12/12

How to build and use ERC-1155 tokens

An intro to the new standard for having many tokens in one

ERC-1155 allows you to send multiple different token classes in one transactions. You can imagine it as transferring Chinese Yuan and US Dollars in a single transfer. ERC-1155 is most commonly known for being used in games, but there are many more use cases for it.

First of all though, what are those multi tokens?

What is ERC-1155?

The key feature introduced by ERC-1155 is multi token support. Let's say we are in a world where each airline has its own token. In a way that's not too far from reality where each airline sits in a different country with its own national currency. Now let's say I wanted to book a flight across the world with three connecting flights from three different airlines. To pay for my flight ticket, I would need to make three separate transactions and pay for each individually.

Many things could go wrong this way. What if I pay for flight 1, but then flight 2 is already booked out? Now I have to find a new connections. In the real world with centralized currencies this job is done by the online travel agency (OTA) or a more old-school traditional agency. In our example though, one would have to create a new smart contract that does all three trades in a row, deploy it and then call the trade function. A bit complicated to only pay for one flight trip, isn't it?

What if instead we created a single multi-token contract which holds the individual tokens for every airline? That's exactly what ERC-1155 allows you to do. Instead of just one token class, we can add as many as we want. Let's look at the functionality in more depth.

ERC-1155 Functions and Features:

1. **Batch Transfer**: Transfer multiple assets in a single call.
2. **Batch Balance**: Get the balances of multiple assets in a single call.
3. **Batch Approval**: Approve all tokens to an address.
4. **EIP-165 Support**: Declare supported interfaces.
5. **Hooks**: Receive tokens hook.
6. **NFT support**: If supply is only 1, treat it as NFT.
7. **Safe Transfer Rules**: Set of rules for secure transfer.

1. Batch Transfer

The batch transfer works very similar to regular ERC-20 transfers. Let's look at the regular ERC-20 transferFrom function:

```
// ERC-20
function transferFrom(address from, address to, uint256 value) external returns (bool);
```

The only difference in ERC-1155 is that we pass the values as an array and we also pass an array of id's. For example given `_ids`=[3, 6, 13] and `_values`=[100, 200, 5], the resulting transfers will be

1. Transfer 100 tokens with id 3 from `_from` to `_to`.
2. Transfer 200 tokens with id 6 from `_from` to `_to`.
3. Transfer 5 tokens with id 13 from `_from` to `_to`.

In ERC-1155 we only have `transferFrom`, no `transfer`. To use it like a regular transfer, just set the `from` address to the address that's calling the function.

- *Why is it called 'safe'-BatchTransferFrom?* - See section 7 below about safe transfer rules.
- *Why is it not returning boolean?* - The transaction reverts on failure, same as the SafeERC-20 implementation we discussed previously.
- *What is the bytes data field?* - Just as in ERC-777, you can pass arbitrary data along with the call which will also be passed to the receive hook.

```
// ERC-1155
function safeBatchTransferFrom(
    address _from,
    address _to,
    uint256[] calldata _ids,
    uint256[] calldata _values,
    bytes calldata _data
) external;
```

2. Batch Balance

The respective ERC-20 balanceOf call likewise has its partner function with batch support. As a reminder, this is the ERC-20 version:

```
// ERC-20
function balanceOf(address owner) external view returns (uint256);
```

Even simpler for the balance call, we can retrieve multiple balances in a single call. We pass the array of owners, followed by the array of token ids.

For example given _ids=[3, 6, 13] and _owners=[0xbeef..., 0x1337..., 0x1111...], the return value will be

[

1. `balanceOf(0xbeef...)`,
2. `balanceOf(0x1337...)`,
3. `balanceOf(0x1111...)`

]

```
// ERC-1155
function balanceOfBatch(
    address[] calldata _owners,
    uint256[] calldata _ids
) external view returns (uint256[] memory);
```

3. Batch Approval

The approvals are slightly different from ERC-20. Instead of approving specific amounts, you simply set an operator to approved or not approved via `setApprovalForAll`.

Reading the current status can be done via `isApprovedForAll`. As you can see, it's an all or nothing. You cannot define how many tokens to approve or even which token class.

This is intentionally designed with simplicity in mind. You can only approve everything for one address. If you need more fine granular control over specific approvals, check out EIP-1761.

```
// ERC-1155
function setApprovalForAll(
    address _operator,
    bool _approved
) external;
```

```
// ERC-1155
function isApprovedForAll(
    address _owner,
    address _operator
) external view returns (bool);
```

4. EIP-165 Support

If you don't know what EIP-165 is yet, see my tutorial here. In short it's way for smart contracts to define what interfaces they support. So for example, does the smart contract support receiving ERC-1155 tokens? If it does, the respective `supportsInterface` function must exist and return true for that contract.

Which brings us directly to the next point: hooks.

5. Receive Hook

Given the EIP-165 support, ERC-1155 supports receive hooks for smart contracts only. The hook function must return a magic predefined bytes4 value which is given as:

```
bytes4(keccak256("onERC1155BatchReceived(address,address,uint256[],uint256[],bytes)"))
```

When the receiving contract returns this value, it is assumed the contract accepts the transfer and knows how to handle the ERC-1155 tokens. Great, no more stuck tokens in a contract!

```
function onERC1155BatchReceived(
    address _operator,
    address _from,
    uint256[] calldata _ids,
    uint256[] calldata _values,
    bytes calldata _data
) external returns(bytes4);
```

6. NFT Support

When a supply is given as just one, the token is essentially a non-fungible token (NFT). And as is standard for ERC-721, you can define a metadata URL. The URL can be read and modified by clients, see here.

7. Safe Transfer Rules

We've touched on a few safe transfer rules already in the previous explanations. But let's look at the most important of the rules:

1. The caller must be approved to spend the tokens for the `_from` address or the caller must equal `_from`.
2. The transfer call must revert if
    1. `_to` address is 0.
    2. length of `_ids` is not the same as length of `_values`.
    3. any of the balance(s) of the holder(s) for token(s) in `_ids` is lower than the respective amount(s) in `_values` sent to the recipient.
    4. any other error occurs.

Note: All batch functions including the hook also exist as version without batch. This is done for gas efficiency, considering transferring just one asset will likely still be the most commonly used way. We've left them out for simplicity in the explanations, incl. safe transfer rules. The names are identical, just remove the 'Batch'.

Implementing our Airline-Tokens with ERC-1155

We can implement the ERC-1155 very easily with the well-documented and audited Openzeppelin contracts. Install them via npm or import the Github URL directly in Remix. The documentation for the Openzeppelin ERC-1155 can be seen here.

Our previous Airline example could implemented like this:

```
import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";

contract AirlineTokens is ERC1155 {
    address public governance;
    uint256 public airlineCount;

    modifier onlyGovernance() {
        require(msg.sender == governance, "only governance can call this");

        _;
    }

    constructor(address governance_) public ERC1155("") {
        governance = governance_;
        airlineCount = 0;
    }

    function addNewAirline(uint256 initialSupply) external onlyGovernance {
        airlineCount++;
        uint256 airlineTokenClassId = airlineCount;

        _mint(msg.sender, airlineTokenClassId, initialSupply, "");
    }
}
```

This is all we need. Now someone can call

```
AirlineTokens.safeBatchTransferFrom(myBuyerAddress, sellerAddress, [airlineId1, airlineId2, airlineId3], [firstFlightPrice, secondFlightPrice, thirdFlightPrice], '').
```

And pay in each airline currency in a single transaction. Great!

Simpler Alternative: ERC-1178

If that was all a bit too complicated and you just want a simple standard that supports multiple fungible tokens in one, check out ERC-1178. It achieves exactly that, but without all the additions from 1155. An implementation updated to Solidity 0.7 can be found here.

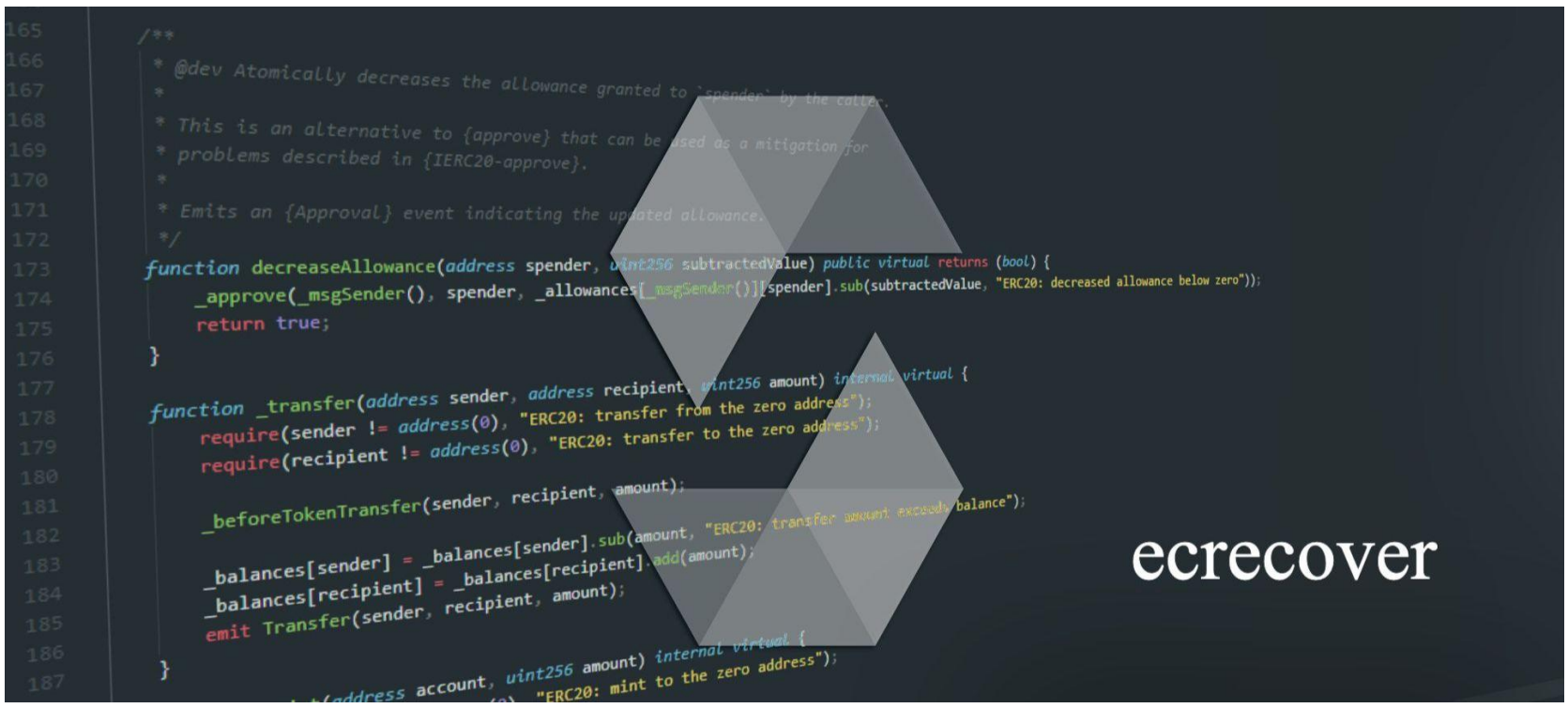
Markus Waas
Solidity Developer

ERC-1155 EIP ERC-20 EIP-165

More great blog posts from Markus Waas

June 2021

06/20



What is ecrecover in Solidity?

A dive into the waters of signatures for smart contracts

Ever wondered what the hell the deal is with the ecrecover command in Solidity? It's all about signatures and keys... What is ecrecover ? You may have seen ecrecover in a Solidity contract before and wondered what exactly the deal with this was. Well you came across the EVM precompile ecrecover ....

READ MORE >

06/13

How to use Binance Smart Chain in your Dapp

Deploying and onboarding users to the Binance Smart Chain (BSC)

Defi has been a major contributor to the Binance Smart Chain taking off recently. Along with increasing gas costs on Ethereum mainnet which are actually at one of the lowest levels since a long time at the time of this writing, but will likely pump again at the next ETH price pump. So how does...

READ MORE >

May 2021

05/29

Using the new Uniswap v3 in your contracts

What's new in Uniswap v3 and how to integrate Uniswap v3

If you're not familiar with Uniswap yet, it's a fully decentralized protocol for automated liquidity provision on Ethereum. An easier-to-understand description would be that it's a decentralized exchange (DEX) relying on external liquidity providers that can add tokens to smart contract pools and...

READ MORE >

LOAD MORE >