Capped STO

- **Contract name:** CappedSTO.sol
- **Type:** STO
- **Compatible Protocol Version:** TBD
- **Associated LucidChart**: TBD

How it Works

Summary: The Capped STO module allows issuers to set up a capped STO for their token distribution. The Capped STO sets a limit on the funding accepted and the number of tokens that will be distributed. Capped STOs are the most typical when it comes to what issuers want for STO structure used in a token sale.

Key Functionalities (as defined in the Smart Contract)

Configure

Summary: The Configure function is used to initialize all of the capped STO contract variables.
The function asks for:

1. The _startTime of the offering
2. The _endTime at which offering ends
3. The_cap Maximum for the number of tokens for sale
4. The _rate of token units a buyer gets per wei / base unit of POLY
5. The_fundRaiseTypes - the type of currency accepted to collect the funds of the STO
6. The _fundsReceiver Ethereum account address to hold the funds raised from the STO.

Other conditions:

1. Requires that the rate of the token is great than zero.
2. Requires that the receiver of funds cannot be the `0x0` address
3. The start date is later than or equal to the present moment
4. The end date is later than the start date
5. The cap should be great that zero
6. The issuer can only select one type of fundraise type.

```
/**
 * @dev Function used to intialize the contract variables
 * @param _startTime Unix timestamp at which offering get star
ted
 * @param _endTime Unix timestamp at which offering get ended
 * @param _cap Maximum No. of tokens for sale
 * @param _rate Token units a buyer gets per wei / base unit o
f POLY
 * @param _fundRaiseTypes Type of currency used to collect the
funds
 * @param _fundsReceiver Ethereum account address to hold the
funds
*/

    function configure(
        uint256 _startTime,
        uint256 _endTime,
        uint256 _cap,
        uint256 _rate,
      FundRaiseType[] _fundRaiseTypes,
        address _fundsReceiver
    )
```

Get In it Function

Summary: This function will return the signature of the above configure function.

```
/**
* @dev This function returns the signature of configure function
*/
    function getInitFunction() public pure returns (bytes4) {
        return bytes4(keccak256("configure(uint256,uint256,uint256,uint256,uint8[],address)"));
    }
```

changeAllowBeneficialInvestments

Summary: This function allows the issuer to set the allowed beneficiary to different than the funder. The only input parameter a boolean to allow or disallow beneficial investments.

```
/**
* @dev Function to set allowBeneficialInvestments (allow beneficiary to be different to funder)
* @param _allowBeneficialInvestments Boolean to allow or disallow beneficial investments
*/


    function changeAllowBeneficialInvestments(bool _allowBeneficialInvestments) public onlyOwner {
        require(_allowBeneficialInvestments != allowBeneficialInvestments, "Does not change value");
        allowBeneficialInvestments = _allowBeneficialInvestments;
        emit SetAllowBeneficialInvestments(allowBeneficialInvestments);
    }
```

buyTokens

Summary: This function is a low level token purchase and should never be overridden. The parameter for this function is the address of the beneficiary who will be performing the token purchase from the STO.

```
/**
* @dev Low level token purchase ***DO NOT OVERRIDE***
* @param _beneficiary Address performing the token purchase
*/
    function buyTokens(address _beneficiary) public payable no
nReentrant {
        if (!allowBeneficialInvestments) {
            require(_beneficiary == msg.sender, "Beneficiary a
ddress does not match msg.sender");
        }

        require(!paused, "Should not be paused");
        require(fundRaiseTypes[uint8(FundRaiseType.ETH)], "Mod
e of investment is not ETH");

        uint256 weiAmount = msg.value;
        _processTx(_beneficiary, weiAmount);

        _forwardFunds();
        _postValidatePurchase(_beneficiary, weiAmount);
    }
```

buyTokensWithPoly

Summary: This function is also for low level token purchasing. The only parameter this function accepts is for the amount of POLY wanted to invest in the STO.

```
/**
* @dev low level token purchase
* @param _investedPOLY Amount of POLY invested
*/
    function buyTokensWithPoly(uint256 _investedPOLY) public n
onReentrant{
        require(!paused, "Should not be paused");
        require(fundRaiseTypes[uint8(FundRaiseType.POLY)], "Mo
de of investment is not POLY");
        _processTx(msg.sender, _investedPOLY);
        _forwardPoly(msg.sender, wallet, _investedPOLY);
        _postValidatePurchase(msg.sender, _investedPOLY);
    }
```

capReached

Summary: This function simply checks whether or not the STO cap has been reached. It returns a simple yes or no.

```
/**
* @dev Checks whether the cap has been reached.
* @return bool Whether the cap was reached
*/
    function capReached() public view returns (bool) {
        return totalTokensSold >= cap;
    }
```

getTokensSold

Summary: This function returns the total number of tokens sold during/after the capped STO.

```
/**
* @dev Return the total no. of tokens sold
*/
    function getTokensSold() public view returns (uint256) {
        return totalTokensSold;
    }
```

getPermissions

Summary: This function returns the permissions flag that are associated with the capped STO.

```
/**
* @dev flags associated with STO
*/
    function getPermissions() public view returns(bytes32[]) {
        bytes32[] memory allPermissions = new bytes32[](0);
        return allPermissions;
    }
```

getSTODetails

Summary: This function returns all of the details for issuers to review about their Capped STO.

When the getSTODetails function is called it will return:

1. The start time of the STO (unixtimestamped)
2. The end time of the STO (unixtimestamped)
3. The number of tokens the capped STO will be allowed to sell to investors during the sale.

4. The overall amount of funds raised
5. The number of investors the STO had/has
6. The total amount of tokens that were sold
7. A notice saying wether the fund raise type was in POLY or Not (returns a simple YES or NO)

```
/**
* @dev Return the STO details
* @return Unixtimestamp at which offering gets started.
* @return Unixtimestamp at which offering ends.
* @return Number of tokens this STO will be allowed to sell to investors.
* @return Amount of funds raised
* @return Number of individual investors this STO have.
* @return Amount of tokens get sold.
* @return Boolean value to justify whether the fund raise type is POLY or not, i.e true for POLY.
*/

    function getSTODetails() public view returns(uint256, uint256, uint256, uint256, uint256, uint256, uint256, bool) {
        return (
            startTime,
            endTime,
            cap,
            rate,
            (fundRaiseTypes[uint8(FundRaiseType.POLY)]) ? fundsRaised[uint8(FundRaiseType.POLY)]: fundsRaised[uint8(FundRaiseType.ETH)],
            investorCount,
            totalTokensSold,
            (fundRaiseTypes[uint8(FundRaiseType.POLY)])
        );
```

```
    }
```

// Internal interface (extensible) //

_ProcessTx

Summary: This function is for processing the purchases from investors addresses as well as verifying the required validations.

```
/**
* @param _beneficiary Address performing the token purchase
* @param _investedAmount Value in wei involved in the purchase
*/
    function _processTx(address _beneficiary, uint256 _investe
dAmount)

  }
```

_pre Validate Purchase

Summary: This function is for validating an incoming purchase. The function requires statements in order to revert the state when conditions are not met. Use super to concatenate validations.
It requires a few inputs:

- That the beneficiary address cannot be the zero address (0x0..)
- That the amount invested cannot be equal to 0.
- That investments must be less than or equal to the cap of the STO

```
/**
* @notice Validation of an incoming purchase.
* @param _beneficiary Address performing the token purchase
```

```
* @param _investedAmount Value in wei involved in the purchase
*/
    function _preValidatePurchase(address _beneficiary, uint25
6 _investedAmount)

}
```

_post Validate Purchase

Summary: This function is used for validation of an executed investor purchase. It observes the state and is used to revert statements to undo rollback when valid conditions are not met.

```
/**
* @notice Validation of an executed purchase.
*/
    function _postValidatePurchase(address /*_beneficiary*/, u
int256 /*_investedAmount*/)

    }
```

_deliver Tokens

Summary: This function is the source of tokens that can be purchased. The function works to override this method to modify the way in which the STO sale ultimately gets and sends its tokens out.

```
/**
* @notice Source of tokens.
* @param _beneficiary Address performing the token purchase
* @param _tokenAmount Number of tokens to be emitted
*/
```

```
    function _deliverTokens(address _beneficiary, uint256 _tok
enAmount) internal {
        require(ISecurityToken(securityToken).mint(_beneficiar
y, _tokenAmount), "Error in minting the tokens");
    }
```

_processPurchase

Summary: This function is executed when an investor purchase has been validated and is ready to be executed.

```
/**
* @notice Executed when a purchase has been validated and is r
eady to be executed. Not necessarily emits/sends tokens.* @par
am _beneficiary Address receiving the tokens
* @param _beneficiary Address receiving the tokens
* @param _tokenAmount Number of tokens to be purchased
*/
    function _processPurchase(address _beneficiary, uint256 _t
okenAmount)


}
```

_update Purchase State

Summary: This function works by overriding for extensions that require an internal state check for validation such as current user contributions, etc...

```
/**
* @notice Overrides for extensions that require an internal st
ate to check if valid
*/
```

```
    function _updatePurchasingState(address /*_beneficiary*/,
uint256 /*_investedAmount*/)


}
```

_get Token Amount

Summary: This function works by overriding in order to extend the way in which ETH is converted to tokens. It takes the input value in wei and converts its to tokens and then returns the amount of tokens that can be purchased with the specified investment amount.

```
/**
* @notice Overrides to extend the way in which ether is conver
ted to tokens.
* @param _investedAmount Value in wei to be converted into tok
ens
* @return Number of tokens that can be purchased with the spec
ified _investedAmount
    */
    function _getTokenAmount(uint256 _investedAmount)


}
```

_forward Funds

Summary: This function simply determines how ETH stored and/or forwarded on investor purchases.

```
/**
* @notice ETH is stored/forwarded on purchases.
*/
    function _forwardFunds() internal {
```

```
        wallet.transfer(msg.value);
    }
```

_forwardPoly

Summary: This function is an internal function that is used to forward the amount of POLY that has been raised to the beneficiary address.

```
/**
* @notice forward the POLY raised to beneficiary address
* @param _beneficiary Address of the funds reciever
* @param _to Address who wants to ST-20 tokens
* @param _fundsAmount Amount invested by _to
*/
    function _forwardPoly(address _beneficiary, address _to, u
int256 _fundsAmount


}
```

Special considerations / notes

None

Troubleshooting / FAQs

None

Know Issues / bugs

None

Changelog

(Pre-v1.4.0)

Added:

- CappedSTO` is now Configurable by choosing the type of fundraise. Either POLY or ETH.
- CappedSTO` takes 3 more constructor arguments fundRaiseType (uint8), the address of the polyToken & address of the fund's receiver.
- buyTokensWithPoly(address _beneficiary, uint256 _investedPoly)` new function added in cappedSTO to facilitate the funds raising with the POLY.

Changed/Fixed:

- Modified CappedSTOFactory to comply with minor interface changes in iSTO. It now uses a mapping named `fundRaiseType` to specify the fundraise type (ETH / POLY)
- Change the `setupCost` of `cappedSTOFactory` from `0 POLY` to `20K POLY`.

Removed:

- n/a

(2.1.0 Updates)

Fixed

- CappedSTO now supports non-divisible tokens. **For example,** if the investor sends 1 ETH and that would yield 200.5 tokens, instead of reverting it mints 200 tokens and gives the ETH difference back for the 0.5 tokens not purchased.
- This STO module now allows issuers to set price of their token higher than 1 ETH/token. The rate is now assumed to be multiplied by 10^18 by the smart contracts.