIPs) and LUKSO standard proposal (LSPs) describe standards for the LUKSO platform, including core protocol specifications, client APIs, and smart contract standards.

A browsable version of all current and draft LIPs can be found on the official LIP folder (https://github.com/lukso-network/LIPs/tree/master/LIPs).

Contributing

1. Review LIP-1 (LIPs/lip-1.md).
2. Fork the repository by clicking "Fork" in the top right.
3. Add your LIP to your fork of the repository. There is a template LIP here (lip-X.md).
4. Submit a Pull Request to LUKSO's LIPs repository (https://github.com/lukso-network/LIPs).

Your first PR should be a first draft of the final LIP. It must meet the formatting criteria enforced by the build (largely, correct metadata in the header). An editor will manually review the first PR for a new LIP and assign it a number before merging it. Make sure you include a discussions-to header with the URL to a discussion forum or open GitHub issue where people can discuss the LIP as a whole.

If your LIP requires images, the image files should be included in a subdirectory of the assets folder for that LIP as follow: assets/lip-X (for lip X). When linking to an image in the LIP, use relative links such as ../assets/lip-X/image.png.

When you believe your LIP is mature and ready to progress past the draft phase, you should do open a PR changing the state of your LIP to 'Final'. An editor will review your draft and ask if anyone objects to its being finalised. If the editor decides there is no rough consensus - for instance, because contributors point out significant issues with the LIP - they may close the PR and request that you fix the issues in the draft before trying again.

LIP Status Terms

- Draft - an LIP that is undergoing rapid iteration and changes.

- Last Call - an LIP that is done with its initial iteration and ready for review by a wide audience.
- Accepted - a core LIP that has been in Last Call for at least 2 weeks and any technical changes that were requested have been addressed by the author. The process for Core Devs to decide whether to encode an LIP into their clients as part of a hard fork is not part of the LIP process. If such a decision is made, the LIP wil move to final.
- Final (non-Core) - an LIP that has been in Last Call for at least 2 weeks and any technical changes that were requested have been addressed by the author.
- Final (Core) - an LIP that the Core Devs have decided to implement and release in a future hard fork or has already been released in a hard fork.
- Deferred - an LIP that is not being considered for immediate adoption. May be reconsidered in the future for a subsequent hard fork.

Preferred Citation Format

The canonical URL for a LIP that has achieved draft status at any point is at https://github.com/lukso-network/LIPs/tree/master/LIPs.

Terminology

The key words below are to be used to describe the specifications of a LIP or LSP standard. This terms are based are to be interpreted as described in [RFC 2119 (https://datatracker.ietf.org/doc/html/rfc2119)](https://datatracker.ietf.org/doc/html/rfc2119).

Terminology	Definition	Synonym
MUST	the definition is an absolute requirement of the specification.	REQUIRED
MUST NOT	the definition is an absolute prohibition of the specification.	FORBIDDEN, PROHIBITED
SHOULD	it is recommended to use and follow the specification, but there may exist valid reasons in particular circumstances where the specification can be ignored. In such cases, the full implications should be understood and carefully weighted before choosing an alternative.	RECOMMENDED
SHOULD NOT	it is not recommended to use the definition specified, but there may exist valid reasons in particular circumstances when the particular behaviour is acceptable or even useful. Before implementing any behaviour described as SHOULD NOT, the full implications should be understood and the case weighted carefully.	NOT RECOMMENDED
COULD	the specification is truly optional. One implementation may choose to include this particular specification because it feels that it enhances the feature, while another implementation may decide to omit it considering unnecessary. An implementation that does not include this optional specification MUST be prepared to interoperate with another implementation which does include this option, though perhaps with reduced functionality. In the same manner, an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)	MAY, OPTIONAL

</file>

<file>

path: /lip-X.md

url: https://github.com/lukso-network/LIPs/blob/main/lip-X.md

---

lip: <to be assigned>
title: <LIP title>
author: <a list of the author's or authors' name(s) and/or username(s), or name(s) and email(s), e.g. (use with the parentheses or triangular brackets): FirstName LastName (@GitHubUsername), FirstName LastName foo@bar.com (mailto:foo@bar.com), FirstName (@GitHubUsername) and GitHubUsername (@GitHubUsername)>
discussions-to: <URL>
status: Draft
type: <Standards Track (Core, Networking, Interface, LSP) | Informational | Meta>
category (*only required for Standard Track): <Core | Interface | LSP>*
created: <date created on, in ISO 8601 (yyyy-mm-dd) format>
*requires (*optional): <LIP number(s)>
replaces (*optional): <LIP number(s)>

<!--You can leave these HTML comments in your merged LIP and delete the visible duplicate text guides, they will not appear and may be helpful to refer to if you edit it again. This is the suggested template for new LIPs. Note that an LIP number will be assigned by an editor. When opening a pull request to submit your LIP, please use an abbreviated title in the filename, lip-draft_title_abbrev.md. The title should be 44 characters or less.-->
This is the suggested template for new LIPs.

Note that an LIP number will be assigned by an editor. When opening a pull request to submit your LIP, please use an abbreviated title in the filename, lip-draft_title_abbrev.md.

The title should be 44 characters or less.

Simple Summary

<!--"If you can't explain it simply, you don't understand it well enough." Provide a simplified and layman-accessible explanation of the LIP.-->
If you can't explain it simply, you don't understand it well enough." Provide a simplified and layman-accessible explanation of the LIP.

Abstract

<!--A short (~200 word) description of the technical issue being addressed.-->
A short (~200 word) description of the technical issue being addressed.

Motivation

<!--The motivation is critical for LIPs that want to change the Lukso protocol. It should clearly explain why the existing protocol specification is inadequate to address the problem that the LIP solves. LIP submissions without sufficient motivation may be rejected outright.-->
The motivation is critical for LIPs that want to change the Lukso protocol. It should clearly explain why the existing protocol specification is inadequate to address the problem that the LIP solves. LIP submissions without sufficient motivation may be rejected outright.

Specification

<!--The technical specification should describe the syntax and semantics of any new feature. The specification should be detailed enough to allow competing, interoperable implementations for any of the current Ethereum platforms (go-ethereum, parity, cpp-ethereum, ethereumj, ethereumjs, and others (https://github.com/ethereum/wiki/wiki/Clients)).-->
The technical specification should describe the syntax and semantics of any new feature. The specification should be detailed enough to allow competing, interoperable implementations for any of the current Ethereum platforms (go-ethereum, parity, cpp-ethereum, ethereumj, ethereumjs, and others (https://github.com/ethereum/wiki/wiki/Clients)).

Rationale

<!--The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.-->
The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, e.g. how the feature is supported in other languages. The rationale may also provide evidence of consensus within the community, and should discuss important objections or concerns raised during discussion.-->

Backwards Compatibility

<!--All LIPs that introduce backwards incompatibilities must include a section describing these incompatibilities and their severity. The LIP must explain how the author proposes to deal with these incompatibilities. LIP submissions without a sufficient backwards compatibility treatise may be rejected outright.-->
All LIPs that introduce backwards incompatibilities must include a section describing these incompatibilities and their severity. The LIP must explain how the author proposes to deal with these incompatibilities. LIP submissions without a sufficient backwards compatibility treatise may be rejected outright.

Test Cases

<!--Test cases for an implementation are mandatory for LIPs that are affecting consensus changes. Other LIPs can choose to include links to test cases if applicable.-->
Test cases for an implementation are mandatory for LIPs that are affecting consensus changes. Other LIPs can choose to include links to test cases if applicable.

Implementation

<!--The implementations must be completed before any LIP is given status "Final", but it need not be completed before the LIP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the principle of "rough consensus and running code" is still useful when it comes to resolving many discussions of API details.-->
The implementations must be completed before any LIP is given status "Final", but it need not be completed before the LIP is accepted. While there is merit to the approach of reaching consensus on the specification and rationale before writing code, the principle of "rough consensus and running code" is still useful when it comes to resolving many discussions of API details.

Copyright

</file>

</repository>

<repository>

lsp-utils

overview

repository: https://github.com/lukso-network/lsp-utils/
userName: lukso-network
repository: lsp-utils
branch: main
date: 2023-11-27T10:52:20+01:00 (1701078740)

<file>

path: /.commitlintrc.json

url: https://github.com/lukso-network/lsp-utils/blob/main/.commitlintrc.json

```
{
   "extends": ["@commitlint/config-conventional"]
}
```

</file>

<file>

path: /.eslintignore

url: https://github.com/lukso-network/lsp-utils/blob/main/.eslintignore

```
types/*
node_modules/*
```

</file>

<file>

path: /.eslintrc.json

url: https://github.com/lukso-network/lsp-utils/blob/main/.eslintrc.json

```
{
    "env": {
        "browser": true,
        "es2021": true
    },
    "extends": ["eslint:recommended", "plugin:@typescript-eslint/recommended"],
    "parser": "@typescript-eslint/parser",
    "parserOptions": {
        "ecmaVersion": 12,
        "sourceType": "module"
    },
    "plugins": ["@typescript-eslint"],
    "rules": {}
}
```

</file>

<file>

path: /.husky/commit-msg

url: https://github.com/lukso-network/lsp-utils/blob/main/.husky/commit-msg

```
#!/bin/sh
. "$(dirname "$0")/_/husky.sh"

npx --no-install commitlint --edit "$1"
```

</file>

<file>

path: /.husky/pre-commit

url: https://github.com/lukso-network/lsp-utils/blob/main/.husky/pre-commit

```
#!/usr/bin/env sh
. "$(dirname -- "$0")/_/husky.sh"

if [[ $(git diff origin/main --name-only src/) != "" ]];
then
    npm test
fi
```

</file>

<file>

path: /.prettierrc

url: https://github.com/lukso-network/lsp-utils/blob/main/.prettierrc

```
{
    "singleQuote": true,
    "tabWidth": 4,
    "printWidth": 100
}
```

</file>

<file>

path: /.versionrc.json

url: https://github.com/lukso-network/lsp-utils/blob/main/.versionrc.json

```
{
  "types": [
    {
      "type": "feat",
      "section": "Features"
    },
    {
      "type": "fix",
      "section": "Bug Fixes"
    },
    {
      "type": "chore",
      "hidden": true
    },
    {
      "type": "docs",
      "hidden": true
    },
    {
      "type": "style",
      "hidden": true
    },
    {
      "type": "refactor",
      "hidden": true
    },
    {
      "type": "perf",
      "hidden": true
    },
    {
      "type": "test",
      "hidden": true
    }
  ],
  "commitUrlFormat": "https://github.com/b00ste/lsp-utils/commits/{{hash}}",
  "compareUrlFormat": "https://github.com/b00ste/lsp-utils/compare/{{previousTag}}...{{currentTag}}"
}
```


</file>

<file>

path: /CHANGELOG.md

url: https://github.com/lukso-network/lsp-utils/blob/main/CHANGELOG.md

Changelog

All notable changes to this project will be documented in this file. See [standard-version (https://github.com/conventional-changelog/standard-version)](https://github.com/conventional-changelog/standard-version) for commit guidelines.

[0.0.2 (https://github.com/b00ste/lsp-utils/compare/v0.0.1...v0.0.2)](https://github.com/b00ste/lsp-utils/compare/v0.0.1...v0.0.2) (2023-10-15)

Features

- add AllowedCalls utils ([53eb309 (https://github.com/b00ste/lsp-utils/commits/53eb309538477251b8751cbecebd4bf96a71f784)](https://github.com/b00ste/lsp-utils/commits/53eb309538477251b8751cbecebd4bf96a71f784))
- add AllowedERC725YDataKeys utils ([4b929a4 (https://github.com/b00ste/lsp-utils/commits/4b929a497ca05a3803deadbcedce788b9834be08)](https://github.com/b00ste/lsp-utils/commits/4b929a497ca05a3803deadbcedce788b9834be08))
- add validityTimestamp util function ([3ecb48a (https://github.com/b00ste/lsp-utils/commits/3ecb48aa8202a5f41cbab8d3cd1d70f53065aab2)](https://github.com/b00ste/lsp-utils/commits/3ecb48aa8202a5f41cbab8d3cd1d70f53065aab2))
- export constants ([87f3dd2 (https://github.com/b00ste/lsp-utils/commits/87f3dd2e9357a4bfb98a7351db94013b9b12afd4)](https://github.com/b00ste/lsp-utils/commits/87f3dd2e9357a4bfb98a7351db94013b9b12afd4))

0.0.1 (2023-10-15)

Features

- create initial lib ([a15b1a4 (https://github.com/b00ste/lsp-utils/commits/a15b1a4ebff3c53dc18c85fecceb3ff35e918309)](https://github.com/b00ste/lsp-utils/commits/a15b1a4ebff3c53dc18c85fecceb3ff35e918309))
  </file>

<file>

path: /README.md

url: https://github.com/lukso-network/lsp-utils/blob/main/README.md

LSP Utils · npm unparseable json response [(https://www.npmjs.com/package/@b00ste/lsp-utils)](https://www.npmjs.com/package/@b00ste/lsp-utils) coverage unknown [(https://coveralls.io/github/b00ste/lsp-utils?branch=main)](https://coveralls.io/github/b00ste/lsp-utils?branch=main)

This package was created with the intent to help developers to use @lukso/lsp-smart-contracts. Its purpose is to provide a series of helper functions for each LSP.

- For more information on LSPs see [Documentation (https://docs.lukso.tech/standards/smart-contracts/introduction)](https://docs.lukso.tech/standards/smart-contracts/introduction) on *docs.lukso.tech (https://docs.lukso.tech/standards/introduction)*.
- For more information on LIPs see [Specification (https://github.com/lukso-network/LIPs)](https://github.com/lukso-network/LIPs)

:warning: *This package is currently in early stages of development,
 use for testing or experimentation purposes only.*

Installation

npm

@b00ste/lsp-utils is available as a npm package (https://www.npmjs.com/package/@b00ste/lsp-utils).

```
npm install @b00ste/lsp-utils
```

cloning the repository

Alternatively you can also clone the repository and install its dependencies to start using the smart contracts.

```
$ git clone https://github.com/b00ste/lsp-utils.git
$ cd ./lsp-utils
$ npm install
```

Usage

in Javascript

You can use the utils by importing them as follow:

ES6 Modules:

```
import { encodeAllowedCalls } from '@b00ste/lsp-utils/dist/lib/es6';

const allowedCalls = encodeAllowedCalls(
    allowedInteractions,
    allowedAddresses,
    allowedStandards,
    allowedFunctions,
);
```

CommonJS

```
cosnt { encodeAllowedCalls } = require('@b00ste/lsp-utils/dist/lib/es5');

const allowedCalls = encodeAllowedCalls(
    allowedInteractions,
    allowedAddresses,
    allowedStandards,
    allowedFunctions
);
```

</file>

<file>

path: /dist/package.json

url: https://github.com/lukso-network/lsp-utils/blob/main/dist/package.json

```
{
  "name": "@b00ste/lsp-utils",
  "version": "0.0.1",
  "description": "A set of TypeScript/JavaScript utils for LUKSO Standard Proposals (LSPs)",
  "author": "Daniel Afteni (B00ste)",
  "license": "MIT",
  "homepage": "https://github.com/b00ste/lsp-utils#readme",
  "bugs": {
    "url": "https://github.com/b00ste/lsp-utils/issues"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/b00ste/lsp-utils.git"
  },
  "main": "lib/es5/index.js",
  "module": "lib/es6/index.js",
  "sideEffects": false,
  "files": [
    "lib",
    "CHANGELOG.md",
    "LICENSE",
    "package.json",
    "README.md"
  ],
  "keywords": [
    "javascript",
    "typecript",
    "lsp-utils",
    "lsp utils",
    "lsp-utilities",
    "lsp utilities",
    "lip-utils",
    "lip utils",
    "lip-utilities",
    "lip utilities",
    "lukso-utils",
    "lukso utils",
    "lukso-utilities",
    "lukso utilities",
    "lukso",
    "lsps",
    "lsp",
    "lips",
    "lip"
  ]
}
```


</file>

<file>

path: /package.json

url: https://github.com/lukso-network/lsp-utils/blob/main/package.json


```
{
  "name": "@b00ste/lsp-utils",
  "version": "0.0.2",
  "description": "A set of TypeScript/JavaScript utils for LUKSO Standard Proposals (LSPs)",
  "author": "Daniel Afteni (B00ste)",
  "license": "MIT",
  "homepage": "https://github.com/b00ste/lsp-utils#readme",
  "bugs": {
    "url": "https://github.com/b00ste/lsp-utils/issues"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/b00ste/lsp-utils.git"
  },
  "files": [
    "dist",
    "./README.md",
    "./CHANGELOG.md",
    "./LICENSE"
  ],
  "scripts": {
    "lint": "eslint . --ext .ts",
    "prettier": "npx prettier -w .",
    "build:types": "npx typechain --target=ethers-v6 --out-dir types ./node_modules/@lukso/lsp-smart-contracts/artifacts/*.json",
    "build": "rm -rf dist/lib && rm -rf types && npm run lint && npm run build:types && bash scripts/ignoreTypesFolder.bash && tsc && tsc --build tsconfig.es5.json",
    "test": "env TS_NODE_COMPILER_OPTIONS='{\"module\": \"commonjs\" }' mocha -r ts-node/register 'tests/**/*.ts'",
    "prepare": "husky install",
    "release": "standard-version",
    "release:minor": "standard-version --release-as minor",
    "release:patch": "standard-version --release-as patch",
    "release:major": "standard-version --release-as major",
    "publish": "git push --follow-tags origin main && npm publish"
  },
  "dependencies": {
    "@commitlint/cli": "^17.8.0",
    "@commitlint/config-conventional": "^17.8.0",
    "prettier": "^3.0.3"
  },
  "devDependencies": {
    "@lukso/lsp-smart-contracts": "^0.11.1",
    "@typechain/ethers-v6": "^0.5.0",
    "@types/chai": "^4.3.8",
    "@types/mocha": "^10.0.2",
    "@typescript-eslint/eslint-plugin": "^6.7.5",
```

```
    "@typescript-eslint/parser": "^6.7.5",
    "chai": "^4.3.10",
    "coveralls": "^3.1.0",
    "ejs": "^3.1.5",
    "eslint": "^7.17.0",
    "ethers": "^6.8.0",
    "husky": "^8.0.0",
    "js-htmlencode": "^0.3.0",
    "jsdoc-parse-plus": "^1.3.0",
    "mocha": "^10.2.0",
    "nyc": "^15.1.0",
    "standard-version": "^9.5.0",
    "ts-node": "^10.9.1",
    "typechain": "^8.3.1",
    "typescript": "^4.1.3"
  },
  "keywords": [
    "javascript",
    "typecript",
    "lsp-utils",
    "lsp utils",
    "lsp-utilities",
    "lsp utilities",
    "lip-utils",
    "lip utils",
    "lip-utilities",
    "lip utilities",
    "lukso-utils",
    "lukso utils",
    "lukso-utilities",
    "lukso utilities",
    "lukso",
    "lsps",
    "lsp",
    "lips",
    "lip"
  ]
}
```

</file>

<file>

path: /scripts/ignoreTypesFolder.bash

url: https://github.com/lukso-network/lsp-utils/blob/main/scripts/ignoreTypesFolder.bash

```
for FILE in types/*.ts;
do
echo '// @ts-nocheck' | cat - $FILE > temp && mv temp $FILE
done

for FILE in types/**/*.ts;
do
echo '// @ts-nocheck' | cat - $FILE > temp && mv temp $FILE
done
```

</file>

<file>

path: /src/IPFS/validateIpfsUrl.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/IPFS/validateIpfsUrl.ts

```
import { defaultIpfsGateway } from '../constants';

/**
 * Returns a valid URL. If it is an IPFS URL (E.g. &amp;nbsp;ipfs://{hash}&amp;nbsp;), the IPFS Gateway will be used to generate a valid link. Otherwise the link is returned.
 *
 * @since v0.0.1
 * @category IPFS
 * @param url The URL that is to be validated.
 * @param ipfsGateway The IPFS Gateway to be used for IPFS URLs.
 * @example
 * validateIpfsUrl('ipfs://{hash}') //=> 'https://2eff.lukso.dev/ipfs/{hash}'
 * validateIpfsUrl('https://google.com/something') //=> 'https://google.com/something'
 * validateIpfsUrl('') //=> ''
 */
export const validateIpfsUrl = (url: string, ipfsGateway?: string): string => {
    return url.startsWith('ipfs://')
        ? ipfsGateway
            ? ipfsGateway + url.replace('ipfs://', '')
            : defaultIpfsGateway + url.replace('ipfs://', '')
        : url;
};
```

</file>

<file>

path: /src/LSP2/decodeAssetUrl.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/decodeAssetUrl.ts

```
import { BytesLike, toUtf8String } from 'ethers';
```

```
/**
 * Decode a JSONURL value content.
 *
 * @since v0.0.1
 * @category LSP2
 * @param assetUrlValue The encoded value as  { "valueContent": "ASSETURL" } .
 *
 * @throws When  assetUrlValue  his composed of less than 36 bytes.
 *
 * @return
 *    
 * {
 *   // The hash digest of the function used to hash the JSON file.
 *   "hashFunction": string
 *   // the hashed bytes value of the JSON file.
 *   "json": string
 *   // The URL where the JSON file is hosted.
 *   "url": string
 * }
 *    
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 *    
 *
decodeAssetUrl("0x6f357c6a2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c0535
68747470733a2f2f676f6f676c652e636f6d2f") //=>
 * // {
 * //   hashFunction: "0x6f357c6a",
 * //   hash: "0x2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c0535",
 * //   url: "https://google.com/"
 * // }
 *    
 */
export const decodeAssetUrl = (assetUrlValue: BytesLike) => {
    const strippedAssetUrlValue = assetUrlValue.toString().substring(2);

    if (strippedAssetUrlValue.length < 72) {
        throw new Error( Invalid 'JSONURL' value. Less than 36 bytes. Value: ${assetUrlValue} );
    }

    const decodedJSONURL = {
        hashFunction:  0x${strippedAssetUrlValue.substring(0, 8)} ,
        hash:  0x${strippedAssetUrlValue.substring(8, 72)} ,
        url: toUtf8String( 0x${strippedAssetUrlValue.substring(72)} ),
    };

    return decodedJSONURL;
};
```

</file>

<file>

path: /src/LSP2/decodeJsonUrl.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/decodeJsonUrl.ts

```
import { BytesLike, toUtf8String } from 'ethers';

/**
 * Decode a JSONURL value content.
 *
 * @since v0.0.1
 * @category LSP2
 * @param jsonUrlValue The encoded value as  { "valueContent": "JSONURL" } .
 *
 * @throws When  jsonUrlValue  his composed of less than 36 bytes.
 *
 * @return
 *    
 * {
 *   // The hash digest of the function used to hash the JSON file.
 *   "hashFunction": string
 *   // the hashed bytes value of the JSON file.
 *   "json": string
 *   // The URL where the JSON file is hosted.
 *   "url": string
 * }
 *    
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 *    
 *
decodeJsonUrl("0x6f357c6a4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a418496
8747470733a2f2f676f6f676c652e636f6d2f") //=>
 * // {
 * //   hashFunction: "0x6f357c6a",
 * //   hash: "0x4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a41849",
 * //   url: "https://google.com/"
 * // }
 *    
 */
export const decodeJsonUrl = (jsonUrlValue: BytesLike) => {
    const stringifiedJsonUrlValue = jsonUrlValue.toString().substring(2);

    if (stringifiedJsonUrlValue.length < 72) {
        throw new Error( Invalid 'JSONURL' value. Less than 36 bytes. Value: ${jsonUrlValue} );
    }

    const decodedJSONURL = {
        hashFunction:  0x${stringifiedJsonUrlValue.substring(0, 8)} ,
```

```
        hash: &amp;nbsp;0x${stringifiedJsonUrlValue.substring(8, 72)}&amp;nbsp;,
        url: toUtf8String(&amp;nbsp;0x${stringifiedJsonUrlValue.substring(72)}&amp;nbsp;),
    };

    return decodedJSONURL;
};
```

</file>

<file>

path: /src/LSP2/encodeAssetUrl.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/encodeAssetUrl.ts

```
import { concat, keccak256, toUtf8Bytes } from 'ethers';

/**
 * Generate a ASSETURL value content.
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param hashFunction The function used to hash the JSON file.
 * @param assetBytes Bytes value of the JSON file.
 * @param url The URL where the JSON file is hosted.
 *
 * @return The encoded value as an &amp;nbsp;ASSETURL&amp;nbsp;.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * encodeAssetUrl(
 *   "keccak256(utf8)",
 *   "{\"name\":\"USDStablecoin\",\"description\":\"Some random description about the token USD
Stablecoin\"}",
 *   "https://google.com/"
 * ) //=>
"0x6f357c6a2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c053568747470733a2f
2f676f6f676c652e636f6d2f"
 */
export const encodeAssetUrl = (hashFunction: string, assetBytes: string, url: string) => {
    const hashFunctionDigest = keccak256(toUtf8Bytes(hashFunction)).substring(0, 10);
    const jsonDigest = keccak256(toUtf8Bytes(assetBytes));

    const ASSETURLValue = concat([hashFunctionDigest, jsonDigest, toUtf8Bytes(url)]);

    return ASSETURLValue;
};
```

</file>

<file>

path: /src/LSP2/encodeJsonUrl.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/encodeJsonUrl.ts

```
import { concat, keccak256, toUtf8Bytes } from 'ethers';

/**
 * Encode a JSONURL value content.
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param hashFunction The function used to hash the JSON file.
 * @param json Bytes value of the JSON file.
 * @param url The URL where the JSON file is hosted.
 *
 * @return The encoded value as an  JSONURL .
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * encodeJsonUrl(
 *   "keccak256(utf8)",
 *   "{\"name\":\"Tom\",\"description\":\"Some random description about Tom\"}",
 *   "https://google.com/"
 * ) //=>
"0x6f357c6a4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a4184968747470733a2f
2f676f6f676c652e636f6d2f"
 */
export const encodeJsonUrl = (hashFunction: string, json: string, url: string) => {
    const hashFunctionDigest = keccak256(toUtf8Bytes(hashFunction)).substring(0, 10);
    const jsonDigest = keccak256(toUtf8Bytes(json));

    const JSONURLValue = concat([hashFunctionDigest, jsonDigest, toUtf8Bytes(url)]);

    return JSONURLValue;
};
```

</file>

<file>

path: /src/LSP2/generateArrayElementKeyAtIndex.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/generateArrayElementKeyAtIndex.ts

```
import { concat, isHexString, toBeHex } from 'ethers';
import { generateArrayKey } from './generateArrayKey';

/**
 * Generates a data key of  { "keyType": "Array" }  at a specific  index .
 arrayKey  can have the following values:
 * 1. An array data key, 32 bytes hex value.
 * 2. An array data key name of type  dataKeyName[]  that will be used to generate an array
data key using  generateArrayKey(arrayKey) .
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param arrayKey The Array data key from which to generate the Array data key at a specific
 index .
 * @param index The index number in the  arrayKey .
 *
 * @return The generated  bytes32  data key of key type Array at a specific  index .
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - generateSingletonKey(arrayDataKeyName, index) //=>  
<bytes16(keccak256(arrayDataKeyName))>:<bytes16(index)> 
 * - generateSingletonKey(arrayDataKey, index) //=>  <bytes16(arrayDataKey)>:
<bytes16(index)> 
 */
export const generateArrayElementKeyAtIndex = (arrayKey: string, index: number) => {
    let arrayKeyHex = arrayKey;

    if (!isHexString(arrayKey, 32)) {
        arrayKeyHex = generateArrayKey(arrayKey);
    }

    const elementInArray = concat([arrayKeyHex.substring(0, 34), toBeHex(index, 16)]);

    return elementInArray;
};
```

</file>

<file>

path: /src/LSP2/generateArrayKey.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/generateArrayKey.ts

```
import { keccak256, toUtf8Bytes } from 'ethers';

/**
 * Generates a data key of  { "keyType": "Array" }  by hashing  arrayKeyName .
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param arrayKeyName The string that will be used to generate a data key of key type Array.
 *
 * @return The generated  bytes32  data key of key type Array.
 *
 * @throws  keyName  has less than 2 characters.
 * @throws  keyName  does not include square brackets  "[]"  at the end of the string.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * generateArrayKey("RandomArrayDataKey[]") //=> keccak256("RandomArrayDataKey[]") = "0x6e9974ec39571e80dcc2ab1fac99097b03bb4617b071cd519a23d38f88f28ffb"
 */
export const generateArrayKey = (arrayKeyName: string) => {
    if (arrayKeyName.length < 2) {
        throw new Error('Array data key name must be longer than 2 characters.');
    }

    if (!arrayKeyName.endsWith('[]')) {
        throw new Error("Missing empty square brackets '[]' at the end of the data key name.");
    }

    return keccak256(toUtf8Bytes(arrayKeyName));
};
```

</file>

<file>

path: /src/LSP2/generateMappingKey.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/generateMappingKey.ts

```
import { concat, isHexString, keccak256, toUtf8Bytes } from 'ethers';

/**
 * Generates a data key of  { "keyType": "Mapping" }  that map  firstPart  to  lastPart .
 *
 *  firstPart  can have the following values:
 * 1. A 10 bytes hex value.
 * 2. A hex value that will be hashed (first 10 bytes will be used).
 * 2. A UTF8 string that will be hashed (first 10 bytes will be used).
```

```
 *
 *  lastPart  can have the following values:
 * 1. An address.
 * 1. A 20 bytes hex value.
 * 2. A hex value that will be hashed (first 20 bytes will be used).
 * 2. A UTF8 string that will be hashed (first 20 bytes will be used).
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param firstPart The word to retrieve the first 10 bytes of its hash.
 * @param lastPart The word to retrieve the first 10 bytes of its hash.
 *
 * @return The generated  bytes32  data key of key type Mapping that map
 firstPart  to a specific  lastPart .
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - generateMappingKey(firstWord, lastWord)
 * //=> <bytes10(keccak256(firstWord))>:<0000>:<bytes20(keccak256(lastWord))> 
 *
 * - generateMappingKey(firstWord, bytes20value)
 * //=> <bytes10(keccak256(firstWord))>:<0000>:<bytes20value> 
 *
 * - generateMappingKey(bytes10Value, lastWord)
 * //=> <bytes10Value>:<0000>:<bytes20(keccak256(lastWord))> 
 *
 * - generateMappingKey(bytes10Value, bytes20value)
 * //=> <bytes10Value>:<0000>:<bytes20value> 
 */
export const generateMappingKey = (firstPart: string, lastPart: string) => {
    let firstPartHex = firstPart;

    if (!isHexString(firstPart, 10)) {
        if (isHexString(firstPart)) {
            firstPartHex = keccak256(firstPart).substring(0, 22);
        } else {
            firstPartHex = keccak256(toUtf8Bytes(firstPart)).substring(0, 22);
        }
    }

    let lastPartHex = lastPart;

    if (!isHexString(lastPart, 20)) {
        if (isHexString(lastPart)) {
            lastPartHex = keccak256(lastPart).substring(0, 42);
        } else {
            lastPartHex = keccak256(toUtf8Bytes(lastPart)).substring(0, 42);
        }
    }

    const mappingDataKey = concat([firstPartHex, '0x0000', lastPartHex]);
```

```
    return mappingDataKey;
};
```

</file>

<file>

path: /src/LSP2/generateMappingWithGroupingKey.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/generateMappingWithGroupingKey.ts

```
import { concat, isHexString, keccak256, toUtf8Bytes } from 'ethers';

/**
 * Generates a data key of &amp;nbsp;{ "keyType": "MappingWithGrouping" }&amp;nbsp; that map
&amp;nbsp;firstPart&amp;nbsp; to &amp;nbsp;middlePart&amp;nbsp; and to &amp;nbsp;lastPart&amp;nbsp;.
 *
 * &amp;nbsp;firstPart&amp;nbsp; can have the following values:
 * 1. A 6 bytes hex value.
 * 2. A hex value that will be hashed (first 6 bytes will be used).
 * 2. A UTF8 string that will be hashed (first 6 bytes will be used).
 *
 * &amp;nbsp;middlePart&amp;nbsp; can have the following values:
 * 1. A 4 bytes hex value.
 * 2. A hex value that will be hashed (first 4 bytes will be used).
 * 2. A UTF8 string that will be hashed (first 4 bytes will be used).
 *
 * &amp;nbsp;lastPart&amp;nbsp; can have the following values:
 * 1. An address.
 * 1. A 20 bytes hex value.
 * 2. A hex value that will be hashed (first 20 bytes will be used).
 * 2. A UTF8 string that will be hashed (first 20 bytes will be used).
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param firstPart The word to retrieve the first 6 bytes of its hash.
 * @param middlePart The word to retrieve the first 4 bytes of its hash.
 * @param lastPart The word to retrieve the first 20 bytes of its hash.
 *
 * @return The generated &amp;nbsp;bytes32&amp;nbsp; data key of key type MappingWithGrouping that map a
&amp;nbsp;firstWord&amp;nbsp; to a &amp;nbsp;secondWord&amp;nbsp; to a specific address &amp;nbsp;addr&amp;nbsp;.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - generateMappingWithGroupingKey(firstWord, middleWord, lastWord)
 * //=> &amp;nbsp;<bytes6(keccak256(firstWord))>:<bytes4(keccak256(middleWord))>:<0000>:
<bytes20(keccak256(lastWord))>&amp;nbsp;
 *
 * - generateMappingWithGroupingKey(firstWord, middleWord, bytes20Value)
```

```
 * //=>  <bytes6(keccak256(firstWord))>:<bytes4(keccak256(middleWord))>:<0000>:
<bytes20Value> 
 *
 * - generateMappingWithGroupingKey(firstWord, bytes4Value, lastWord)
 * //=>  <bytes6(keccak256(firstWord))>:<bytes4Value>:<0000>:
<bytes20(keccak256(lastWord))> 
 *
 * - generateMappingWithGroupingKey(firstWord, bytes4Value, bytes20Value)
 * //=>  <bytes6(keccak256(firstWord))>:<bytes4Value>:<0000>:<bytes20Value> 
 *
 * - generateMappingWithGroupingKey(bytes6Value, middleWord, lastWord)
 * //=>  <bytes6Value>:<bytes4(keccak256(middleWord))>:<0000>:
<bytes20(keccak256(lastWord))> 
 *
 * - generateMappingWithGroupingKey(bytes6Value, middleWord, bytes20Value)
 * //=>  <bytes6Value>:<bytes4(keccak256(middleWord))>:<0000>:<bytes20Value> 
 *
 * - generateMappingWithGroupingKey(bytes6Value, bytes4Value, lastWord)
 * //=>  <bytes6Value>:<bytes4Value>:<0000>:<bytes20(keccak256(lastWord))> 
 *
 * - generateMappingWithGroupingKey(bytes6Value, bytes4Value, bytes20Value)
 * //=>  <bytes6Value>:<bytes4Value>:<0000>:<bytes20Value> 
 */
export const generateMappingWithGroupingKey = (
    firstPart: string,
    middlePart: string,
    lastPart: string,
) => {
    let firstPartHex = firstPart;

    if (!isHexString(firstPart, 6)) {
        if (isHexString(firstPart)) {
            firstPartHex = keccak256(firstPart).substring(0, 14);
        } else {
            firstPartHex = keccak256(toUtf8Bytes(firstPart)).substring(0, 14);
        }
    }

    let middlePartHex = middlePart;

    if (!isHexString(middlePart, 4)) {
        if (isHexString(middlePart)) {
            middlePartHex = keccak256(middlePart).substring(0, 10);
        } else {
            middlePartHex = keccak256(toUtf8Bytes(middlePart)).substring(0, 10);
        }
    }

    let lastPartHex = lastPart;

    if (!isHexString(lastPart, 20)) {
        if (isHexString(lastPart)) {
```

```
        lastPartHex = keccak256(lastPart).substring(0, 42);
      } else {
        lastPartHex = keccak256(toUtf8Bytes(lastPart)).substring(0, 42);
      }
    }

    const mappingWithGroupingDataKey = concat([firstPartHex, middlePartHex, '0x0000', lastPartHex]);

    return mappingWithGroupingDataKey;
};
```

</file>

<file>

path: /src/LSP2/generateSingletonKey.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/generateSingletonKey.ts

```
import { keccak256, toUtf8Bytes } from 'ethers';

/**
 * Generates a data key of &amp;nbsp;{ "keyType": "Singleton" }&amp;nbsp; by hashing the string
&amp;nbsp;keyName&amp;nbsp;.
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param keyName The string to hash to generate a Singleton data key.
 *
 * @return The generated &amp;nbsp;bytes32&amp;nbsp; data key of key type Singleton.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * generateSingletonKey("RandomDataKey") //=> keccak256("RandomKeyName") =
"0xb0c92ac98a2a422f33a3e130e3fa6e922195f0a0a991999638140123 51f906cb"
 */
export const generateSingletonKey = (keyName: string) => {
    return keccak256(toUtf8Bytes(keyName));
};
```

</file>

<file>

path: /src/LSP2/isCompactBytesArray.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/isCompactBytesArray.ts

```
import { BytesLike, isHexString, toNumber } from 'ethers';
```

```
/**
 * Verify if &amp;nbsp;data&amp;nbsp; is a valid array of value encoded as a &amp;nbsp;CompactBytesArray&amp;nbsp;
according to the LSP2 &amp;nbsp;CompactBytesArray&amp;nbsp; valueType specification.
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param compactBytesArray The bytes value to verify.
 *
 * @return &amp;nbsp;true&amp;nbsp; if the &amp;nbsp;data&amp;nbsp; is correctly encoded CompactBytesArray,
&amp;nbsp;false&amp;nbsp; otherwise.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - isCompactBytesArray("0x0002cafe000abeefdeadbeef0000cafe") //=> true
 * - isCompactBytesArray("0x0002cafecafe000abeefdeadbeef0000cafe") //=> false
 * - isCompactBytesArray("0x0002") //=> false
 */
export const isCompactBytesArray = (compactBytesArray: BytesLike) => {
  if (!isHexString(compactBytesArray)) {
    throw new Error(&amp;nbsp;'compactBytesArray' is not hex. Value: '${compactBytesArray}'&amp;nbsp;);
  }

  /**
   * Pointer will always land on these values:
   *
   * ↓↓↓↓
   * 0003 a00000
   * 0005 fff83a0011
   * 0020 aa0000000000000000000000000000000000000000000000000000000000cafe
   * 0012 bb00000000000000000000000000beef
   * 0019 cc0000000000000000000000000000000000000000deed
   * ↑↑↑↑
   *
   * The pointer can only land on the length of the following bytes value.
   */
  let pointer = 0;
  const strippedCompactBytesArray = compactBytesArray.substring(2);

  /**
   * Check each length byte and make sure that when you reach the last length byte.
   * Make sure that the last length describes exactly the last bytes value and you do not get out of bounds.
   */
  while (pointer < strippedCompactBytesArray.length) {
    if (pointer + 4 >= strippedCompactBytesArray.length) return false;
    const elementLength = strippedCompactBytesArray.substring(pointer, pointer + 4);

    pointer += toNumber(&amp;nbsp;0x${elementLength}&amp;nbsp;) * 2 + 4;
  }
  if (pointer === strippedCompactBytesArray.length) return true;
  return false;
```

```
};
```

</file>

<file>

path: /src/LSP2/isValidArrayLengthValue.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/isValidArrayLengthValue.ts

```
import { BytesLike, isHexString } from 'ethers';

/**
 * Validates if the bytes &amp;nbsp;arrayLength&amp;nbsp; are exactly 16 bytes long, and are of the exact size of an LSP2 Array length value
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param arrayLength Plain bytes that should be validated.
 *
 * @return &amp;nbsp;true&amp;nbsp; if the value is 16 bytes long, &amp;nbsp;false&amp;nbsp; otherwise.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - isValidArrayLengthValue("0x00000000000000000000000000000001") //=> true
 * - isValidArrayLengthValue("0x0000000000000000000000000000a3b") //=> true
 * - isValidArrayLengthValue("0x000000000000000000000000004a") //=> false
 * - isValidArrayLengthValue("0x00000000000000000000000000000000f60a") //=> false
 */
export const isValidArrayLengthValue = (arrayLength: BytesLike) => {
    if (isHexString(arrayLength, 16)) {
        return true;
    }

    return false;
};
```

</file>

<file>

path: /src/LSP2/removeElementFromArrayAndMap.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/removeElementFromArrayAndMap.ts

```
// ethers
import { BytesLike, concat, isHexString, toBeHex } from 'ethers';

// types
```

```
import { UniversalProfile } from '../../types';
import { generateArrayElementKeyAtIndex } from './generateArrayElementKeyAtIndex';
import { generateMappingKey } from './generateMappingKey';

/**
 * Generates Data Key/Value pairs for removing an element from an LSP2 Array and a mapping Data Key.
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param erc725YContract The ERC725Y contract.
 * @param arrayKey The Data Key of Key Type Array.
 * @param newArrayLength The new Array Length for the  arrayKey .
 * @param removedElementIndexKey The Data Key of Key Type Array Index for the removed element.
 * @param removedElementIndex the index of the removed element.
 * @param removedElementMapKey The Data Key of a mapping to be removed.
 *
 * @return A set of data keys & data values that can be used to update an array and map in ERC725Y
storage.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - removeLastElementFromArrayAndMap(...) //=> { dataKeys: BytesLike[], dataValues: BytesLike[] }
 */
export const removeElementFromArrayAndMap = async (
    erc725YContract: UniversalProfile,
    arrayKey: BytesLike,
    newArrayLength: number,
    removedElementIndexKey: BytesLike,
    removedElementIndex: number,
    removedElementMapKey: BytesLike,
) => {
    if (!isHexString(arrayKey, 32)) {
        throw new Error( 'arrayKey' data key is not of length 32 bytes. Value: '${arrayKey}' );
    }

    if (!isHexString(removedElementIndexKey, 32)) {
        throw new Error(
             'removedElementIndexKey' data key is not of length 32 bytes. Value:
'${removedElementIndexKey}' ,
        );
    }

    if (!isHexString(removedElementMapKey, 32)) {
        throw new Error(
             'removedElementMapKey' data key is not of length 32 bytes. Value:
'${removedElementMapKey}' ,
        );
    }

    const dataKeys: BytesLike[] = [];
    const dataValues: BytesLike[] = [];
```

```
  // store the number of received assets decremented by 1
  dataKeys[0] = arrayKey;
  dataValues[0] = toBeHex(newArrayLength, 16);

  // remove the data value for the map key of the element
  dataKeys[1] = removedElementMapKey;
  dataValues[1] = '';

  // Generate the key of the last element in the array
  const lastElementIndexKey = generateArrayElementKeyAtIndex(arrayKey, newArrayLength);

  // Get the data value from the key of the last element in the array
  const lastElementIndexValue = await erc725YContract.getData(lastElementIndexKey);

  if (!isHexString(lastElementIndexValue, 20)) {
    throw new Error(
       'lastElementIndexValue' data key is not of length 20 bytes. Value:
'${lastElementIndexValue}' ,
    );
  }

  // Set data value of the last element instead of the element from the array that will be removed
  dataKeys[2] = removedElementIndexKey;
  dataValues[2] = lastElementIndexValue;

  // Remove the data value for the swapped array element
  dataKeys[3] = lastElementIndexKey;
  dataValues[3] = '';

  // Generate mapping key for the swapped array element
  const lastElementMapKey = generateMappingKey(removedElementMapKey, lastElementIndexValue);

  // Generate the mapping value for the swapped array element
  const lastElementMapValue = concat([
    (await erc725YContract.getData(lastElementMapKey)).substring(0, 10),
    toBeHex(removedElementIndex, 16),
  ]);

  // Update the map value of the swapped array element to the new index
  dataKeys[4] = lastElementMapKey;
  dataValues[4] = lastElementMapValue;

  return { dataKeys, dataValues };
};
```

</file>

<file>

path: /src/LSP2/removeLastElementFromArrayAndMap.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP2/removeLastElementFromArrayAndMap.ts

```
import { BytesLike, isHexString, toBeHex } from 'ethers';

/**
 * Generates Data Key/Value pairs for removing the last element from an LSP2 Array and a mapping Data Key.
 *
 * @since v0.0.1
 * @category LSP2
 *
 * @param arrayKey The Data Key of Key Type Array.
 * @param newArrayLength The new Array Length for the  arrayKey .
 * @param removedElementIndexKey The Data Key of Key Type Array Index for the removed element.
 * @param removedElementMapKey The Data Key of a mapping to be removed.
 *
 * @return A set of data keys & data values that can be used to update an array and map in ERC725Y storage.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md
 * @example
 * - removeLastElementFromArrayAndMap(...) //=> { dataKeys: BytesLike[], dataValues: BytesLike[] }
 */
export const removeLastElementFromArrayAndMap = (
    arrayKey: BytesLike,
    newArrayLength: number,
    removedElementIndexKey: BytesLike,
    removedElementMapKey: BytesLike,
) => {
    if (!isHexString(arrayKey, 32)) {
        throw new Error( 'arrayKey' data key is not of length 32 bytes. Value: '${arrayKey}' );
    }

    if (!isHexString(removedElementIndexKey, 32)) {
        throw new Error(
             'removedElementIndexKey' data key is not of length 32 bytes. Value: '${removedElementIndexKey}' ,
        );
    }

    if (!isHexString(removedElementMapKey, 32)) {
        throw new Error(
             'removedElementMapKey' data key is not of length 32 bytes. Value: '${removedElementMapKey}' ,
        );
    }

    const dataKeys: BytesLike[] = [];
    const dataValues: BytesLike[] = [];

    // store the number of received assets decremented by 1
    dataKeys[0] = arrayKey;
```

```
    dataValues[0] = toBeHex(newArrayLength, 16);

    // remove the data value for the map key of the element
    dataKeys[1] = removedElementMapKey;
    dataValues[1] = '';

    // remove the data value for the map key of the element
    dataKeys[2] = removedElementIndexKey;
    dataValues[2] = '';

    return { dataKeys, dataValues };
};
```

</file>

<file>

path: /src/LSP3/getProfileMetadata.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP3/getProfileMetadata.ts

```
import { ERC725YDataKeys } from '@lukso/lsp-smart-contracts';

// constants
import { defaultLSP3ProfileMetadata } from '../constants';

// IPFS Utils
import { validateIpfsUrl } from '../IPFS/validateIpfsUrl';

// LSP2 Utils
import { decodeJsonUrl } from '../LSP2/decodeJsonUrl';

// LSP3 Utils
import { isProfileMetadata } from './isProfileMetadata';

// types
import { UniversalProfile } from '../../types';

/**
 * Returns a object of type LSP3ProfileMetadata.
 *
 * @since v0.0.1
 * @category LSP3
 *
 * @param unviersalProfile The instance of a Unviersal Profile contract.
 *
 * @throws When the fetched data is not  LSP3ProfileMetadata .
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-3-Profile-Metadata.md
 * @example
 *    
```

```
 * getMetadata(UniversalProfile) //=>
 * // {
 * //   LSP3Profile: {
 * //     description: "",
 * //     links: [],
 * //     name: "",
 * //     tags: []
 * //   }
 * // }
 *    
 */
export const getProfileMetadata = async (unviersalProfile: UniversalProfile) => {
    const profileMetadataDataValue = await unviersalProfile.getData(
        ERC725YDataKeys.LSP3.LSP3Profile,
    );

    const JSONURL = decodeJsonUrl(profileMetadataDataValue);

    const profileDataURL = validateIpfsUrl(JSONURL.url);

    let profileData;

    await fetch(profileDataURL)
        .then(async (result) => await result.json())
        .then((result) => (profileData = result));

    if (!isProfileMetadata(profileData)) {
        throw new Error('Fetched data is not an  LSP3ProfileMetadata  object.');
    }

    return profileData ? profileData : defaultLSP3ProfileMetadata;
};
```

</file>

<file>

path: /src/LSP3/isProfileMetadata.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP3/isProfileMetadata.ts

```
import { LSP3ProfileMetadata } from '../constants';

/**
 * Returns  true  is the passed object is an LSP3 Profile Metadata,  false 
otherwise.
 *
 * @since v0.0.1
 * @category LSP3
 * @param object - The object that is to be checked.
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-3-Profile-Metadata.md
 * @example
 *    
 * isProfileMetadata({ LSP3Profile: { description: "", links: [], name: "", tags: [] } }) //=> true
 * isProfileMetadata({ description: "", links: [], name: "", tags: [] }) //=> false
 *    
 */
export const isProfileMetadata = (
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    object: any,
): object is LSP3ProfileMetadata => {
    return (
        'LSP3Profile' in object &&
        'description' in object.LSP3Profile &&
        'links' in object.LSP3Profile &&
        'name' in object.LSP3Profile &&
        'tags' in object.LSP3Profile
    );
};
```

</file>

<file>

path: /src/LSP5/generateReceivedAssetKeys.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP5/generateReceivedAssetKeys.ts

```
import { BytesLike, concat, isHexString, toBeHex, toNumber } from 'ethers';

//  @luksp/lsp-smart-contracts  constants
import { ERC725YDataKeys } from '@lukso/lsp-smart-contracts';

// LSP2 utils
import { isValidArrayLengthValue } from '../LSP2/isValidArrayLengthValue';
import { generateMappingKey } from '../LSP2/generateMappingKey';
import { generateArrayElementKeyAtIndex } from '../LSP2/generateArrayElementKeyAtIndex';

// types
import { UniversalProfile } from '../../types';

/**
```

```
 * Generate an array of Data Key/Value pairs to be set on the receiver address after receiving assets.
 *
 * @since v0.0.1
 * @category LSP5
 *
 * @param erc725YContract The contract instance of the asset reciever.
 * @param assetAddress The address of the asset being received (_e.g: an LSP7 or LSP8 token_).
 * @param assetInterfaceId The interfaceID of the asset being received.
 *
 * @return A set of LSP5 data keys & data values that can be used to update an array and map in ERC725Y
storage.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-5-ReceivedAssets.md
 * @example
 * - generateReceivedAssetKeys(...) //=> { lsp5DataKeys: BytesLike[], lsp5DataValues: BytesLike[] }
 */
export const generateReceivedAssetKeys = async (
    erc725YContract: UniversalProfile,
    assetAddress: BytesLike,
    assetInterfaceId: BytesLike,
) => {
    if (isHexString(assetAddress, 20)) {
        throw new Error( 'assetAddress' bytes length is not 20. Value: ${assetAddress} );
    }

    if (isHexString(assetInterfaceId, 4)) {
        throw new Error( 'assetInterfaceId' bytes length is not 4. Value: ${assetInterfaceId} );
    }

    // ---  LSP5ReceivedAssets[]  Array ---
    let currentArrayLengthBytes = await erc725YContract.getData(
        ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length,
    );

    // CHECK that the value of  LSP5ReceivedAssets[]  Array length is a valid
 uint128  (16 bytes long)
    if (!isValidArrayLengthValue(currentArrayLengthBytes)) {
        if (currentArrayLengthBytes === '0x' || currentArrayLengthBytes === '') {
            // if it's the first asset received and nothing is set (= 0x)
            // we need to convert it to:  0x00000000000000000000000000000000 
            // to safely cast to a uint128 of length 0
            currentArrayLengthBytes = toBeHex(0, 16);
        } else {
            // otherwise the array length is invalid
            throw new Error(
                 'LSP5ReceivedAssets[]' length invalid. Value: ${currentArrayLengthBytes} ,
            );
        }
    }

    // CHECK for potential overflow
    if (currentArrayLengthBytes ===  0x${'ff'.repeat(16)} ) {
```

```
      throw new Error(
        &amp;nbsp;'LSP5ReceivedAssets[]' length reached max value. Value:
${currentArrayLengthBytes}&amp;nbsp;,
      );
    }

    const currentArrayLength = toNumber(currentArrayLengthBytes);

    // --- &amp;nbsp;LSP5ReceivedAssetsMap:<assetAddress>&amp;nbsp; ---

    const mapDataKey = generateMappingKey(
      ERC725YDataKeys.LSP5.LSP5ReceivedAssetsMap,
      assetAddress.toString(),
    );

    // CHECK that the map value is not already set in the storage for the newly received asset
    // If that's the case, the asset is already registered. Do not try to update.
    const fetchedMapValue = await erc725YContract.getData(mapDataKey);
    if (!(fetchedMapValue === '0x') && !(fetchedMapValue === '')) {
      throw new Error(&amp;nbsp;Asset already registred. Asset address: ${assetAddress}&amp;nbsp;);
    }

    // --- LSP5 Data Keys &amp; Values ---

    const lsp5DataKeys: BytesLike[] = [];
    const lsp5DataValues: BytesLike[] = [];

    // Increment &amp;nbsp;LSP5ReceivedAssets[]&amp;nbsp; Array length
    lsp5DataKeys[0] = ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length;
    lsp5DataValues[0] = toBeHex(currentArrayLength + 1, 16);

    // Add asset address to &amp;nbsp;LSP5ReceivedAssets[index]&amp;nbsp;, where index == previous array length
    lsp5DataKeys[1] = generateArrayElementKeyAtIndex(
      ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length,
      currentArrayLength,
    );
    lsp5DataValues[1] = assetAddress;

    // Add interfaceId + index as value under &amp;nbsp;LSP5ReceivedAssetsMap:<assetAddress>&amp;nbsp;
    lsp5DataKeys[2] = mapDataKey;
    lsp5DataValues[2] = concat([assetInterfaceId, currentArrayLengthBytes]);

    return { lsp5DataKeys, lsp5DataValues };
};
```

</file>

<file>

path: /src/LSP5/generateSentAssetKeys.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP5/generateSentAssetKeys.ts

```
// ethers
import { BytesLike, toBeHex, toNumber } from 'ethers';

//  @lukso/lsp-smart-contracts  constants
import { ERC725YDataKeys } from '@lukso/lsp-smart-contracts';

// LSP2 utils
import { isValidArrayLengthValue } from '../LSP2/isValidArrayLengthValue';
import { generateMappingKey } from '../LSP2/generateMappingKey';
import { generateArrayElementKeyAtIndex } from '../LSP2/generateArrayElementKeyAtIndex';
import { removeLastElementFromArrayAndMap } from '../LSP2/removeLastElementFromArrayAndMap';
import { removeElementFromArrayAndMap } from '../LSP2/removeElementFromArrayAndMap';

// types
import { UniversalProfile } from '../../types';

/**
 * Generate an array of Data Key/Value pairs to be set on the sender address after sending assets.
 *
 * @since v0.0.1
 * @category LSP5
 *
 * @param erc725YContract The contract instance of the asset sender.
 * @param assetAddress The address of the asset that is being sent.
 *
 * @return A set of LSP5 data keys & data values that can be used to update an array and map in ERC725Y
storage.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-5-ReceivedAssets.md
 * @example
 * - generateSentAssetKeys(...) //=> { lsp5DataKeys: BytesLike[], lsp5DataValues: BytesLike[] }
 */
export const generateSentAssetKeys = async (
    erc725YContract: UniversalProfile,
    assetAddress: BytesLike,
) => {
    // ---  LSP5ReceivedAssets[]  Array ---
    const currentArrayLengthBytes = await erc725YContract.getData(
        ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length,
    );

    // CHECK that the value of  LSP5ReceivedAssets[]  Array length is a valid
 uint128  (16 bytes long)
    if (!isValidArrayLengthValue(currentArrayLengthBytes)) {
        throw new Error( 'LSP5ReceivedAssets[]' length invalid. Value:
${currentArrayLengthBytes} );
    }

    // CHECK for potential underflow
    if (currentArrayLengthBytes === toBeHex(0, 16)) {
```

```
    throw new Error("'LSP5ReceivedAssets[]' length is 0. Cannot remove asset.");
  }

  const newArrayLength = toNumber(currentArrayLengthBytes) - 1;

  // ---  LSP5ReceivedAssetsMap:<assetAddress>  ---

  const removedElementMapKey = generateMappingKey(
    ERC725YDataKeys.LSP5.LSP5ReceivedAssetsMap,
    assetAddress.toString(),
  );

  // Query the ERC725Y storage of the LSP0-ERC725Account
  const mapValue = await erc725YContract.getData(removedElementMapKey);

  // CHECK if no map value was set for the asset to remove.
  // If that's the case, there is nothing to remove. Do not try to update.
  if (mapValue === '' || mapValue === '0x') {
    throw new Error(
       Asset is not registerd length is 0. Cannot remove asset. Asset address:
'${assetAddress}' ,
    );
  }
  if (mapValue.length !== 20) {
    throw new Error(
       Registered asset has invalid data in the map. Asset address: '${assetAddress}'. Map data:
'${mapValue}'. ,
    );
  }

  // Extract index of asset to remove from the map value
  const removedElementIndex = toNumber( 0x${mapValue.substring(10)} );

  const removedElementIndexKey = generateArrayElementKeyAtIndex(
    ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length,
    removedElementIndex,
  );

  if (removedElementIndex === newArrayLength) {
    return removeLastElementFromArrayAndMap(
      ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length,
      newArrayLength,
      removedElementIndexKey,
      removedElementMapKey,
    );
  } else if (removedElementIndex < newArrayLength) {
    return removeElementFromArrayAndMap(
      erc725YContract,
      ERC725YDataKeys.LSP5['LSP5ReceivedAssets[]'].length,
      newArrayLength,
      removedElementIndexKey,
      removedElementIndex,
```

```
            removedElementMapKey,
        );
    } else {
        // If index is bigger than the array length, out of bounds
        throw new Error(
             Element index is out of the array bounds. Array length: '${
                newArrayLength + 1
            }'. Asset index: '${removedElementIndex}' ,
        );
    }
};
```

</file>

<file>

path: /src/LSP6/createValidityTimestamp.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP6/createValidityTimestamp.ts

```
import { BigNumberish, concat, toBeHex } from 'ethers';

/**
 * Create a  validityTimestamp  that can be used in  LSP6.executeRelayCall(...) 
 *
 * @since v0.0.1
 * @category LSP6
 * @param startingTimestamp The timestamp after which a relay call can be executed.
 * @param endingTimestamp The timestamp after which a relay call cannot be executed.
 *
 * @throws When teh bytes value of either  startingTimestamp  or
 endingTimestamp  exceeds 16 bytes.
 *
 * @return A hex value of 32 bytes that contains both starting & ending timestamps.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-6-KeyManager.md
 * @example
 *    
 * createValidityTimestamp(5, 10) //=>
 0x00000000000000000000000000000005000000000000000000000000000000a 
 *    
 */
export const createValidityTimestamp = (
    startingTimestamp: number | BigNumberish,
    endingTimestamp: number | BigNumberish,
) => {
    if (toBeHex(startingTimestamp).length > 34) {
        throw new Error( The hex value of the number: '${startingTimestamp}' exeeeds 16
bytes );
    }

    if (toBeHex(endingTimestamp).length > 34) {
        throw new Error( The hex value of the number: '${endingTimestamp}' exeeeds 16 bytes );
    }

    return concat([toBeHex(startingTimestamp, 16), toBeHex(endingTimestamp, 16)]);
};
```

</file>

<file>

path: /src/LSP6/decodeAllowedCalls.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP6/decodeAllowedCalls.ts

```
import { BytesLike, isHexString, toNumber } from 'ethers';

/**
 * Decode AllowedCalls encoded as  { "valueType": "bytes32[CompactBytesArray]" } .
 *
```

```
 * @since v0.0.1
 * @category LSP6
 * @param allowedCalls A list of allowed calls as &amp;nbsp;{ "valueType": "bytes32[CompactBytesArray]"
}&amp;nbsp;.
 *
 * @throws
 * - When the value of &amp;nbsp;allowedCalls&amp;nbsp; is not hex.
 * - When the bytes length of any allowed call is different from 32.
 * - When the length of an element reaches past the length of &amp;nbsp;allowedCalls&amp;nbsp;
 *
 * @return The allowed interactions, addresses, functions and standards that were encoded.
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-6-KeyManager.md
 * @example
 * &amp;nbsp;&amp;nbsp;&amp;nbsp;
 *
decodeAllowedCalls("0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c00200
0000003cafecafecafecafecafecafecafecafecafecafe24871b3d44c028fe") //=>
 * // {
 * //   allowedInteractions: ["0x00000002", "0x00000003"],
 * //   allowedAddresses: ["0xcafecafecafecafecafecafecafecafecafecafe",
"0xcafecafecafecafecafecafecafecafecafecafe"],
 * //   allowedStandards: ["0x24871b3d", "0x24871b3d"],
 * //   allowedFunctions: ["0x7f23690c", "0x44c028fe"],
 * // }
 *
 * decodeAllowedCalls("0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c")
//=>
 * // {
 * //   allowedInteractions: ["0x00000002"],
 * //   allowedAddresses: ["0xcafecafecafecafecafecafecafecafecafecafe"],
 * //   allowedStandards: ["0x24871b3d"],
 * //   allowedFunctions: ["0x7f23690c"],
 * // }
 * &amp;nbsp;&amp;nbsp;&amp;nbsp;
 */
export const decodeAllowedCalls = (allowedCalls: BytesLike) => {
    if (!isHexString(allowedCalls)) {
        throw new Error(&amp;nbsp;'${allowedCalls}' is not hex&amp;nbsp;);
    }

    const strippedAllowedCalls = allowedCalls.substring(2);
    let pointer = 0;

    const allowedInteractions: BytesLike[] = [];
    const allowedAddresses: BytesLike[] = [];
    const allowedStandards: BytesLike[] = [];
    const allowedFunctions: BytesLike[] = [];

    while (pointer < strippedAllowedCalls.length) {
        const bytesLength = toNumber(
            '0x' + strippedAllowedCalls.toString().substring(pointer, pointer + 4),
```

```
      );
      // increment to skip length
      pointer += 4;

      if (bytesLength !== 32) {
        throw new Error( Invalid length. Length: '${bytesLength}'. Must be 32. );
      }

      const stringLength = bytesLength * 2;

      if (pointer + stringLength > strippedAllowedCalls.length) {
        throw new Error(
          'Out of bounds, length of an element reaches past the length of  allowedCalls ',
        );
      }

      const allowedCall = strippedAllowedCalls
        .toString()
        .substring(pointer, pointer + stringLength);

      allowedInteractions.push( 0x${allowedCall.substring(0, 8)} );
      allowedAddresses.push( 0x${allowedCall.substring(8, 48)} );
      allowedStandards.push( 0x${allowedCall.substring(48, 56)} );
      allowedFunctions.push( 0x${allowedCall.substring(56)} );

      // increment to skip element
      pointer += stringLength;
    }

    return { allowedInteractions, allowedAddresses, allowedStandards, allowedFunctions };
};
```

</file>

<file>

path: /src/LSP6/decodeAllowedERC725YDataKeys.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP6/decodeAllowedERC725YDataKeys.ts

```
import { BytesLike, isHexString, toNumber } from 'ethers';

/**
 * Decode AllowedERC725YDataKeys encoded as  { "valueType": "bytes[CompactBytesArray]"
} .
 *
 * @since v0.0.1
 * @category LSP6
 * @param allowedERC725YDataKeys A list of allowed calls as  { "valueType":
"bytes[CompactBytesArray]" } .
 *
```

```
* @throws
* - When the value of &amp;nbsp;allowedERC725YDataKeys&amp;nbsp; is not hex.
* - When the bytes length of any allowed ERC725Y data key is 0 or bigger than 32.
* - When the length of an element reaches past the length of &amp;nbsp;allowedERC725YDataKeys&amp;nbsp;
*
* @return The allowed ERC725Y data keys that were encoded.
*
* @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-6-KeyManager.md
* @example
* &amp;nbsp;&amp;nbsp;&amp;nbsp;
* decodeAllowedERC725YDataKeys("0x0002cafe000abeefdeadbeef0000cafe") //=>
* // [
* //   "0xcafe",
* //   "0xbeefdeadbeef0000cafe"
* // ]
* &amp;nbsp;&amp;nbsp;&amp;nbsp;
*/
export function decodeAllowedERC725YDataKeys(allowedERC725YDataKeys: BytesLike) {
  if (!isHexString(allowedERC725YDataKeys)) {
    throw new Error(&amp;nbsp;'${allowedERC725YDataKeys}' is not hex&amp;nbsp;);
  }

  const strippedAllowedERC725YDataKeys = allowedERC725YDataKeys.substring(2);
  let pointer = 0;

  const dataKeys: BytesLike[] = [];

  while (pointer < strippedAllowedERC725YDataKeys.length) {
    const bytesLength = toNumber(
      '0x' + strippedAllowedERC725YDataKeys.toString().substring(pointer, pointer + 4),
    );
    // increment to skip length
    pointer += 4;

    if (bytesLength === 0 || bytesLength > 32) {
      throw new Error(
        &amp;nbsp;Invalid length. Length: '${bytesLength}'. Must be bigger than 0 and smaller than 32.&amp;nbsp;,
      );
    }

    const stringLength = bytesLength * 2;

    if (pointer + stringLength > strippedAllowedERC725YDataKeys.length) {
      throw new Error(
        'Out of bounds, length of an element reaches past the length of
&amp;nbsp;allowedERC725YDataKeys&amp;nbsp;',
      );
    }

    dataKeys.push(
      '0x' +
        strippedAllowedERC725YDataKeys
```

```
                .toString()
                .substring(pointer, pointer + stringLength),
        );

        // increment to skip element
        pointer += stringLength;
    }

    return dataKeys;
}
```

</file>

<file>

path: /src/LSP6/encodeAllowedCalls.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP6/encodeAllowedCalls.ts

```
import { BytesLike, isHexString } from 'ethers';

/**
 * Enocde a list of data keys as  { "valueType": "bytes32[CompactBytesArray]" } . The result can be user for  AddressPermissions:AllowedCalls:<address> 
 *
 * @since v0.0.1
 * @category LSP6
 * @param allowedInteractions A list of allowed interactions.
 * @param allowedAddresses A list of allowed addresses.
 * @param allowedStandards A list of allowed standards.
 * @param allowedFunctions A list of allowed functions.
 *
 * @throws
 * - When the arrays passed as parameters don't have the same length.
 * - When one of  allowedInteractions[index]  has a bytes length different from 4.
 * - When one of  allowedAddresses[index]  has a bytes length different from 20.
 * - When one of  allowedStandards[index]  has a bytes length different from 4.
 * - When one of  allowedFunctions[index]  has a bytes length different from 4.
 *
 * @return The compacted array of allowed calls as  { "valueType": "bytes32[CompactBytesArray]" } .
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-6-KeyManager.md
 * @example
 *    
 * encodeAllowedCalls(
 *   ["0x00000002", "0x00000003"],
 *   ["0xcafecafecafecafecafecafecafecafecafecafe", "0xcafecafecafecafecafecafecafecafecafecafe"]
 *   ["0x24871b3d", "0x24871b3d"],
 *   ["0x7f23690c", "0x44c028fe"],
 * ) //=>
```

```
"0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c002000000003cafecafecafe
cafecafecafecafecafecafecafecafe24871b3d44c028fe"
 * encodeAllowedCalls(
 *   ["0x00000002"],
 *   ["0xcafecafecafecafecafecafecafecafecafecafe"]
 *   ["0x24871b3d"],
 *   ["0x7f23690c"],
 * ) //=> "0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c"
 *    
 */
export function encodeAllowedCalls(
    allowedInteractions: BytesLike[],
    allowedAddresses: BytesLike[],
    allowedStandards: BytesLike[],
    allowedFunctions: BytesLike[],
) {
    if (
        allowedInteractions.length !== allowedAddresses.length &&
        allowedAddresses.length !== allowedStandards.length &&
        allowedStandards.length !== allowedFunctions.length
    ) {
        throw new Error(
            'Arrays must have the same length. ' +
                 'allowedInteractions' length: ${allowedInteractions}.   +
                 'allowedAddresses' length: ${allowedAddresses}.   +
                 'allowedStandards' length: ${allowedStandards}.   +
                 'allowedFunctions' length: ${allowedFunctions}. ,
        );
    }

    let result = '0x0020';

    for (let ii = 0; ii < allowedStandards.length; ii++) {
        // remove "0x" prefixes

        if (!isHexString(allowedInteractions[ii])) {
            throw new Error(
                 Allowed interaction is not hex. Allowed interaction: '${allowedInteractions[ii]}' ,
            );
        }
        if (!isHexString(allowedInteractions[ii], 4)) {
            throw new Error(
                 Allowed interactions has invalid bytes length. Must be 4. Length: ${
                    allowedInteractions[ii].length / 2 - 1
                } ,
            );
        }

        const allowedInteraction = allowedInteractions[ii].toString().substring(2);

        if (!isHexString(allowedAddresses[ii])) {
            throw new Error(
```

```
             Allowed address is not hex. Allowed address: '${allowedAddresses[ii]}' ,
        );
    }
    if (!isHexString(allowedAddresses[ii], 20)) {
        throw new Error(
             Allowed interactions has invalid bytes length. Must be 20. Length: ${
                allowedAddresses[ii].length / 2 - 1
            } ,
        );
    }

    const allowedAddress = allowedAddresses[ii].toString().substring(2).toLowerCase();

    if (!isHexString(allowedStandards[ii])) {
        throw new Error(
             Allowed standard is not hex. Allowed standard: '${allowedStandards[ii]}' ,
        );
    }
    if (!isHexString(allowedStandards[ii], 4)) {
        throw new Error(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
                allowedStandards[ii].length / 2 - 1
            } ,
        );
    }

    const allowedStandard = allowedStandards[ii].toString().substring(2);

    if (!isHexString(allowedFunctions[ii])) {
        throw new Error(
             Allowed function is not hex. Allowed function: '${allowedFunctions[ii]}' ,
        );
    }
    if (!isHexString(allowedFunctions[ii], 4)) {
        throw new Error(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
                allowedFunctions[ii].length / 2 - 1
            } ,
        );
    }

    const allowedFunction = allowedFunctions[ii].toString().substring(2);

    result = result + allowedInteraction + allowedAddress + allowedStandard + allowedFunction;

    if (ii != allowedStandards.length - 1) {
        result = result + '0020';
    }
}

return result;
}
```

</file>

<file>

path: /src/LSP6/encodeAllowedERC725YDataKeys.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/LSP6/encodeAllowedERC725YDataKeys.ts

```
import { BytesLike, isHexString, toBeHex } from 'ethers';

/**
 * Enocde a list of data keys as  { "valueType": "bytes[CompactBytesArray]" } . The result can be user for  AddressPermissions:AllowedERC725YDataKeys:<address> 
 *
 * @since v0.0.1
 * @category LSP6
 * @param dataKeys A list of data keys.
 *
 * @throws
 * - When one of  dataKeys[index]  is not hex.
 * - When one of  dataKeys[index]  has a length of 0 bytes or bigger thsn 32 bytes.
 *
 * @return The compacted array of data keys as  { "valueType": "bytes[CompactBytesArray]" } .
 *
 * @see https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-6-KeyManager.md
 * @example
 *    
 * encodeAllowedERC725YDataKeys([
 *   "0xcafe",
 *   "0xbeefdeadbeef0000cafe"
 * ]) //=> "0x0002cafe000abeefdeadbeef0000cafe"
 *    
 */
export const encodeAllowedERC725YDataKeys = (dataKeys: BytesLike[]) => {
    let compactedBytes = '0x';

    for (let i = 0; i < dataKeys.length; i++) {
        const dataKey = dataKeys[i];

        if (!isHexString(dataKey)) {
            throw new Error( '${dataKey}' is not hex );
        }

        const strippedDataKey = dataKey.replace('0x', '');

        if (strippedDataKey.length === 0 || strippedDataKey.length > 64) {
            throw new Error(
                 Invalid length. Length: '${
                    strippedDataKey.length / 2
                }'. Must be bigger than 0 and smaller than 32. ,
            );
        }

        compactedBytes += toBeHex(strippedDataKey.length, 2).substring(2) + strippedDataKey;
    }

    return compactedBytes;
};
```

</file>

<file>

path: /src/constants/index.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/constants/index.ts

```
export interface LSP3ProfileMetadata {
    LSP3Profile: {
        description: string;
        links: string[];
        name: string;
        tags: string[];
        profileImage?: {
            width: number;
            height: number;
            hashFunction: string;
            hash: string;
            url: string;
        }[];
        backgroundImage?: {
            width: number;
            height: number;
            hashFunction: string;
            hash: string;
            url: string;
        }[];
    };
}

export const defaultLSP3ProfileMetadata: LSP3ProfileMetadata = {
    LSP3Profile: {
        description: '',
        links: [],
        name: '',
        tags: [],
    },
};

export const defaultIpfsGateway = 'https://2eff.lukso.dev/ipfs/';
```

</file>

<file>

path: /src/index.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/src/index.ts

```ts
// ------ Constants ------
export { LSP3ProfileMetadata, defaultIpfsGateway, defaultLSP3ProfileMetadata } from './constants';

// ------ IPFS ------
export { validateIpfsUrl } from './IPFS/validateIpfsUrl';

// ------ LSP2 ------
export { decodeAssetUrl } from './LSP2/decodeAssetUrl';
export { decodeJsonUrl } from './LSP2/decodeJsonUrl';
export { encodeAssetUrl } from './LSP2/encodeAssetUrl';
export { encodeJsonUrl } from './LSP2/encodeJsonUrl';
export { generateArrayElementKeyAtIndex } from './LSP2/generateArrayElementKeyAtIndex';
export { generateArrayKey } from './LSP2/generateArrayKey';
export { generateMappingKey } from './LSP2/generateMappingKey';
export { generateMappingWithGroupingKey } from './LSP2/generateMappingWithGroupingKey';
export { generateSingletonKey } from './LSP2/generateSingletonKey';
export { isCompactBytesArray } from './LSP2/isCompactBytesArray';
export { isValidArrayLengthValue } from './LSP2/isValidArrayLengthValue';
export { removeElementFromArrayAndMap } from './LSP2/removeElementFromArrayAndMap';
export { removeLastElementFromArrayAndMap } from './LSP2/removeLastElementFromArrayAndMap';

// ------ LSP3 ------
export { getProfileMetadata } from './LSP3/getProfileMetadata';
export { isProfileMetadata } from './LSP3/isProfileMetadata';

// ------ LSP5 ------
export { generateReceivedAssetKeys } from './LSP5/generateReceivedAssetKeys';
export { generateSentAssetKeys } from './LSP5/generateSentAssetKeys';

// ------ LSP6 ------
export { decodeAllowedCalls } from './LSP6/decodeAllowedCalls';
export { encodeAllowedCalls } from './LSP6/encodeAllowedCalls';
export { decodeAllowedERC725YDataKeys } from './LSP6/decodeAllowedERC725YDataKeys';
export { encodeAllowedERC725YDataKeys } from './LSP6/encodeAllowedERC725YDataKeys';
export { createValidityTimestamp } from './LSP6/createValidityTimestamp';
```

</file>

<file>

path: /tests/IPFS/validateIpfsUrl.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/IPFS/validateIpfsUrl.test.ts

```
import { expect } from 'chai';
import { keccak256, toUtf8Bytes } from 'ethers';
import { defaultIpfsGateway, validateIpfsUrl } from '../../src';

describe('validateIpfsUrl', () => {
    it('IPFS URL with default gateway', () => {
        const randomHash = keccak256(toUtf8Bytes('RandomHash')).substring(2);

        expect(validateIpfsUrl(&amp;nbsp;ipfs://${randomHash}&amp;nbsp;)).to.equal(defaultIpfsGateway +
randomHash);
    });

    it('IPFS URL with custom gateway', () => {
        const randomHash = keccak256(toUtf8Bytes('RandomHash')).substring(2);
        const customGateway = 'https://custom.gateway.io/';

        expect(validateIpfsUrl(&amp;nbsp;ipfs://${randomHash}&amp;nbsp;, customGateway)).to.equal(
            customGateway + randomHash,
        );
    });

    it('Normal URL without custom gateway', () => {
        const url = 'https://google.com';

        expect(validateIpfsUrl(url)).to.equal(url);
    });

    it('Normal URL with custom gateway', () => {
        const url = 'https://google.com';
        const customGateway = 'https://custom.gateway.io/';

        expect(validateIpfsUrl(url, customGateway)).to.equal(url);
    });
});
```

</file>

<file>

path: /tests/LSP2/decodeAssetUrl.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/decodeAssetUrl.test.ts

```
import { expect } from 'chai';
import { decodeAssetUrl } from '../../src';

describe('decodeAssetUrl', () => {
    it('should pass if ASSETURL has bytes length bigger than 36', () => {
        const ASSETURL =

'0x6f357c6a2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c053568747470733a2f
2f676f6f676c652e636f6d2f';

        const expectedJsonUrl = {
            hashFunction: '0x6f357c6a',
            hash: '0x2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c0535',
            url: 'https://google.com/',
        };

        expect(decodeAssetUrl(ASSETURL)).to.deep.equal(expectedJsonUrl);
    });

    it('should pass if ASSETURL has bytes length equal to 36', () => {
        const ASSETURL =
            '0x6f357c6a2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c0535';

        const expectedJsonUrl = {
            hashFunction: '0x6f357c6a',
            hash: '0x2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c0535',
            url: '',
        };

        expect(decodeAssetUrl(ASSETURL)).to.deep.equal(expectedJsonUrl);
    });

    it('should throw if ASSETURL has bytes length smaller than 36', () => {
        const ASSETURL = '0xcafedeafbeef';

        expect(() => decodeAssetUrl(ASSETURL)).to.throw(
             Invalid 'JSONURL' value. Less than 36 bytes. Value: ${ASSETURL} ,
        );
    });
});
```

</file>

<file>

path: /tests/LSP2/decodeJsonUrl.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/decodeJsonUrl.test.ts

```
import { expect } from 'chai';
import { decodeJsonUrl } from '../../src';

describe('decodeJsonUrl', () => {
  it('should pass if JSONURL has bytes length bigger than 36', () => {
    const JSONURL =

'0x6f357c6a4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a4184968747470733a2f
2f676f6f676c652e636f6d2f';

    const expectedJsonUrl = {
      hashFunction: '0x6f357c6a',
      hash: '0x4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a41849',
      url: 'https://google.com/',
    };

    expect(decodeJsonUrl(JSONURL)).to.deep.equal(expectedJsonUrl);
  });

  it('should pass if JSONURL has bytes length equal to 36', () => {
    const JSONURL =
      '0x6f357c6a4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a41849';

    const expectedJsonUrl = {
      hashFunction: '0x6f357c6a',
      hash: '0x4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a41849',
      url: '',
    };

    expect(decodeJsonUrl(JSONURL)).to.deep.equal(expectedJsonUrl);
  });

  it('should throw if JSONURL has bytes length smaller than 36', () => {
    const JSONURL = '0xcafedeafbeef';

    expect(() => {
      decodeJsonUrl(JSONURL);
    }).to.throw( Invalid 'JSONURL' value. Less than 36 bytes. Value: ${JSONURL} );
  });
});
```

</file>

<file>

path: /tests/LSP2/encodeAssetUrl.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/encodeAssetUrl.test.ts

```
import { expect } from 'chai';
import { encodeAssetUrl } from '../../src';

describe('encodeAssetUrl', () => {
    it('should pass when encoding any  hashFunction ,  assetBytes ,
 url ', () => {
        expect(
            encodeAssetUrl(
                'keccak256(utf8)',
                '{"name":"USDStablecoin","description":"Some random description about the token USD
Stablecoin"}',
                'https://google.com/',
            ),
        ).to.equal(

'0x6f357c6a2a04850096912391bbb0966a624519e8f5d797df2a2c47425e892c25e00c053568747470733a2f
2f676f6f676c652e636f6d2f',
        );
    });
});
```

</file>

<file>

path: /tests/LSP2/encodeJsonUrl.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/encodeJsonUrl.test.ts

```
import { expect } from 'chai';
import { encodeJsonUrl } from '../../src';

describe('encodeJsonUrl', () => {
    it('should pass when encoding any  hashFunction ,  json ,  url ', ()
=> {
        expect(
            encodeJsonUrl(
                'keccak256(utf8)',
                '{"name":"Tom","description":"Some random description about Tom"}',
                'https://google.com/',
            ),
        ).to.equal(

'0x6f357c6a4dade694d7dd4081f46073e99ce898a9b53cf6988452904de7db5cc704a4184968747470733a2f
2f676f6f676c652e636f6d2f',
        );
    });
});
```

</file>

<file>

path: /tests/LSP2/generateArrayElementKeyAtIndex.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/generateArrayElementKeyAtIndex.test.ts

```
import { expect } from 'chai';
import { concat, keccak256, toBeHex, toUtf8Bytes } from 'ethers';
import { generateArrayElementKeyAtIndex } from '../../src';

describe('generateArrayElementKeyAtIndex', () => {
    it('should pas if &amp;nbsp;arrayKey&amp;nbsp; is a UTF8 string', () => {
        const arrayKeyName = 'SomeArray[]';
        const index = 2;

        expect(generateArrayElementKeyAtIndex(arrayKeyName, index)).to.equal(
            concat([keccak256(toUtf8Bytes(arrayKeyName)).substring(0, 34), toBeHex(index, 16)]),
        );
    });

    it('should pas if &amp;nbsp;arrayKey&amp;nbsp; is a hex string', () => {
        const arrayKey = keccak256(toUtf8Bytes('SomeArray[]'));
        const index = 2;

        expect(generateArrayElementKeyAtIndex(arrayKey, index)).to.equal(
            concat([arrayKey.substring(0, 34), toBeHex(index, 16)]),
        );
    });
});
```

</file>

<file>

path: /tests/LSP2/generateArrayKey.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/generateArrayKey.test.ts

```
import { expect } from 'chai';
import { keccak256, toUtf8Bytes } from 'ethers';
import { generateArrayKey } from '../../src';

describe('generateArrayKey', () => {
    it('should pass if &amp;nbsp;arrayKey&amp;nbsp; is a UTF8 string', () => {
        const arrayKeyName = 'SomeArray[]';

        expect(generateArrayKey(arrayKeyName)).to.equal(keccak256(toUtf8Bytes(arrayKeyName)));
    });

    it("should pass if &amp;nbsp;arrayKey&amp;nbsp; is '[]'", () => {
        const arrayKeyName = '[]';

        expect(generateArrayKey(arrayKeyName)).to.equal(keccak256(toUtf8Bytes(arrayKeyName)));
    });

    it('should revert if &amp;nbsp;arrayKey&amp;nbsp; is a UTF8 string with a single character', () => {
        const arrayKeyName = 'S';

        expect(() => generateArrayKey(arrayKeyName)).to.throw(
            'Array data key name must be longer than 2 characters.',
        );
    });

    it("should revert if &amp;nbsp;arrayKey&amp;nbsp; is a UTF8 string without '[]' at the end", () => {
        const arrayKeyName = 'SomeArray';

        expect(() => generateArrayKey(arrayKeyName)).to.throw(
            "Missing empty square brackets '[]' at the end of the data key name.",
        );
    });
});
```

</file>

<file>

path: /tests/LSP2/generateMappingKey.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/generateMappingKey.test.ts

```
import { expect } from 'chai';
import { generateMappingKey } from '../../src';
import { concat, keccak256, toUtf8Bytes } from 'ethers';

describe('generateMappingKey', () => {
  it('should pass and generate a mapping key.  firstPart  as UTF8 &  lastPart  as UTF8', () => {
    const firstPart = 'firstPart';
    const lastPart = 'lastPart';

    expect(generateMappingKey(firstPart, lastPart)).to.equal(
      concat([
        keccak256(toUtf8Bytes(firstPart)).substring(0, 22),
        '0x0000',
        keccak256(toUtf8Bytes(lastPart)).substring(0, 42),
      ]),
    );
  });

  it('should pass and generate a mapping key.  firstPart  as hex &  lastPart  as UTF8', () => {
    const firstPart = keccak256(toUtf8Bytes('firstPart')).substring(0, 22);
    const lastPart = 'lastPart';

    expect(generateMappingKey(firstPart, lastPart)).to.equal(
      concat([firstPart, '0x0000', keccak256(toUtf8Bytes(lastPart)).substring(0, 42)]),
    );
  });

  it('should pass and generate a mapping key.  firstPart  as UTF8 &  lastPart  as hex', () => {
    const firstPart = 'firstPart';
    const lastPart = keccak256(toUtf8Bytes('lastPart')).substring(0, 42);

    expect(generateMappingKey(firstPart, lastPart)).to.equal(
      concat([keccak256(toUtf8Bytes(firstPart)).substring(0, 22), '0x0000', lastPart]),
    );
  });

  it('should pass and generate a mapping key.  firstPart  as hex &  lastPart  as hex', () => {
    const firstPart = keccak256(toUtf8Bytes('firstPart')).substring(0, 22);
    const lastPart = keccak256(toUtf8Bytes('lastPart')).substring(0, 42);

    expect(generateMappingKey(firstPart, lastPart)).to.equal(
      concat([firstPart, '0x0000', lastPart]),
    );
  });
});
```

</file>

<file>

path: /tests/LSP2/generateMappingWithGroupingKey.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/generateMappingWithGroupingKey.test.ts

```
import { expect } from 'chai';
import { generateMappingWithGroupingKey } from '../../src';
import { concat, keccak256, toUtf8Bytes } from 'ethers';

describe('generateMappingWithGroupingKey', () => {
  it('should pass and generate a mapping with grouping key.  firstPart  as UTF8 &
 middlePart  as UTF8 &  lastPart  as UTF8', () => {
    const firstPart = 'firstPart';
    const middlePart = 'middlePart';
    const lastPart = 'lastPart';

    expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
      concat([
        keccak256(toUtf8Bytes(firstPart)).substring(0, 14),
        keccak256(toUtf8Bytes(middlePart)).substring(0, 10),
        '0x0000',
        keccak256(toUtf8Bytes(lastPart)).substring(0, 42),
      ]),
    );
  });

  it('should pass and generate a mapping with grouping key.  firstPart  as UTF8 &
 middlePart  as UTF8 &  lastPart  as hex', () => {
    const firstPart = 'firstPart';
    const middlePart = 'middlePart';
    const lastPart = keccak256(toUtf8Bytes('lastPart')).substring(0, 42);

    expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
      concat([
        keccak256(toUtf8Bytes(firstPart)).substring(0, 14),
        keccak256(toUtf8Bytes(middlePart)).substring(0, 10),
        '0x0000',
        lastPart,
      ]),
    );
  });

  it('should pass and generate a mapping with grouping key.  firstPart  as UTF8 &
 middlePart  as hex &  lastPart  as UTF8', () => {
    const firstPart = 'firstPart';
    const middlePart = keccak256(toUtf8Bytes('middlePart')).substring(0, 10);
    const lastPart = 'lastPart';

    expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
```

```
        concat([
          keccak256(toUtf8Bytes(firstPart)).substring(0, 14),
          middlePart,
          '0x0000',
          keccak256(toUtf8Bytes(lastPart)).substring(0, 42),
        ]),
      );
    });

    it('should pass and generate a mapping with grouping key.  firstPart  as UTF8 &
 middlePart  as hex &  lastPart  as UTF8', () => {
      const firstPart = 'firstPart';
      const middlePart = keccak256(toUtf8Bytes('middlePart')).substring(0, 10);
      const lastPart = keccak256(toUtf8Bytes('lastPart')).substring(0, 42);

      expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
        concat([
          keccak256(toUtf8Bytes(firstPart)).substring(0, 14),
          middlePart,
          '0x0000',
          lastPart,
        ]),
      );
    });

    it('should pass and generate a mapping with grouping key.  firstPart  as hex &
 middlePart  as UTF8 &  lastPart  as UTF8', () => {
      const firstPart = keccak256(toUtf8Bytes('firstPart')).substring(0, 14);
      const middlePart = 'middlePart';
      const lastPart = 'lastPart';

      expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
        concat([
          firstPart,
          keccak256(toUtf8Bytes(middlePart)).substring(0, 10),
          '0x0000',
          keccak256(toUtf8Bytes(lastPart)).substring(0, 42),
        ]),
      );
    });

    it('should pass and generate a mapping with grouping key.  firstPart  as hex &
 middlePart  as UTF8 &  lastPart  as hex', () => {
      const firstPart = keccak256(toUtf8Bytes('firstPart')).substring(0, 14);
      const middlePart = 'middlePart';
      const lastPart = keccak256(toUtf8Bytes('lastPart')).substring(0, 42);

      expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
        concat([
          firstPart,
          keccak256(toUtf8Bytes(middlePart)).substring(0, 10),
          '0x0000',
```

```
            lastPart,
          ]),
      );
    });

    it('should pass and generate a mapping with grouping key.  firstPart  as hex &
 middlePart  as hex &  lastPart  as UTF8', () => {
      const firstPart = keccak256(toUtf8Bytes('firstPart')).substring(0, 14);
      const middlePart = keccak256(toUtf8Bytes('middlePart')).substring(0, 10);
      const lastPart = 'lastPart';

      expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
        concat([
          firstPart,
          middlePart,
          '0x0000',
          keccak256(toUtf8Bytes(lastPart)).substring(0, 42),
        ]),
      );
    });

    it('should pass and generate a mapping with grouping key.  firstPart  as hex &
 middlePart  as hex &  lastPart  as hex', () => {
      const firstPart = keccak256(toUtf8Bytes('firstPart')).substring(0, 14);
      const middlePart = keccak256(toUtf8Bytes('middlePart')).substring(0, 10);
      const lastPart = keccak256(toUtf8Bytes('lastPart')).substring(0, 42);

      expect(generateMappingWithGroupingKey(firstPart, middlePart, lastPart)).to.equal(
        concat([firstPart, middlePart, '0x0000', lastPart]),
      );
    });
});
```

</file>

<file>

path: /tests/LSP2/generateSingletonKey.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/generateSingletonKey.test.ts

```
import { expect } from 'chai';
import { generateSingletonKey } from '../../src';
import { keccak256, toUtf8Bytes } from 'ethers';

describe('generateSingletonKey', () => {
    it('should pass and return the hash', () => {
        const keyName = 'SignletonKey';

        expect(generateSingletonKey(keyName)).to.equal(keccak256(toUtf8Bytes(keyName)));
    });
});
```

</file>

<file>

path: /tests/LSP2/isCompactBytesArray.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/isCompactBytesArray.test.ts

```
import { expect } from 'chai';
import { isCompactBytesArray } from '../../src';

describe('isCompactBytesArray', () => {
    it('should throw if passed value is not hex', () => {
        const compactBytesArray = 'Some string';

        expect(() => isCompactBytesArray(compactBytesArray)).to.throw(
             'compactBytesArray' is not hex. Value: '${compactBytesArray}' ,
        );
    });

    it('should pass and return true if valid', () => {
        const compactBytesArray = '0x0002cafe000abeefdeadbeef0000cafe';

        expect(isCompactBytesArray(compactBytesArray)).to.be.true;
    });

    it('should pass and return false if invalid', () => {
        const compactBytesArray = '0x0002cafecafe000abeefdeadbeef0000cafe';

        expect(isCompactBytesArray(compactBytesArray)).to.be.false;
    });

    it('should pass and return false if invalid', () => {
        const compactBytesArray = '0x0002';

        expect(isCompactBytesArray(compactBytesArray)).to.be.false;
    });
});
```

</file>

<file>

path: /tests/LSP2/isValidArrayLengthValue.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/isValidArrayLengthValue.test.ts

```
import { expect } from 'chai';
import { isValidArrayLengthValue } from '../../src';

describe('isValidArrayLengthValue', () => {
    it('should pass and return true if value is exactly 16 bytes', () => {
        const arrayLength = '0x0000000000000000000000000000a3b';

        expect(isValidArrayLengthValue(arrayLength)).to.be.true;
    });

    it('should pass and return true if value is shorter than 16 bytes', () => {
        const arrayLength = '0x00000000000000000000000004a';

        expect(isValidArrayLengthValue(arrayLength)).to.be.false;
    });

    it('should pass and return true if value is longer than 16 bytes', () => {
        const arrayLength = '0x0000000000000000000000000000000f60a';

        expect(isValidArrayLengthValue(arrayLength)).to.be.false;
    });
});
```

</file>

<file>

path: /tests/LSP2/removeElementFromArrayAndMap.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/removeElementFromArrayAndMap.test.ts

```
describe('removeElementFromArrayAndMap', () => {
    it('WIP -- add tests', () => {});
});
```

</file>

<file>

path: /tests/LSP2/removeLastElementFromArrayAndMap.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP2/removeLastElementFromArrayAndMap.test.ts

```
describe('removeLastElementFromArrayAndMap', () => {
    it('WIP -- add tests', () => {});
});
```

</file>

<file>

path: /tests/LSP3/getProfileMetadata.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP3/getProfileMetadata.test.ts

```
describe('getProfileMetadata', () => {
    it('WIP -- add tests', () => {});
});
```

</file>

<file>

path: /tests/LSP3/isProfileMetadata.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP3/isProfileMetadata.test.ts

```
import { expect } from 'chai';
import { isProfileMetadata } from '../../src';

describe('isProfileMetadata', () => {
    it('should pass and return true if an object of type  LSP3ProfileMetadata  is passed', () => {
        const object = { LSP3Profile: { description: '', links: [], name: '', tags: [] } };

        expect(isProfileMetadata(object)).to.be.true;
    });

    it('should pass and return false if an object of type other than  LSP3ProfileMetadata  is passed', () => {
        const object = { description: '', links: [], name: '', tags: [] };

        expect(isProfileMetadata(object)).to.be.false;
    });
});
```

</file>

<file>

path: /tests/LSP5/generateReceivedAssetKeys.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP5/generateReceivedAssetKeys.test.ts

```
describe('generateReceivedAssetKeys', () => {
    it('WIP -- add tests', () => {});
});
```

</file>

<file>

path: /tests/LSP5/generateSentAssetKeys.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP5/generateSentAssetKeys.test.ts

```
describe('generateSentAssetKeys', () => {
    it('WIP -- add tests', () => {});
});
```

</file>

<file>

path: /tests/LSP6/createValidityTimestamp.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP6/createValidityTimestamp.test.ts

```
import { expect } from 'chai';
import { createValidityTimestamp } from '../../src';
import { concat, toBeHex } from 'ethers';

describe('createValidityTimestamp', () => {
  it('should pass if the passed timestamps are valid', () => {
    const startingTimestamp = Math.round(new Date().getTime() / 1000);
    const endingTimestamp = startingTimestamp + 60;

    expect(createValidityTimestamp(startingTimestamp, endingTimestamp)).to.equal(
      concat([toBeHex(startingTimestamp, 16), toBeHex(endingTimestamp, 16)]),
    );
  });

  it('should throw if the bytes value of  startingTimestamp  is bigger than 16 bytes', () => {
    const startingTimestamp = '0xffffffffffffffffffffffffffffffff';
    const endingTimestamp = Math.round(new Date().getTime() / 1000);

    expect(() => createValidityTimestamp(startingTimestamp, endingTimestamp)).to.throw(
       The hex value of the number: '${startingTimestamp}' exeeceds 16 bytes ,
    );
  });

  it('should throw if the bytes value of  endingTimestamp  is bigger than 16 bytes', () => {
    const startingTimestamp = Math.round(new Date().getTime() / 1000);
    const endingTimestamp = '0xffffffffffffffffffffffffffffffff';

    expect(() => createValidityTimestamp(startingTimestamp, endingTimestamp)).to.throw(
       The hex value of the number: '${endingTimestamp}' exeeceds 16 bytes ,
    );
  });
});
```

</file>

<file>

path: /tests/LSP6/decodeAllowedCalls.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP6/decodeAllowedCalls.test.ts

```
import { expect } from 'chai';
import { decodeAllowedCalls } from '../../src';

describe('decodeAllowedCalls', () => {
  it('should throw if passed  allowedCalls  is not hex', () => {
    const allowedCalls = 'allowedCalls';

    expect(() => decodeAllowedCalls(allowedCalls)).to.throw( '${allowedCalls}' is not hex );
  });
```

```
  it('should throw if a length in the passed  allowedCalls  is 0', () => {
    const allowedCalls =
      '0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c0000';

    expect(() => decodeAllowedCalls(allowedCalls)).to.throw(
       Invalid length. Length: '${0}'. Must be 32. ,
    );
  });

  it('should throw if a length in the passed  allowedCalls  is smaller than 32', () => {
    const allowedCalls =

'0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c001000000003cafecafecafecafecafecafe';

    expect(() => decodeAllowedCalls(allowedCalls)).to.throw(
       Invalid length. Length: '${16}'. Must be 32. ,
    );
  });

  it('should throw if a length in the passed  allowedCalls  is bigger than 32', () => {
    const allowedCalls =

'0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c004000000003cafecafecafecafecafecafecafecafecafecafe24871b3d44c028fe00000003cafecafecafecafecafecafecafecafecafe24871b3d44c028fe';

    expect(() => decodeAllowedCalls(allowedCalls)).to.throw(
       Invalid length. Length: '${64}'. Must be 32. ,
    );
  });

  it('should throw if a length in the passed  allowedCalls  is bigger than its element', () => {
    const allowedCalls =

'0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c0020cafecafecafecafecafecafecafecafecafecafe';

    expect(() => decodeAllowedCalls(allowedCalls)).to.throw(
      'Out of bounds, length of an element reaches past the length of  allowedCalls ',
    );
  });

  it('should pass if  allowedCalls  is valid', () => {
    const allowedCalls =

'0x002000000002cafecafecafecafecafecafecafecafecafecafe24871b3d7f23690c002000000003cafecafecafecafecafecafecafecafecafecafe24871b3d44c028fe';

    const decodedAllowedCalls = {
      allowedInteractions: ['0x00000002', '0x00000003'],
      allowedAddresses: [
```

```
            '0xcafecafecafecafecafecafecafecafecafecafe',
            '0xcafecafecafecafecafecafecafecafecafecafe',
        ],
        allowedStandards: ['0x24871b3d', '0x24871b3d'],
        allowedFunctions: ['0x7f23690c', '0x44c028fe'],
    };

    expect(decodeAllowedCalls(allowedCalls)).to.deep.equal(decodedAllowedCalls);
  });
});
```

</file>

<file>

path: /tests/LSP6/decodeAllowedERC725YDataKeys.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP6/decodeAllowedERC725YDataKeys.test.ts

```
import { expect } from 'chai';
import { decodeAllowedERC725YDataKeys } from '../../src';

describe('decodeAllowedERC725YDataKeys', () => {
  it('should throw if passed  allowedERC725YDataKeys  is not hex', () => {
    const allowedERC725YDataKeys = 'allowedERC725YDataKeys';

    expect(() => decodeAllowedERC725YDataKeys(allowedERC725YDataKeys)).to.throw(
       '${allowedERC725YDataKeys}' is not hex ,
    );
  });

  it('should throw if a length in the passed  allowedERC725YDataKeys  is 0', () => {
    const allowedERC725YDataKeys = '0x0004deadbeef0000cafe';

    expect(() => decodeAllowedERC725YDataKeys(allowedERC725YDataKeys)).to.throw(
       Invalid length. Length: '${0}'. Must be bigger than 0 and smaller than 32. ,
    );
  });

  it('should throw if a length in the passed  allowedERC725YDataKeys  is bigger than 32', () =>
{
    const allowedERC725YDataKeys =

'0x0004deadbeef0040cafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafe';

    expect(() => decodeAllowedERC725YDataKeys(allowedERC725YDataKeys)).to.throw(
       Invalid length. Length: '${64}'. Must be bigger than 0 and smaller than 32. ,
    );
  });
```

```
  it('should throw if a length in the passed &amp;nbsp;allowedERC725YDataKeys&amp;nbsp; is bigger than its element', () => {
    const allowedERC725YDataKeys = '0x0004deadbeef0020cafecafecafecafe';

    expect(() => decodeAllowedERC725YDataKeys(allowedERC725YDataKeys)).to.throw(
      'Out of bounds, length of an element reaches past the length of &amp;nbsp;allowedERC725YDataKeys&amp;nbsp;',
    );
  });

  it('should pass if &amp;nbsp;allowedERC725YDataKeys&amp;nbsp; is valid', () => {
    const allowedERC725YDataKeys =

'0x0004deadbeef0020cafecafecafecafecafe00cafecafecafecafe00cafecafecafecafecafecafe0001aa0014beefbeefbeefbeefbeefbeefbeefbeefbeefbeef';

    const decodedDataKeys = [
      '0xdeadbeef',
      '0xcafecafecafecafecafe00cafecafecafecafe00cafecafecafecafecafecafe',
      '0xaa',
      '0xbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef',
    ];


expect(decodeAllowedERC725YDataKeys(allowedERC725YDataKeys)).to.deep.equal(decodedDataKeys);
  });
});
```

</file>

<file>

path: /tests/LSP6/encodeAllowedCalls.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP6/encodeAllowedCalls.test.ts

```
import { expect } from 'chai';
import { encodeAllowedCalls } from '../../src';

describe('encodeAllowedCalls', () => {
  describe('testing &amp;nbsp;allowedInteractions&amp;nbsp;', () => {
    it('should throw if any element of &amp;nbsp;allowedInteractions&amp;nbsp; is not hex', () => {
      const allowedInteractions = ['allowedInteractions'];
      const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
      const allowedStandards = ['0x24871b3d'];
      const allowedFunctions = ['0x7f23690c'];

      expect(() =>
        encodeAllowedCalls(
          allowedInteractions,
          allowedAddresses,
```

```
                allowedStandards,
                allowedFunctions,
            ),
        ).to.throw(
             Allowed interaction is not hex. Allowed interaction: '${allowedInteractions[0]}' ,
        );
    });

    it('should throw if any element of  allowedInteractions  has bytes lenght bigger than 4', ()
=> {
        const allowedInteractions = ['0x0000000200'];
        const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
        const allowedStandards = ['0x24871b3d'];
        const allowedFunctions = ['0x7f23690c'];

        expect(() =>
            encodeAllowedCalls(
                allowedInteractions,
                allowedAddresses,
                allowedStandards,
                allowedFunctions,
            ),
        ).to.throw(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
                allowedInteractions[0].length / 2 - 1
            } ,
        );
    });

    it('should throw if any element of  allowedInteractions  has bytes lenght smaller than 4', ()
=> {
        const allowedInteractions = ['0x000002'];
        const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
        const allowedStandards = ['0x24871b3d'];
        const allowedFunctions = ['0x7f23690c'];

        expect(() =>
            encodeAllowedCalls(
                allowedInteractions,
                allowedAddresses,
                allowedStandards,
                allowedFunctions,
            ),
        ).to.throw(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
                allowedInteractions[0].length / 2 - 1
            } ,
        );
    });
});

describe('testing  allowedAddresses ', () => {
```

```
    it('should throw if any element of  allowedAddresses  is not hex', () => {
      const allowedInteractions = ['0x00000002'];
      const allowedAddresses = ['allowedAddresses'];
      const allowedStandards = ['0x24871b3d'];
      const allowedFunctions = ['0x7f23690c'];

      expect(() =>
        encodeAllowedCalls(
          allowedInteractions,
          allowedAddresses,
          allowedStandards,
          allowedFunctions,
        ),
      ).to.throw( Allowed address is not hex. Allowed address: '${allowedAddresses[0]}' );
    });

    it('should throw if any element of  allowedAddresses  has bytes lenght bigger than 20', ()
=> {
      const allowedInteractions = ['0x00000002'];
      const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe0000'];
      const allowedStandards = ['0x24871b3d'];
      const allowedFunctions = ['0x7f23690c'];

      expect(() =>
        encodeAllowedCalls(
          allowedInteractions,
          allowedAddresses,
          allowedStandards,
          allowedFunctions,
        ),
      ).to.throw(
         Allowed interactions has invalid bytes length. Must be 20. Length: ${
          allowedAddresses[0].length / 2 - 1
        } ,
      );
    });

    it('should throw if any element of  allowedAddresses  has bytes lenght smaller than 20', ()
=> {
      const allowedInteractions = ['0x00000002'];
      const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
      const allowedStandards = ['0x24871b3d'];
      const allowedFunctions = ['0x7f23690c'];

      expect(() =>
        encodeAllowedCalls(
          allowedInteractions,
          allowedAddresses,
          allowedStandards,
          allowedFunctions,
        ),
      ).to.throw(
```

```
             Allowed interactions has invalid bytes length. Must be 20. Length: ${
              allowedAddresses[0].length / 2 - 1
            } ,
          );
        });
      });

      describe('testing  allowedStandards ', () => {
        it('should throw if any element of  allowedStandards  is not hex', () => {
          const allowedInteractions = ['0x00000002'];
          const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
          const allowedStandards = ['allowedStandards'];
          const allowedFunctions = ['0x7f23690c'];

          expect(() =>
            encodeAllowedCalls(
              allowedInteractions,
              allowedAddresses,
              allowedStandards,
              allowedFunctions,
            ),
          ).to.throw( Allowed standard is not hex. Allowed standard: '${allowedStandards[0]}' );
        });

        it('should throw if any element of  allowedStandards  has bytes lenght bigger than 4', () =>
{
          const allowedInteractions = ['0x00000002'];
          const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
          const allowedStandards = ['0x24871b3d00'];
          const allowedFunctions = ['0x7f23690c'];

          expect(() =>
            encodeAllowedCalls(
              allowedInteractions,
              allowedAddresses,
              allowedStandards,
              allowedFunctions,
            ),
          ).to.throw(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
              allowedStandards[0].length / 2 - 1
            } ,
          );
        });

        it('should throw if any element of  allowedStandards  has bytes lenght smaller than 4', () =>
{
          const allowedInteractions = ['0x00000002'];
          const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
          const allowedStandards = ['0x24871b'];
          const allowedFunctions = ['0x7f23690c'];
```

```
        expect(() =>
            encodeAllowedCalls(
                allowedInteractions,
                allowedAddresses,
                allowedStandards,
                allowedFunctions,
            ),
        ).to.throw(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
                allowedStandards[0].length / 2 - 1
            } ,
        );
    });
});

describe('testing  allowedFunctions ', () => {
    it('should throw if any element of  allowedFunctions  is not hex', () => {
        const allowedInteractions = ['0x00000002'];
        const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
        const allowedStandards = ['0x24871b3d'];
        const allowedFunctions = ['allowedFunctions'];

        expect(() =>
            encodeAllowedCalls(
                allowedInteractions,
                allowedAddresses,
                allowedStandards,
                allowedFunctions,
            ),
        ).to.throw( Allowed function is not hex. Allowed function: '${allowedFunctions[0]}' );
    });

    it('should throw if any element of  allowedFunctions  has bytes lenght bigger than 4', () => {
        const allowedInteractions = ['0x00000002'];
        const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
        const allowedStandards = ['0x24871b3d'];
        const allowedFunctions = ['0x7f23690c00'];

        expect(() =>
            encodeAllowedCalls(
                allowedInteractions,
                allowedAddresses,
                allowedStandards,
                allowedFunctions,
            ),
        ).to.throw(
             Allowed interactions has invalid bytes length. Must be 4. Length: ${
                allowedFunctions[0].length / 2 - 1
            } ,
        );
    });
```

```
    it('should throw if any element of  allowedFunctions  has bytes lenght smaller than 4', () =>
{
      const allowedInteractions = ['0x00000002'];
      const allowedAddresses = ['0xcafecafecafecafecafecafecafecafecafecafe'];
      const allowedStandards = ['0x24871b3d'];
      const allowedFunctions = ['0x7f2369'];

      expect(() =>
        encodeAllowedCalls(
          allowedInteractions,
          allowedAddresses,
          allowedStandards,
          allowedFunctions,
        ),
      ).to.throw(
         Allowed interactions has invalid bytes length. Must be 4. Length: ${
          allowedFunctions[0].length / 2 - 1
        } ,
      );
    });
  });

  it('should pass if  allowedInteractions ,  allowedAddresses ,
 allowedStandards  &  allowedFunctions  are valid', () => {
    const allowedInteractions = ['0x00000002', '0x00000003'];
    const allowedAddresses = [
      '0xcafecafecafecafecafecafecafecafecafecafe',
      '0xcafecafecafecafecafecafecafecafecafecafe',
    ];
    const allowedStandards = ['0x24871b3d', '0x24871b3d'];
    const allowedFunctions = ['0x7f23690c', '0x44c028fe'];

    const expectedAllowedCalls =

'0x002000000002cafecafecafecafecafecafecafecafecafe24871b3d7f23690c002000000003cafecafecafe
cafecafecafecafecafecafecafe24871b3d44c028fe';

    expect(
      encodeAllowedCalls(
        allowedInteractions,
        allowedAddresses,
        allowedStandards,
        allowedFunctions,
      ),
    ).to.equal(expectedAllowedCalls);
  });
});
```

</file>

<file>

path: /tests/LSP6/encodeAllowedERC725YDataKeys.test.ts

url: https://github.com/lukso-network/lsp-utils/blob/main/tests/LSP6/encodeAllowedERC725YDataKeys.test.ts

```
import { expect } from 'chai';
import { encodeAllowedERC725YDataKeys } from '../../src';

describe('encodeAllowedERC725YDataKeys.test', () => {
  it('should throw if a passed data key is UTF8', () => {
    const dataKeys = ['data key'];

    expect(() => encodeAllowedERC725YDataKeys(dataKeys)).to.throw(
      &amp;nbsp;'${dataKeys[0]}' is not hex&amp;nbsp;,
    );
  });

  it('should throw if a passed data key has 0 bytes', () => {
    const dataKeys = ['0x'];

    expect(() => encodeAllowedERC725YDataKeys(dataKeys)).to.throw(
      &amp;nbsp;Invalid length. Length: '${
        dataKeys[0].length - 2
      }'. Must be bigger than 0 and smaller than 32.&amp;nbsp;,
    );
  });

  it('should throw if a passed data key has more than 32 bytes', () => {
    const dataKeys = [
      '0xcafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafecafe',
    ];

    expect(() => encodeAllowedERC725YDataKeys(dataKeys)).to.throw(
      &amp;nbsp;Invalid length. Length: '${
        dataKeys[0].length / 2 - 1
      }'. Must be bigger than 0 and smaller than 32.&amp;nbsp;,
    );
  });
});
```

</file>

<file>

path: /tsconfig.es5.json

url: https://github.com/lukso-network/lsp-utils/blob/main/tsconfig.es5.json

```
{
   "compilerOptions": {
      "module": "commonjs",
      "target": "ES5",
      "lib": ["ES2020", "DOM"],
      "declaration": true,
      "outDir": "./dist/lib/es5",
      "moduleResolution": "NodeNext",
      "skipLibCheck": true
   },
   "include": ["src/**/*"],
   "exclude": ["node_modules", "types/**/*"]
}
```

</file>

<file>

path: /tsconfig.json

url: https://github.com/lukso-network/lsp-utils/blob/main/tsconfig.json

```
{
   "compilerOptions": {
      "module": "NodeNext",
      "target": "ES5",
      "lib": ["ES2020", "DOM"],
      "declaration": true,
      "outDir": "./dist/lib/es6",
      "moduleResolution": "NodeNext",
      "skipLibCheck": true
   },
   "include": ["src/**/*"],
   "exclude": ["node_modules", "types/**/*"]
}
```

</file>

</repository>