Polkadot Runtime Specification

Web3 Foundation

February 2020

1 Polkadot Transactions

Summary

Module	Function	Weight	Description
Attestations	more_attestations	None	Provide candidate receipts
Claims	deposit_event	None	Deposit one of this module's events
Claims	claim	1'000'000	Make a claim
Claims	mint_claim	30'000	Add a new claim, if you are root
Crowdfund	deposit_event	None	Deposit one of this modules's events
Crowdfund	create	100'000	Create a new campaign
Crowdfund	contribute	None	Contribute to a crowd sale
Crowdfund	fix_deploy_data	None	Set the deploy data of funded parachain
Crowdfund	onboard	None	Complete onboarding process
Crowdfund	begin_retirement	None	Retire a fund when it has lost its slot
Crowdfund	withdraw	None	Withdraw full balance
Crowdfund	dissolve	None	Remove a fund
Parachains	set_heads	1'000'000	Provide candidate receipts
Registrar	deposit_event	None	Deposit one of this module's events
Registrar	register_para	5'000,000	Register a parachain with given code
Registrar	deregister_para	10'000	Deregister a parachain
Registrar	set_thread_count	None	Reset the number of parathreads
Registrar	register_parathread	None	Register a parathread for immediate use
Registrar	select_parathread	None	Place a bid to progress parathread
Registrar	deregister_parathread	None	Deregister a parathread
Registrar	swap	None	Swap a parachain with another
Slots	deposit_event	None	Deposit one of this module's events
Slots	new_auction	100'000	Create a new aution
Slots	bid	500'000	Make a new bid for deploying
Slots	bid_renew	500'000	Make a new bid for renewing
Slots	set_offboarding	1'000'000	Set the off-boarding information
Slots	fix_deploy_data	500'000	Set the information for deployment
Slots	elaborate_deploy_data	5'000'000	Note a new parachains's code

1.1 Attestations

1.1.1 more_attestations

Prototype:

```
(func more_attestations
    (param $origin) (param $more) (return dispatch))
```

Weight

Weight: None

Description

Provide candidate receipts for parachains, in ascending order by id.

1.2 Claims

1.2.1 deposit_event

Prototype:

```
(func deposit_event
    (param $origin) (param $more))
```

Weight

Weight: None

Description

Deposit one of this module's events by using the default implementation.

1.2.2 claim

Prototype:

```
(func claim
    (param $origin) (param $dest) (param $sig))
```

Weight

Weight: 1'000'000

Description

Make a claim.

1.2.3 mint_claim

Prototype:

```
(func mint_claim
    (param $origin) (param $who) (param $value) (param $vesting_schedule))
```

Weight

Weight: 30'000

Description

Add a new claim, if you are root.

1.3 Crowdfund

1.3.1 deposit_event

Prototype:

```
(func deposit_event)
```

Weight

Weight: None

Description

Deposit one of this module's events by using the default implementation.

1.3.2 create

Prototype:

```
(func create
    (param $origin) (param $cap) (param $first_slot)
    (param $last_slot) (param $end))
```

Weight

Weight: 100'000

Description

Create a new crowdfunding campaign for a parachain slot deposit for the current auction.

1.3.3 contribute

Prototype:

```
(func contribute
    (param $origin) (param $index) (param $value))
```

Weight

Weight: None

Description

Contribute to a crowd sale. This will transfer some balance over to fund a parachain slot. It will be withdrawable in two instances: the parachain becomes retired; or the slot is.

1.3.4 fix_deploy_data

Prototype:

```
(func fix_deploy_data
    (param $origin) (param $index) (param $code_hash)
    (param $initial_head_data))
```

Weight

Weight: None

Description

Set the deploy data of the funded parachain if not already set. Once set, this cannot be changed again.

- `origin` must be the fund owner.
- `index` is the fund index that origin owns and whose deploy data will be set.
- `code_hash` is the hash of the parachain's Wasm validation function.
- `initial_head_data` is the parachain's initial head data.

1.3.5 onboard

Prototype:

```
(func onboard
    (param $origin) (param $index) (param $para_id))
```

Weight

Weight: None

Description

Complete onboarding process for a winning parachain fund. This can be called once by any origin once a fund wins a slot and the fund has set its deploy data (using `fix_deploy_data`).

- `index` is the fund index that origin owns and whose deploy data will be set.
- `para_id` is the parachain index that this fund won.

1.3.6 begin_retirement

Prototype:

```
(func begin_retirement
    (param $origin) (param $index))
```

Weight

Weight: None

Description

Note that a successful fund has lost its parachain slot, and place it into retirement.

1.3.7 withdraw

Prototype:

```
(func withdraw
    (param $origin) (param $index))
```

Weight

Weight: None

Description

Withdraw full balance of a contributor to an unsuccessful or off-boarded fund.

1.3.8 dissolve

Prototype:

```
(func dissolve
    (param $origin) (param $index))
```

Weight

Weight: None

Description

Remove a fund after either: it was unsuccessful and it timed out; or it was successful but it has been retired from its parachain slot. This places any deposits that were not withdrawn into the treasury.

1.4 Parachains

1.4.1 set_heads

Prototype:

```
(func set_heads
    (param $origin) (param $heads) (return dispatch))
```

Weight

Weight: 1'000'000

Description

Provide candidate receipts for parachains, in ascending order by id.

1.5 Registrar

1.5.1 deposit_event

Prototype:

```
(func deposit_event)
```

Weight

Weight: None

Description

Deposit one of this module's events by using the default implementation.

1.5.2 register_para

Prototype:

```
(func register_para
    (param $origin) (param $id) (param $info)
    (param $code) (param $initial_head_data)
    (return dispatch))
```

Weight

Weight: 5'000'000

Description

Register a parachain with given code. Fails if given ID is already used.

1.5.3 deregister_para

Prototype:

```
(func deregister_para
    (param $origin) (param $id) (return dispatch))
```

Weight

Weight: 10'000

Description

Deregister a parachain with given id.

1.5.4 set_thread_count

Prototype:

```
(func set_thread_count
    (param $origin) (param $count))
```

Weight

Weight: None

Description

Reset the number of parathreads that can pay to be scheduled in a single block.

- `count` is the number of parathreads.

Must be called from Root origin.

1.5.5 register_parathread

Prototype:

```
(func register_parathread
    (param $origin) (param $code) (param $initial_head_data)
```

Weight

Weight: None

Description

Register a parathread for immediate use. Must be sent from a Signed origin that is able to have ParathreadDeposit reserved. `code` and `initial_head_data` are used to initialize the parathread's state.

1.5.6 select_parathread

Prototype:

```
(func select_parathread
    (param $origin) (param $id)
    (param $collator) (param $head_hash))
```

Weight

Weight: None

Description

Place a bid for a parathread to be progressed in the next block. This is a kind of special transaction that should be heavily prioritized in the transaction pool according to the ‘value‘; only ‘ThreadCount‘ of them may be presented in any single block.

1.5.7 deregister_parathread

Prototype:

```
(func deregister_parathread
    (param $origin))
```

Weight

Weight: None

Description

Deregister a parathread and retrieve the deposit. Must be sent from a Parachain `origin` which is currently a parathread. Ensure that before calling this that any funds you want emptied from the parathread's account is moved out; after this it will be impossible to retrieve them (without governance intervention).

1.5.8 swap

Prototype:

```
(func swap
    (param $origin) (param $other))
```

Weight

Weight: None

Description

Swap a Parachain with another Parachain or parathread. The origin must be a Parachain. The swap will happen only if there is already an opposite swap pending. If there is not, the swap will be stored in the pending swaps map, ready for a later confirmatory swap. The `ParaId`‘s remain mapped to the same head data and code so external code can rely on `ParaId` to be a long-term identifier of a notional Parachain. However, their scheduling info (i.e. whether they're a parathread or parachain), auction information and the auction deposit are switched.

1.6 Slots

1.6.1 deposit_event

Prototype:

```
(func deposit_event)
```

Weight

Weight: None

Description

Deposit one of this module's events by using the default implementation.

1.6.2 new_auction

Prototype:

```
(func new_auction
    (param $origin) (param $duration) (param $lease_period_index))
```

Weight

Weight: 100'000

Description

Create a new auction. This can only happen when there isn't already an auction in progress and may only be called by the root origin. Accepts the 'duration' of this auction and the 'lease_period_index' of the initial lease period of the four that are to be auctioned.

1.6.3 bid

Prototype:

```
(func bid
    (param $origin) (param $sub) (param $auction_index)
    (param $first_slot) (param $last_slot) (param $amount))
```

Weight

Weight: 500'000

Description

Make a new bid from an account (including a parachain account) for deploying a new parachain. Multiple simultaneous bids from the same bidder are allowed only as long as all active bids overlap each other (i.e. are mutually exclusive). Bids cannot be redacted.

- `sub` is the sub-bidder ID, allowing for multiple competing bids to be made by (and funded by) the same account.
- `auction_index` is the index of the auction to bid on. Should just be the present value of 'AuctionCounter'.
- `first_slot` is the first lease period index of the range to bid on. This is the absolute lease period index value, not an auction-specific offset.

- `last_slot` is the last lease period index of the range to bid on. This is the absolute lease period index value, not an auction-specific offset.
- `amount` is the amount to bid to be held as deposit for the parachain should the bid win. This amount is held throughout the range.

1.6.4 bid_renew

Prototype:

```
(func bid_renew
    (param $origin) (param $auction_index) (param $first_slot)
    (param $last_slot) (param $amount))
```

Weight

Weight: 500'000

Description

Make a new bid from a parachain account for renewing that (pre-existing) parachain. The origin **must** be a parachain account. Multiple simultaneous bids from the same bidder are allowed only as long as all active bids overlap each other (i.e. are mutually exclusive). Bids cannot be redacted.

- `auction_index` is the index of the auction to bid on. Should just be the present value of `AuctionCounter`.
- `first_slot` is the first lease period index of the range to bid on. This is the absolute lease period index value, not an auction-specific offset.
- `last_slot` is the last lease period index of the range to bid on. This is the absolute lease period index value, not an auction-specific offset.
- `amount` is the amount to bid to be held as deposit for the parachain should the bid win. This amount is held throughout the range.

1.6.5 set_offboarding

Prototype:

```
(func set_offboarding
    (param $origin) (param $dest))
```

Weight

Weight: 1'000'000

Description

Set the off-boarding information for a parachain. The origin **must** be a parachain account.

- `dest` is the destination account to receive the parachain's deposit.

1.6.6 fix_deploy_data

Prototype:

```
(func fix_deploy_data
    (param $origin) (param $sub) (param $para_id)
    (param $code_hash) (param $initial_head_data))
```

Weight

Weight: 500'000

Description

Set the deploy information for a successful bid to deploy a new parachain.

- `origin` must be the successful bidder account.
- `sub` is the sub-bidder ID of the bidder.
- `para_id` is the parachain ID allotted to the winning bidder.
- `code_hash` is the hash of the parachain's Wasm validation function.
- `initial_head_data` is the parachain's initial head data.

1.6.7 elaborate_deploy_data

Prototype:

```
(func elaborate_deploy_data
    (param $origin) (param $para_id) (param $code))
```

Weight

Weight: 5'000'000

Description

Note a new parachain's code. This must be called after `fix_deploy_data` and `code` must be the preimage of the `code_hash` passed there for the same `para_id`. This may be called before or after the beginning of the parachain's first lease period. If called before then the parachain will become active at the first block of its starting lease period. If after, then it will become active immediately after this call.

- `origin` is irrelevant.
- `para_id` is the parachain ID whose code will be elaborated.
- `code` is the preimage of the registered `code_hash` of `para_id`.

2 Weights

Polkadot blocks have a limited window for block producers to create a block, limited amount of data that can be included per block and overall practical limit to storage footprint of the blockchain. Therefore, Polkadot has introduced a Weight system that tells block producers how "heavy" an extrinsic is. Given the maximum block weight, and the weight of the individual extrinsic in a transaction pool, the block producer can select a set of extrinsics that allows them to saturate the block while not going over the limits.

Additionally, Polkadot has introduced an available block ratio which ensures that only a portion of the total maximum block weight is used for regular transactions. This also introduces the concept of operational transactions which are system critical operations that can use the rest of the available block weight. TODO: define operational transactions

Example: considering the maximum block weight is 1'000'000, the weight of individual transactions is 1'000 and the available block ration is 20%, about 200 transfers per block could be possible.

```
1'000'000 * .20 / 1'000 = 200 transfers per block
```

2.1 Fees

To bring the weight system to the users of Polkadot, a tightly couples fee system is introduced. Users will pay a transaction fee proportional to the weight of the call they are making.

```
total_fee = base_fee + length_fee + weight_fee
```

TODO: explain more about fees: base fee, length, define types, etc.

2.2 Calculating weights

Weights should be accurately calculated relatively to the runtime functions computational and resource complexity.

Following operations must be considered:

- Storage Operations (read, write, mutate, etc.)
- Codec Operations (serializing/deserializing, etc.)
- Search / Sort / Notable computational
- Calls to other functions

2.2.1 Storage Operations

2.2.2 Codec Operations

2.2.3 Search / Sort / Notable Operations

2.2.4 Calls to other functions

2.3 Combining the Data