[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 0 - Getting Started](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Getting Started](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Blockchain 101](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Blockchain History](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Public and Managed Blockchains](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Consensus in Distributed Networks](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Cryptography](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Self-Assessment Quiz](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Go Introduction - First Steps](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Go Basics](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Go Interfaces](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Control Structures in Go](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Arrays and Slices in Go](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Standard Packages in Go](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Concurrency in Go](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Docker Introduction](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[The Interchain Ecosystem](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Getting ATOM and Staking It](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[A Blockchain App Architecture](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Accounts](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Transactions](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Messages](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Modules](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Protobuf](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Multistore and Keepers](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[BaseApp](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Queries](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Events](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Context](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Testing](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Relaying with IBC](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Interchain Security](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Bridges](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Migrations](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 1 Quiz](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 2 - First Steps](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[First Steps](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Setup Your Work Environment](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Run a Node, API, and CLI](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Ignite CLI](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Store Object](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create and Save a Game Properly](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Add a Way to Make a Move](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Emit Game Information](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Record the Game Winner](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 2 Exercise](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[What is IBC?](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC Token Transfer](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC Tooling](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[What is CosmJS?](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Your First CosmJS Actions](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Compose Complex Transactions](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Learn to Integrate Keplr](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Put Your Games in Order](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Auto-Expiring Games](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Let Players Set a Wager](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Handle wager payments](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Integration tests](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Incentivize Players](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Help Find a Correct Move](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Understand IBC Denoms](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Go Relayer](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Hermes Relayer](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create Custom Objects](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Get an External GUI](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Backend Script for Game Indexing](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[IBC Application Developer Introduction](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Create a Leaderboard Chain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Run in Production](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Prepare the Software to Run](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Prepare a Validator and Keys](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Prepare Where the Node Starts](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Prepare and Do Migrations](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Simulate Production in Docker](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Tally Player Info After Production](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Simulate a Migration in Docker](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Final Exam](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[What's Next?](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

[Continue Your Interchain Journey](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html)

Docs Version Switcher

On this page

[Scaffold a leaderboard chain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#scaffold-a-leaderboard-chain)

[IBC application module requirements](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#ibc-application-module-requirements)

[Implementing the IBCModule interface](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#implementing-the-ibcmodule-interface)

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#make-a-module-ibc-enabled) **Make a Module IBC-Enabled**

In this section, you will build a conceptual Cosmos SDK blockchain with one module: first as a regular module, and second as an IBC module. This will introduce you to what makes a module IBC-enabled.

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#scaffold-a-leaderboard-chain) Scaffold a leaderboard chain

By now you should be familiar with scaffolding a chain with Ignite CLI. If not, check out the [Create Your Own Chain](https://ida.interchain.io/hands-on-exercise/1-ignite-cli/) chapter.

To begin, scaffold a leaderboard chain:



Copy

$ ignite scaffold chain github.com/cosmonaut/leaderboard

This creates a chain with x/leaderboard as a regular SDK module.

Next, scaffold another chain (for example in another git branch), but this time add the --no-module flag:



Copy

$ ignite scaffold chain github.com/cosmonaut/leaderboard --no-module

Now add the x/leaderboard module as an IBC module with the --ibc flag:



Copy

$ ignite scaffold module leaderboard --ibc

The output you see on the terminal when the module has finished scaffolding already gives a sense of what has to be implemented to create an IBC module:



Copy

modify app/app.go

modify proto/leaderboard/genesis.proto

create proto/leaderboard/packet.proto

modify testutil/keeper/leaderboard.go

modify x/leaderboard/genesis.go

create x/leaderboard/module\_ibc.go

create x/leaderboard/types/events\_ibc.go

modify x/leaderboard/types/genesis.go

modify x/leaderboard/types/keys.go



The code in this section was scaffolded with Ignite CLI v0.22.1. This version includes ibc-go v3 as a dependency, which has [reached past end-of-life (opens new window)↗](https://github.com/cosmos/ibc-go/blob/143e5d85e9d2fc5fc2fc53adc42127a439ee2b79/RELEASES.md#stable-release-policy) and is no longer actively supported.   
  
**It is thus highly discouraged to deploy any code in production using ibc-go code scaffolded by Ignite CLI v0.22.1.**   
  
<!--- TODO: add link to section on upgrading ibc-go version --->

[All actively supported versions of ibc-go (opens new window)↗](https://github.com/cosmos/ibc-go/blob/main/RELEASES.md#stable-release-policy) have reached past v3, so there may be some differences compared to the code in this section. For documentation on the latest version of ibc-go, please refer to the [ibc-go docs (opens new window)↗](https://ibc.cosmos.network/main/ibc/apps/apps.html).   
  
For example, channel callbacks from v4 onwards now return a *version* string next to an error:



Copy

func (im IBCModule) OnChanOpenInit(ctx sdk.Context,

order channeltypes.Order,

connectionHops []string,

portID string,

channelID string,

channelCap \*capabilitytypes.Capability,

counterparty channeltypes.Counterparty,

version string,

) (string, error) {

...

return version, nil

}

For a more detailed view, you can now compare both versions with a git diff.



To make use of git diffs to check the changes, be sure to commit between different (scaffolding) actions.



Copy

$ git diff <commit\_hash\_1> <commit\_hash\_2>

You can use git or GitHub to visualize the git diffs or alternatively use [diffy.org (opens new window)↗](https://diffy.org/).

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#ibc-application-module-requirements) IBC application module requirements

What does Ignite CLI do behind the scenes when creating an IBC module for you? What do you need to implement if you want to upgrade a regular custom application module to an IBC-enabled module?

The required steps to implement can be found in the [ibc-go docs (opens new window)↗](https://ibc.cosmos.network/main/ibc/apps/apps.html). There you will find:



**To have your module interact over IBC you must:**

* Implement the IBCModule interface:
  + Channel (opening) handshake callbacks
  + Channel closing handshake callbacks
  + Packet callbacks
* Bind to a port(s).
* Add keeper methods.
* Define your packet data and acknowledgement structs as well as how to encode/decode them.
* Add a route to the IBC router.

Now take a look at the git diff and see if you can recognize the steps listed above.

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#implementing-the-ibcmodule-interface) Implementing the IBCModule interface



For a full explanation, visit the [ibc-go docs (opens new window)↗](https://ibc.cosmos.network/main/ibc/apps/ibcmodule.html).

The Cosmos SDK expects all IBC modules to implement the [IBCModule interface (opens new window)↗](https://github.com/cosmos/ibc-go/tree/main/modules/core/05-port/types/module.go). This interface contains all of the callbacks IBC expects modules to implement. This includes callbacks related to:

* Channel handshake (OnChanOpenInit, OnChanOpenTry, OnChanOpenAck, and OnChanOpenConfirm)
* Channel closing (OnChanCloseInit and OnChanCloseConfirm)
* Packets (OnRecvPacket, OnAcknowledgementPacket, and OnTimeoutPacket).

Ignite CLI implements this in the file x/leaderboard/module\_ibc.go.

****

**x/leaderboard/module\_ibc.go**

Copy

// OnChanOpenInit implements the IBCModule interface

func (am AppModule) OnChanOpenInit(

ctx sdk.Context,

order channeltypes.Order,

connectionHops []string,

portID string,

channelID string,

chanCap \*capabilitytypes.Capability,

counterparty channeltypes.Counterparty,

version string,

) error {

// Require portID is the portID module is bound to

boundPort := am.keeper.GetPort(ctx)

if boundPort != portID {

return sdkerrors.Wrapf(porttypes.ErrInvalidPort, "invalid port: %s, expected %s", portID, boundPort)

}

if version != types.Version {

return sdkerrors.Wrapf(types.ErrInvalidVersion, "got %s, expected %s", version, types.Version)

}

// Claim channel capability passed back by IBC module

if err := am.keeper.ClaimCapability(ctx, chanCap, host.ChannelCapabilityPath(portID, channelID)); err != nil {

return err

}

return nil

}

// OnChanOpenTry implements the IBCModule interface

func (am AppModule) OnChanOpenTry(

ctx sdk.Context,

order channeltypes.Order,

connectionHops []string,

portID,

channelID string,

chanCap \*capabilitytypes.Capability,

counterparty channeltypes.Counterparty,

counterpartyVersion string,

) (string, error) {

// Require portID is the portID module is bound to

boundPort := am.keeper.GetPort(ctx)

if boundPort != portID {

return "", sdkerrors.Wrapf(porttypes.ErrInvalidPort, "invalid port: %s, expected %s", portID, boundPort)

}

if counterpartyVersion != types.Version {

return "", sdkerrors.Wrapf(types.ErrInvalidVersion, "invalid counterparty version: got: %s, expected %s", counterpartyVersion, types.Version)

}

// Module may have already claimed capability in OnChanOpenInit in the case of crossing hellos

// (ie chainA and chainB both call ChanOpenInit before one of them calls ChanOpenTry)

// If the module can already authenticate the capability then the module already owns it so we don't need to claim

// Otherwise, the module does not have channel capability and we must claim it from IBC

if !am.keeper.AuthenticateCapability(ctx, chanCap, host.ChannelCapabilityPath(portID, channelID)) {

// Only claim channel capability passed back by IBC module if we do not already own it

if err := am.keeper.ClaimCapability(ctx, chanCap, host.ChannelCapabilityPath(portID, channelID)); err != nil {

return "", err

}

}

return types.Version, nil

}

// OnChanOpenAck implements the IBCModule interface

func (am AppModule) OnChanOpenAck(

ctx sdk.Context,

portID,

channelID string,

\_,

counterpartyVersion string,

) error {

if counterpartyVersion != types.Version {

return sdkerrors.Wrapf(types.ErrInvalidVersion, "invalid counterparty version: %s, expected %s", counterpartyVersion, types.Version)

}

return nil

}

// OnChanOpenConfirm implements the IBCModule interface

func (am AppModule) OnChanOpenConfirm(

ctx sdk.Context,

portID,

channelID string,

) error {

return nil

}

// OnChanCloseInit implements the IBCModule interface

func (am AppModule) OnChanCloseInit(

ctx sdk.Context,

portID,

channelID string,

) error {

// Disallow user-initiated channel closing for channels

return sdkerrors.Wrap(sdkerrors.ErrInvalidRequest, "user cannot close channel")

}

// OnChanCloseConfirm implements the IBCModule interface

func (am AppModule) OnChanCloseConfirm(

ctx sdk.Context,

portID,

channelID string,

) error {

return nil

}

// OnRecvPacket implements the IBCModule interface

func (am AppModule) OnRecvPacket(

ctx sdk.Context,

modulePacket channeltypes.Packet,

relayer sdk.AccAddress,

) ibcexported.Acknowledgement {

var ack channeltypes.Acknowledgement

// this line is used by starport scaffolding # oracle/packet/module/recv

var modulePacketData types.LeaderboardPacketData

if err := modulePacketData.Unmarshal(modulePacket.GetData()); err != nil {

return channeltypes.NewErrorAcknowledgement(sdkerrors.Wrapf(sdkerrors.ErrUnknownRequest, "cannot unmarshal packet data: %s", err.Error()).Error())

}

// Dispatch packet

switch packet := modulePacketData.Packet.(type) {

// this line is used by starport scaffolding # ibc/packet/module/recv

default:

errMsg := fmt.Sprintf("unrecognized %s packet type: %T", types.ModuleName, packet)

return channeltypes.NewErrorAcknowledgement(errMsg)

}

// NOTE: acknowledgement will be written synchronously during IBC handler execution.

return ack

}

// OnAcknowledgementPacket implements the IBCModule interface

func (am AppModule) OnAcknowledgementPacket(

ctx sdk.Context,

modulePacket channeltypes.Packet,

acknowledgement []byte,

relayer sdk.AccAddress,

) error {

var ack channeltypes.Acknowledgement

if err := types.ModuleCdc.UnmarshalJSON(acknowledgement, &ack); err != nil {

return sdkerrors.Wrapf(sdkerrors.ErrUnknownRequest, "cannot unmarshal packet acknowledgement: %v", err)

}

// this line is used by starport scaffolding # oracle/packet/module/ack

var modulePacketData types.LeaderboardPacketData

if err := modulePacketData.Unmarshal(modulePacket.GetData()); err != nil {

return sdkerrors.Wrapf(sdkerrors.ErrUnknownRequest, "cannot unmarshal packet data: %s", err.Error())

}

var eventType string

// Dispatch packet

switch packet := modulePacketData.Packet.(type) {

// this line is used by starport scaffolding # ibc/packet/module/ack

default:

errMsg := fmt.Sprintf("unrecognized %s packet type: %T", types.ModuleName, packet)

return sdkerrors.Wrap(sdkerrors.ErrUnknownRequest, errMsg)

}

ctx.EventManager().EmitEvent(

sdk.NewEvent(

eventType,

sdk.NewAttribute(sdk.AttributeKeyModule, types.ModuleName),

sdk.NewAttribute(types.AttributeKeyAck, fmt.Sprintf("%v", ack)),

),

)

switch resp := ack.Response.(type) {

case \*channeltypes.Acknowledgement\_Result:

ctx.EventManager().EmitEvent(

sdk.NewEvent(

eventType,

sdk.NewAttribute(types.AttributeKeyAckSuccess, string(resp.Result)),

),

)

case \*channeltypes.Acknowledgement\_Error:

ctx.EventManager().EmitEvent(

sdk.NewEvent(

eventType,

sdk.NewAttribute(types.AttributeKeyAckError, resp.Error),

),

)

}

return nil

}

// OnTimeoutPacket implements the IBCModule interface

func (am AppModule) OnTimeoutPacket(

ctx sdk.Context,

modulePacket channeltypes.Packet,

relayer sdk.AccAddress,

) error {

var modulePacketData types.LeaderboardPacketData

if err := modulePacketData.Unmarshal(modulePacket.GetData()); err != nil {

return sdkerrors.Wrapf(sdkerrors.ErrUnknownRequest, "cannot unmarshal packet data: %s", err.Error())

}

// Dispatch packet

switch packet := modulePacketData.Packet.(type) {

// this line is used by starport scaffolding # ibc/packet/module/timeout

default:

errMsg := fmt.Sprintf("unrecognized %s packet type: %T", types.ModuleName, packet)

return sdkerrors.Wrap(sdkerrors.ErrUnknownRequest, errMsg)

}

return nil

}

Additionally, in the module.go file, the following line (and the corresponding import) will be added:



Copy

var (

\_ module.AppModule = AppModule{}

\_ module.AppModuleBasic = AppModuleBasic{}

// this line is used by starport scaffolding # ibc/module/interface

+ \_ porttypes.IBCModule = IBCModule{}

)

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#channel-handshake-version-negotiation) Channel handshake version negotiation

Application modules are expected to verify the versioning used during the channel handshake procedure:

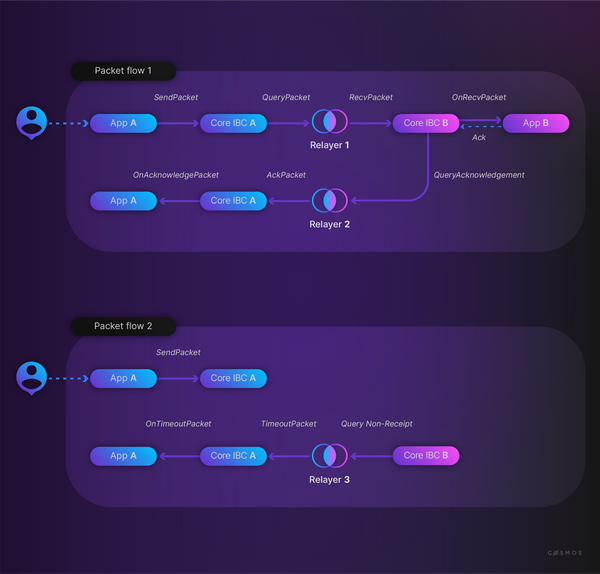
* OnChanOpenInit will verify that the relayer-chosen parameters are valid and perform any custom INIT logic.
  + It may return an error if the chosen parameters are invalid, in which case the handshake is aborted. If the provided version string is non-empty, OnChanOpenInit should return the version string if valid or an error if the provided version is invalid.
  + **If the version string is empty, OnChanOpenInit is expected to return a default version string representing the version(s) it supports.** If there is no default version string for the application, it should return an error if the provided version is an empty string.
* OnChanOpenTry will verify the relayer-chosen parameters along with the counterparty-chosen version string and perform custom TRY logic.
  + If the relayer-chosen parameters are invalid, the callback must return an error to abort the handshake. If the counterparty-chosen version is not compatible with this module's supported versions, the callback must return an error to abort the handshake.
  + If the versions are compatible, the try callback must select the final version string and return it to core IBC.OnChanOpenTry may also perform custom initialization logic.
* OnChanOpenAck will error if the counterparty selected version string is invalid and abort the handshake. It may also perform custom ACK logic.



Versions must be strings but can implement any versioning structure. Often a simple template is used that combines the name of the application and an iteration number, like leaderboard-1 for the leaderboard IBC module.   
  
However, the version string can also include metadata to indicate attributes of the channel you are supporting, like applicable middleware and the underlying app version. An example of this is the version string for middleware, which is discussed in this [IBC section](https://ida.interchain.io/academy/3-ibc/10-ibc-mw-develop.html).

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#packet-callbacks) Packet callbacks

The general application packet flow was discussed in [a previous section (opens new window)↗](https://tutorials.cosmos.network/academy/3-ibc/3-channels.html#application-packet-flow). As a refresher, let's take a look at the diagram:



The packet callbacks in the packet flow can now be identified by investigating the IBCModule interface.

##### [#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#sending-packets) Sending packets

Modules **do not send packets through callbacks**, since modules initiate the action of sending packets to the IBC module, as opposed to other parts of the packet flow where messages sent to the IBC module must trigger execution on the port-bound module through the use of callbacks. Thus, to send a packet a module simply needs to call SendPacket on the IBCChannelKeeper.



In order to prevent modules from sending packets on channels they do not own, IBC expects modules to pass in the correct channel capability for the packet's source channel.



For advanced readers, more on capabilities can be found in the [ibc-go docs (opens new window)↗](https://ibc.cosmos.network/main/ibc/overview.html#capabilities) or the [ADR on the Dynamic Capability Store (opens new window)↗](https://github.com/cosmos/cosmos-sdk/blob/6aaf83c894e917836a047b0399dd70a95fd2710d/docs/architecture/adr-003-dynamic-capability-store.md).

##### [#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#receiving-packets) Receiving packets

To handle receiving packets, the module must implement the OnRecvPacket callback. This gets invoked by the IBC module after the packet has been proved valid and correctly processed by the IBC keepers. Thus, the OnRecvPacket callback only needs to worry about making the appropriate state changes given the packet data without worrying about whether the packet is valid or not.

Modules may return to the IBC handler an acknowledgement which implements the Acknowledgement interface. The IBC handler will then commit this acknowledgement of the packet so that a relayer may relay the acknowledgement back to the sender module.

The state changes that occurred during this callback will only be written if:

* The acknowledgement was successful as indicated by the Success() function of the acknowledgement.
* The acknowledgement returned is nil, indicating that an asynchronous process is occurring.



Applications that process asynchronous acknowledgements must handle reverting state changes when appropriate. Any state changes that occurred during the OnRecvPacket callback will be written for asynchronous acknowledgements.

In x/leaderboard/module\_ibc.go scaffolded by Ignite CLI you find OnRecvPacket:



Copy

// OnRecvPacket implements the IBCModule interface

func (am AppModule) OnRecvPacket(

ctx sdk.Context,

modulePacket channeltypes.Packet,

relayer sdk.AccAddress,

) ibcexported.Acknowledgement {

var ack channeltypes.Acknowledgement

// this line is used by starport scaffolding # oracle/packet/module/recv

var modulePacketData types.LeaderboardPacketData

...

// Dispatch packet

switch packet := modulePacketData.Packet.(type) {

// this line is used by starport scaffolding # ibc/packet/module/recv

default:

errMsg := fmt.Sprintf("unrecognized %s packet type: %T", types.ModuleName, packet)

return channeltypes.NewErrorAcknowledgement(errMsg)

}

// NOTE: acknowledgement will be written synchronously during IBC handler execution.

return ack

}

The *dispatch packet* switch statement is added by Ignite CLI. As it is stated in the docs, strictly speaking, you only need to decode the packet data (which is discussed in an upcoming section) and return the acknowledgement after processing the packet. However, the structure provided by Ignite CLI is useful to get set up but can be changed according to the preference of the developer.

As a reminder, this is the Acknowledgement interface:



Copy

// Acknowledgement defines the interface used to return

// acknowledgements in the OnRecvPacket callback.

type Acknowledgement interface {

Success() bool

Acknowledgement() []byte

}

##### [#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#acknowledging-packets) Acknowledging packets

The last step of the packet flow depends on whether you have a happy path, when the packet has been successfully relayed, or a timeout when something went wrong.

After a module writes an Acknowledgement, a relayer can relay it back to the sender module. The sender module can then process the acknowledgement using the OnAcknowledgementPacket callback. The contents of the Acknowledgement are entirely up to the modules on the channel (just like the packet data); however, it may often contain information on whether the packet was successfully processed, along with some additional data that could be useful for remediation if the packet processing failed.

Since the modules are responsible for agreeing on an encoding/decoding standard for packet data and acknowledgements, IBC will pass in the acknowledgements as []byte to this callback. The callback is responsible for decoding the acknowledgement and processing it.

In x/leaderboard/module\_ibc.go scaffolded by Ignite CLI you will find OnAcknowledgementPacket:



Copy

// OnAcknowledgementPacket implements the IBCModule interface

func (am AppModule) OnAcknowledgementPacket(

ctx sdk.Context,

modulePacket channeltypes.Packet,

acknowledgement []byte,

relayer sdk.AccAddress,

) error {

var ack channeltypes.Acknowledgement

... //validation

// this line is used by starport scaffolding # oracle/packet/module/ack

var modulePacketData types.LeaderboardPacketData

... //validation

var eventType string

// Dispatch packet

switch packet := modulePacketData.Packet.(type) {

// this line is used by starport scaffolding # ibc/packet/module/ack

default:

... //errMsg

}

ctx.EventManager().EmitEvent(

sdk.NewEvent(

...

),

)

switch resp := ack.Response.(type) {

case \*channeltypes.Acknowledgement\_Result:

ctx.EventManager().EmitEvent(

sdk.NewEvent(

...

),

)

case \*channeltypes.Acknowledgement\_Error:

ctx.EventManager().EmitEvent(

sdk.NewEvent(

...

),

)

}

return nil

}

Again, the structure to dispatch the packet with the switch statement as well as the switch statement for the ack (success case or error case) have been structured by Ignite CLI where the [docs (opens new window)↗](https://ibc.cosmos.network/main/ibc/apps/ibcmodule.html#acknowledging-packets) offer more freedom to the developer to implement decoding and processing of the ack.



The events that are being emitted are defined in x/leaderboard/types/events\_ibc.go.

##### [#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#timing-out-packets) Timing out packets

If the timeout for a packet is reached before the packet is successfully received, or the counterparty channel end is closed before the packet is successfully received, then the receiving chain can no longer process it. Thus, the sending chain must process the timeout using OnTimeoutPacket to handle this situation. Again the IBC module will verify that the timeout is indeed valid, so our module only needs to implement the state machine logic for what to do once a timeout is reached and the packet can no longer be received.

In x/leaderboard/module\_ibc.go scaffolded by Ignite CLI you will find OnTimeoutPacket:



Copy

// OnTimeoutPacket implements the IBCModule interface

func (am AppModule) OnTimeoutPacket(

ctx sdk.Context,

modulePacket channeltypes.Packet,

relayer sdk.AccAddress,

) error {

var modulePacketData types.LeaderboardPacketData

... // validation

// Dispatch packet

switch packet := modulePacketData.Packet.(type) {

// this line is used by starport scaffolding # ibc/packet/module/timeout

default:

... // errMsg

}

return nil

}

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#binding-to-a-port) Binding to a port

Every IBC module binds to a port, with a unique portID which denotes the type of application.



The portID does not refer to a certain numerical ID, like localhost:8080 with a portID 8080. Rather it refers to the application module to which the port binds. For IBC modules built with the Cosmos SDK it defaults to the module's name, and for CosmWasm contracts, it defaults to the contract address.

Currently, ports must be bound on app initialization. In order to bind modules to their respective ports on initialization, the following needs to be implemented:

1. Port ID in the GenesisState proto definition:



Copy

@@ proto/leaderboard/genesis.proto

// GenesisState defines the leaderboard module's genesis state.

message GenesisState {

Params params = 1 [(gogoproto.nullable) = false];

+ string port\_id = 2;

// this line is used by starport scaffolding # genesis/proto/state

}

1. Port ID as a key in the module store in x/leaderboard/types/keys.go:



Copy

@@ const in x/leaderboard/types/keys.go

// MemStoreKey defines the in-memory store key

MemStoreKey = "mem\_leaderboard"

+ // Version defines the current version the IBC module supports

+ Version = "leaderboard-1"

+ // PortID is the default port id that module binds to

+ PortID = "leaderboard"

+ )

+ var (

+ // PortKey defines the key to store the port ID in store

+ PortKey = KeyPrefix("leaderboard-port-")

)



By default, the portID is indeed set to the module name, and the application version is set to <modulename>-n with n as an incrementing value.

1. Port ID in the x/leaderboard/types/genesis.go:



Copy

// DefaultGenesisState returns a GenesisState with "transfer" as the default PortID.

func DefaultGenesisState() \*GenesisState {

return &GenesisState{

+ PortId: PortID,

// additional k-v fields

}

}

// Validate performs basic genesis state validation returning an error upon any

// failure.

func (gs GenesisState) Validate() error {

+ if err := host.PortIdentifierValidator(gs.PortId); err != nil {

+ return err

+ }

//addtional validations

return gs.Params.Validate()

}

1. Binding of the IBC module to the port in x/leaderboard/genesis.go:



Copy

@@ InitGenesis

func InitGenesis(ctx sdk.Context, k keeper.Keeper, genState types.GenesisState) {

// this line is used by starport scaffolding # genesis/module/init

+ k.SetPort(ctx, genState.PortId)

+ // Only try to bind to port if it is not already bound, since we may already own

+ // port capability from capability InitGenesis

+ if !k.IsBound(ctx, genState.PortId) {

+ // module binds to the port on InitChain

+ // and claims the returned capability

+ err := k.BindPort(ctx, genState.PortId)

+ if err != nil {

+ panic("could not claim port capability: " + err.Error())

+ }

}

k.SetParams(ctx, genState.Params)

}

@@ ExportGenesis

func ExportGenesis(ctx sdk.Context, k keeper.Keeper) \*types.GenesisState {

genesis := types.DefaultGenesis()

genesis.Params = k.GetParams(ctx)

+ genesis.PortId = k.GetPort(ctx)

// this line is used by starport scaffolding # genesis/module/export

return genesis

}

Where:



Copy

// IsBound checks if the module is already bound to the desired port

func (k Keeper) IsBound(ctx sdk.Context, portID string) bool {

\_, ok := k.scopedKeeper.GetCapability(ctx, host.PortPath(portID))

return ok

}

// BindPort defines a wrapper function for the port Keeper's function in

// order to expose it to the module's InitGenesis function

func (k Keeper) BindPort(ctx sdk.Context, portID string) error {

cap := k.portKeeper.BindPort(ctx, portID)

return k.ClaimCapability(ctx, cap, host.PortPath(portID))

}

The module binds to the desired port(s) and returns the capabilities.

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#keeper) Keeper

Previous steps sometimes referenced keeper methods that deal with binding to and getting and setting a port, claiming and authenticating capabilities. These methods need to be added to the keeper.

For a full overview, check out the [ibc-go docs (opens new window)↗](https://ibc.cosmos.network/main/ibc/apps/keeper.html) and compare it with the x/leaderboard/keeper/keeper.go file.

You will notice that Ignite CLI uses a custom cosmosibckeeper package which you can find [here (opens new window)↗](https://github.com/ignite/cli/tree/v0.22.1/ignite/pkg/cosmosibckeeper).

[#Copy link](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#routing-and-app-go) Routing and **app.go**

When looking at app.go you will see some minor additions, the most prominent of which is adding a route to the Leaderboard module on the IBC Router.



Copy

@@ func New in app/app.go

ibcRouter := ibcporttypes.NewRouter()

ibcRouter.AddRoute(ibctransfertypes.ModuleName, transferIBCModule)

ibcRouter.AddRoute(monitoringptypes.ModuleName, monitoringModule)

+ ibcRouter.AddRoute(leaderboardmoduletypes.ModuleName, leaderboardModule)

// this line is used by starport scaffolding # ibc/app/router

app.IBCKeeper.SetRouter(ibcRouter)

synopsis

To summarize, this section has explored:

* How to build an SDK blockchain, as a regular module.
* How to build an SDK blockchain, as an IBC module.
* How application modules verify the versioning used during the channel handshake procedure.
* Packet callbacks, and specifically how sending, receiving, acknowledging, and timing out is handled in the packet flow.
* How to bind to a port, and the necessity of adding appropriate keeper methods to the keeper.



Copy

previous

[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/3-ibc-app-intro.html)

**[IBC Application Developer Introduction](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/3-ibc-app-intro.html)**

up next

**[Adding Packet and Acknowledgment Data](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/5-ibc-app-packets.html)**

[[](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/5-ibc-app-packets.html)](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/5-ibc-app-packets.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

[](https://ida.interchain.io/ida-course/discord-info.html)

On this page

[Scaffold a leaderboard chain](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#scaffold-a-leaderboard-chain)

[IBC application module requirements](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#ibc-application-module-requirements)

[Implementing the IBCModule interface](https://ida.interchain.io/hands-on-exercise/5-ibc-adv/4-ibc-app-steps.html#implementing-the-ibcmodule-interface)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)