Scheduled Checkpoint

- **Introduced in:** 2.0.0 (Experimental)
- **Contract name: ScheduledCheckpoint.sol**
- **Type:** Mixed - Transfer Manager and Checkpoint

How it works

Allows a user to schedule checkpoints that will be created in the future.

Checkpoints can be scheduled by specifying the schedule name, the start time of the schedule and the interval of the schedule.

This is useful if you want to guarantee having a checkpoint created every Monday at 12:00 (so that this checkpoint can be used for dividends or voting) but don't want to have to remember to manually create a checkpoint each Monday, or run the risk that it is not created at exactly the correct time.

The module works by checking, every time a token transfer occurs, whether there are any pending checkpoints that have been scheduled. This means that whilst the wall-clock time that the checkpoint is created may not correspond to the scheduled time, the logical time (with respect to token balances) will do.

A consequence of this approach is that if there have been no token transfers following a scheduled checkpoint, the checkpoint will not be created (until there is a token transfer). The module provides an `updateAll` and `update` function that can be used to manually trigger and pending updates in this scenario.

NB - since in this implementation every schedule is checked for every token transfer, having a large number of schedules will increase gas costs for all token transfers. This module is designed as an experimental module to give an indication of how this functionality could be built, rather than being used as is.

Key functionalities (as defined in the Smart Contract)

Initialization

There is no initialisation for this module

Creating Schedules

To create a schedule the function:
`function addSchedule(bytes32 _name, uint256 _startTime, uint256 _interval) external`
can be used.

A checkpoint will be created at `_startTime`, and every `_interval` thereafter (both specified in seconds) until the schedule is removed.

Removing Schedules

To remove a schedule, the function:
`function removeSchedule(bytes32 _name) external`
can be used.

Once removed any future checkpoints associated with this schedule will not be created.

Updating Schedules

Since the ScheduledCheckpoint contract only checks and updates checkpoints every time a transfer is made, it is possible that a checkpoint will be overdue (past the wall-clock time it is as-of). In some cases this is fine as it means that querying the current balances of the token contract will be the logical equivalent of querying at the missing checkpoint (since by definition no token transfers have taken place since the checkpoint should have been created) but the contract provides methods that can be used to force update all pending checkpoints.

To update a single named checkpoint, you can use:
`function update(bytes32 _name) external onlyOwner`
and to update all checkpoints you can use:
`function updateAll() onlyOwner external`

Schedule Information

To get information about a schedule, the function:

`function getSchedule(bytes32 _name) view external returns(bytes32, uint256, uint256, uint256, uint256[], uint256[], uint256[])`
can be used.

This returns the data:

```
        schedules[_name].name,
        schedules[_name].startTime,
        schedules[_name].nextTime,
        schedules[_name].interval,
        schedules[_name].checkpointIds,
        schedules[_name].timestamps,
        schedules[_name].periods
```

The last three return arguments provide a list of checkpoints that have been created using this schedule, and the associated checkpoint ids. If, between any scheduled checkpoints, no token transfers take place, the `periods` value will record the number of empty periods across which the corresponding `checkpointIds` value is valid.

Special considerations / notes

Experimental module - designed as a proof of concept not for direct usage.

Troubleshooting / FAQs

None

Know Issues / bugs

None