How to customize ArbOS on your Orbit chain

PUBLIC PREVIEW, MAINNET READY Orbit chains are nowMainnet ready ! Note that Orbit is still apublic preview capability - the Orbit product and its supporting documentation may change significantly as we capture feedback from readers like you.

To provide feedback, click theRequest an update button at the top of this documentjoin the Arbitrum Discord , or reach out to our team directly by completingthis form . caution

Customizations require expertise

Customizing your chain is a core benefit of building with Arbitrum Orbit. We strongly recommend that teams interested in customizations work alongside a partner with ArbOS and Nitro software expertise, such as a Rollup-as-a-Service team.

Working alongside an experienced Orbit operator can help your team navigate the complex tradeoff space of rollup customizations, which can include performance, security, and cost considerations. Offchain Labs is positioned to train and enable Rollup-as-a-Service in their work with clients to scale support to the Orbit ecosystem as a whole. As such, Offchain Labs does not necessarily have the capacity to review code changes made by individual Orbit chains.

We encourage you to leverage your in-house expertise, collaborate with expert partners, and allocate appropriate resources for both an initial implementation (including an audit) and ongoing maintenance and security management of your customization.

Cases where you may want to consider customizing your own ArbOS upgrade

1. When you want to make changes to your Nitro code that affect the State Transition Function, or STF (you may refer to theCustomize STF docs
2.), and
3. If your desired changes need to be made to a live and operational Orbit chain

If your changes meet both those 2 points, then you will need a custom ArbOS upgrade.

Also, if you made changes to a live and operational chain and want to upgrade them later in the future, then you will likely need an ArbOS upgrade to facilitate the upgrade.

Where should I insert ArbOS Upgrade related code?

Below, you will find 4 examples of ArbOS-related code changes and, generally, how to make them:

1. Add a new method to exsiting precompile on a specific ArbOS version:

After you addsayHi() toArbSys.go according to the guide incustomize precompile option 1 , you need continue to modifyprecompile.go .

For example, the original code is:

ArbSys :=

insert (MakePrecompile (pgen . ArbSysMetaData ,

& ArbSys { Address : types . ArbSysAddress })) arbos . ArbSysAddress = ArbSys . address arbos . L2ToL1TransactionEventID = ArbSys . events ["L2ToL1Transaction"] . template . ID arbos . L2ToL1TxEventID = ArbSys . events ["L2ToL1Tx"] . template . ID You need to append the following code to it:

ArbSys :=

insert (MakePrecompile (pgen . ArbSysMetaData ,

& ArbSys { Address : types . ArbSysAddress })) arbos . ArbSysAddress = ArbSys . address arbos . L2ToL1TransactionEventID = ArbSys . events ["L2ToL1Transaction"] . template . ID arbos . L2ToL1TxEventID = ArbSys . events ["L2ToL1Tx"] . template . ID // The arbos version control logic ArbOwner . methodsByName ["SayHi"] . arbosVersion =

{ The arbos version you want to activate this method } In this way, this method will be executed normally and return results only after you update ArbOS to the target version.

2. Create a new precompile contract on a specific ArbOS version

After you add a new precompile namedArbHi according to the guide incustomize precompile option 2 and make changes toprecompile.go , you also need to make the following changes:

ArbHi :=

insert (MakePrecompile (pgen . ArbHiMetaData ,

& ArbHi { Address : types . ArbHiAddress }))

// types.ArbHiAddress here is an example address // Set activate version to the precompile ArbHi . arbosVersion =

{ The arbos version you want to activate this precompile } // Set activate version to all method for

_ , method :=

range ArbHi . methods { method . arbosVersion =

{ The arbos version you want to activate this precompile } } In this way, ArbHi and all its methods will be activated after the ArbOS version you set.

3. Create a new ArbOS state on a specific ArbOS version

After you add a new statemyNumber according to the guide incustomize precompile Option 5 , you also need to rewriteUpgradeArbosVersion inarbosstate.go : Add your expected ArbOS version to the switch case statement of nextArbosVersion. Here we will take ArbOS V21 as an example:

ensure :=

func (err error)

{ if err !=

nil

{ message := fmt . Sprintf ("Failed to upgrade ArbOS version %v to version %v: %v" , state . arbosVersion , state . arbosVersion + 1 , err ,) panic (message) } } nextArbosVersion := state . arbosVersion +

1 switch nextArbosVersion { case

1 : //..... case

2 : //.... //..... case

21 : // Set your new ArbOS state value here ensure (state . SetNewMyNumber (

{ random number })) //.... } Here, we will ensure that the initial value of (a random number) after ArbOS is upgraded to V21.

caution It should be noted that when you initialize the state (initial code is incustomize precompile Option 5), you need to initialize it to 0 or null value first to avoid a potential re-org on your chain. Also, please make sure your program cannot call thestate.SetNewMyNumber or other functions may change the value ofmyNumber before ArbOS V21. To prevent this, if you are using an external call to the precompile contract to change the value, you can refer topoint 1 orpoint 2 to set the activation time of the precompile contract method. If your nitro code needs to call this method to change the state, you can continue readingpoint 4 .

4. Any changes in the STF logic that will affect the final execution result

If you change the logic in STF, it will cause the execution result of the transaction to be different, you need to keep the original execution logic and put the new logic into another branch. You can use anif else statement to control it.

For example, we change theSayHi return after upgrading the ArbOS version:

// The method in ArbHi precompile func

(con * ArbHi)

SayHi (c ctx , evm mech)

(string ,

error)

{ if p . state . ArbOSVersion ( )

=

{ The arbos you want to upgrade to }

{ // Your new logic code return

"hi, new ArbOS version" ,

nil }

else

{ // The old logic code needs to be kept return

"hi" ,

nil } } In the above code, we use precompiles as an example. Some logic might also affect the STF, such as the methods in[block_process.go](block_process.go) ,internal_tx.go ,tx_processor.go and so on. The aforementioned ArbOS control methods will be needed for interacting with different versions of the STF logic.

Backward compatibility Wasm module roots are backward compatible, so upgrading them before an ArbOS version upgrade will not disrupt your chain's functionality.

Upgrade WASM Module Root on the parent chain

After you have made your custom ArbOS changes, you will need to update the WASM Module root recorded on the parent chain. This is because an Arbitrum chain may execute the validation and fraud proofs on the parent chain. Please continue reading[customize stf](customize stf) for follow-up operations.

Schedule ArbOS upgrade

After you add ArbOS version control to the Nitro code, you can update ArbOS. You can refer to the documen[ArbOS upgrade](ArbOS upgrade) to upgrade. It is recommended that teams who opt to version control their custom ArbOS version choose an ArbOS version number that builds on top of the canonical releases shipped by Offchain Labs. As an example, if your team is customizing ArbOS 31, we recommend versioning your ArbOS version as ArbOS 32. It should be noted that if you set a higher ArbOS version as the upgrade target, all the features added between the current and target versions will be activated. For example, if your current version is ArbOS v18 and you set the target version to v25, all the features between v18 and v25 will be loaded. Edit this page Last updatedonOct 30, 2024 Previous Manage the fee collectors Next Implement Circle bridged USDC