Dev Activity Log pt. 2 - Partner Chain Integration

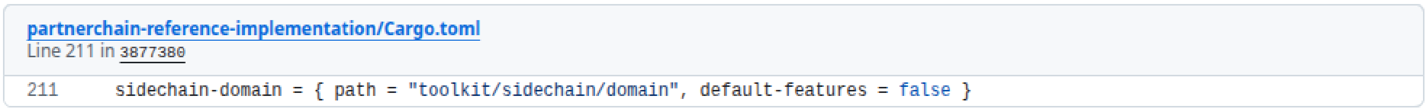
Part 1 - Code Integration

The first step of the process is to include the relevant code modifications.

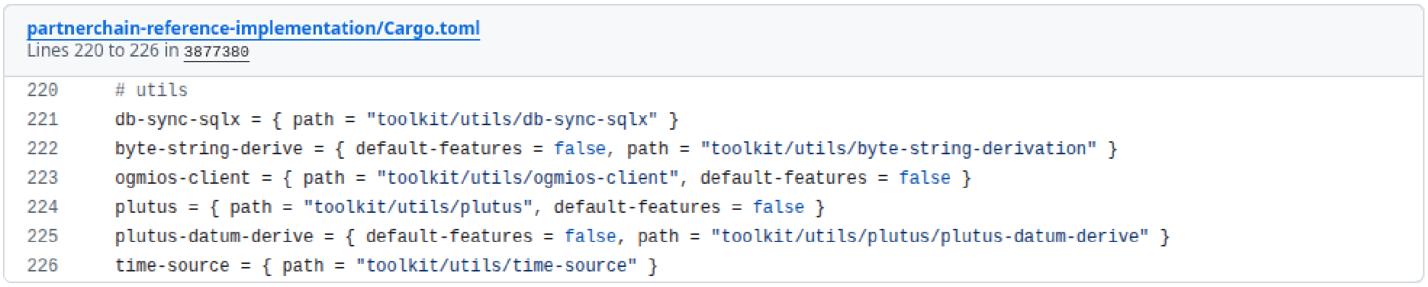
The Substrate node uses IOG's Partner Chain SDK to connect with Cardano. This SDK provides a wide variety of utilities for the node. We are not using all of the functionalities provided (many of them are oriented to the account-model), but the part of them that is both important for the partner chains creation/usage and generic in the ledger model: the `partner-chains-cli` module, along with the `cli`, which are meant for the creation of the partner chain, the initial set up of candidates, and the registration of candidates. Moreover, we've modified some parts to better adjust to our needs.

Let's get started with the modifications:

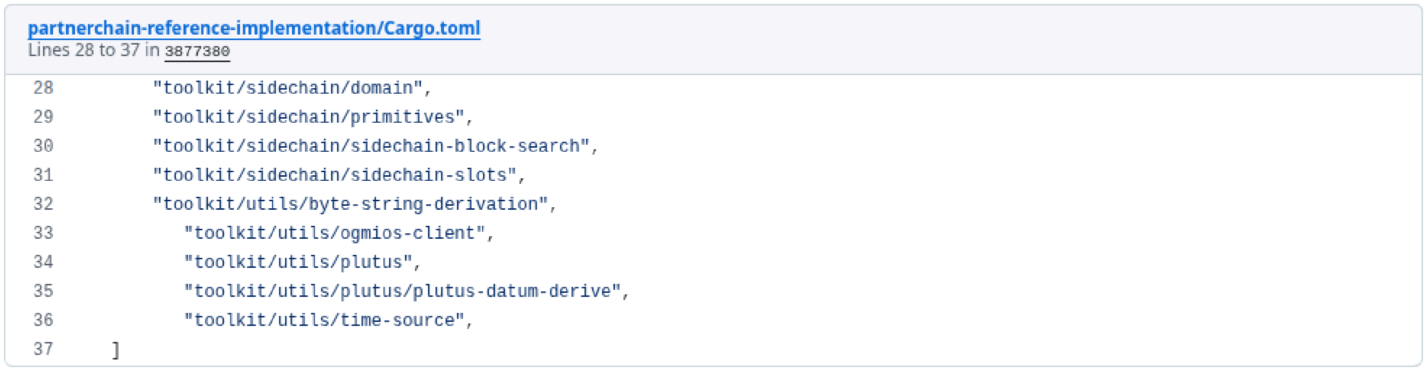
1. Add `sidechain-domain` and `utils` from Partner-chains SDK `toolkit`. Add as workspace members and as workspace dependencies. Add additional dependencies.


partnerchain-reference-implementation/Cargo.toml
Line 211 in 3877380

```toml
sidechain-domain = { path = "toolkit/sidechain/domain", default-features = false }
```


partnerchain-reference-implementation/Cargo.toml
Lines 220 to 226 in 3877380

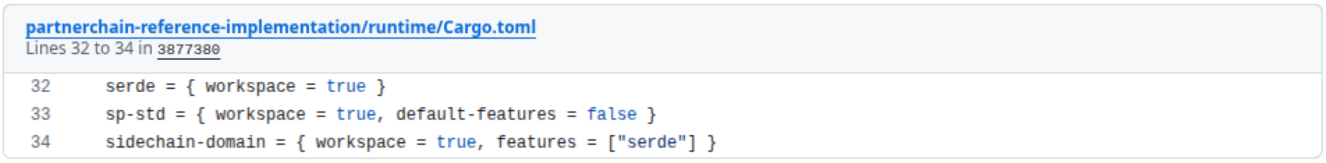
```toml
# utils
db-sync-sqlx = { path = "toolkit/utils/db-sync-sqlx" }
byte-string-derive = { default-features = false, path = "toolkit/utils/byte-string-derivation" }
ogmios-client = { path = "toolkit/utils/ogmios-client", default-features = false }
plutus = { path = "toolkit/utils/plutus", default-features = false }
plutus-datum-derive = { default-features = false, path = "toolkit/utils/plutus/plutus-datum-derive" }
time-source = { path = "toolkit/utils/time-source" }
```


partnerchain-reference-implementation/Cargo.toml
Lines 28 to 37 in 3877380

```toml
    "toolkit/sidechain/domain",
    "toolkit/sidechain/primitives",
    "toolkit/sidechain/sidechain-block-search",
    "toolkit/sidechain/sidechain-slots",
    "toolkit/utils/byte-string-derivation",
       "toolkit/utils/ogmios-client",
       "toolkit/utils/plutus",
       "toolkit/utils/plutus/plutus-datum-derive",
       "toolkit/utils/time-source",
]
```

2. Add `CrossChainKey` implementation to runtime:

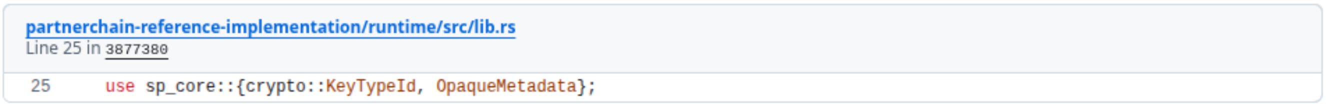
    1. Add deps to `Cargo.toml`.



partnerchain-reference-implementation/runtime/Cargo.toml
Lines 32 to 34 in 3877380

```
serde = { workspace = true }
sp-std = { workspace = true, default-features = false }
sidechain-domain = { workspace = true, features = ["serde"] }
```

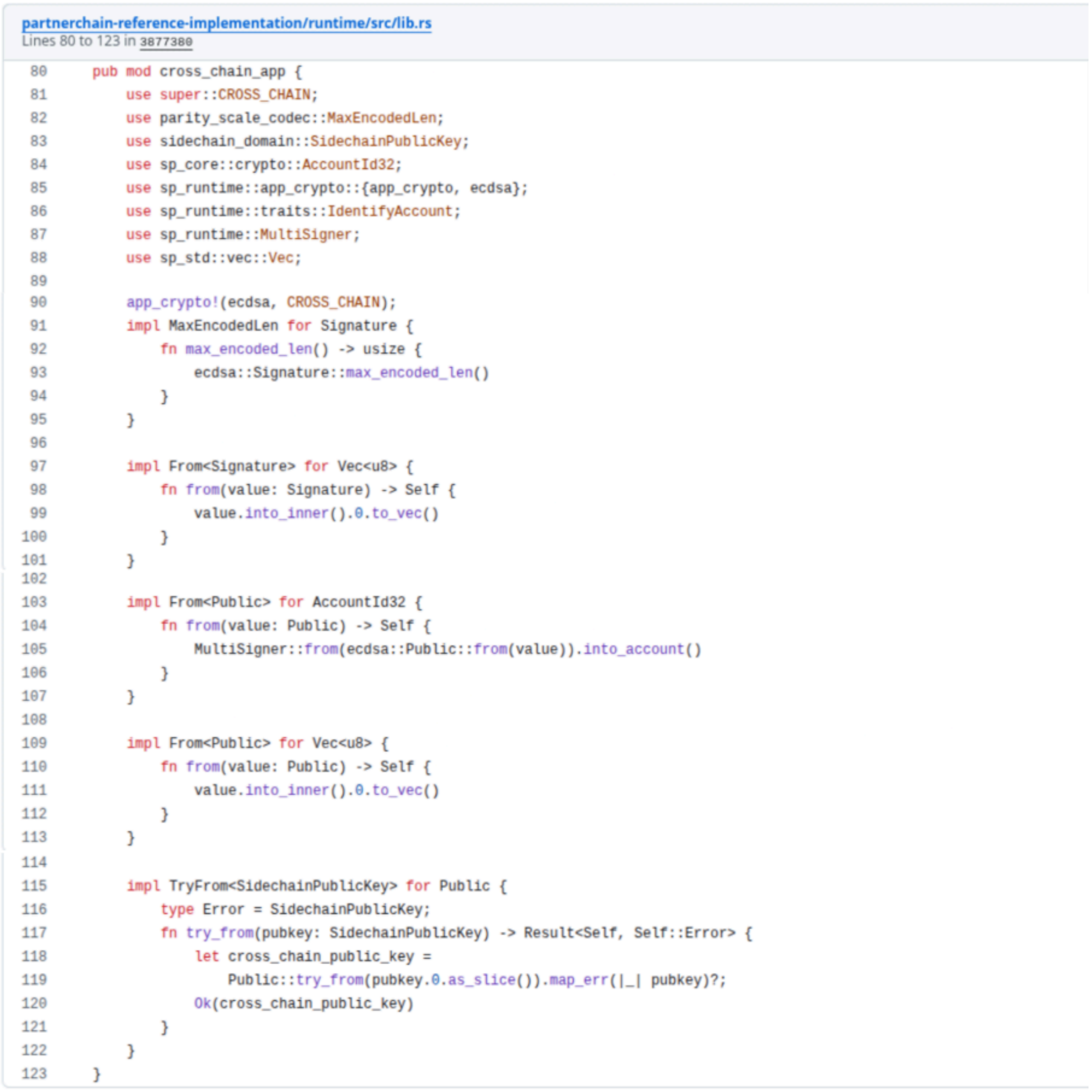
    2. Import `KeyTypeId` from `sp_core`.



partnerchain-reference-implementation/runtime/src/lib.rs
Line 25 in 3877380

```
use sp_core::{crypto::KeyTypeId, OpaqueMetadata};
```

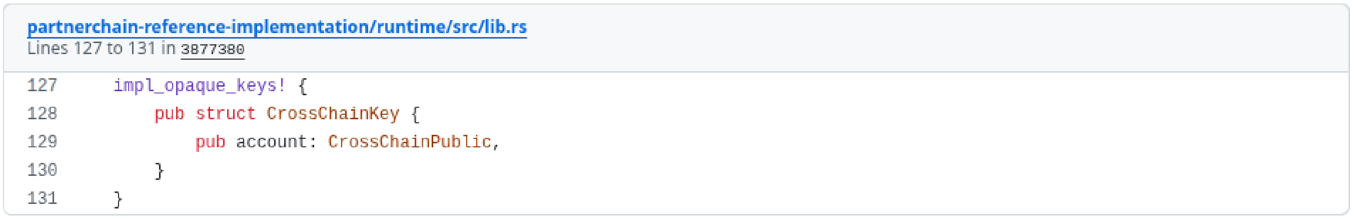
    3. Implement `cross_chain_app` in `opaque` module.



partnerchain-reference-implementation/runtime/src/lib.rs
Lines 80 to 123 in 3877380

```
pub mod cross_chain_app {
    use super::CROSS_CHAIN;
    use parity_scale_codec::MaxEncodedLen;
    use sidechain_domain::SidechainPublicKey;
    use sp_core::crypto::AccountId32;
    use sp_runtime::app_crypto::{app_crypto, ecdsa};
    use sp_runtime::traits::IdentifyAccount;
    use sp_runtime::MultiSigner;
    use sp_std::vec::Vec;

    app_crypto!(ecdsa, CROSS_CHAIN);
    impl MaxEncodedLen for Signature {
        fn max_encoded_len() -> usize {
            ecdsa::Signature::max_encoded_len()
        }
    }

    impl From<Signature> for Vec<u8> {
        fn from(value: Signature) -> Self {
            value.into_inner().0.to_vec()
        }
    }

    impl From<Public> for AccountId32 {
        fn from(value: Public) -> Self {
            MultiSigner::from(ecdsa::Public::from(value)).into_account()
        }
    }

    impl From<Public> for Vec<u8> {
        fn from(value: Public) -> Self {
            value.into_inner().0.to_vec()
        }
    }

    impl TryFrom<SidechainPublicKey> for Public {
        type Error = SidechainPublicKey;
        fn try_from(pubkey: SidechainPublicKey) -> Result<Self, Self::Error> {
            let cross_chain_public_key =
                Public::try_from(pubkey.0.as_slice()).map_err(|_| pubkey)?;
            Ok(cross_chain_public_key)
        }
    }
}
```

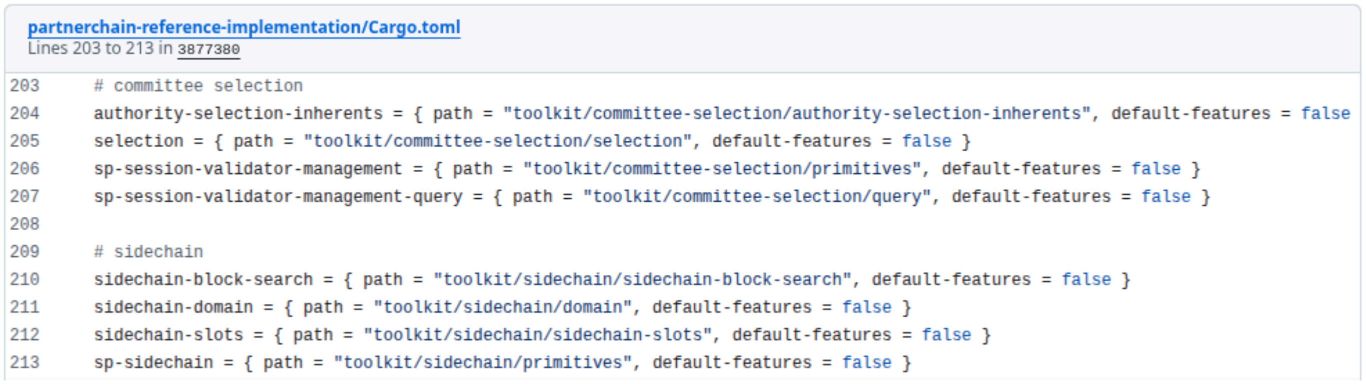
4. Define `CrossChainKey` within `impl_opaque_keys` macro.



partnerchain-reference-implementation/runtime/src/lib.rs
Lines 127 to 131 in 3877380

```rust
    impl_opaque_keys! {
        pub struct CrossChainKey {
            pub account: CrossChainPublic,
        }
    }
```

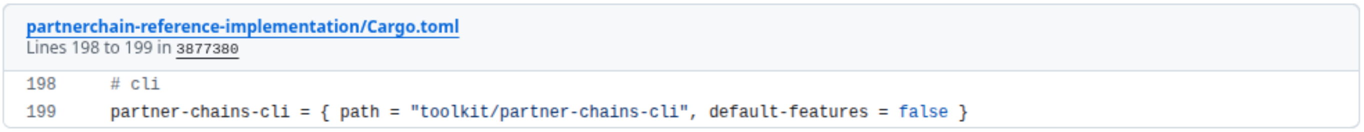
3. Add `authoritiy_selection_inherents`, `primitives`, `query`, `selection` from `CommitteeSelection`. Add `primitives`, `block-search` and `slots` from `sidechain`.



partnerchain-reference-implementation/Cargo.toml
Lines 203 to 213 in 3877380

```toml
# committee selection
authority-selection-inherents = { path = "toolkit/committee-selection/authority-selection-inherents", default-features = false
selection = { path = "toolkit/committee-selection/selection", default-features = false }
sp-session-validator-management = { path = "toolkit/committee-selection/primitives", default-features = false }
sp-session-validator-management-query = { path = "toolkit/committee-selection/query", default-features = false }

# sidechain
sidechain-block-search = { path = "toolkit/sidechain/sidechain-block-search", default-features = false }
sidechain-domain = { path = "toolkit/sidechain/domain", default-features = false }
sidechain-slots = { path = "toolkit/sidechain/sidechain-slots", default-features = false }
sp-sidechain = { path = "toolkit/sidechain/primitives", default-features = false }
```

4. Add `partner-chains-cli`.



partnerchain-reference-implementation/Cargo.toml
Lines 198 to 199 in 3877380

```toml
# cli
partner-chains-cli = { path = "toolkit/partner-chains-cli", default-features = false }
```

Here we modified some things, namely the removal of references to pallets.

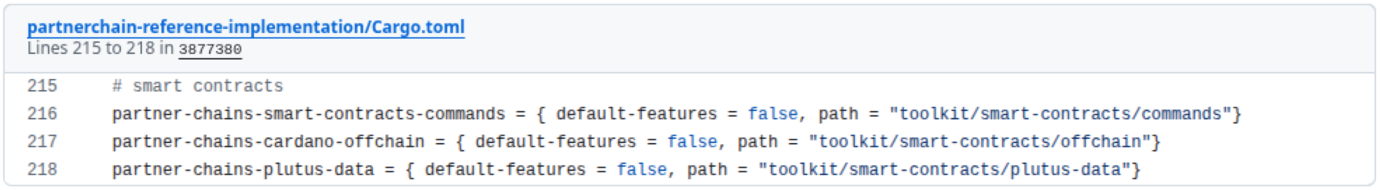
1. In `Cargo.toml`: remove pallets as dependencies.

```rust
# pallet-session-validator-management = { workspace = true, features = ["std"]
}
# pallet-partner-chains-bridge = { workspace = true, features = ["std"] }
# sp-partner-chains-bridge = { workspace = true, features = ["std"] }
# pallet-partner-chains-session = { workspace = true, features = ["std"] }
# pallet-sidechain = { workspace = true, features = ["std"] }
# pallet-governed-map = { workspace = true, features = ["std"] }
# sp-governed-map = { workspace = true, features = ["std"] }
```

2. In `create_chain_spec`, remove the following lines:

```rust
use sp_core::ecdsa;
use sp_runtime::AccountId32;
// /// Returns [pallet_sidechain::GenesisConfig] derived from the config
// pub fn pallet_sidechain_config<T: pallet_sidechain::Config>(
//  &self,
//  slots_per_epoch: sidechain_slots::SlotsPerEpoch,
// ) -> pallet_sidechain::GenesisConfig<T> {
//  pallet_sidechain::GenesisConfig {
//      genesis_utxo: self.genesis_utxo,
//      slots_per_epoch,
//      _config: PhantomData,
//  }
// }

// /// Returns [pallet_partner_chains_session::GenesisConfig] derived from the
config, using initial permissioned candidates
// /// as initial validators
// pub fn pallet_partner_chains_session_config<T:
pallet_partner_chains_session::Config>(
//  &self,
// ) -> pallet_partner_chains_session::GenesisConfig<T>
// where
//  T::ValidatorId: From<AccountId32>,
//  T::Keys: From<Keys>,
// {
//  pallet_partner_chains_session::GenesisConfig {
//      initial_validators: self
//          .initial_permissioned_candidates_parsed
//          .iter()
//          .map(|c| (c.account_id_32().into(), c.keys.clone().into()))
//          .collect::<Vec<_>>(),
//  }
// }

// /// Returns [pallet_session_validator_management::GenesisConfig] derived
from the config using initial permissioned candidates
// /// as initial authorities
// pub fn pallet_session_validator_management_config<
//  T: pallet_session_validator_management::Config,
// >(
//  &self,
// ) -> pallet_session_validator_management::GenesisConfig<T>
// where
//  T::AuthorityId: From<ecdsa::Public>,
//  T::AuthorityKeys: From<Keys>,
//  T::CommitteeMember:
//      From<authority_selection_inherents::CommitteeMember<T::AuthorityId,
T::AuthorityKeys>>,
// {
//  pallet_session_validator_management::GenesisConfig {
//      initial_authorities: self
//          .initial_permissioned_candidates_parsed
//          .iter()
//          .map(|c| {
//              authority_selection_inherents::CommitteeMember::permissioned(
//                  c.sidechain.into(),
//                  c.keys.clone().into(),
//              )
//              .into()
//          })
//          .collect::<Vec<_>>(),
//      main_chain_scripts: sp_session_validator_management::MainChainScripts {
//          committee_candidate_address:
self.committee_candidate_address.clone(),
//          d_parameter_policy_id: self.d_parameter_policy_id.clone(),
//          permissioned_candidates_policy_id:
self.permissioned_candidates_policy_id.clone(),
//      },
//  }
// }

// /// Returns [pallet_partner_chains_bridge::GenesisConfig] derived from the
config
// pub fn bridge_config<T: pallet_partner_chains_bridge::Config>(
//  &self,
// ) -> pallet_partner_chains_bridge::GenesisConfig<T> {
//  pallet_partner_chains_bridge::GenesisConfig {
//      main_chain_scripts: Some(sp_partner_chains_bridge::MainChainScripts {
//          token_policy_id: self.bridge_token_policy.clone(),
//          token_asset_name: self.bridge_token_asset_name.clone(),
//          illiquid_circulation_supply_validator_address: self
//              .illiquid_circulation_supply_validator_address
//              .clone(),
//      }),
//      initial_checkpoint: Some(self.genesis_utxo),
//      _marker: PhantomData,
//  }
// }

// /// Returns [pallet_governed_map::GenesisConfig] derived from the config
// pub fn governed_map_config<T: pallet_governed_map::Config>(
//  &self,
// ) -> pallet_governed_map::GenesisConfig<T> {
//  pallet_governed_map::GenesisConfig {
//      main_chain_scripts:
self.governed_map_validator_address.as_ref().and_then(|addr| {
//          self.governed_map_asset_policy_id.as_ref().map(|policy| {
//              sp_governed_map::MainChainScriptsV1 {
//                  validator_address: addr.clone(),
//                  asset_policy_id: policy.clone(),
//              }
//          })
//      }),
//      _marker: PhantomData,
//  }
// }
```

2. Add `offchain`, `commands` and `plutus-data` from `smart-contracts`.



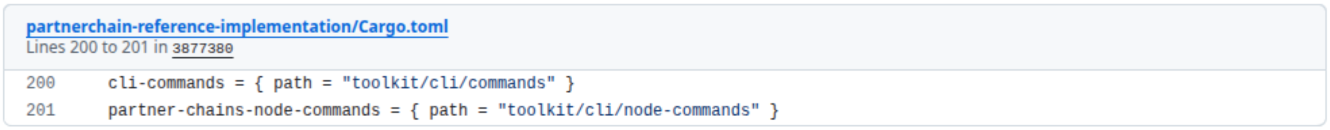
partnerchain-reference-implementation/Cargo.toml
Lines 215 to 218 in 3877380

```
# smart contracts
partner-chains-smart-contracts-commands = { default-features = false, path = "toolkit/smart-contracts/commands"}
partner-chains-cardano-offchain = { default-features = false, path = "toolkit/smart-contracts/offchain"}
partner-chains-plutus-data = { default-features = false, path = "toolkit/smart-contracts/plutus-data"}
```

Here we also needed to remove some things for some functionalities that we won't support. There are many changes to the original SDK here so if you're following our pallet-less approach it might be more convenient to just copy from our source code, but the modifications are detailed below anyways:

1. Remove references and implementations of `from_ogmios_utxo`, `AssetIdExt`, `get_asset_amount` in `offchain/src/csl`.
2. Remove the following modules and their references in lib.rs:
    1. `versioning_system`.
    2. `bridge`.
    3. `reserve`.

2. Add `node-commands` and `commands` from `cli`.


partnerchain-reference-implementation/Cargo.toml
Lines 200 to 201 in 3877380

```
cli-commands = { path = "toolkit/cli/commands" }
partner-chains-node-commands = { path = "toolkit/cli/node-commands" }
```

Again some modules were removed as they don't fit our implementation.

1. Node commands

    1. Remove imports

        1. `authority-selection-inherent` import
        2. `address_association_signatures` and `block_producer_metadata_signatures` from `cli-commands`
        3. `Decode, Encode`
        4. `sc_service::task_manager;`
        5. `use sp_api::ProvideRuntimeApi;`
        6. `use sp_blockchain::HeaderBackend;`
        7. `use sp_runtime::AccountId32;`
        8. `use sp_runtime::DeserializeOwned;`
        9. `use sp_runtime::Serialize;`
        10. `use sp_session_validator_management::CommitteeMember as CommitteeMemberT;`
        11. `use sp_session_validator_management::SessionValidatorManagementApi;`
        12. `use sp_session_validator_management_query::SessionValidatorManagementQuery;`
        13. `use sp_session_validator_management_query::commands::*;`
        14. `use sp_sidechain::{GetGenesisUtxo, GetSidechainStatus};`
        15. `use std::sync::Arc;`
        16. `Future`
        17. `SubstrateCli` from `sc_cli`

    2. In `SubCommand::PartnerChains` remove the following:

        1. `PartnerChainsAddress` type parameter

            2. `SidechainParams`
            3. `RegistrationStatus`
            4. `AriadneParameters`
            5. `SignAddressAssociation`
            6. `SignBlockProducerMetadata`

        2. Remove `REGISTRATION_STATUS_AFTER_HELP`.

        3. In `run`:

            1. Remove `Cli`, `CommitteeMember`, `Client`, `BlockProducerMetadata`, `PartnerchainAddress` type parameters, and its respective constraints on the function.
            2. Remove `cli` and `get_deps` function parameters.
            3. Remove `SidechainParams`, `RegistrationStatus`, `AriadneParameters`, `SignAddressAssociation` and `SignBlockProducerMetadata` command.
            4. Remove `print_Result` (optional but it will cause an *unused* warning).

    2. `commands`:

        1. `address_association_signatures`, `get_genesis_utxo` and `block_producer_metadata_signatures` mod inclusions and their respective files.

2. Add `partner-chains` command to the node.

    1. Add dependencies to the node:

partnerchain-reference-implementation/node/Cargo.toml
Lines 45 to 46 in 3877380

```
partner-chains-node-commands = { workspace = true }
partner-chains-cli = { workspace = true }
```

    2. In `cli`:

        1. Imports

partnerchain-reference-implementation/node/src/cli.rs
Lines 1 to 3 in 3877380

```
use griffin_partner_chains_runtime::opaque::SessionKeys;
use partner_chains_cli::{KeyDefinition, AURA, GRANDPA};
use partner_chains_node_commands::{PartnerChainRuntime, PartnerChainsSubcommand};
```

    2. Define `PartnerChainRuntime`’s `WizardBindings`:

partnerchain-reference-implementation/node/src/cli.rs
Lines 12 to 13 in `3877380`

```rust
#[derive(Debug, Clone)]
pub struct WizardBindings;
```

    3. Add `PartnerChains` command.

partnerchain-reference-implementation/node/src/cli.rs
Lines 101 to 109 in `3877380`

```rust
#[derive(Debug, clap::Subcommand)]
pub enum Subcommand {
    /// Key management cli utilities
    #[command(subcommand)]
    Key(sc_cli::KeySubcommand),

    #[clap(flatten)]
    PartnerChains(PartnerChainsSubcommand<WizardBindings>),

```

3. In `command`:

    1. In the run function add parsing and implementation for `PartnerChain` command.

partnerchain-reference-implementation/node/src/command.rs
Lines 47 to 63 in `3877380`

```rust
/// Parse and run command line arguments
pub fn run() -> sc_cli::Result<()> {
    let cli = Cli::from_args();

    match &cli.subcommand {
        Some(Subcommand::Key(cmd)) => cmd.run(&cli),
        Some(Subcommand::PartnerChains(cmd)) => {
            partner_chains_node_commands::run::<
                // _,
                griffin_core::types::OpaqueBlock,
                // _,
                // _,
                // BlockProducerMetadataType,
                WizardBindings,
                // AccountId32,
            >(cmd.clone())
        }
```

2. Add `MaybeFromCandidateKeys` trait implementation for `SessionKeys`.

    1. Add `authority-selection-inherents` dependency on runtime.

partnerchain-reference-implementation/runtime/Cargo.toml
Line 35 in `3877380`

```toml
authority-selection-inherents = { workspace = true }
```

add also as `std` feature:

partnerchain-reference-implementation/runtime/Cargo.toml
Lines 40 to 43 in `3877380`

```
[features]
default = ["std"]
std = [
	"authority-selection-inherents/std",
```

2. Implement trait in `opaque`.

partnerchain-reference-implementation/runtime/src/lib.rs
Lines 124 to 126 in `3877380`

```

impl MaybeFromCandidateKeys for SessionKeys {}

```

3. `create-chain-spec`: The command was modified to generate the `genesis.json` file that we need for Griffin.

    1. Imports in `Cargo.toml`:

partnerchain-reference-implementation/toolkit/partner-chains-cli/Cargo.toml
Lines 44 to 45 in `3877380`

```
griffin-core = { workspace = true }
partner-chains-plutus-data = { workspace = true }
```

    2. In `create_chain_spec/mod`:

        1. Add imports

partnerchain-reference-implementation/toolkit/partner-chains-cli/src/create_chain_spec/mod.rs
Lines 8 to 9 in `3877380`

```
use griffin_core::genesis::config_builder::GenesisConfig;
use partner_chains_plutus_data::permissioned_candidates::permissioned_candidates_to_plutus_data;
```

        2. Modify main function.

        3. Add constant default genesis to build from.

Part 2 - Usage

The inclusion of the partner-chain toolkit allows us to create some UTxOs in Cardano that rule some aspects of the partner chain. We'll briefly go over what these are and then we'll go through the instructions to set up the partner chain using the wizards.

The UTxOs that will be created on the cardano side are the following;

- Governance init UTxO: identifies the partner chain, it's unique for each chain.
- Permissioned-candidates UTxO: holds the list of permissioned candidates for the list. It can be updated.
- D-Parameter UTxO: holds the D-Parameter, which is the ratio of permissioned to registered candidates on the committee. It is updatable as well.

Instructions:

Build the node

You can build the `griffin-partner-chains-node` with the following command:

Shell
```
cargo build –release -p griffin-partner-chains-node
```

Generate-keys

The first time you set up a chain you'll need to run the generate-keys command:

Shell
```
./griffin-partner-chains-node wizards generate-keys
```

This command creates three keys: a partner-chain key, a grandpa key and an aura key. The first one is an identifier for the node and the last two are used for the committee. If you run this command and there are keys present already, it will prompt you to decide whether you want new ones or not.

Prepare-configuration

This command requires Ogmios so make sure to have that service on before running it.

This wizard sets up the initial governance UTxO and also fills the `pc-chain-config.json` file. Once you start the wizard:

```shell
./griffin-partner-chains-node wizards prepare-configuration
```

It will prompt you to fill out some information:

1. Update the bootnodes and provide the public IP or host name for your node.
2. Provide the information to connect to the Ogmios instance.
3. Provide the required keys and select a UTxO that will be used to initialize the chain.
4. Provide and confirm initial multisig governance authorities. This can be skipped and confirmed later using other commands, but we won't document this approach.
5. Configure native token. Native tokens aren't supported, so anything set here won't have any impact on the sidechain.
6. Store the main chain configuration, this is, the `pc-chain-config.json`.

After the wizard finishes, you need to include the permissioned candidates on the `initial_permissioned_candidates` field of the aforementioned file. Here you may include the keys that were generated in the previous step, for example. But you might also need to gather the keys of the other permissioned candidates of the chain. Once the file has been adequately modified, you may move on to the next step.

Setup-main-chain-state

This command requires Ogmios so make sure to have that service on before running it.

This command sets up the DParameter and the Initial Permissioned Candidates list on the main chain.

```shell
./griffin-partner-chains-node wizards setup-main-chain-state
```

It will prompt you to provide the Ogmios instance. Then, as it doesn't find a UTxO with the permissioned candidates, it will ask whether to create the UTxO or not. If you choose to set the UTxO, it will prompt for the signing key and to choose a UTxO for consumption. Afterwards, the wizard will ask for the DParam values. Firstly, for the Permissioned Candidates and then for the Registered ones. After this is finished, the DParam will be set on the mainchain.

Create-chain-spec

This command is different from the original Partner Chain's one, as we've modified it to build the chain specification that we need for Griffin. The functionality is similar in that it reads the `pc-chain-config.json` file to obtain the chain information and builds the chain specification, which in our case is the set of genesis UTxOs. Before running this step we need to have the registered candidates' keys as well, and they need to be added to the permissioned candidate field on the configuration file.

Shell
```
./griffin-partner-chains-node wizards create-chain-spec
```

After reading the candidates list from the configuration, it will prompt you to choose if you want to use it for the genesis.

Example with Devnet

Before starting testing on Cardano, you might find it useful to follow these previous steps connecting with a local devnet, using test keys. The stack comes with both Substrate and Main chain keys for 5 nodes, but in this example we will be using only 3 of them. For this walkthrough we will be using Docker to set up a stack of utilities needed for the setup. We need to have a functioning instance of a Cardano node, connected to DB-sync and PostgreSQL instances, along with an Ogmios instance.

To initialize the stack, you need to navigate to the `dev/local-environment`.

Shell
```
cd dev/local-environment/
```

Here we can do `docker compose up -d`; this might take some time the first time you run it. After it finishes pulling the images and starting the services, it will take a bit to correctly initialize every service. Once every service is initialized, we can start configuring the chain.

We will be starting 3 nodes. The first one will correspond to the chain builder, and the other two will be other permissioned candidates.

Setup

Go back to the repo base path (`cd ../../`)

We will create three directories, one for each node:

```
Shell
mkdir node1 node2 node3
```

We need to copy the node executable to each of these folders.

```
Shell
cp target/release/griffin-partner-chains-node node1
cp target/release/griffin-partner-chains-node node2
cp target/release/griffin-partner-chains-node node3
```

And we also need to create a sub folder `chains` for each node, that will in turn have a `local_testnet` sub folder. This will hold the keystore, db and network key of the chain:

```
Shell
mkdir -p node1/chains/local_testnet
mkdir -p node2/chains/local_testnet
mkdir -p node3/chains/local_testnet
```

We also need to copy the keys for the chain builder. We will copy the keys present in the Cardano folder of the dev environment, as we will use `funded_address` to sign and pay for the partner chain transactions in Cardano.

```
Shell
cp -r dev/local-environment/configurations/cardano/keys node1
```

As the other nodes will be permissioned they won't need to sign any transactions on the Cardano side, so we won't need any keys for them.

1. Generate-keys

Run the `generate-keys` command within each node's sub folder:

```
Shell
cd node1/
```

```
./griffin-partner-chains-node wizards generate-keys
```

Then make sure to appropriately point to the node directory as base path (`./`):

```None
This 🧙 wizard will generate the following keys and save them to your node's
keystore:
→ ecdsa Cross-chain key
→ sr25519 AURA key
→ ed25519 Grandpa key
It will also generate a network key for your node if needed.

> Enter the node base path ./
```

This will generate a set of 3 keys like the following:

```JSON
{
  "partner_chains_key":
"0x031c995372ba274cd5c88ea22e342b6557d36d7a0adf768fb3fcad8debe1282819",
  "keys": {
    "aura":
"0x74691960a862d3fd979b8cd7fbc6802caede3a16850da0a1d972aa292cfe8358",
    "gran":
"0x7ae1e582882c736d524199d6fcd82a3644089702733746bed90d121ca9e8ce80"
  }
}
```

This command creates a `partner-chains-public-keys.json` file that saves the keys. After repeating this step for each node, we will have the set of keys needed for the configuration of the chain.

2. Prepare configuration

This step must be run within our chain builder's node folder, in this case `node1`. We also need to have Ogmios available so if you haven't initialized the stack yet, it's time to do so with `docker compose up -d` on the `dev/local-environment` folder. Once Ogmios is ready to receive connections we can run the wizard:

```
Shell
./griffin-partner-chains-node wizards prepare-configuration
```

First we configure the bootnode and its access point. Make sure to select this directory as the node base path (`./`).

```
None
This 🧙 wizard will:
* establish single bootnode configuration (to be later included in chain-spec
file)
* choose Genesis UTXO on Cardano
* initialize Partner Chains Governance on Cardano
* establish Partner Chains Smart Contracts addresses and policies (to be later
included in chain-spec file)
> Enter the node base path ./
> Do you want to configure a single bootnode with your peer id
'12D3KooWCwp2Mnd9xrTjbbu4voHRBMmdyzTqYwQJh9YBsXvq99LP'? Yes
> Your bootnode should be accessible via: hostname
> Enter bootnode TCP port 3033
> Enter bootnode hostname localhost
Bootnode saved successfully. Keep in mind that you can manually modify
pc-chain-config.json, to edit bootnodes.
```

Then we will set up the genesis UTxO. First we need to configure the Ogmios connection:

```
None
Now, let's set up the genesis UTXO. It identifies the partner chain. This
wizard will query Ogmios for your UTXOs using the address derived from the
payment signing key. This signing key will be then used for spending the
genesis UTXO in order to initialize the chain governance. Please provide
required data.
> Select Ogmios protocol (http/https) http
> Enter the Ogmios hostname localhost
> Enter the Ogmios port 1337
```

Then it will prompt for the signing key file, and make you choose from that address one UTxO to use as Genesis:

```None
> Enter the path to the payment signing key file keys/funded_address.skey
⚙️ Querying UTXOs of
addr_test1vr5vxqpnpl3325cu4zw55tnapjqzzx78pdrnk8k5j7wl72c6y08nd from Ogmios at
http://localhost:1337...
> Select an UTXO to use as the genesis UTXO
f8fbe7316561e57de9ecd1c86ee8f8b512a314ba86499ba9a584bfa8fe2edc8d#0 (1000000000
lovelace)
```

And for the initial governance set:

```None
Please provide the initial chain governance key hashes:
> Enter the space separated keys hashes of the initial Multisig Governance
Authorities 0xe8c300330fe315531ca89d4a2e7d0c80211bc70b473b1ed4979dff2b
> Enter the Initial Multisig Governance Threshold 1
> Governance will be initialized with:
Governance authorities:
       0xe8c300330fe315531ca89d4a2e7d0c80211bc70b473b1ed4979dff2b
Threshold: 1
Do you want to continue? Yes
```

We are using a default address in this case, so the commands' default prompt is good. If you wanted to use another address, you'd need to input the corresponding public key.

Moreover, if you input more than one public key, or if you input a public key that doesn't correspond with the signing key, the wizard will create a multisig governance for the rest of the governance commands. For the sake of simplicity, we won't show this approach here.

After agreeing to continue, the wizard will submit the initial governance transaction:

```None
2025-10-23T17:38:44.073757719-03:00 INFO
partner_chains_cardano_offchain::init_governance - ✉️ Submitter address:
addr_test1vr5vxqpnpl3325cu4zw55tnapjqzzx78pdrnk8k5j7wl72c6y08nd
2025-10-23T17:38:44.073782826-03:00 INFO
partner_chains_cardano_offchain::init_governance - 💱 4 UTXOs available
2025-10-23T17:38:44.240917905-03:00 INFO
partner_chains_cardano_offchain::init_governance - ✅ Transaction submitted.
ID: 611b0f6cbcdf0f137f2018e2774ba4390b96200baec879a3e71a43cb6d4c9e2c
```

```
2025-10-23T17:38:44.241007204-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'611b0f6cbcdf0f137f2018e2774ba4390b96200baec879a3e71a43cb6d4c9e2c#0'
2025-10-23T17:38:49.246943249-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'611b0f6cbcdf0f137f2018e2774ba4390b96200baec879a3e71a43cb6d4c9e2c#0'
2025-10-23T17:38:49.251409241-03:00 INFO
partner_chains_cardano_offchain::await_tx - Transaction output
'611b0f6cbcdf0f137f2018e2774ba4390b96200baec879a3e71a43cb6d4c9e2c'
Governance initialized successfully for UTXO:
f8fbe7316561e57de9ecd1c86ee8f8b512a314ba86499ba9a584bfa8fe2edc8d#0
```

A file will be created as well (`pc-chain-config.json`), with the chain configuration:

```JSON
{
  "bootnodes": [

"/dns/localhost/tcp/3033/p2p/12D3KooWCwp2Mnd9xrTjbbu4voHRBMmdyzTqYwQJh9YBsXvq99
LP"
  ],
  "cardano": {
    "active_slots_coeff": 0.4,
    "epoch_duration_millis": 120000,
    "first_epoch_number": 0,
    "first_epoch_timestamp_millis": 1761249030000,
    "first_slot_number": 0,
    "security_parameter": 5,
    "slot_duration_millis": 1000
  },
  "cardano_addresses": {
    "bridge": {
      "asset": {
        "asset_name": "0x",
        "policy_id":
"0x00000000000000000000000000000000000000000000000000000000"
      },
      "illiquid_circulation_supply_validator_address":
"addr_test1wptv33aalq7dpnkd8fpgnelsxn0sa49h2dqw0wzj3g7e97cnm4p0v"
    },
    "committee_candidates_address":
"addr_test1wrnxhtkrgseddsx90c99c0g6kwprm6l8ztay50qfxwtyamg56ulc7",
    "d_parameter_policy_id":
"0x984d9f65827b0ecaf407725aa8a2bfdd8086ccaf684179a5ece6b21e",
    "governed_map": {
      "policy_id":
"0x72c20375b1429c39cfe01c88065a5110ac7592f31cf57ebcff4d5498",
      "validator_address":
"addr_test1wqnwyett6a4squsfe0zyns65dhtsrf8cjkqug7pg5n0g2ucgnwaj0"
    },
    "permissioned_candidates_policy_id":
"0xa5dce5600d8c898c72d69928db3731c6efd0368bf4402f0a1ae68d2c"
  },
  "chain_parameters": {
    "genesis_utxo":
"f8fbe7316561e57de9ecd1c86ee8f8b512a314ba86499ba9a584bfa8fe2edc8d#0"
  },
  "initial_governance": {
    "authorities": [
      "0xe8c300330fe315531ca89d4a2e7d0c80211bc70b473b1ed4979dff2b"
    ],
    "threshold": 1
  },
  "initial_permissioned_candidates": []
}
```

3. Setup-main-chain-state

Before running this wizard, we need to modify the previously mentioned file to include our permissioned candidates' keys (the keys we generated at the beginning):

```JSON
"initial_permissioned_candidates": [
  {
    "partner_chains_key":
"0x036195477f75452bc298b3bfc9bb3e84f33cbd3fc95c9d2a1fcf8980493f3c0a1a",
    "keys": {
      "aura":
"0x9a18472abce2625ed1c042909c802f5bb28bcb6821b39115be35fdc8ad3fd947",
      "gran":
"0xd9eac2dadcfe5c339a5fe7bd2edb1ed81bd6885ebe6df061fe0459b186894648"
    }
  },
  {
    "partner_chains_key":
"0x036336457437fa2cf806bc830007a24949fa2751f856b67501e2992eea9c155b35",
    "keys": {
      "aura":
"0x503d1c528caa6e251d6f0704c8c123f8dc2b406d76cfde71ee1c1b578df8b734",
      "gran":
"0x47b7b6682795111d59ebefd41fd5261e2a23b756e9a89361e333ab5def653b3e"
    }
  },
  {
    "partner_chains_key":
"0x020aa543bb236cd1f10baf98fdb5d978e0cec5ce9e0ef26e5a202bd6d6ea3fc940",
    "keys": {
      "aura":
"0x2658f7871b44d5bfaec78697672febd7972d1bf80ac13a3235d38e5a6e19692e",
      "gran":
"0xdfa706f73659f51a659a405f622f0bba65a0822e068e230349936d0a1d4eecf6"
    }
  }
]
```

This is an example of what it looks like with the keys I generated.

Once we have this we can run the next wizard (also within `node1/`):

```Shell
./griffin-partner-chains-node wizards setup-main-chain-state
```

You should still have an Ogmios instance running, this is the first thing the wizard asks for:

```None
This wizard will set or update D-Parameter and Permissioned Candidates on the
main chain. Setting either of these costs ADA!
Will read the current D-Parameter and Permissioned Candidates from the main
chain using Ogmios client.
> Select Ogmios protocol (http/https) http
> Enter the Ogmios hostname localhost
> Enter the Ogmios port 1337
```

Since this is the first time running this command for this chain, the permissioned candidates list on chain is empty, and the wizard asks if you want to update it with the candidates on the configuration file. Here we need to use the same signing key that we used for the chain setup.

```
None
List of permissioned candidates is not set on Cardano yet.
> Do you want to set/update the permissioned candidates on the main chain with
values from configuration file? Yes
> Enter the path to the payment signing key file keys/funded_address.skey
2025-10-23T17:49:32.257962365-03:00 INFO
partner_chains_cardano_offchain::permissioned_candidates - There aren't any
permissioned candidates. Preparing transaction to insert.
2025-10-23T17:49:32.391697248-03:00 INFO
partner_chains_cardano_offchain::multisig - 'Insert Permissioned Candidates'
transaction submitted:
b7df75650d53865422534c8e14ea32b386ad4626ad1d79c61626aa7b7d501ab5
2025-10-23T17:49:32.391776440-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'b7df75650d53865422534c8e14ea32b386ad4626ad1d79c61626aa7b7d501ab5#0'
2025-10-23T17:49:37.397929702-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'b7df75650d53865422534c8e14ea32b386ad4626ad1d79c61626aa7b7d501ab5#0'
2025-10-23T17:49:37.401745736-03:00 INFO
partner_chains_cardano_offchain::await_tx - Transaction output
'b7df75650d53865422534c8e14ea32b386ad4626ad1d79c61626aa7b7d501ab5'
Permissioned candidates updated. The change will be effective in two main chain
epochs.
```

Afterwards, it checks for the D-Parameter and again, it's not set. So we will be prompted to fill it out if we want to. Here we set a ratio of 3 permissioned candidates and 0 registered, simply because that's the amount of nodes we will be turning on.

```
None
> Do you want to set/update the D-parameter on the main chain? Yes
> Enter P, the number of permissioned candidates seats, as a non-negative
integer. 3
> Enter R, the number of registered candidates seats, as a non-negative
integer. 0
> Enter the path to the payment signing key file keys/funded_address.skey
2025-10-23T17:50:23.428976515-03:00 INFO
partner_chains_cardano_offchain::d_param - There is no D-parameter set.
Inserting new one.
2025-10-23T17:50:23.572951836-03:00 INFO
partner_chains_cardano_offchain::multisig - 'Insert D-parameter' transaction
submitted: 29b848247023a18cb97ddb39ee8ef3c859b2b34be59160527e3e44a87945155e
2025-10-23T17:50:23.573049815-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'29b848247023a18cb97ddb39ee8ef3c859b2b34be59160527e3e44a87945155e#0'
```

```
2025-10-23T17:50:28.578153078-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'29b848247023a18cb97ddb39ee8ef3c859b2b34be59160527e3e44a87945155e#0'
2025-10-23T17:50:28.585816840-03:00 INFO
partner_chains_cardano_offchain::await_tx - Transaction output
'29b848247023a18cb97ddb39ee8ef3c859b2b34be59160527e3e44a87945155e'
2025-10-23T17:50:28.585873149-03:00 INFO
partner_chains_cardano_offchain::await_tx - Probing for transaction output
'29b848247023a18cb97ddb39ee8ef3c859b2b34be59160527e3e44a87945155e#0'
2025-10-23T17:50:28.589846603-03:00 INFO
partner_chains_cardano_offchain::await_tx - Transaction output
'29b848247023a18cb97ddb39ee8ef3c859b2b34be59160527e3e44a87945155e'
D-parameter updated to (3, 0). The change will be effective in two main chain
epochs.
Done. Please remember that any changes to the Cardano state can be observed
immediately, but from the Partner Chain point of view they will be effective in
two main chain epochs.
```

4. Create-chain-spec

The last step of the wizard helps us create the genesis file that we will need to start the chain. This wizard reads the candidates from the list in the configuration file and adds them to a UTxO in the genesis set for the chain. Within `node1/` run:

```Shell
./griffin-partner-chains-node wizards create-chain-spec
```

Then confirm the provided values:

```None
This wizard will create a genesis.json to use as chain spec, using the
candidates found in the the provided configuration
If the chain includes registered candidates, you need to obtain their keys and
add them to the permissioned candidates list in the configuration as well, to
set up the genesis accordingly. You need to have all the candidate's keys
before moving on, or else they won't be able to participate in the chain.
Chain parameters:
- Genesis UTXO:
f8fbe7316561e57de9ecd1c86ee8f8b512a314ba86499ba9a584bfa8fe2edc8d#0
Candidates:
- Partner Chains Key:
0x036195477f75452bc298b3bfc9bb3e84f33cbd3fc95c9d2a1fcf8980493f3c0a1a, aura:
```

```
0x9a18472abce2625ed1c042909c802f5bb28bcb6821b39115be35fdc8ad3fd947, gran:
0xd9eac2dadcfe5c339a5fe7bd2edb1ed81bd6885ebe6df061fe0459b186894648
- Partner Chains Key:
0x036336457437fa2cf806bc830007a24949fa2751f856b67501e2992eea9c155b35, aura:
0x503d1c528caa6e251d6f0704c8c123f8dc2b406d76cfde71ee1c1b578df8b734, gran:
0x47b7b6682795111d59ebefd41fd5261e2a23b756e9a89361e333ab5def653b3e
- Partner Chains Key:
0x020aa543bb236cd1f10baf98fdb5d978e0cec5ce9e0ef26e5a202bd6d6ea3fc940, aura:
0x2658f7871b44d5bfaec78697672febd7972d1bf80ac13a3235d38e5a6e19692e, gran:
0xdfa706f73659f51a659a405f622f0bba65a0822e068e230349936d0a1d4eecf6
> Do you want to continue? Yes
genesis.json file has been created.
Committee candidates will be found at
0000000000000000000000000000000000000000000000000000000000, with the
corresponding Authorities token
The rest of the UTxOs can be modified to have the genesis set you need.
If you are the governance authority, you can distribute it to the validators.
```

The genesis file, called `new-genesis.json` by the wizard, has to be shared with the rest of the nodes.

```shell
cp new-genesis.json ../node2/
cp new-genesis.json ../node3/
```

5. Start the nodes

First we need to move the `network` and `keystore` folders from each node base path to the corresponding `chains/local_testnet/` folder, as that is where the node will look for them before executing.

```shell
mv node1/network node1/chains/local_testnet/
mv node1/keystore node1/chains/local_testnet/
mv node2/network node2/chains/local_testnet/
mv node2/keystore node2/chains/local_testnet/
mv node3/network node3/chains/local_testnet/
mv node3/keystore node3/chains/local_testnet/
```

To initialize node1, run (within `node1/`):

```
Shell
./griffin-partner-chains-node \
 --validator \
 --chain=new-genesis.json \
 --base-path . \
 --port 3033 \
 --rpc-port=9944 \
 --rpc-cors=all \
 --unsafe-rpc-external \
 --prometheus-port=9615 \
 --prometheus-external
```

To initialize node2, you will need to copy the ID of node1, the bootnode we are connecting to. You can find it in the output of the previous command, or by checking the `pc-chain-config.json` file within node1's base path. In this case it is `12D3KooWCwp2Mnd9xrTjbbu4voHRBMmdyzTqYwQJh9YBsXvq99LP`. Now run (within `node2/`):

```
Shell
./griffin-partner-chains-node \
 --validator \
 --chain=new-genesis.json \
 --base-path . \
 --port 3034 \
 --rpc-port=9945 \
 --rpc-cors=all \
 --unsafe-rpc-external \
 --prometheus-port=9616 \
 --prometheus-external \
 --bootnodes
/dns/localhost/tcp/3033/p2p/12D3KooWCwp2Mnd9xrTjbbu4voHRBMmdyzTqYwQJh9YBsXvq99L
P
```

To initialize node3 connecting to node1, we proceed similarly (within `node3/`):

```
Shell
./griffin-partner-chains-node \
 --validator \
 --chain=new-genesis.json \
 --base-path . \
 --port 3035 \
 --rpc-port=9946 \
 --rpc-cors=all \
 --unsafe-rpc-external \
 --prometheus-port=9617 \
 --prometheus-external \
 --bootnodes
/dns/localhost/tcp/3033/p2p/12D3KooWCwp2Mnd9xrTjbbu4voHRBMmdyzTqYwQJh9YBsXvq99L
P
```

You may need to modify the port numbers if they’re already in use by some other service on your computer, just make sure to change the mentions appropriately.

Restarting the chain

If you wish to scrub the previous chain, you can use the `purge-chain` command:

Shell
```
./griffin-partner-chains-node purge-chain -d .
```

This command removes the database stored at `chains/local_testnet` located in the current directory.

None
```
./griffin-partner-chains-node purge-chain -d .
Are you sure to remove "./chains/local_testnet/paritydb"? [y/N]: y
"./chains/local_testnet/paritydb" removed.
```