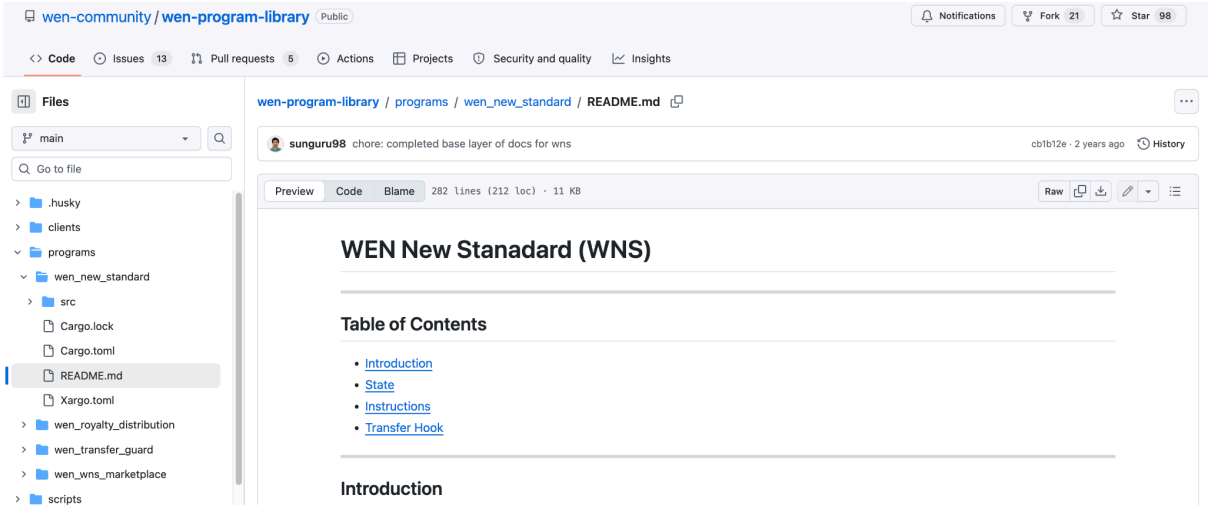
source
https://github.com/wen-community/wen-program-library/blob/main/programs/wen_new_standard/README.md

wen-community/wen-program-library Public

wen-program-library
/programs
/wen_new_standard

/README.md



WEN New Stanadard (WNS)

Table of Contents

Introduction

The WEN New Standard program was created to leverage Token extensions with custom royalty enforcements. The motivation behind this program is to foster an ecosystem around the newly available extensions without needing much of external PDAs to resort towards. Group and token members with royalties bound within results in an optimal yet essential structure for both Fungible, SemiFungible and NonFungible assets.

State

There exists a total of 5 state PDAs within the program's boundaries, in which the group and member custom accounts would be migrated to the mint address when Group extensions are live over mainnet.

- Manager - Currently this PDA is responsible for holding the collection and it's members' mint account and extension's authority to programatically change any extension configurations or to aid in any mutation under the mint account.

```
struct Manager {
    // bump: u8
}
```

- Group - Each collection mint account is considered a group. Whenever a collection is initialized/created, a new `Group` PDA is stored.

```
struct TokenGroup {
    /// The authority that can update the group
    pub update_authority: Pubkey,
    /// The token mint
    pub mint: Pubkey,
    /// The current size of the group (current member count)
    pub size: u32,
    /// The maximum number of group members
    pub max_size: u32,
}
```

- Member - Every sub collection/member mint account is considered a member. A `TokenMember` PDA is initialized when a new member mint account is created and stored with the required fields mentioned below.

```
struct TokenGroupMember {
```

```
    /// The token mint
    pub mint: Pubkey,
    /// The group PDA which is under
    pub group: Pubkey,
    /// The index at which the member has joined the group
    pub member_number: u32,
}
```

- Approve Transfer Account - An approve transfer account is currently a clock's slot verification account used inside the transfer hook, to check if the PDA has been initalized/assigned the right slot in the same transaction as the transfer instruction is invoked. This makes sure, that a transfer instruction cannot be called as is from another program without setting this PDA's state right. In other words, an approve account is a validity checker for any transfer instruction between any member token accounts.

```
struct ApproveAccount {
    // The current slot where the transfer is about to happen
    slot: u64,
}
```

- Extra Meta Account List - This account technically isn't created from scratch under this program, but since it still is being initialized under WNS program as it's owner, we mention this as well. The `ExtraMetaAccountList` PDA is also used for transfer hooks. This PDA is initialized only when royalties are being added. If no royalties are required, then the transfer hook will not be pointed to WNS, which results to not needing the `ExtraMetaAccountList`'s initialization.

Program Instructions

1. `init_manager_account` - The first instruction required to be invoked by the program deployer to set a manager PDA account responsible for critical updates and mutations over the program.

Accounts required

- payer [signer, writable]
- manager [writable]
- system_program []

2. `create_group_account` - Allows a user to create a Token22 NFT with a custom made Group PDA to support `GroupPointerExtension`. This will be migrated to

native `InitializeGroup` instruction in Token extensions program once it's available. This instruction primarily focusses on creating the mint account, initializing the required extensions, metadata and minting a token to the receiver.

Accounts required

- payer [signer, writable]
- authority [signer]
- group [writable]
- mint [signer, writable]
- mint_token_account [writable]
- manager
- system_program []
- associated_token_program []
- token_extensions_program []

3. `update_group_account` - Allows the group authority to update the group configurations or the base metadata

Accounts required

- payer [signer, writable]
- authority [signer]
- group [writable]
- mint [signer, writable]
- system_program []
- token_extensions_program []

4. `create_mint_account` - Once the transfer is completed, the vault keeps a note of `claim_data` of which creator requires how much percentage of the vault royalty funds. Allows any `creator` to withdraw their share.

Accounts required

- payer [signer, writable]
- authority [signer]
- receiver []
- mint [signer, writable]
- mint_token_account [writable]
- manager []
- system_program []
- associated_token_program []
- token_extensions_program []

5. `add_mint_to_group` - Once the transfer is completed, the vault keeps a note of `claim_data` of which creator requires how much percentage of the vault royalty funds. Allows any `creator` to withdraw their share.

Accounts required

- payer [signer, writable]
- authority [signer]
- group [writable]
- member [writable]
- mint []
- manager []
- system_program []
- token_extensions_program []

6. `burn_mint_account` - Allows the token member token account and the mint account (via token extensions close_authority) to be burnt.

Accounts required

- payer [signer, writable]
- user [signer]
- mint [writable]
- mint_token_account [writable]
- manager []
- token_extensions_program []

7. `freeze_mint_account` - The token member mint account must have a delegated authority (through token instructions approve) in order to freeze the token the mint_token_account

Accounts required

- user []
- delegate_authority [signer, writable]
- mint []
- mint_token_account [writable]
- manager []
- token_extensions_program []

8. `thaw_mint_account` - Same as `freeze_mint_account`, but for unfreezing. Allows the user to keep in custody for staking/escrow purposes without having the need to transfer.

Accounts required

- user []
- delegate_authority [signer, writable]
- mint []
- mint_token_account [writable]
- manager []
- token_extensions_program []

9. `add_royalties` - Allowing the creator of the NFT collection to opt in for royalties, by updating the metadata of each member NFT with the creators list/share and the basis points that would be transferred for royalty share.

Accounts required

- payer [signer, writable]
- authority [signer]
- mint [writable]
- extra_meta_account [writable]
- system_program []
- token_extensions_program []

10. `modify_royalties` - Allows for any modification over the already present royalty configurations.

Accounts required

- payer [signer, writable]
- authority [signer]
- mint [writable]
- system_program []
- token_extensions_program []

11. `add_metadata` - Allows either a collection or member NFT to add additional metadata based on the nature of the NFT. Each instruction invoke would add one entry to the tuple vector.

Accounts required

- payer [signer, writable]
- authority []
- mint [writable]
- token_extensions_program []
- system_program []

12. `remove_metadata` - Allows either a collection or member NFT to remove any field in additional metadata.

Accounts required

- payer [signer, writable]
- authority []
- mint [writable]
- token_extensions_program []
- system_program []

13. `approve_transfer` - When a transfer for an NFT is invoked via a protocol CPI, an approve transfer procedure must be present before the transfer instruction in the same transaction. This function sets the blockchain's current slot and checks in the transfer hook, whether the slot is the same or expired. By this way, we could ensure that WNS was called alongside to enforce royalties, and not bypassed. The approve transfer also makes sure to transfer the royalty funds to the distribution program's PDA if the transfer instruction is via a CPI. Else it's a regular non-enforced transfer.

Accounts required

- payer [signer, writable]
- authority [signer, writable]
- mint []
- payment_mint []
- approve_account [writable]
- distribution_account[writable]
- distribution_token_account [writable]
- authority_token_account[writable]
- system_program []
- distribution_program []
- token_program []

Lifecycle of program

The lifecycle of WNS program can be split into two sides. Group and GroupMember. Since we have discussed in detail about the royalty-distribution program in it's respective docs, we will just focus over the WNS side of things. For a group account, the following procedures take place.

1. We `initialize` a group mint account and create a custom `TokenGroup` PDA
2. We have the options to `update` or `add/remove` any additional metadata.

Speaking of a token group member account, the following procedures take place

1. We `initialize` a member mint account with the same ways like a group account
2. A member mint account can be added to a `group`, resulting in creating a custom `TokenGroupMember` PDA
3. A member mint account can be configured with royalties that are embedded onchain with the necessary attributes.
4. A member mint account can also have it's metadata added or removed
5. Any delegate to the member NFT can have the rights to freeze/thaw the token accounts
6. We also have the option to enforce royalty for a particular NFT through `approve_transfer`

Transfer Hook

WNS utilizes transfer hook extension to make sure the program is being utilized as a part of transfer instructions. The way how it works is that, the `ExtraMetaAccountList` is being initialized while the user/creator wishes to opt in for royalties. The account is serialized with one account (being the approve PDA), to let the Token extensions program know that the same is going to be utilized during hook `execute` function.

```
pub fn get_meta_list(approve_account: Pubkey) -> Vec<ExtraAccountMeta> {
    vec![ExtraAccountMeta {
        discriminator: 0,
        address_config: approve_account.to_bytes(),
        is_signer: false.into(),
        is_writable: true.into(),
    }]
}

// initialize the extra metas account
let extra_metas_account = &ctx.accounts.extra_metas_account;
let metas =
get_meta_list(get_approve_account_pda(ctx.accounts.mint.key()));
let mut data = extra_metas_account.try_borrow_mut_data()?;
ExtraAccountMetaList::init::<ExecuteInstruction>(&mut data, &metas)?;
```

Then during the `execute` function, the required accounts are passed via anchor's remaining accounts and checked if the slot has been set right or not. If yes, it's reset back to the original value and made sure for next seamless transfer.

```
if ctx.remaining_accounts.is_empty() {
    return Err(MetadataErrors::MissingApproveAccount.into());
}
let mut approve_account: ApproveAccount = AnchorDeserialize::deserialize(
    &mut &ctx.remaining_accounts[0].try_borrow_mut_data()?[8..],
)?;
```

```
if approve_account.slot == Clock::get()?.slot {
    // mark approve account as used by setting slot to 0
    approve_account.slot = 0;
    AnchorSerialize::serialize(
        &approve_account,
        &mut &mut ctx.remaining_accounts[0].try_borrow_mut_data()?[8..],
    )?;
    Ok(())
} else {
    Err(MetadataErrors::ExpiredApproveAccount.into())
}
```

source
https://github.com/wen-community/wen-program-library/blob/main/programs/wen_new_standard/README.md

More about and Sources

How is this different from Token-2022

Token-2022 is the technical name and GitHub repository for the new version of the SPL Token Program released by Solana Labs. Token Extensions, in turn, are the new capabilities implemented by this token standard. Think of them as a set of new functions and features now available at the Token Program level.

Technical Manifesto: Token-2022 (Token Extensions) and Wen New Standard (WNS)

PART 1. Token-2022 (Token Extensions): The Institutional Base Layer

Definition and Architectural Status

Security & Audits: The core program code has successfully undergone 5 independent security audits by industry-leading firms: *Halborn, Zellic, NCC, Trail of Bits*, and *OtterSec*.

Token-2022 is the technical name and GitHub repository for the new SPL (Solana Program Library) token program developed and released by Solana Labs.

Token Extensions is the functional and ecosystem-facing name for the next-generation capabilities unlocked by this new program directly at the blockchain protocol level.

Backward Compatibility: While the protocol maintains backward compatibility with the legacy `SPL Token` standard, decentralized applications (dApps) must be upgraded to support tokens minted with specific extensions.

Core Native Extensions and Their Purposes

Token-2022 replaces fragmented third-party smart contracts with a unified, native set of features programmed directly at the asset's mint initialization:

- **Confidential Transfers:** Masking account balances and transaction amounts using ElGamal encryption. It restricts viewing rights to a *single auditor public key* to ensure regulatory compliance. Enabled network-wide via the *Agave 2.0* validator client client client.
- **Transfer Fees:** A native capability allowing token issuers to configure a percentage or flat fee deducted automatically on every transfer for direct monetization.

- **Transfer Hooks:** Intercepting token transfers to invoke a custom Solana program. This forces a programmatic verification check, enabling rule-based transfer execution like KYC/AML verification or white/blacklisting.
- **Permanent Delegation:** Granting the token issuer absolute authority to transfer, freeze, or burn tokens from any account. This is critical for Real-World Assets (RWA) and compliant stablecoins (such as Paxos USDP).
- **Required Metadata:** Forcing token-level data payload attachments directly within the mint account. This binds accounting, attribution, or asset properties natively to the transaction itself.

WNS (Wen New Standard): The Application-Level NFT Layer

1. Architectural Essence

WNS (Wen New Standard) is an open-source asset representation specification and tooling framework designed by the *Wen Developer Community*.
It is explicitly built to standardize **NFTs (Non-Fungible Tokens)** on top of Token-2022 (Token Extensions), solving missing application-layer requirements of the raw standard.

2. Core Components of the Wen Program Library (WPL)

The WNS specification is encoded and enforced through two crucial on-chain programs:

1. `wen_new_standard` **Program**
    - **Program ID:** look in official source community tech github source main https://github.com/wen-community/ folder https://github.com/wen-community/wen-program-library
    - **Purpose:** Complements Token Extensions by adding native grouping structures (*Token Groups*) and establishing default configurations for required NFT extensions. It provides a standardized CLI for collection minting and management operations.
2. `wen_royalty_distribution` **Program**
    - **Program ID:** look in official source community tech github source main https://github.com/wen-community/ folder https://github.com/wen-community/wen-program-library
    - **Purpose:** Handles the programmatic allocation and distribution of creator royalties. It **strictly enforces royalty enforcement** by leveraging Token-2022's **Transfer Hook** extension. Because this logic executes on-chain during the transfer state itself, marketplaces can no longer bypass creator fees.

      Wen New Standart

      https://github.com/wen-community/wen-program-library/blob/main/programs/wen_new_standard/README.md

---

The Direct Link: Token-2022 and WNS Synergy

The relationship between these two technologies is strictly hierarchical (**L1 Protocol Program -> Application Framework**):

| Solana Base Layer (L1 Consensus)
|
+----------
| Token-2022 Program / Token Extensions (Infrastructure)
| - Transfer Hooks - Metadata - Confidential
+----------

|
Natively implements & utilizes features
v
+----------
| WNS: Wen New Standard (Application)
| - wen_new_standard (NFT Creation, Token Groups/Collections)
| - wen_royalty_distribution (Enforced Royalty Transfer Hook)
+---------------------------------------

The Shift from Legacy Architecture: In the legacy Solana standard, NFTs required complex, separate indexers and metadata states (e.g., Metaplex). WNS moves properties into a **native state**. It abstracts Token-2022's primitive building blocks into a plug-and-play NFT standard.

Abstract Primitives to Functional Execution: Token-2022 introduces the abstract concept of a *Transfer Hook* (allowing custom code to execute during an asset transfer). WNS takes this primitive tool and configures it into `wen_royalty_distribution`, a specialized contract that calculates and claims creator percentages in real time.

Optimized Metadata Layouts: By utilizing Token-2022's *Required Metadata* extension, WNS anchors media links and attributes directly inside the mint account state. This design removes redundant account layouts and keeps storage architecture thin.

If Token-2022 is an infrastructure toolkit built to satisfy large financial institutions and enterprise stablecoin issuers (via privacy controls, delegation, and hooks), then WNS is the Web3 community's manifesto. WNS proves that those exact same institutional control mechanics (transfer hooks) can be re-engineered to protect digital artists, making royalty avoidance technically impossible at the protocol level.

The Direct Technical Link: github.com & Token-2022

The **Wen Foundation (Wen Community)** GitHub organization does not merely consume the `Token-2022` standard—its flagship **`wen-program-library` (WPL)** acts as a production-grade abstraction layer written in the **Anchor** framework. It is built specifically to automate, govern, and enforce Token-2022 extensions for Non-Fungible Tokens (NFTs).

The direct technical architecture relies on three primary connection vectors:

1. **Manifest Layer Dependencies:** The core `wen_new_standard` smart contracts directly import `spl-token-2022` and `spl-token-metadata-interface` crates. WNS serves as an orchestrator, calculating byte layouts and managing Cross-Program Invocations (CPI) directly into the native Token-2022 program state.
2. **Native Extension Configurations:**
    - Instead of utilizing heavy external metadata accounts (like legacy standards), WNS initializes the mint by embedding the native **Metadata Pointer** and **Metadata** extensions directly inside the single token account layout according to Token-2022 protocol rules.
    - Multi-asset asset groupings (Collections) are established utilizing Token-2022's native **Token Group** and **Token Group Member** extensions, keeping index structures lean.
3. **Transfer Hook Standardization:** The `wen_royalty_distribution` program explicitly implements the `spl-transfer-hook-interface`. By hooking into the native Token-2022 transfer flow, it intercepts execution states to enforce immutable, on-chain royalty split distributions.

Production Anchor/Rust Template: Enforced Royalty & Transfer Hook

The following **Anchor (v0.30.1+)** program implements a custom **Transfer Hook** extension. Unlike vanilla transfer hooks that only validate text states, this production sample captures token transfer events and **strictly enforces a royalty payout** in native SOL or SPL tokens directly to a designated creator account before allowing the core transfer to settle.

example
Cargo.toml Configurations

toml

```
[dependencies]
anchor-lang = { version = "0.30.1", features = ["init-if-needed"] }
anchor-spl = { version = "0.30.1", features = ["token_2022"] }
spl-transfer-hook-interface = "0.6.3"
spl-tlv-account-resolution = "0.6.3"
```

Use the code with caution. This is an example of the capabilities, not a final offer for use. Use API data according to the protocol github.

example

Smart Contract Implementation (`src/lib.rs`)
rust

```
use anchor_lang::prelude::*;
use anchor_lang::solana_program::program::invoke;
use anchor_lang::solana_program::system_instruction;
use anchor_spl::token_interface::{TokenAccount, Mint};
use anchor_spl::token_2022::spl_token_2022;

declare_id!("HookRoyalty11111111111111111111111111111111");

#[program]
pub mod enforced_royalty_hook {
    use super::*;

    /// Initializes the layout describing what additional accounts this hook requires.
    /// This pattern mimics WNS implementations to ensure client indexers resolve metadata addresses.
    pub fn initialize_extra_account_meta_list(ctx: Context<InitializeExtraAccountMetaList>) -> Result<()> {
        // Define additional account constraints needed during execution
        // Index 0: Creator Wallet (Receiver of Royalties)
        // Index 1: System Program (Required for native SOL transfers)
        let account_metas = vec![
            spl_transfer_hook_interface::instruction::ExtraAccountMeta::new_with_pubkey(
                &ctx.accounts.creator.key(),
                false, // Is not a signer
                true,  // Is writable (receives SOL)
            ).map_err(|_| ProgramError::InvalidArgument)?,
            spl_transfer_hook_interface::instruction::ExtraAccountMeta::new_with_pubkey(
                &anchor_lang::solana_program::system_program::ID,
                false,
                false,
            ).map_err(|_| ProgramError::InvalidArgument)?,
        ];

        let mut data = ctx.accounts.extra_account_meta_list.data.borrow_mut();
```

```
spl_transfer_hook_interface::state::ExtraAccountMetaList::init::<spl_transfer_hook_interface::instruction::Execute
Instruction>(
            &mut data,
            &account_metas,
        )?;

        msg!("Royalty Extra Account Meta List Initialized.");
        Ok(())
    }

    /// The definitive consensus instruction invoked by Token-2022 via CPI on every single transfer.
    pub fn execute(ctx: Context<TransferHookExecute>, amount: u64) -> Result<()> {
        msg!("Enforced Royalty Hook Triggered. Transferring amount: {}", amount);

        // SECURITY VULNERABILITY MITIGATION:
        // Ensure this instruction is strictly called by the official Token-2022 Program, not direct callers.
        let token_program_info = ctx.accounts.token_program.to_account_info();
        if token_program_info.key != &anchor_spl::token_2022::ID {
            return Err(ProgramError::IncorrectProgramId.into());
        }

        // Bypassing protection: Prevent zero-token transfer exploits
        if amount == 0 {
            return Err(ProgramError::InvalidArgument.into());
        }

        // ENFORCED ROYALTY LOGIC (SOL-Based Demonstration):
        // Fixed royalty calculation: e.g., 0.01 SOL (10,000,000 Lamports) mandatory protocol tax per transfer
        let royalty_fee_lamports: u64 = 10_000_000;

        let source_wallet_info = ctx.accounts.owner.to_account_info();
        let creator_wallet_info = ctx.accounts.creator.to_account_info();
        let system_program_info = ctx.accounts.system_program.to_account_info();

        // Validate account availability and execute runtime transfer
        if source_wallet_info.lamports() < royalty_fee_lamports {
            msg!("Error: Source account has insufficient SOL to pay required creator royalties.");
            return Err(ProgramError::InsufficientFunds.into());
        }

        msg!("Deducting royalty tax from transfer origin...");
        invoke(
            &system_instruction::transfer(
                source_wallet_info.key,
                creator_wallet_info.key,
                royalty_fee_lamports,
            ),
            &[
                source_wallet_info,
                creator_wallet_info,
                system_program_info,
            ],
        )?;

        msg!("Royalty payout successfully enforced on-chain. Transfer authorized.");
        Ok(())
    }
```

```
}

#[derive(Accounts)]
pub struct InitializeExtraAccountMetaList<'info> {
    #[account(mut)]
    pub payer: Signer<'info>,

    /// PDA account holding the instruction resolution array. Matches Token-2022 deterministic derivation.
    #[account(
        init,
        space = 8 + 4 + (2 * 35), // Space for 2 ExtraAccountMeta elements
        seeds = [b"extra-account-metas", mint.key().as_ref()],
        bump,
        payer = payer
    )]
    /// CHECK: Validated via deterministic protocol seeds
    pub extra_account_meta_list: AccountInfo<'info>,

    pub mint: InterfaceAccount<'info, Mint>,

    /// The hardcoded target wallet destined to harvest royalties
    /// CHECK: Arbitrary reader account destination
    pub creator: AccountInfo<'info>,

    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
#[instruction(amount: u64)]
pub struct TransferHookExecute<'info> {
    pub source: InterfaceAccount<'info, TokenAccount>,
    /// CHECK: Evaluated by the core Token-2022 validation steps
    pub mint: AccountInfo<'info>,
    pub destination: InterfaceAccount<'info, TokenAccount>,
    /// CHECK: Main transaction signer/authority
    pub owner: AccountInfo<'info>,

    /// CHECK: Automatically resolved by Token-2022 using the "extra-account-metas" seed
    pub extra_account_meta_list: AccountInfo<'info>,

    /// EXTRA ACCOUNT index 0: Must match the initialization target account
    /// CHECK: Account specified during setup to receive funds
    #[account(mut)]
    pub creator: AccountInfo<'info>,

    /// EXTRA ACCOUNT index 1: Required for processing native transfers
    pub system_program: Program<'info, System>,

    pub token_program: Program<'info, anchor_spl::token_2022::Token2022>,
}
```

Use the code with caution. This is an example of the capabilities, not a final offer for use. Use API data according to the protocol github.

How it works in practice: You deploy this contract to the Solana network. When you create (mint) your new token via the CLI or script, you specify this program's address as a Transfer Hook extension. initialize_extra_account_meta_list is called, registering the wallet rules. Now, when any marketplace, DEX, or user attempts to transfer your token using the standard transfer_checked command, the Token-2022 program will

automatically make an internal call (CPI) to your execute function. If your function returns an error (e.g., Err), the entire token transfer transaction is completely rolled back.

Source https://solana.stackexchange.com/questions/7360/transfer-hook-in-token-2022-how-to-actually-transfer

Example

Client TypeScript/JavaScript Implementation

When interacting with assets protected by Token-2022 Transfer Hooks, client applications (DApps, Marketplaces) cannot use legacy token transfer instructions. The runtime requires looking up the `ExtraAccountMetaList` on-chain to append the supplementary validation accounts dynamically before submission.

The code below utilizes the official stable Solana libraries to dynamically parse and resolve the required royalty hook accounts.

JavaScript Script (`transfer-with-hook.ts`)

typescript

```
import {
  Connection,
  PublicKey,
  Transaction,
  sendAndConfirmTransaction,
  Keypair
} from '@solana/web3.js';
import {
  createTransferCheckedWithTransferHookInstruction,
  TOKEN_2022_PROGRAM_ID
} from '@solana/spl-token';

async function executeRoyaltyTokenTransfer() {
  // Initialize connection to Solana Mainnet or Devnet RPC Endpoint
  const connection = new Connection("https://solana.com", "confirmed");

  // Mocking transfer actor keys
  const senderKeypair = Keypair.generate();
  const mintAddress = new PublicKey("MintAddressHere11111111111111111111111111");
  const sourceTokenAccount = new PublicKey("SourceTokenAccountPubkey");
  const destinationTokenAccount = new PublicKey("DestTokenAccountPubkey");

  const decimals = 0; // NFT specific
  const amountToTransfer = BigInt(1);

  console.log("Resolving required on-chain transfer hook instruction accounts...");

  // This single helper utility queries the Token-2022 program state, locates the
  // 'extra-account-metas' PDA, extracts the creator wallet + system program info,
  // and dynamically builds the complete transaction instruction package.
  const transferInstruction = await createTransferCheckedWithTransferHookInstruction(
    connection,
    sourceTokenAccount,
    mintAddress,
    destinationTokenAccount,
    senderKeypair.publicKey,
    amountToTransfer,
    decimals,
```

```
    undefined, // Automatically resolves the associated transfer hook program ID from mint
    "confirmed",
    TOKEN_2022_PROGRAM_ID
  );

  const transaction = new Transaction().add(transferInstruction);

  console.log("Broadcasting transaction to Solana cluster. Royalty payment will auto-execute.");

  // Send transaction safely across the cluster
  // const signature = await sendAndConfirmTransaction(connection, transaction, [senderKeypair]);
  // console.log(`Transaction Settled Successfully. Signature: ${signature}`);
}
```

Use the code with caution. This is an example of the capabilities, not a final offer for use. Use API data according to the protocol github.

Reference Documentation

For deep technical validation and ongoing framework adjustments regarding production `Token-2022` integrations, visit the verified official platform channels:

- Learn more about implementation patterns directly on Solana Core Solutions: Token Extensions.
  open link https://solana.com/solutions/token-extensions
- View production code wrappers and open-source packages directly within the Wen Foundation Repositories (WPL).
  open link https://github.com/wen-community/wen-program-library
- View If you need a TypeScript/JavaScript script to build a translation transaction that takes this hook into account https://solana.com/vi/developers/guides/token-extensions/transfer-hook

---

Comparative Framework: Legacy SPL vs. Token-2022 Compliance

Compliance Dimension	Legacy SPL Token Standard	Token-2022 / WNS Extension Framework
Real-time KYC Validation	Impossible without external application wrappers.	**Native execution** via protocol-level `Transfer Hooks`.
Asset Reclamation & Recovery	Not supported; requires total contract freezing.	**Natively supported** via `Permanent Delegate` authority.
Native Revenue/Tax Modeling	Requires customized and auditable smart contracts.	**Embedded in token design** using `Transfer Fees`.
Default Security Isolation	Open and transacting to all addresses by default.	**Restricted on creation** via `Default Account State`.
On-Chain Audit Trails	Optional; dependent on client dApp implementation.	**Natively enforced** using `Memo Transfer` logic.

Creation of the WNS Specification: The Wen Foundation engineered the **WNS (Wen New Standard)**, an optimized, lightweight NFT standard constructed entirely on top of the Token-2022 Program on GitHub.

Native NFT Fractionalization: Founder *Meow* wrote a digital poem called *"A Love Letter to Wen Bros"*. This poem was minted as a WNS NFT and broken down into **1 trillion fungible $WEN tokens** using native Token-2022 logic. Holding individual $WEN tokens represents ownership of a microscopic fraction of that foundational digital asset.

Enterprise Proof-of-Concept: By executing an airdrop to over **1 million active Solana wallets** simultaneously, WEN stress-tested Token-2022's data layouts under peak network loads. This real-world execution proved to institutional token issuers (such as Paxos for the USDP stablecoin and GMO Trust) that Token-2022's compliance structures remain stable and efficient during periods of extreme traffic congestion.

WEN ($WEN) Crypto Cat — The First Mass-Scale Field Test

WEN ($WEN) is a cat-themed community token created by the core team behind the leading Solana aggregator, Jupiter DEX. Launched in January 2024, WEN was designed as a production-scale stress test for both the Token-2022 standard and Jupiter's LFG Launchpad before conducting the massive JUP token distribution.

WEN's practical implementation validated the standard through specific technical milestones up about.

And Solana Token-2022 Architecture & Built-In Compliance

Before Token-2022, enforcing regulatory rules (like geographic restrictions or KYC checks) required developers to build custom smart contract wrappers around standard SPL tokens. This introduced security risks and fragmented user experiences.

The Solana Token Extensions Spec embeds enterprise-grade compliance tools directly into the protocol layer using modular **Type-Length-Value (TLV) data extensions**. The key compliance extensions include: Source https://solana.com/docs/tokens/extensions

Memo Transfers: This extension strictly requires a text memo to be attached to every single transaction. If the text log (such as an invoice ID or audit code) is missing, the transfer fails natively at t

Transfer Hooks: This is the cornerstone of automated compliance. Every time a token transfer is initiated, a secondary verification program is natively triggered. The transfer will execute only if the program verifies specific parameters, enabling on-chain, real-time KYC/AML validation, geographic whitelisting, or sanction screening.

Permanent Delegate: This extension gives the token issuer an immutable authority to manage token supply across any wallet. Issuers can freeze, seize, or burn tokens without the wallet owner's private keys. This capability is mandatory for regulated financial entities to comply with court orders or reverse fraudulent hacks.

Default Account State: When a new user account is generated for a token, it is automatically initialized in a "Frozen" state. The user cannot transact with the asset until they perform an external verification step (e.g., identity check on a portal) to unfreeze the account.

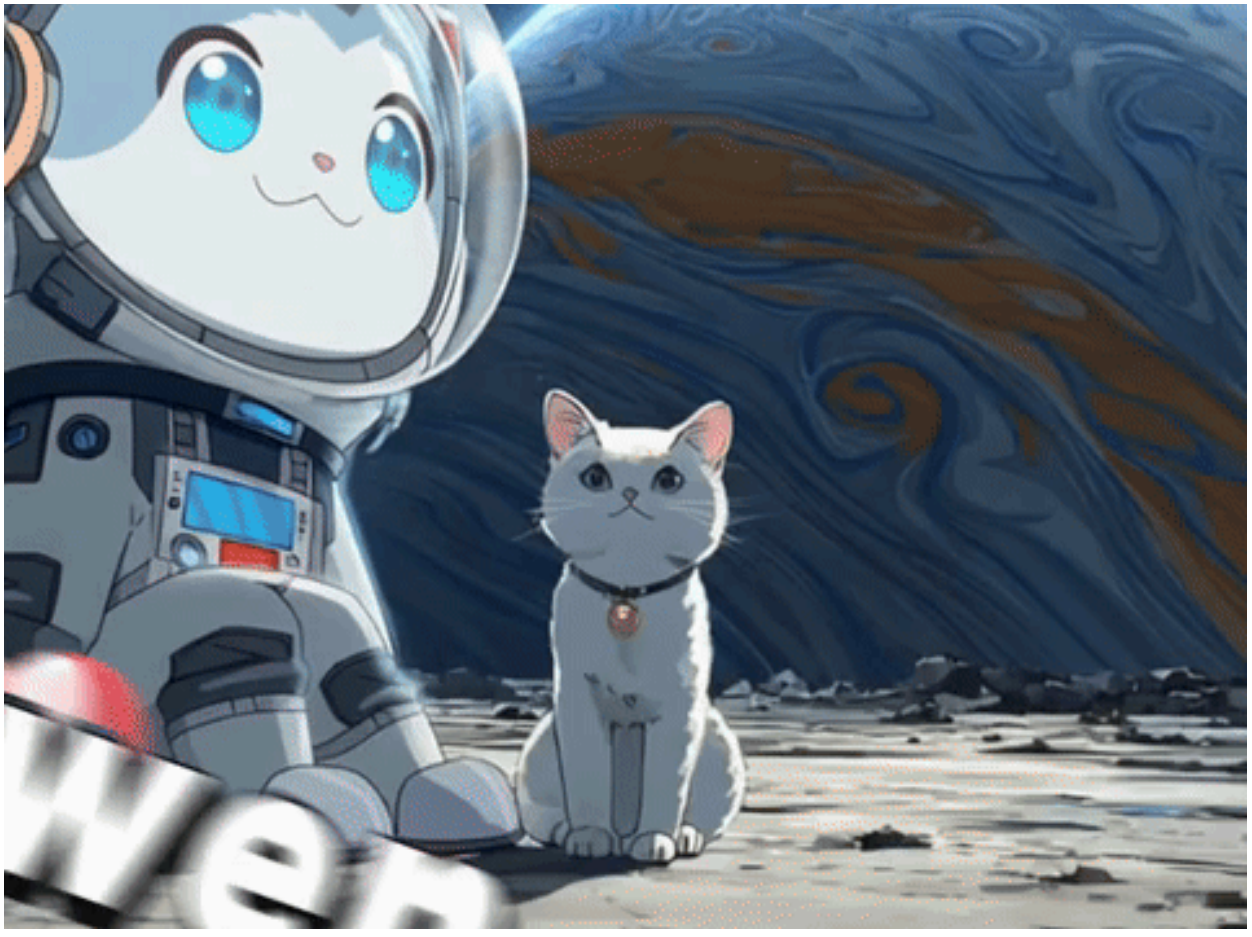
Solana Token-2022 and WEN ($WEN): The Definitive Guide to Built-In Compliance and Live Mass Testing

The **Token-2022 standard** (officially referred to as **Token Extensions**) and the **WEN ($WEN)** meme coin share a fundamental technical link. Rather than being a typical speculative asset, project WEN historically served as the **primary mass-testing infrastructure** to prove the stability and capabilities of this new token standard directly on the Solana Blockchain.

WNS and Token-2022: technical architecture

Token-2022 is the official name of the new SPL token program released by Solana Labs, also known by its functional name Token Extensions. The program has undergone 5 independent security audits (Halborn, Zellic, NCC Group, Trail of Bits, OtterSec) per Solana's official FAQ; more recent independent audit registries also list a Certora audit.

WNS is an open NFT-representation standard, built by the **Wen Community on top of Token-2022, addressing application-layer gaps the raw protocol leaves open** (collection grouping, royalties, metadata). Its architecture consists of two programs: `wen_new_standard` (NFT-group creation and management via 5 state PDAs — Manager, Group, Member, Approve Transfer Account, Extra Meta Account List) and `wen_royalty_distribution` (enforced royalty payouts via Transfer Hook). The core mechanism is the Transfer Hook: on every token-transfer attempt, Token-2022 calls out via CPI to an external verification program; if the check (in WNS, an approval "slot" match) fails, the entire transfer transaction is rolled back.



Practical application of WNS/Token-2022 (per additional sources)

This architecture's real-world use splits into two confirmed tiers.

Tier 1 — the standard's stress test (2024). The WEN memecoin, launched by the Jupiter team in January 2024, used WNS to fractionalize the founder's NFT poem ("A Love Letter to Wen Bros") into one trillion fungible $WEN tokens, airdropped to over 1 million active wallets simultaneously — independently confirmed by DL News and Gate Learn. This was the first mass-scale production test of these Token-2022/WNS structures under peak load.

Tier 2 — institutional application (2026). Shinhan Asset Management (roughly $96.6B AUM) signed a four-party **non-binding** MOU on August 21, 2026 with the Solana Foundation, Etherfuse and Orca to run a **proof-of-concept** — testing the full issuance-to-distribution cycle for a KRW-denominated short-term bond fund aimed at overseas institutional investors, modeled on BlackRock's BUIDL. The PoC specifically exercises the functions Token-2022's architecture provides — KYC/AML checks, issuance, and on-chain distribution. Confirmed by four independent outlets: The Block, CryptoTimes, BigGo Finance, TechTimes.

An important caveat missing from the original description: this is not an exclusive choice of Solana. Exactly one week earlier, on August 14, 2026, Shinhan signed a nearly identical MOU with rival blockchain **Plume** for the same product — parallel infrastructure testing ahead of Korea's Security Token Offering (STO) law taking effect in February 2027. Source: BigGo Finance, TechTimes.

Separately, on the regulatory-recognition side for SOL as an asset — staking ETF filings (Bitwise Solana Staking ETF, form dated Aug. 13, 2026) are filed and registered with the SEC via EDGAR, an independent confirmation of institutional movement toward the SOL asset more broadly, unrelated to WNS/Token-2022 specifically.

This underscores the uncompromising approach to the creation of open data, its assessment, and reassessment, with a strict focus on standardizing objectivity. After all, this isn't about causal laurels and victories, or the commercial component of Solana, but rather something more unusual for a typical economy. It's about an open model that changes the system of interest rates for instruments and infrastructure. Therefore, without any coercion or push for integration, these institutions and the economic entity made their choice. We simply document this history with our own eyes, observe this assessment openly, and draw conclusions.

All sources

Technical sources (from the WNS-README document):

GitHub — WNS README:

https://github.com/wen-community/wen-program-library/blob/main/programs/wen_new_standard/README.md
GitHub — wen-program-library repository (root): https://github.com/wen-community/wen-program-library
Solana.com — Token Extensions (Solutions): https://solana.com/solutions/token-extensions
Solana.com — Token Extensions Docs: https://solana.com/docs/tokens/extensions
Solana.com — Transfer Hook Guide: https://solana.com/vi/developers/guides/token-extensions/transfer-hook
Solana StackExchange — Transfer Hook in Token-2022:
https://solana.stackexchange.com/questions/7360/transfer-hook-in-token-2022-how-to-actually-transfer

News sources on the Shinhan case:

7. The Block: https://www.theblock.co/news/regulation/2026-08-21-south-korea-shinhan-partners-solana-412420
8. TechTimes: https://www.techtimes.com/articles/325238/20260821/shinhan-bets-korean-won-solana-tokenized-fund-targets-dollar-dominated-rwa-market.htm
9. CryptoTimes: https://www.cryptotimes.io/2026/08/21/shinhan-taps-solana-to-test-korean-won-tokenized-fund/
10. BigGo Finance: https://finance.biggo.com/news/4991f187-02e5-49b5-8d85-b6728d1f43f4

Regulatory Sources:

11. SEC EDGAR — Bitwise Solana Staking ETF: https://www.sec.gov/Archives/edgar/data/0002045872/000119312526349138/bsol-20260813.htm
12. SEC EDGAR — Grayscale Sui Staking ETF: https://www.sec.gov/Archives/edgar/data/2034012/000203401226000009/424b3_gsui_12312025.htm

WEN/WNS Historical Sources:

13. DL News: https://www.dlnews.com/articles/defi/jupiter-wen-airdrop-introduces-new-solana-nft-standard/
14. Gate Learn: https://www.gate.com/learn/articles/what-is-wen-all-you-need-to-know-about-wen/5046
15. CoinMarketCap AI: https://coinmarketcap.com/cmc-ai/wen/what-is/
16. wen-crypto.com: https://wen-crypto.com/

Connection: from the emergence of the WNS/Token-2022 standard to real-world practice in South Korea

[Token-2022, program released by Solana Labs]
| (native Transfer Hook, protocol-level KYC/AML primitives,
| 5 independent audits: Halborn, Zellic, NCC, Trail of Bits, OtterSec)
v
[WNS — Wen New Standard, built by the Wen Community]
| (application layer on top of Token-2022: Group/Member PDAs,
| enforced royalties via wen_royalty_distribution)
v
[$WEN, January 2024 — real-world load stress test]
| (1 trillion tokens, airdrop to 1M+ wallets,
| first proof that Transfer Hook holds under mass load)
v
[Institutional interest in Token-2022 as compliance infrastructure]
| (PYUSD, USDG, EURC and other regulated stablecoins
| are already built on Token-2022, per independent market reviews)
v
[South Korea, August 2026 — Shinhan Asset Management]
| (non-binding PoC on Solana using the same
| Token-2022 compliance primitives: KYC/AML, issuance, on-chain liquidity)
|
+– **IN PARALLEL**: an identical PoC with Plume (Aug 14, 2026) –> hedged infrastructure choice
v
[Korea's STO law expected to take effect, February 2027]

token2022-korea-flow-wns-wen-sol

Token-2022 – WNS by Wen community – SPL by Solana – roadmap 2022-2024-2026

The same principle applies here, adding the collection of sources and compiling a data pie for historical notes. That's how the world works; it doesn't stand still, and the speed of movement is only a small part of our understanding. An uncompromising approach to the creation of open data, its assessment, and repeated assessments, with a strict focus on standardizing objectivity. After all, this isn't about causal laurels and victories, or the commercial component of Solan, but rather something a bit more unusual for a typical economy. It's about an open model that changes the system of interest rates for instruments, for infrastructure. Therefore, without any coercion or push for integration, the aforementioned institutions and economic entities have made their choice. We simply record this history with our own eyes, examine this assessment openly, and draw conclusions. And you decide for yourself whether this benefits or harms.

More about https://wen-crypto.com/links-wen-cat/

Disclaimer and Data Transparency Statement: Important Notice: This analytical note is compiled solely based on verified historical data, official government announcements, and technical documentation of blockchain protocols. The content of the material is completely free of elements of fiction, speculative theories, neural network imagination, or hypothetical scenarios. The information has been collected, structured, and analyzed as a strict record of real historical and corporate events. Regulatory and Government Track: The dates for the testing phases of the wholesale CBDC (Project Hangang) and the integration of tokenized bonds correspond to official releases from the Bank of Korea (BOK) and the Ministry of Economy and Finance (MOEF). The legislative amendments are implemented in strict compliance with the regulations of the South Korean Financial Services Commission (FSC). Institutional Track: The conclusion of the Quadripartite Agreement (MOU) on August 21, 2026, between Shinhan Asset Management, Solana Foundation, Etherfuse, and Orca is a confirmed corporate event in the real asset market (RWA). Technical Track: The architectural features of the Token-2022 extensions (Transfer Hooks, Permanent Delegate) and the launch parameters of the WNS standard of the WEN memcoin (as a network stress testing tool) are presented based on the open source code of the Solana Labs repositories and the Jupiter ecosystem documentation. This post is created for informational and research purposes only. This material does not constitute investment, financial, legal, or professional advice. Each reader and specialist in the field independently evaluates the degree of practical value, applicability, and fundamental usefulness of the presented analysis, based on their own criteria and an independent audit of the primary sources.