Smart Contract Overview

This guide provides an overview of smart contract functionality for the smart contract components. You can also find contract addresses on OP Mainnet.

L1 contracts

DelayedVetoable(opens in a new tab)

This contract enables a delay before a call is forwarded to a target contract, and during the delay period the call can be vetoed by the authorized vetoer. This contract does not support value transfers, only data is forwarded. Additionally, this contract cannot be used to forward calls with data beginning with the function selector of the queuedAt(bytes32) function. This is because of input validation checks which solidity performs at runtime on functions which take an argument.

L1CrossDomainMessenger(opens in a new tab)

The L1CrossDomainMessenger is a message passing interface between L1 and L2 responsible for sending and receiving data on the L1 side. Users are encouraged to use this interface instead of interacting with lower-level contracts directly.

L1ERC721Bridge(opens in a new tab)

TheL1ERC721bridge is a contract which works together with the L2 ERC721 bridge to make it possible to transfer ERC721 tokens from Ethereum to Optimism. This contract acts as an escrow for ERC721 tokens deposited into L2.

L1StandardBridge(opens in a new tab)

⚠ This contract is not intended to support all variations of ERC20 tokens. Examples of some token types that may not be properly supported by this contract include, but are not limited to: tokens with transfer fees, rebasing tokens, and tokens with blocklists. TheL1StandardBridge is responsible for transferring ETH and ERC20 tokens between L1 and L2. In the case that an ERC20 token is native to L1, it will be escrowed within this contract. If the ERC20 token is native to L2, it will be burnt. Before Bedrock, ETH was stored within this contract. After Bedrock, ETH is instead stored inside the OptimismPortal contract.

L2OutputOracle(opens in a new tab)

TheL2OutputOracle contains an array of L2 state outputs, where each output is a commitment to the state of the L2 chain. Other contracts like theOptimismPortal use these outputs to verify information about the state of L2.

OptimismPortal(opens in a new tab)

TheOptimismPortal is a low-level contract responsible for passing messages between L1 and L2. Messages sent directly to theOptimismPortal have no form of replayability. Users are encouraged to use theL1CrossDomainMessenger for a higher-level interface.

ProtocolVersions(opens in a new tab)

TheProtocolVersions contract is used to manage superchain protocol version information.

ResourceMetering(opens in a new tab)

ResourceMetering implements an EIP-1559 style resource metering system where pricing updates automatically based on current demand.

SuperchainConfig(opens in a new tab)

TheSuperchainConfig contract is used to manage configuration of global superchain values.

SystemConfig(opens in a new tab)

TheSystemConfig contract is used to manage configuration of an OP Stack network. All configuration is stored on L1 and picked up by L2 as part of the derivation of the L2 chain.

L2 contracts (predeploys)

BaseFeeVault(opens in a new tab)

TheBaseFeeVault accumulates the base fee that is paid by transactions.

GasPriceOracle(opens in a new tab)

This contract maintains the variables responsible for computing the L1 portion of the total fee charged on L2. Before Bedrock, this contract held variables in state that were read during the state transition function to compute the L1 portion of the transaction fee. After Bedrock, this contract now simply proxies the L1Block contract, which has the values used to compute the L1 portion of the fee in its state.

The contract exposes an API that is useful for knowing how large the L1 portion of the transaction fee will be. The following events were deprecated with Bedrock:

- event OverheadUpdated(uint256 overhead);
- event ScalarUpdated(uint256 scalar);
- event DecimalsUpdated(uint256 decimals);

L1Block(opens in a new tab)

TheL1Block predeploy gives users access to information about the last known L1 block. Values within this contract are updated once per epoch (every L1 block) and can only be set by the "depositor" account, a special system address. Depositor account transactions are created by the protocol whenever we move to a new epoch.

L1FeeVault(opens in a new tab)

TheL1FeeVault accumulates the L1 portion of the transaction fees.

L2CrossDomainMessenger(opens in a new tab)

TheL2CrossDomainMessenger is a high-level interface for message passing between L1 and L2 on the L2 side. Users are generally encouraged to use this contract instead of lower level message passing contracts.

L2ERC721Bridge(opens in a new tab)

⚠ Do not bridge an ERC721 that was originally deployed on Optimism. This bridge ONLY supports ERC721s originally deployed on Ethereum. Users will need to wait for the one-week challenge period to elapse before their Optimism-native NFT can be refunded on L2. TheL2ERC721Bridge is a contract which works together with the L1 ERC721 bridge to make it possible to transfer ERC721 tokens from Ethereum to Optimism. This contract acts as a minter for new tokens when it hears about deposits into the L1 ERC721 bridge. This contract also acts as a burner for tokens being withdrawn.

L2StandardBridge(opens in a new tab)

⚠ This contract is not intended to support all variations of ERC20 tokens. Examples of some token types that may not be properly supported by this contract include, but are not limited to: tokens with transfer fees, rebasing tokens, and tokens with blocklists. TheL2StandardBridge is responsible for transferring ETH and ERC20 tokens between L1 and L2. In the case that an ERC20 token is native to L2, it will be escrowed within this contract. If the ERC20 token is native to L1, it will be burnt.

L2ToL1MessagePasser(opens in a new tab)

There is a legacy contract under this same name. TheL2ToL1MessagePasser is a dedicated contract where messages that are being sent from L2 to L1 can be stored. The storage root of this contract is pulled up to the top level of the L2 output to reduce the cost of proving the existence of sent messages.

SequencerFeeVault(opens in a new tab)

The SequencerFeeVault is the contract that holds any fees paid to the Sequencer during transaction processing and block production.

Legacy Contracts

Those are legacy contracts from the old version of the OP Stack.

AddressManager(opens in a new tab)

AddressManager is a legacy contract that was used in the old version of the Optimism system to manage a registry of string names to addresses. We now use a more standard proxy system instead, but this contract is still necessary for backwards compatibility with several older contracts.

DeployerWhitelist(opens in a new tab)

DeployerWhitelist is a legacy contract that was originally used to act as a whitelist of addresses allowed to the Optimism network. TheDeployerWhitelist has since been disabled, but the code is kept in state for the sake of full backwards compatibility. As of the Bedrock upgrade, theDeployerWhitelist is completely unused by the Optimism system and could, in theory, be removed entirely.

L1BlockNumber(opens in a new tab)

L1BlockNumber is a legacy contract that fills the roll of theOVM_L1BlockNumber contract in the old version of the Optimism system. Only necessary for backwards compatibility. If you want to access the L1 block number going forward, you should use theL1Block contract instead.

L1ChugSplashProxy(opens in a new tab)

Basic ChugSplash proxy contract for L1. Very close to being a normal proxy but has added functionssetCode andsetStorage for changing the code or storage of the contract. Note for future developers: do NOT make anything in this contract 'public' unless you know what you're doing. Anything public can potentially have a function signature that conflicts with a signature attached to the implementation contract. Public functions SHOULD always have theproxyCallIfNotOwner modifier unless there's somereally good reason not to have that modifier. And there almost certainly is not a good reason to not have that modifier. Beware!

LegacyERC20ETH(opens in a new tab)

LegacyERC20ETH is a legacy contract that held ETH balances before the Bedrock upgrade. All ETH balances held within this contract were migrated to the state trie as part of the Bedrock upgrade. Functions within this contract that mutate state were already disabled as part of the EVM equivalence upgrade.

LegacyMessagePasser(opens in a new tab)

TheLegacyMessagePasser was the low-level mechanism used to send messages from L2 to L1 before the Bedrock upgrade. It is now deprecated in favor of the newMessagePasser .

LegacyMintableERC20(opens in a new tab)

The legacy implementation of theOptimismMintableERC20 . This contract is deprecated and should no longer be used.

ResolvedDelegateProxy(opens in a new tab)

ResolvedDelegateProxy is a legacy proxy contract that makes use of theAddressManager to resolve the implementation address. We're maintaining this contract for backwards compatibility so we can manage all legacy proxies where necessary.

Releases

The full smart contract release process is documented in themonorepo(opens in a new tab) . All production releases are always tags, versioned as/v and contract releases you'll see them tagged asop-contract/vX.X.X .

⚠ For contract releases, refer to the GitHub release notes for a given release, which will list the specific contracts being released—not all contracts are considered production ready within a release , and many are under active development. These release pages are linked below.

Tags of the formv , such asv1.1.4 , indicate releases of all Go code only, andDO NOT include smart contracts.

v1.0.0 - Bedrock

The Bedrock protocol upgrade was designed to minimize the amount of code in the OP Stack, pushed it as close as possible to Ethereum-Equivalence, and most importantly making the stack modular.

- Bedrock release notes(opens in a new tab)
- ...
- Governance post(opens in a new tab)

v1.1.0 - ProtocolVersions

The Protocol Version documents the progression of the total set of canonical OP Stack specifications. Components of the OP Stack implement the subset of their respective protocol component domain, up to a given Protocol Version of the OP Stack.

The Protocol Version is NOT a hardfork identifier, but rather indicates software-support for a well-defined set of features introduced in past and future hardforks, not the activation of said hardforks.

The Protocol Version only applies to the Protocol specifications with theSuperchain Targets(opens in a new tab) specified within. This versioning is independent of the Semver versioning used in OP Stack smart contracts, and the Semver-versioned reference software of the OP-Stack. This is an optional feature. * ProtocolVersions release notes(opens in a new tab)

v1.2.0 - SuperchainConfig with Extended Pause Functionality

The SuperchainConfig contract is used to manage global configuration values for multiple OP Chains within a single Superchain network.

- SuperchainConfig and Extended Pause release notes(opens in a new tab)
- Governance post(opens in a new tab)

v1.3.0 - Multi-Chain Prep (MCP)

This protocol upgrade strengthens the security and upgradeability of the Superchain by enabling L1 contracts to be upgraded atomically across multiple chains in a single transaction. This upgrade also extends theSystemConfig to contain the addresses of the contracts in the network, allowing users to discover the system's contract addresses programmatically.

- MCP release notes(opens in a new tab)
- Governance post(opens in a new tab)

Rollup Overview Deposit Flow