🤖

# 0.8.0 Release - To - Services

## Constants.js

- Replace most of the string errors with Custom errors. The selector for the custom errors is present in `constants.js` .
- `ALL_PERMISSIONS` was all permissions except `DELEGATECALL` and `SUPER_DELEGATECALL`, now we add `REENTRANCY` also as an exception.
- `ERC725YKeys` const function was renamed to `ERC725YDataKeys`,
  Example: [https://github.com/lukso-network/universalprofile-extension/blob/94d569d674eb6a021643ed00559659b6da24d2d4/src/popup/utils/ethUrlUtils.ts#L2](https://github.com/lukso-network/universalprofile-extension/blob/94d569d674eb6a021643ed00559659b6da24d2d4/src/popup/utils/ethUrlUtils.ts#L2)

## LSP0

- InterfaceId was: `0xeb6be62e` changed to : `0x66767497`

- Change in the `UniversalReceiver` event.
  Switching the `returnedValue` with `receivedData`, event signature didn't change.

Before:

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

After:

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

- Instead of just having the `UniversalReceiverDelegate` reacting on `universalReceiver(typeId,data)` function calls, `MappedUniversalReceiverDelegate` is introduced as a contract that can be set to react on certain typeIds.

  Main differences:
  - `UniversalReceiverDelegate` react on the whole call regardless of the typeId
  - `MappedUniversalReceiverDelegate` react on calls with specific typeIds, if set by the owner.

The `MappedUniversalReceiverDelegate` has a specific data key:

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

- The return value of the `universalReceiver` function was the return value of the UniversalReceiverDelegate. `0x` in case it didn't exist.

  New behavior: The return value of the `universalReceiver` function is both return values of `UniversalReceiverDelegate` and `MappedUniversalReceiverDelegate` encoded as bytes.
- Before: `ValueReceived` was emitted whenever the contract receive ether.

  After: `ValueReceived` event is also emitted in the payable function when the call is associated with value (ether/LYX) such as `execute` function, `execute` Batch, `universalReceiver` function.
- Change the ownership management from `ClaimOwnership` to `LSP14Ownable2Step`:
  - Changing function name from `claimOwnership` to `acceptOwnership`.

  - Change the event name from `RenounceOwnershipInitiated` to `RenounceOwnershipStarted`.

  - `renounceOwnership` function is now 2 step process where once it's called, the second call should happen after a delay of 100 block. If the second call was not made in another 100 block after the delay, the process is reset.

  - Introduce `OwnershipTransferStarted` and `OwnershipRenounced` events.
- Added `execute` batch function.
- Function overloading for `execute` batch, in `web3.js`/ether.js overloaded functions should be written:

```
// web3.js example

// execute
myContract.methods['execute(uint256,address,uint256,bytes)'](OPERATION_CALL, target.address, 2WEI, "0x").send();
// execute Array
myContract.methods['execute(uint256[],address[],uint256[],bytes[])']([OPERATION_CALL, OPERATION_CREATE], [target.address, ZERO_ADDRESS], [2W

// OR

// execute
myContract.methods['0x44c028fe'](OPERATION_CALL, target.address, 2WEI, "0x").send();
// execute Array
myContract.methods['0x13ced88d']([OPERATION_CALL, OPERATION_CREATE], [target.address, ZERO_ADDRESS], [2WEI, 0WEI], ["0x", CONTRACT_BYTECODE
```

- LSP0 supports the following interfaceIds:
  - `ERC725X`: before: `0x44c028fe`  after: `0x570ef073`
  - `ERC725Y`: The same  `0x714df77c`
  - `ERC1271`: The same `0x1626ba7e`
  - `LSP1`: The same `0x6bb56a14`
  - `LSP14`: before: `0xa375e9c6` after: `0x94be5999`
  - `LSP17Extendable`: New: `0xa918fa6b`

- The event `ContractCreated` changed from: (Added bytes32 salt)
  `event ContractCreated(uint256 indexed operationType, address indexed contractAddress, uint256 indexed value)`

  to:

  `event ContractCreated(uint256 indexed operationType, address indexed contractAddress, uint256 indexed value, bytes32 salt)`

  In case of `CREATE2` operation the salt will be the provided salt by the user.
  In case of `CREATE` operation the salt will be `bytes32(0)`.

- Sending random data to an LSP0 before was permissible, the call would pass.

  New behavior: If you send data to the LSP0:
  - if `data.length` < 4, the call will pass and return.

  - if `data.length` =< 4, and the first 4 bytes are 0s, the call will pass and return.

  - if `data.length` =< 4, and the first 4 bytes are not 0, (look like a normal function selector):
    - The function selector will be checked against an extension.
    - If the extension exist for the function selector (first 4 bytes), then LSP0 call the extension and return its return value.
    - If the extension don’t exist for the function selector, then the call will revert.

## LSP1

- Switch the name of `universalReceiverDelegate(..)` function to `universalReceiver(..)`
- Removed `LSP1UniversalReceiverDelegate` interfaceId

## LSP2

- Added the compact bytes array type, more information here: [https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md#bytescompactbytesarray](https://github.com/lukso-network/LIPs/blob/main/LSPs/LSP-2-ERC725YJSONSchema.md#bytescompactbytesarray)

## LSP6

- InterfaceId was: `0xc403d48f` changed to : `0xfb437414`

- The event `Executed` changed

  from
  `Executed(uint256 indexed value, bytes4 selector)`

  to
  `Executed(bytes4 indexed selector, uint256 indexed value)`

- Signed message in executeRelayCall has a new format:

  Before, the signed message was formed of: `uint256 chainId`, `address KeyManager`, `uint256 nonce` and `bytes payloadToExecute` was hashed according to solidity keccak256.

Then signed with the normal signature process of `keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n"` provided by `ethers.js` with `signer.signMessage` / `web3.js` with `web3.eth.personal.sign`.

```
let hash = ethers.utils.solidityKeccak256(
["uint256", "address", "uint256", "bytes"],
[
HARDHAT_CHAINID,
context.keyManager.address,
nonceBefore,
executeRelayCallPayload,
]
);

let signature = await signer.signMessage(ethers.utils.arrayify(hash));

await context.keyManager.connect(relayer)
      .executeRelayCall(signature, nonceBefore, executeRelayCallPayload);
```

New behavior:

The data is formed of:
`0x19 <0x00> <KeyManager address> <LSP6_VERSION> <chainId> <nonce> <value> <payload>`

with:

`0x19`: byte intended to ensure that the `signed_data` is not valid RLP.

`0x00`: version 0 of the EIP191.

`KeyManager address`: The address of the Key Manager executing the payload.

`LSP6_VERSION`: Version relative to the LSP6KeyManager defined as a uint256 equal to `6`.

`chainId`: The chainId of the blockchain where the Key Manager is deployed, as a uint256.

`nonce`: The nonce to sign the payload with, as a uint256.

`value`: The amount of native token to transfer to the linked target contract alongside the call.

`payload`: The payload to be executed.

`LUKSO` developed a signer tool for this method:
https://github.com/lukso-network/tools-eip191-signer

For the KeyManager purpose you can use as follow:

```
eip191Signer.signDataWithIntendedValidator(
  validatorAddress,  // We Put address of LSP6KeyManager here:
  message, // Packed encoded of: <LSP6VERSION><chainId><nonce><value><payload>
  signingKey, // The private key of the signer
);
```

- Introduce `execute(uint256[], bytes[])` batch and `executeRelayCall(bytes[], uint256[], uint256[], bytes[])` batch.

  Since there is function overloading, using `web3.js` or `ether.js` there is a need to call these functions using function signatures, like the example above for `execute` on `LSP0`.

```
// Example

// execute single
```

```
await context.keyManager.connect(signer)
    ["execute(bytes)"](payload)
   );

// execute batch

const tx = await context.keyManager.connect(context.owner)
     ["execute(uint256[],bytes[])"]([0, 0, 0], batchExecutePayloads);



// executeRelayCall single
await context.keyManager
.connect(relayer)
["executeRelayCall(bytes,uint256,bytes)"](
signature,
nonceBefore,
executeRelayCallPayload
);

// executeRelayCall batch

await context.keyManager.connect(context.owner)
["executeRelayCall(bytes[],uint256[],uint256[],bytes[])"](
signatures,
nonces,
values,
payloads
)
```

- Change the Data key `AllowedERC725YKeys` to `AllowedERC725YDataKeys` which is `0x4b80742de2bf90b8b485` to `0x4b80742de2bf866c2911`.
- Change all references of plain “Keys” to “Data Keys” (The term keys should be replaced with data keys also on services side, to avoid confusion with keys as controllers)
- Allow `bytes32(0)` data key to be set a data value through the LSP6KeyManager
- New Order of the Permission:

  Before:

```
const PERMISSIONS = {
CHANGEOWNER:          "0x0000000000000000000000000000000000000000000000000000000000000001", // .... .... .... 0001
CHANGEPERMISSIONS:    "0x0000000000000000000000000000000000000000000000000000000000000002", // .... .... .... 0010
ADDPERMISSIONS:       "0x0000000000000000000000000000000000000000000000000000000000000004", // .... .... .... 0100
SETDATA:              "0x0000000000000000000000000000000000000000000000000000000000000008", // .... .... .... 1000
CALL:                 "0x0000000000000000000000000000000000000000000000000000000000000010", // .... .... 0001 ....
STATICCALL:           "0x0000000000000000000000000000000000000000000000000000000000000020", // .... .... 0010 ....
DELEGATECALL:         "0x0000000000000000000000000000000000000000000000000000000000000040", // .... .... 0100 ....
DEPLOY:               "0x0000000000000000000000000000000000000000000000000000000000000080", // .... .... 1000 ....
TRANSFERVALUE:        "0x0000000000000000000000000000000000000000000000000000000000000100", // .... 0001 .... ....
SIGN:                 "0x0000000000000000000000000000000000000000000000000000000000000200", // .... 0010 .... ....
ENCRYPT:              "0x0000000000000000000000000000000000000000000000000000000000000400", // .... 0100 .... ....
SUPER_SETDATA:        "0x0000000000000000000000000000000000000000000000000000000000000800", // .... 1000 .... ....
SUPER_TRANSFERVALUE:  "0x0000000000000000000000000000000000000000000000000000000000001000", // 0001 .... .... ....
SUPER_CALL:           "0x0000000000000000000000000000000000000000000000000000000000002000", // 0010 .... .... ....
SUPER_STATICCALL:     "0x0000000000000000000000000000000000000000000000000000000000004000", // 0100 .... .... ....
SUPER_DELEGATECALL:   "0x0000000000000000000000000000000000000000000000000000000000008000", // 1000 .... .... ....
}
```

After:

```
const PERMISSIONS = {
CHANGEOWNER                      :"0x0000000000000000000000000000000000000000000000000000000000000001",
ADDCONTROLLER                    :"0x0000000000000000000000000000000000000000000000000000000000000002",
CHANGEPERMISSIONS                :"0x0000000000000000000000000000000000000000000000000000000000000004",
ADDEXTENSIONS                    :"0x0000000000000000000000000000000000000000000000000000000000000008",
CHANGEEXTENSIONS                 :"0x0000000000000000000000000000000000000000000000000000000000000010",
ADDUNIVERSALRECEIVERDELEGATE     :"0x0000000000000000000000000000000000000000000000000000000000000020",
CHANGEUNIVERSALRECEIVERDELEGATE  :"0x0000000000000000000000000000000000000000000000000000000000000040",
REENTRANCY                       :"0x0000000000000000000000000000000000000000000000000000000000000080",
SUPER_TRANSFERVALUE              :"0x0000000000000000000000000000000000000000000000000000000000000100",
TRANSFERVALUE                    :"0x0000000000000000000000000000000000000000000000000000000000000200",
SUPER_CALL                       :"0x0000000000000000000000000000000000000000000000000000000000000400",
CALL                             :"0x0000000000000000000000000000000000000000000000000000000000000800",
SUPER_STATICCALL                 :"0x0000000000000000000000000000000000000000000000000000000000001000",
STATICCALL                       :"0x0000000000000000000000000000000000000000000000000000000000002000",
SUPER_DELEGATECALL               :"0x0000000000000000000000000000000000000000000000000000000000004000",
DELEGATECALL                     :"0x0000000000000000000000000000000000000000000000000000000000008000",
DEPLOY                           :"0x0000000000000000000000000000000000000000000000000000000000010000",
SUPER_SETDATA                    :"0x0000000000000000000000000000000000000000000000000000000000020000",
SETDATA                          :"0x0000000000000000000000000000000000000000000000000000000000040000",
ENCRYPT                          :"0x0000000000000000000000000000000000000000000000000000000000080000",
DECRYPT                          :"0x0000000000000000000000000000000000000000000000000000000000100000",
SIGN                             :"0x0000000000000000000000000000000000000000000000000000000000200000",
}
```

- Change Permission name from `ADDPERMISSIONS` to `ADDCONTROLLER` .
  Variable in `constant.js` has been renamed to `ADDCONTROLLER`.
- To set the `UniversalReceiverDelegate` data keys:

  - Before: Permission needed was `SETDATA`
  - After: Permission needed is `ADDUNIVERSALRECEIVERDELEGATE`

  To change the address of the UniversalReceiverDelegate or remove it, Permission needed is `CHANGEUNIVERSALRECEIVERDELEGATE`

- Our default `UniversalReceiverDelegate` needs `Reentrancy` Permission because it reenter the UniversalProfile and `setData` on it.

- Super Permissions are inlined after their relevant permission, not at the end.

- Changed the `AllowedStandard/AllowedAddresses/AllowedFunctions` to `AllowedCalls`

Before we had these three data keys:

```
{
"name": "AddressPermissions:AllowedAddresses:<address>",
"key": "0x4b80742de2bfc6dd6b3c0000<address>",
"keyType": "MappingWithGrouping",
"valueType": "address[]",
"valueContent": "Address"
},

{
"name": "AddressPermissions:AllowedFunctions:<address>",
"key": "0x4b80742de2bf8efea1e80000<address>",
"keyType": "MappingWithGrouping",
"valueType": "bytes4[]",
"valueContent": "Bytes4"
},

{
"name": "AddressPermissions:AllowedStandards:<address>",
"key": "0x4b80742de2bf3efa94a30000<address>",
"keyType": "MappingWithGrouping",
"valueType": "bytes4[]",
"valueContent": "Bytes4"
},
```

Now their functionalities is combined under this data Key:

```
{
"name": "AddressPermissions:AllowedCalls:<address>",
"key": "0x4b80742de2bf393a64c70000<address>",
"keyType": "MappingWithGrouping",
"valueType": "(bytes4,address,bytes4)[CompactBytesArray]",
"valueContent": "(Bytes4,Address,Bytes4)"
}
```

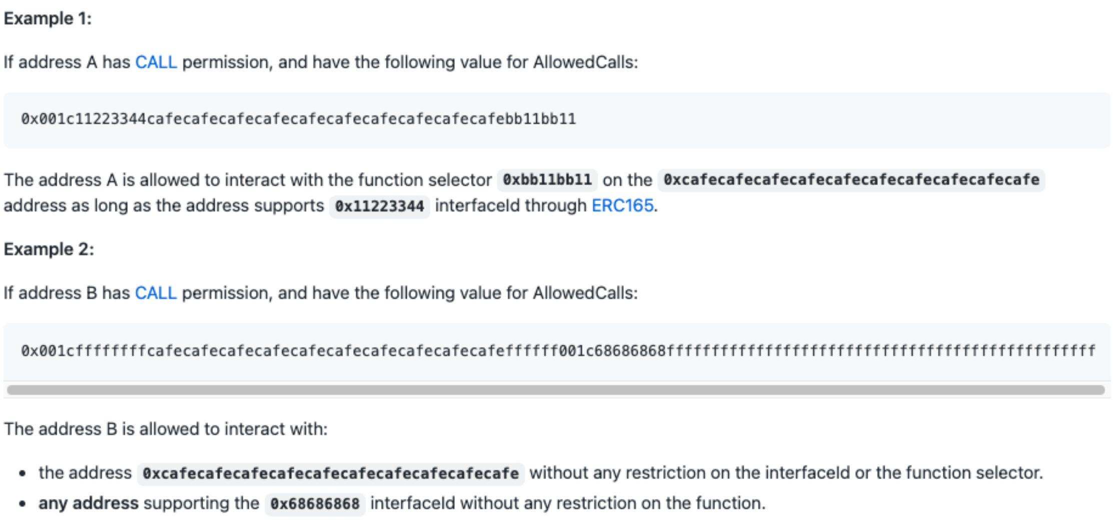
Each entry (allowed call) is made of three elements concatenated together as a tuple that forms a final `bytes28` long value.

The full list of allowed calls MUST be constructed as a CompactBytesArray according to LSP2-ERC725YJSONSchema as follow:

`<001c> <bytes4 allowedInterfaceId> <bytes20 allowedAddress> <bytes4 allowedFunction> <001c> ... <001c> ...`

> NB: the three dots ... are placeholders for <bytes4 allowedInterfaceId> <bytes20 allowedAddress> <bytes4 allowedFunction> and used for brievity.

- `001c`: **001c** in decimals is **28**, which is the sum of bytes length of the three elements below concatenated together.
- `allowedInterfaceId`: The ERC165 interface id being supported by the contract called from the target.
- `allowedAddress`: The address called by the target contract.
- `allowedFunction`: The function selector being called on the contract called by the target contract.



**Example 1:**

If address A has CALL permission, and have the following value for AllowedCalls:

```
0x001c11223344cafecafecafecafecafecafecafecafecafecafebb11bb11
```

The address A is allowed to interact with the function selector `0xbb11bb11` on the `0xcafecafecafecafecafecafecafecafecafecafe` address as long as the address supports `0x11223344` interfaceId through ERC165.

**Example 2:**

If address B has CALL permission, and have the following value for AllowedCalls:

```
0x001cffffffffcafecafecafecafecafecafecafecafecafecafeffffff001c68686868ffffffffffffffffffffffffffffffffffffffffffffffff
```

The address B is allowed to interact with:

- the address `0xcafecafecafecafecafecafecafecafecafecafe` without any restriction on the interfaceId or the function selector.
- **any address** supporting the `0x68686868` interfaceId without any restriction on the function.

These can be done with the help of `erc725.js` once it's released.

- Before: If there is no `AllowedCalls`, everything was whitelisted.

  After: If there is no `AllowedCalls`, no calls are allowed.
- Change verification logic for `AllowedERC725YDataKey.`

## Before:

```
{
"name": "AddressPermissions:AllowedERC725YKeys:<address>",
"key": "0x4b80742de2bf90b8b4850000<address>",
"keyType": "MappingWithGrouping",
"valueType": "bytes32[]",
"valueContent": "Bytes32"
}
```

Each data key defined in the array MUST be 32 bytes long. It is possible to set a range of allowed ERC725Y data keys (**= partial data keys**), by setting:

- some part of the data keys as the exact data key bytes
- the rest of the data key bytes as 0 bytes.

The 0 bytes part will represent a part that is dynamic. Below is an example based on a LSP2 Mapping key type, where first word = `SupportedStandards`, and second word = `LSP3UniversalProfile`.

`name: "SupportedStandards:LSP3UniversalProfile"`

`key: 0xeafec4d89fa9619884b60000abe425d64acd861a49b8ddf5c0b6962110481f38`

By setting the value to `0xeafec4d89fa9619884b600000000000000000000000000000000000000000000` in the list of allowed ERC725Y data keys, one address can set any data key **starting with the first word** `SupportedStandards:...`.

## After

```
{
"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:

`<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 key type, where first word = `SupportedStandards`, and second word = `LSP3UniversalProfile`.

`name: "SupportedStandards:LSP3UniversalProfile"`

`key: 0xeafec4d89fa9619884b60000abe425d64acd861a49b8ddf5c0b6962110481f38`

**Example 1:**

- If address A has SETDATA permission, and have the following value for `AllowedERC725YDataKeys`:

```
> 0x 0020 eafec4d89fa9619884b60000abe425d64acd861a49b8ddf5c0b6962110481f38
> 0x0020eafec4d89fa9619884b60000abe425d64acd861a49b8ddf5c0b6962110481f38
```

> 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 eafec4d89fa9619884b6 0020 beefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef`
`>0x000aeafec4d89fa9619884b60020beefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeefbeef`

> `000a` (10 in decimals) is the length of the `eafec4d89fa9619884b6` 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 `0xeafec4d89fa9619884b6`.

By setting the value to `0xeafec4d89fa9619884b6` in the list of allowed ERC725Y data keys, one address can set any data key **starting with the first word** **`SupportedStandards:...`**.

## LSP7

- InterfaceId was: `0x5fcaac27` changed to : `0xda1f85e4`
- Changing `bool` param to `bool[]` in token `transferBatch(..)`
- `LSP7AmountExceedsBalance(address,address,uint256)` changed to `LSP7AmountExceedsBalance(uint256,address,uint256)` . And the error hash from `0xc5e194ab` to `0x08d47949`

## LSP8

- Add `onReceived` for `safeTransferFrom` of `LSP8CompatibleERC721`
- Changing `bool` param to `bool[]` in token `transferBatch(..)`
- InterfaceId was: `0x49399145` changed to : `0x622e7a01`

The specifications at LIP repo are updated, feel free to go there to seek more information.