AutoRenewSubscriptionLicense Contract Documentation

The AutoRenewSubscriptionLicense contract is an Ethereum smart contract written in Solidity that manages software licenses as ERC721 tokens. The contract uses the OpenZeppelin library for secure and standardized contract development.

Contract Overview

The contract manages software licenses as non-fungible tokens (NFTs), where each token represents a unique software license. The contract allows for the purchase, minting, updating, and reactivation of these licenses. The licenses are auto-renewable, and the contract also provides functionality for checking the status of a license, transferring licenses, and withdrawing funds.

Contract Details

State Variables

The contract maintains several state variables to manage the licenses:

- **s_tokenCounter**: A counter for the total number of tokens minted.
- **s_licensePrice**: The price of a license.
- **s_gasPrice**: The gas price for transactions.
- **i_companyName**: The name of the company issuing the licenses.
- **i_licenseType**: The type of license (set to "Subscription").
- **i_licenseName**: The name of the license.
- **i_tokenAddress**: The address of the ERC20 token used for payments.
- **i_periodSecond**: The duration of the subscription in seconds.
- **i_licenseAgreementUrl**: The URL of the license agreement.

Constructor

The constructor initializes the contract with the company name, license name, license agreement URL, token address, license price, subscription period, and gas price.

Functions

The contract includes several functions:

- **buyToken()**: Allows a user to purchase a license. The function checks that the user has enough balance and allowance, mints a new token, and transfers the license price from the user to the contract.

- **mintToken(address customer)**: Allows the contract owner to mint a new token and assign it to a customer.
- **updateSubscription(uint256 tokenId)**: Allows a user to update their subscription. The function checks that the subscription is not active and that the user has enough balance and allowance, then extends the subscription period.
- **reactivateSubscription(uint256 tokenId)**: Allows a user to reactivate their subscription. The function checks that the subscription is not active and that the user has enough balance and allowance, then sets a new expiration timestamp.
- **tokenURI(uint256 tokenId)**: Returns a URI for a given token ID, which points to a JSON file with the token's metadata.
- **getTokenCounter()**, **getLicensePrice()**, **getGasPrice()**, **getSubscritptionTimePeriod()**, **getExpirationTime(uint256 tokenId)**, **isSubscriptionActive(uint256 tokenId)**, **isTransferAllowed(uint256 tokenId)**: Getter functions that return the values of the corresponding state variables.
- **withdraw()**: Allows the contract owner to withdraw the balance of the contract.
- **transferFrom(address from, address to, uint256 tokenId)**, **safeTransferFrom(address from, address to, uint256 tokenId)**, **safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data)**: Functions that override the corresponding ERC721 functions to implement custom transfer logic.
- **updateLicensePrice(uint256 newPrice)**: Allows the contract owner to update the license price.
- **allowTransfer(uint256 tokenId), restrictTransfer(uint256 tokenId)**: Functions that allow the contract owner to enable or disable transfers for a specific token.
- **cancelSubscription(uint256 tokenId)**: Allows a user to cancel their subscription.

Events

The contract emits two events:

- **UpdatedSubscriptionToken(uint256 indexed tokenId, uint256 licensePrice)**: This event is triggered when a subscription token is updated. It includes the token ID and the license price.
- **NewSubscriptionToken(uint256 indexed tokenId, uint256 licensePrice)**: This event is emitted when a new subscription token is minted. It provides detailed information about the creation of the new subscription token, including the token ID and the price of the license associated with it.

Inherited Functionality

The **AutoRenewSubscriptionLicense** contract inherits from the **ERC721** and **Ownable** contracts from the OpenZeppelin library.

- **ERC721**: This is a standard for non-fungible tokens (NFTs) on Ethereum, meaning tokens that are unique and not interchangeable. The **ERC721** standard provides basic functionality to track and transfer NFTs. The **AutoRenewSubscriptionLicense** contract uses this standard to represent software licenses as NFTs.

- **Ownable**: This is a contract module that sets an **owner** state variable and provides modifier **onlyOwner** which can be applied to functions to restrict their use to the owner. The **AutoRenewSubscriptionLicense** contract uses this module to restrict certain functions (like minting new tokens) to the owner of the contract.

Interactions with ERC20 Tokens

The **AutoRenewSubscriptionLicense** contract interacts with an ERC20 token contract for payments. The address of the ERC20 token contract is passed to the **AutoRenewSubscriptionLicense** contract during deployment. The **IERC20** interface from the OpenZeppelin library is used to interact with the ERC20 token contract. The **SafeERC20** library is used to safely call ERC20 functions.

Users can buy licenses by sending ERC20 tokens to the **AutoRenewSubscriptionLicense** contract. The contract checks the user's ERC20 balance and allowance before accepting the payment. The contract owner can withdraw the accumulated ERC20 tokens from the contract.