wFIAT: local stablecoins by Ripio

Version 1.0 - October 2025

Abstract

In this paper we introduce wFIAT stablecoins, tokens issued by a Ripio group affiliate that are pegged 1:1 to the value of local currencies of certain Latin American countries.

Contents

What is wFIAT?

wFIAT is a common denominator for several stablecoins of certain Latam local currencies, such as wARS, wCOP, wBRL, etc. wFIAT is issued by a Ripio group affiliate and is fully-collateralized in local fiat currencies and/or local currency-denominated instruments in regulated institutions, in order to maintain a 1:1 value ratio with the local currency to which the specific stablecoin is pegged.
wFIAT allows token holders to transact, save, and build on decentralized networks using a stablecoin pegged 1:1 to their local currency.

What is wFIAT used for?

- **On-chain payments & remittances.** Send and receive wFIAT instantly, 24/7, with minimal fees.
- **Fiat on-/off-ramp.** Bridge between traditional banking rails and crypto rails via Ripio's platforms and other VASP platforms.
- **DeFi & dApp integration.** Use wFIAT as collateral for defi operations, liquidity pools, and payment rails.

How many wFIAT stablecoins are in circulation?

wFIAT stablecoins have an initial floating supply. Each wFIAT stablecoin is backed by reserves held in local regulated institutions.

Who is behind wFIAT?

wFIAT are issued by a Ripio group affiliate.

Economic model & collateralization

Each wFIAT stablecoin is pegged 1:1 to the local fiat currency and fully backed by reserves held within regulated institutions.

New tokens are minted by the issuer only upon sufficient collateralization. Conversely, tokens are burned when deemed necessary by the issuer, such as in cases when market liquidity is withdrawn.

All issued tokens remain fully collateralized, with the corresponding reserves held in segregated accounts of issuer's affiliated entities. Strict reconciliation processes are kept, and issuer may publish independent third-party attestations periodically to ensure transparency and proof of backing.

This structure ensures that wFIAT maintains its 1:1 peg to the corresponding local fiat currency, while giving the issuer the flexibility to manage liquidity and market conditions efficiently across both centralized and decentralized environments.

Where can I buy wFIAT?

- **Ripio's platforms:** Purchase will be made available in Ripio's platforms with available funds, either fiat or crypto, and in other VASP platforms.

- **Decentralized Exchanges:** Swap a cryptocurrency or token for wFIAT on Uniswap, QuickSwap, etc.

Technical architecture

wFIAT is implemented as an ERC-20 compliant smart contract deployed on multiple networks, including Ethereum, Base and Worldchain, that manages minting and burning operations.
The smart contract includes:

- **Mint/Burn Controls:** Only addresses with the MINTER_ROLE can mint tokens..

- **Transparency:** Balances and supply are publicly verifiable on-chain.

- **Interoperability:** As a standard ERC-20 token, wFIAT integrates seamlessly with wallets, DEXs, and DeFi protocols.

- **Audits:** Built entirely with OpenZeppelin's battle-tested, industry-standard libraries, with publicly accessible source code for full transparency.

- **No e-money or deposits:** As pure ERC20 stablecoins, wFIAT stablecoins do not represent user deposits, they are not e-money in any jurisdiction, and a token holder cannot claim any funds or any obligations from the issuer (a token holder could only sell the token in the market –there is no "redemption" right).

Networks Supported

wFIAT stablecoins are available on:

Ethereum Mainnet

- **wARS —** 0x0DC4F92879B7670e5f4e4e6e3c801D229129D90D

- **wMXN** — 0x337E7456B420bD3481e7FA61fA9850343d610d34
- **wBRL** — 0xD76f5Faf6888e24D9F04Bf92a0c8B921FE4390e0
- **wCOP** — 0x8a1D45e102e886510e891d2Ec656a708991e2D76

Worldchain

- **wARS** — 0x0DC4F92879B7670e5f4e4e6e3c801D229129D90D
- **wMXN** — 0x337E7456B420bD3481e7FA61fA9850343d610d34
- **wBRL** — 0xD76f5Faf6888e24D9F04Bf92a0c8B921FE4390e0
- **wCOP** — 0x8a1D45e102e886510e891d2Ec656a708991e2D76

Base

- **wARS** — 0x0DC4F92879B7670e5f4e4e6e3c801D229129D90D
- **wMXN** — 0x337e7456b420bd3481e7fa61fa9850343d610d34
- **wBRL** — 0xD76f5Faf6888e24D9F04Bf92a0c8B921FE4390e0
- **wCOP** — 0x8a1D45e102e886510e891d2Ec656a708991e2D76

External Technical Specification

> Solidity ^0.8.27 · OpenZeppelin Upgradeable ^5.x · UUPS proxy pattern

1. Abstract

wFIAT is an upgradeable ERC-20 token that includes burning, pausability, EIP-2612 Permit approvals, role-based access control, and UUPS upgrade authorization. The contract is designed to be deployed behind a UUPS proxy and initialized once via `initialize`.

2. Versioning & Dependencies

- **Compiler:** `pragma solidity ^0.8.27;`
- **OpenZeppelin (upgradeable) modules:**
  - `ERC20Upgradeable`
  - `ERC20BurnableUpgradeable`
  - `ERC20PausableUpgradeable`
  - `AccessControlUpgradeable`
  - `ERC20PermitUpgradeable`
  - `UUPSUpgradeable`
- **Proxy pattern:** UUPS (upgrade logic resides in the implementation; proxy delegates).

3. Inheritance Structure

```
None
classDiagram
    Initializable <|-- wFIAT
    ERC20Upgradeable <|-- wFIAT
    ERC20BurnableUpgradeable <|-- wFIAT
    ERC20PausableUpgradeable <|-- wFIAT
    AccessControlUpgradeable <|-- wFIAT
    ERC20PermitUpgradeable <|-- wFIAT
    UUPSUpgradeable <|-- wFIAT
```

4. Roles

wFIAT uses `AccessControlUpgradeable` with the following role constants:

Constant	Value	Purpose
`DEFAULT_ADMIN_ROLE`	`0x00…00` (OZ default)	Administer all roles through `grantRole`/`revokeRole`/`renounceRole`.
`PAUSER_ROLE`	`keccak256("PAUSER_ROLE")`	Authorize `pause()` and `unpause()`.
`MINTER_ROLE`	`keccak256("MINTER_ROLE")`	Authorize `mint(address,uint256)`.
`UPGRADER_ROLE`	`keccak256("UPGRADER_ROLE")`	Authorize `_authorizeUpgrade(address)`; required for `upgradeTo` / `upgradeToAndCall`.

5. Initialization

The implementation constructor disables initializers:

```
None
constructor() { _disableInitializers(); }
```

The proxy-side initializer must be called exactly once:

```
None
function initialize(
    address defaultAdmin,
    address pauser,
    address minter,
    address upgrader,
    string memory tokenName,
    string memory tokenSymbol
) public initializer;
```

Effects:

1. Initializes inherited modules in this order:
    - `__ERC20_init(tokenName, tokenSymbol)`
    - `__ERC20Burnable_init()`
    - `__ERC20Pausable_init()`
    - `__AccessControl_init()`
    - `__ERC20Permit_init(tokenName)`
    - `__UUPSUpgradeable_init()`
2. Grants roles to the provided addresses: `DEFAULT_ADMIN_ROLE`, `PAUSER_ROLE`, `MINTER_ROLE`, `UPGRADER_ROLE`.

6. Token Parameters

- **Name / Symbol:** supplied via `initialize` (e.g., `"wFIAT"`, `"WFIAT"`).
- **Decimals:** inherits ERC20 default of 18.
- **Initial supply:** `0` (supply is produced via `mint`).

7. Pausable Behavior

wFIAT inherits `ERC20PausableUpgradeable`. While paused, the internal hook `_update` reverts, which blocks:

- `transfer`
- `transferFrom`
- `approve`-driven state changes that rely on `_update`
- `mint`
- `burn` and `burnFrom`

Control functions:

```
None
function pause()   public onlyRole(PAUSER_ROLE);
function unpause() public onlyRole(PAUSER_ROLE);
```

8. Minting & Burning

- Minting

```
None
function mint(address to, uint256 amount) public
onlyRole(MINTER_ROLE);
```

Mints `amount` tokens to `to` and increases `totalSupply`.

- Burning
  - `burn(uint256 amount)` allows the caller to burn their own balance.
  - `burnFrom(address account, uint256 amount)` allows an approved spender to burn from `account` up to the allowance.

Both operations are blocked while paused due to `_update` guarding.

9. Permit (EIP-2612)

`ERC20PermitUpgradeable` enables gasless approvals via EIP-2612:

```
None
function permit(
  address owner,
  address spender,
  uint256 value,
  uint256 deadline,
  uint8 v,
  bytes32 r,
  bytes32 s
) external;
```

- The EIP-712 domain name equals `tokenName` passed to `initialize`.
- Successful calls update `allowance(owner, spender)` and consume `nonces(owner)`.

10. Upgradeability (UUPS)

Upgrade entry points (`upgradeTo`, `upgradeToAndCall`) are provided by `UUPSUpgradeable`. Authorization is enforced by the internal override:

```
None
function _authorizeUpgrade(address newImplementation)
  internal
  override
  onlyRole(UPGRADER_ROLE)
{}
```

11. Overrides

To compose ERC20 base logic with pausable checks, wFIAT overrides `_update`:

```
None
function _update(address from, address to, uint256 value)
  internal
  override(ERC20Upgradeable, ERC20PausableUpgradeable)
{
  super._update(from, to, value);
}
```

12. Public & External API (Summary)

ERC-20

Function	Signature	Description
`name`	`function name() view returns (string)`	Token name.
`symbol`	`function symbol() view returns (string)`	Token symbol.
`decimals`	`function decimals() view returns (uint8)`	Returns 18.
`totalSupply`	`function totalSupply() view returns (uint256)`	Total minted minus burned.
`balanceOf`	`function balanceOf(address) view returns (uint256)`	Holder balance.
`transfer`	`function transfer(address,uint256) returns (bool)`	Transfer when not paused.
`allowance`	`function allowance(address,address) view returns`	Remaining spender allowance.

	`(uint256)`	
`approve`	`function approve(address,uint256) returns (bool)`	Set allowance.
`transferFrom`	`function transferFrom(address,address,uint256) returns (bool)`	Move tokens using allowance.

Burning

Function	Signature	Description
`burn`	`function burn(uint256)`	Burn caller balance.
`burnFrom`	`function burnFrom(address,uint256)`	Burn from `account` using allowance.

Pausable

Function	Signature	Description
`pause`	`function pause()`	Requires `PAUSER_ROLE`.
`unpause`	`function unpause()`	Requires `PAUSER_ROLE`.

Permit (EIP-2612)

Function	Signature	Description
`permit`	`function permit(address,address,uint256,uint256,uint8,bytes32,bytes32)`	Off-chain signed approval.
`nonces`	`function nonces(address) view returns (uint256)`	Owner-specific nonce.

DOMAIN_SEPARATOR	function DOMAIN_SEPARATOR() view returns (bytes32)	EIP-712 domain separator.

Access Control

Function	Signature	Description
hasRole	function hasRole(bytes32,address) view returns (bool)	Role check.
grantRole	function grantRole(bytes32,address)	Assign role (admin only).
revokeRole	function revokeRole(bytes32,address)	Remove role (admin only).
renounceRole	function renounceRole(bytes32,address)	Caller renounces a role.
getRoleAdmin	function getRoleAdmin(bytes32) view returns (bytes32)	Admin role for a role.

UUPS Upgradeability

Function	Signature	Description
upgradeTo	function upgradeTo(address)	Upgrade implementation (requires UPGRADER_ROLE via _authorizeUpgrade).
upgradeToAndCall	function upgradeToAndCall(address,bytes)	Upgrade and call (same authorization).

13. Events

- **ERC-20**
  - `event Transfer(address indexed from, address indexed to, uint256 value);`
  - `event Approval(address indexed owner, address indexed spender, uint256 value);`
- **AccessControl**
  - `event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed newAdminRole);`
  - `event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender);`
  - `event RoleRevoked(bytes32 indexed role, address indexed account, address indexed sender);`