Caching contracts with Stylus

Stylus is designed for fast computation and efficiency. However, the initialization process when entering a contract can be resource-intensive and time-consuming.

This initialization process, if repeated frequently, may lead to inefficiencies. To address this, we have implemented a caching strategy. By storing frequently accessed contracts in memory, we can avoid repeated initializations. This approach saves resources and time, significantly enhancing the speed and efficiency of contract execution.

Note that Stylus smart contracts will need to be re-activated once per year (365 days) or whenever a upgrade to Stylus (which will always involve an ArbOS upgrade), even if they are in the cache. This re-activation can be done usingcargo-stylus , a cargo subcommand for building, verifying, and deploying Arbitrum Stylus WASM contracts in Rust.

CacheManager contract

The core component of our caching strategy is theCacheManager contract . This smart contract manages the cache, interacts with precompiles, and determines which contracts should be cached. TheCacheManager can hold approximately 4,000 contracts in memory.

TheCacheManager defines how contracts remain in the cache and how they compete with other contracts for cache space. Its primary purpose is to reduce high initialization costs, ensuring efficient contract activation and usage. The contract includes methods for adding and removing cache entries, querying the status of cached contracts, and managing the lifecycle of cached data.

Key features

TheCacheManager plays a crucial role in our caching strategy by keeping a specific set of contracts in memory rather than retrieving them from disk. This significantly reduces the activation time for frequently accessed contracts. TheCacheManager contract is an on-chain contract that accepts bids for inserting contract code into the cache. It then calls a precompile that loads or unloads the contracts in theArbOS cache, which follows the on-chain cache but operates locally in the client and marks the contract as in or out of the cache in theArbOS state.

The cache operates through an auction system where dApp developers submit bids to insert their contracts into the cache. If the cache is at capacity, lower bids are evicted to make space for higher bids. The cache maintains a minimum heap of bids forcodeHashes , with bids encoded asbid << 64 + index , whereindex represents the position in the list of all bids. When an insertion exceeds the cache's maximum size, items are popped off the minimum heap and deleted until there is enough space to insert the new item. Contracts with equal bids will be popped in a random order, while the smallest bid is evicted first.

To ensure that developers periodically pay to maintain their position in the cache, we use a global decay parameter computed bydecay = block.timestamp * _decay . This inflates the value of bids over time, making newer bids more valuable.

Cache access and costs

During activation, we compute the contract's initialization costs for both non-cached and cached initialization. These costs take into account factors such as the number of functions, types, code size, data length, and memory usage. It's important to note that accessing an uncached contract does not automatically add it to theCacheManager 's cache. Only explicit calls to theCacheManager contract will add a contract to the cache. If a contract is removed from the cache, calling the contract becomes more expensive unless it is re-added.

To see how much gas contract initialization would cost, you need to callprogramInitGas(address) from theArbWasm precompile . This function returns both the initialization cost when the contract is cached and when it is not.

How to use the CacheManager API

This section provides a practical guide for interacting with theCacheManager contract API, either directly or through thecargo stylus command-line tool.

Step 1: Determine the minimum bid

Before placing a bid, it's important to know the minimum bid required to cache the Stylys contract. This can be done using thegetMinBid function, or using thecargo stylus cache suggest-bid command.

Method 1: Direct smart contract call

uint192 minBid = cacheManager . getMinBid (contractAddress) ; Method 2: Cargo stylus command

Note that here, [contractAddress] is the address of the Stylus contract you want to cache.

cargo stylus cache suggest-bid [contractAddress]

Step 2: Place a bid

You can place a bid using either of the following methods:

Method 1: Direct smart contract call

Here,bidAmount is the amount you want to bid, andcontractAddress is the address of the Stylus contract you're bidding for.

cacheManager . placeBid { value : bidAmount } (contractAddress) ; Method 2: Cargo stylus command

You can place a bid using thecargo stylus cache bid command:

cargo stylus cache bid < --private-key-path < PRIVATE_KEY_PATH

| --private-key < PRIVATE_KEY | --keystore-path < KEYSTORE_PATH

[contractAddress]

[bidAmount] * [contractAddress] * : The address of the Stylus contract you want to cache. * [bidAmount] * : The amount you want to bid. If not specified, the default bid is 0.

If you specify a bid amount usingcargo stylus , it will automatically validate that the bid is greater than or equal to the result of thegetMinBid function. If the bid is insufficient, the command will fail, ensuring that only valid bids are placed.

Step 3: Check cache status

To check if a specific address is cached, you can use thecargo stylus status command:

cargo stylus cache status --address = [contractAddress]

Additional information

- Pausing Bids: TheCacheManager
- contract has anisPaused
- state that can be toggled by the owner to prevent or allow new bids.
- Checking Cache Size: You can monitor the current cache size and decay rate using thegetCacheSize
- andgetDecayRate
- functions respectively.

By following these steps, you can effectively interact with theCacheManager contract, either directly through smart contract calls or using thecargo stylus command-line tool. This ensures that your bids meet the necessary requirements for caching programs on the network, optimizing your contracts for faster and more efficient execution. Edit this page Last updatedonJan 27, 2025 Previous Verify contracts Next Verify on Arbiscan