SET V2

Smart Contracts. Audit

Mikhail Vladimirov and Dmitry Khovratovich

25th August 2020

This document describes the audit process of the SET smart contracts performed by ABDK Consulting.

1. Introduction

We've been asked to review the Set V2 smart contracts given in separate files.

2. IssuanceModule

In this section we describe issues found in the IssuanceModule.sol.

2.1 Critical Flaws

This section lists critical flaws, which were found in the smart contract.

1. ~~Line 146: the calling of the `executeIssuanceDeposit` may internally increase position balances. It happens not just by sending component tokens to Set token smart contracts, but in a more profitable way for the caller, for example by repaying a flash loan. This will allow the caller to use both component tokens in some profitable way and also the same component tokens to issue Set tokens. The `nonReentrant` modifier doesn't help here, as internal transactions may interact with the same token through some other module. Here is the negative scenario:~~
    - ~~Attacker invokes specially crafted smart contract A.~~
    - ~~A obtains a flash loan of component tokens from a Set token smart contract.~~
    - ~~A invokes issueFromContract on issuance module, providing A's address as `_issueContract`.~~
    - ~~The issuance module calls the `executeIssuanceDeposit` function on A.~~
    - ~~A repays flash loan, thus increasing component balances of the Set token.~~

- A returns control to the issuance module.
- The Issuance module verifies that component balances were increased, mints Set tokens and send them to the attacker.

2. Line 148: there is no guarantee that the Set still has the same components here as it had two lives above, so comparing positions with components addresses could lead to unexpected results.
3. Line 150: the `validateExpectedComponentIncrease` check doesn't take the `int` account that Set token may have several positions for the same component.

2.2 Documentation Issues

This section lists documentation issues, which were found in the smart contract.

1. Line 52, 53: the events are usually named via nouns, such as `SetTokenIssuance`.
2. Line 229: the module is called the `IssuanceModule` not the `BasicIssuanceModule`.

2.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 52, 53: the `_to` and `_redeemer` parameter should probably be indexed to make it easier to track token balance changes.
2. Line 249: the returning single array of structs with two fields instead of `address[] memory, uint256[] memory` would be more efficient.
3. Line 313 315: in the code there should be the check for arrays that they have the same length.

2.4 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision

1. Line 124: it is unclear why an alternative deposit mechanism is needed.The smart contract should have no problems using the `issue` function.
2. Line 186: the redeeming is not possible in case any of the components are not in default state. It is also true for the components that are not currently in default state. Is it fine?
3. Line 342: perhaps, would it be better to use strict inequality instead of the `>=`.

2.5 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 115: there should be `deposit` instead of `deposits`.
3. Line 168: unlike `issue` function, this function doesn't have the `_to` parameter, so component tokens are always sent to `msg.sender`. Consider adding `_to` parameter for consistency.

3. PriceOracle

In this section we describe issues found in the PriceOracle.sol.

3.1 Documentation Issues

This section lists documentation issues, which were found in the smart contract.

1. Line 45-50: events are usually named using nouns. Here some examples:
    - `NewPair` or `PairAddition` instead of `PairAdded`
    - `PairRemoval` instead of `PairRemoved`
    - `PairChange` or `PairEdit` instead of `PairEdited`
    - `NewAdapter` or `AdapterAddition` instead of `AdapterAdded`
    - `AdapterRemoval` instead of `AdapterRemoved`
    - `MasterQuoteAssetChange` or `MasterQuoteAssetEdit` instead of `MasterQuoteAssetEdited`
2. Line 45-50: next parameters should be indexed:
    - `_assetOne`
    - `_assetTwo`
    - `_oracle`
    - `_newOracle`
    - `_oldOracle`
    - `_adapter`
    - `_newMasterQuote`
    - `_oldMasterQuote`
3. Line 64: the comment is incorrect. There should be the `IOracleAdapter` instead of the `address`. Also the storing the `adapters` in an array is suboptimal, as checking for whether a given address is in the adapters list requires linear scan over storage. Consider turning into a mapping.

3.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 79: the constructor may add some adapters, pairs, and sets master quote asset, but doesn't emit corresponding events. This makes it impossible to reconstruct the smart contract's state from the event log. Consider logging corresponding events in the constructor.
2. Line 83-85: passing a single array of structs with three fields would be more efficient and would make length checks unnecessary.
3. Line 98, 153, 171: the operation `oracles[_assetOnes[i]][_assetTwos[i]]` may overwrite an already registered oracle. Consider adding an explicit check for whether adapter is already set, before setting it. Also, it is possible to set zero oracle address in the `_oracles[i]` while zero address has special meaning. Consider adding an explicit check that oracle address is not zero.
4. Line 170, 187: the value `oracles[_assetOne][_assetTwo]` was just read from the storage in the previous statement. Consider reading once and caching in a local variable.
5. Line 200, 215: the `adapters.contains(_adapter)` call searches through a storage array which may consume lots of gas. Consider using mapping instead of array.
6. Line 203, 218: the line is deadly inefficient, as it reads all the adapters into memory, appends the new adapter, and then writes all adapters back into storage.
7. Line 232: the event will be emitted even if the new master quote asset is the same as the current one.
8. Line 240: the `getAdapters` function is redundant, as the adapter's storage variable is already public.
9. Line 264: the `oracles[_assetTwo][_assetOne]` should be read only if directOracle was not found.
10. Line 304: the `_getDirectOrInversePrice(_assetTwo, masterQuoteAsset)` should be called only if priceFoundOne is true.

3.3 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision

3.4 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 45: there is no good reason to prefix the event parameter names like `_assetOne` with underscore. These names may not clash with state variable names or any other identifiers. Consider removing underscores.

4. InvokeLib

In this section we describe issues found in the InvokeLib.sol

4.1 Moderate Flaws

This section lists moderate flaws, which were found in the smart contract.

1. ~~Line 55, 77: the returned value is ignored. Unsuccessful calls may be treated as a successful one.~~

4.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. ~~Line 98: the `_quantity > 0` skips zero-quantity transfer, but doesn't skip trivial transfers having `_setToken == _to`. Consider skipping such transfers as well, as they are anyway not handled properly by the code below.~~
2. ~~Line 109: the check `newBalance == existingBalance.sub(_quantity)` will most probably fail in the case `_setToken == _to`.~~

4.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 32: unlike most of the libraries in the same repository, the name of this library ends with `Lib` suffix. This may confuse people, as they could think that other libraries, named without such suffix, are not libraries. Consider using consistent naming.~~
3. ~~Line 39, 60: the comment is incorrect. This is not enforced in the function. Consider removing this statement from the documentation.~~
4. ~~Line 48, 69, 92: the `address` should be IERC20.~~
5. ~~Line 83: the statement is confusing. Consider rewriting it.~~

5. Controller

In this section we describe issues found in the Controller.sol.

5.1 Moderate Flaws

This section lists moderate flaws, which were found in the smart contract.

1. Line164: the same ID could be specified for several different resources. The contract will map such ID only to the latest such resource, leaving all other resources with no IDs assigned. Consider checking that `resourceId[_resourceIds[i]]` is zero before setting it.
2. Line 256, 316, 329, 351, 366: the next lines is deadly inefficient:
   - `sets = sets.append(_setToken)`
   - `factories = factories.append(_factory)`
   - `modules = modules.append(_module)`
   - `modules = modules.remove(_module)`
   - `resources = resources.append(_resource)`

   They read the `registered sets`, the `factories`,the `modules` and the `resources` from storage into memory, then copy them from one in-memory array into another, appending new `registered sets`, `factories`, `modules` and `resources`, and then write all `registered sets`, `factories`, `modules` and `resources` back into storage. Efficient way to remove an element from a storage array is to copy the last array element into the slot occupied by the element that ought to be removed, and then decrease array length by one.
3. Line 301, 331, 370: the `isFactory`, the `isModule` and the `isResource` functions remove only the first occurrence of given factory from factories array, other occurrences may still exist, so setting to false the `isFactory[_factory]`, the `isModule[_module]` and the `isResource[resourceToRemove]` could make the contract's state inconsistent.

5.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 34: the `Controller` contract should implement the `IController` interface.
2. Line 39: the event definitions should probably be moved into the `IController` interface.
3. Line 41-50: next parameters should be indexed: `_factory`, `_oldFeeRecipient`, `_module`, `_resource`, `_id`, `uint256 _id`, `address _resource`, `_setToken`.
4. Line 80-88: the arrays look redundant. The contract itself only uses isXXX mappings, and for on-chain applications it would be too expensive to iterate over these arrays. Off-chain application may query events to obtain the whole list of sets, factories, modules, or resources. Consider removing these arrays.

5. Line 91-94: the mappings could be replaced with a single mapping from an address to a bit mask, with separate bits for the `isSet`,the `isFactory`, the `isModule`, and the `isResource` flags. This way would make cheaper tests such as in the `onlyModuleOrResource` or the `isSystemContrac`.
6. Line 108: no access level specified for the `isInitialized` storage variable. So internal access level will be used by default. Consider explicitly specifying access level.
7. Line 132:the `initialize` function doesn't emit the `FactoryAdded`, the `ModuleAdded`, or the `ResourceAdded` events, thus making it impossible to reconstruct the full lists of factories, modules, and resources from events. Consider emitting events from this function.
8. Line 135-136: passing the `_resources` and the `_resourceIds` as a single array of structs would make calls cheaper and would make length check unnecessary.
9. Line 154, 158, 163: the same factory address could be added several times. Consider checking that the `isFactory[_factories[i]]`, the `isModule[_modules[i]]` and the `isResource[_resources[i]]` is false before setting it to true.
10. Line 162: the `_resources.length == _resourceIds.length` check should be put outside the loop.
11. Line 190: the `_quantity > 0` check is already performed by `ExplicitERC20.transferFrom`. Probably no reason to use this check.
12. Line 224: perhaps there is no need to forbid empty batches. Note, that zero-quantity transfers are allowed.
13. Line 385, 402: the `fees[_module][_feeType] == 0` check is not reliable as zero seems to be a valid fee value.
14. Line 389, 406: the `FeeEdited` event is emitted even if the new fee percentage is zero.
15. Line 399, 418: the `editFee` function differs from the `addFee` only in precondition it checks. However, the check is non-reliable. Consider merging these two functions into one.
16. Line 389, 406, 418: the `FeeEdited` and the `FeeRecipientChanged` events are emitted even when the new fee percentage is the same as the current one.
17. Line 423, 434, 438, 442, 446: the `fee` mapping is already public. The next functions are redundant: `getModuleFee`, `getFactories`, `getModule`, `getResources`, `getSets`.

5.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 23, 36: the `SafeMath.sol` file is not used. Consider removing it.
3. Line 73: the word `If` in the `onlyIfInitialized` modifier looks redundant.
4. Line 181, 210: the `address` should be IERC20.

5. Line 41-50: the events are usually named with nouns. Here are some examples:
    - `NewFactory` or `FactoryAddition` instead of `FactoryAdded`
    - `FactoryRemoval` instead of `FactoryRemoved`
    - `FeeChange` instead of `FeeEdited`
    - `NewFeeRecipient` or `FeeRecipientChange` instead of `FeeRecipientChanged`
    - `NewModule` or `ModuleAddition` instead of `ModuleAdded`
    - `ModuleRemoval` instead of `ModuleRemoved`
    - `NewResource` or `ResourceAddition` instead of `ResourceAdded`
    - `ResourceRemoval` instead of `ResourceRemoved`
    - `NewSet`, `SetCreation` or `SetAddition` instead of `SetAdded`
    - `SetRemova` instead of `SetRemoved`
6. Line 41-50: there is no good reason to prefix event parameter names with underscore, as these names may not clash with state variable names or any other identifiers. Consider removing underscores.

6. SetToken

In this section we describe issues found in the SetToken.sol.

6.1 Moderate Flaws

This section lists moderate flaws, which were found in the smart contract.

1. Line 154: it is possible to specify the same component address several times, which will lead to a Set token with several positions sharing the same component address. This Set token will not work properly.
2. Line 213: perhaps, there should be `positions.length` instead of `positions.length.add(1)`.

6.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 39: the `SetToken` contract should implement `ISetToken` interface.
2. Line 44: the next events should probably be moved into the `ISetToken` interface: `Invoked`, `ModuleAdded`, `ModuleRemoved`, `ModuleInitialized`, `ManagerEdited`, `PositionChanged`. Also events usually named via nouns. Here some examples:
    - `Invocation`
    - `NewModule` or `ModuleAddition`
    - `ModuleRemoval`

- ~~`ModuleInitialization`~~
- ~~`NewManager` or `ManagerChange`~~
- ~~`PositionChange`~~

3. ~~Line 50, 52: the `_oldManager` and the `_index` parameters are probably redundant.~~
4. ~~Line 113: the `isLocked` variable is probably redundant.The locker `!= address(0)` could probably be used instead.~~
5. ~~Line 132-133, 252-253, 295-296: passing a single array of struct with two fields would be more efficient and would make length check unnecessary. Also, it is not checked that these arrays are of the same size (for the line 132-133).~~
6. ~~Line 175-176: two modifiers often go together. Consider wrapping them into one modifier.~~
7. ~~Line 234, 277: the `_validateInputtedIndex(_index, positions.length)` and the `_validateInputtedIndex(_index, positions.length)` checks are redundant. It will be checked by the compiler in the next statement anyway.~~
8. ~~Line 194, 397: the `ManagerEdited` and the `PositionChanged` will be emitted even if the new unit is the same as the current one too. The `PositionChanged` also logs all position attributes, even though most of them didn't change. Consider having separate event for unit change.~~
9. ~~Line 268: the line `modules = modules.remove(_module)` is deadly inefficient. It reads all the modules into memory, removes one module, and then writes all modules back into storage.~~
10. ~~Line 402: the `getPositions` and the `getModules` functions are redundant as positions are already public.~~
11. ~~Line 413: the name of the function `isModule` is confusing, as this function does not treat pending modules as modules. Consider renaming to `isInitializedModule`.~~

6.4 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. ~~Line 101: is the `address[] public modules` array really needed? The smart contract doesn't use it.~~
2. ~~Line 182: should the `Envoked` event include returned data as well?~~

6.5 Other Issues

This section lists other minor issues which were found in the token smart contract.

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~

2. Line 46: there is no good reason to prefix the `_target` event parameter names with underscore, as these names may not clash with state variable names or any other identifiers. Consider removing underscores. Also this parameter should be indexed.
3. Line 54: the `_component` parameter should be indexed.

7. StreamingFeeModule

In this section we describe issues found in the StreamingFeeModule.sol.

7.1 Arithmetic Overflow Issues

This section lists issues of smart contracts related to the arithmetic overflows.

Line 230: phantom overflow is possible in `mul`, i.e. situation, when the final calculation result would fit into the destination type, but some intermediate calculations overflow.

7.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 60-62: the next parameters should be indexed: `_setToken`, `_oldStreamingFee`, `_oldFeeRecipient`.
2. Line 85: the message `Fee must be > 0` is confusing, as one could think that it tells about streaming fee accumulated since the last accruing, rather than about fee percentage. Consider rewriting the message.
3. Line 192: the formula in the comment is incorrect. It doesn't take fee compounding into account. The correct formula would be `annualFee^(secondsPassed/secondsInYear)`. Actually, it would be more efficient to store 1-second fee percentage rather than annual fee percentage, and use exponentiation by squaring to calculate compound fee for an arbitrary time interval. Such compounding would take only about 2K gas.
4. Line 298: the expression `PreciseUnitMath.preciseUnit().sub(_inflationFee).toInt256()` is used twice. Consider calculating before the `if` statement and storing in a local variable.
5. Line 304-313: the `_feeRecipient`, the `_lastStreamingFeeTimestamp` and the `_streamingFeePercentage` functions are redundant. The `feeStates` storage variable is already public.

7.3 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. ~~Line 79: in Set protocol v1, each Set token has natural unit property, and adjusting natural unit up had the same effect as adjusting units of all components down. Why is the natural unit concept removed?~~
2. ~~Line 120, 171: perhaps there is no need to try to forbid the `_settings.feeRecipient != address(0)` and the `_newFeeRecipient != address(0)`. It is anyway possible to specify the fee recipient address owned by nobody.~~

7.4 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. Line 210: around the `feeQuantity + totalSupply` there should be parentheses.
3. ~~Line 60-62: the events are usually named via nouns. Here are some examples:~~
   - ~~`FeeActualization` instead of `FeeActualized`~~
   - ~~`FeeUpdate` instead of `StreamingFeeUpdated`~~
   - ~~`FeeRecipientUpdate` instead of `FeeRecipientUpdated`~~
4. ~~Line 60: there is no good reason to prefix event parameters like `_setToken` names with underscore, as these names may not clash with state variable names or any other identifiers. Consider removing underscores.~~
5. ~~Line 264: there should be `increase` instead of `increaase`.~~
6. ~~Line 228: the comment `All negative unit positions will be rounded away from 0, and all positive units positions will * be rounded towards 0` can be simplified: `all positions will be rounded down.`~~

8. IntegrationRegistry

In this section we describe issues found in the IntegrationRegistry.sol.

8.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 37-39, 41: the `_adapter`, the `_integrationName` and the `_newAdapter` parameters should be indexed:
2. Line 42: the `_oldAdapter` parameter is redundant.
3. Line 43: the `_integrationName` parameter looks like an identifier of the integration and should probably be indexed.
4. Line 100-102, 160-162: passing these lines as a single array of structs would make calls cheaper and would make length checks unnecessary.
5. Line 142, 194: the value `integrations[_module][hashedName]` was already read from storage two statements above. Consider reading once and caching in a local variable.
6. Line 188, 206, 218: passing the hash of the integration name here would make calls cheaper for smart contracts.
7. Line 232: the `keccak256(abi.encodePacked(_name))` could be simplified to `keccak256(bytes(_name))`.

8.2 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

Line 110: why are the empty batches forbidden by `modulesCount > 0`?

8.3 Other Issues

This section lists other minor issues which were found in the token smart contract

1. Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 37-39: the events are usually named via nouns. Here some examples:
   - `NewIntegration` or `IntegrationAddition` instead of `IntegrationAdded`
   - `IntegrationRemoval` instead of `IntegrationRemoved`
   - `IntegrationEdit` or `IntegrationChange` instead of `IntegrationEdited`
3. Line 37: there is no good reason to prefix event parameters like `_module` names with underscore, as these names may not clash with state variable names or any other identifiers. Consider removing underscores.
4. Line 231: the name is confusing as it looks like a getter, while this is a pure function. Consider renaming to something like `nameHash`.

9. PositionLib

In this section we describe issues found in the PositionLib.sol.

9.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 86: the `positionIsFound &&` is redundant, as positionIsFound is guaranteed to be true here.
2. Line 145: the caller has already provided all the position attributes, the only results the caller may be interested in are position index and unit. Returning the whole `PositionData` seems redundant.

9.2 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. Line 63: is it possible for the `Set` to have several positions for the same component tokens? If no, then it would be more efficient to use component address as a position identifier, rather than position index.
2. Line 77: the `editDefaultPosition` function is not used in other contracts being reviewed. The function may do very different things depending on circumstances and parameters passed. Is this function really needed? If so, consider splitting it into several functions each doing one thing.
3. Line 83, 90: the newly created and edited position by the `pushPosition` and the `editPositionUnit` may be undercollateralized or make collateral inaccessible. Is this fine?
4. Line 87: removing the position by `removePosition` that is still collateralized may make collateral inaccessible. Is this fine?
5. Line 122: is it guaranteed that the default position module is zero `address(0)` and the data is empty?
6. Line 126: does the comment `the four specified fields: component, module, state, and data` mean that component could be non-unique within a Set token? It is guaranteed that the combination of these four fields is unique?
7. Line 153-154: is it really necessary to do the checks in these lines in case the position is in the default state? When looking for a non-default position, it seems that the caller needs to know all the position attributes in order to find it.

8. ~~Line 176: does position with zero unit according to the check `_unit >= 0` really make sense?~~
9. ~~Line 202, 212: the `getDefaultTotalNotional` and the `getDefaultPositionUnit` functions are basically an alias for the `preciseMul`. Is it really needed?~~
10. ~~Line 217: it is unclear what this function calculates, as new position unit could be calculated from the post total notional and Set total supply~~.

9.3 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 35: unlike most of the libraries in the same repository, the name of the `PositionLib` library ends with the `Lib` suffix. This may confuse people. They could think that other libraries, named without such suffix, are not libraries. Consider using consistent naming.~~

10. PreciseUnitMath

In this section we describe issues found in the PreciseUnitMath.sol.

10.1 Major Flaws

We have identified potentially faulty behaviour in smart contracts, all related to multiple-voting scenario:

~~Line 187: the condition returns 0 if a and b are zero, while for all other values of a, the function reverts on `b==0`~~.

10.2 Arithmetic Overflow Issues

This section lists issues of smart contracts related to the arithmetic overflows.

~~Line 103, 118, 136, 156, 158, 173, 187: the phantom overflow is possible in `mul` operation, i.e. situation, when the final result would fit into the destination type, but some intermediate calculations overflow.~~

10.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 38: defining the `PRECISE_UNIT` constant as uint64 or uint248 would make it directly assignable to both, uint256 and int256, so the second constant would not be probably needed.
2. ~~Line 125: the implementation of the `preciseMulCeil` function could be simplified to: `return a.mul(b).add(PRECISE_UNIT - 1).div(PRECISE_UNIT);`~~
3. ~~Line 155: the condition `a > 0 && b > 0 || a < 0 && b < 0` could be simplified as `a ^ b >= 0`.~~
4. ~~Line 156: the check `.sub(1).div(PRECISE_UNIT_INT).add(1)`could be simplified as `.add(PRECISE_UNIT_INT - 1).div(PRECISE_UNIT_INT)`.~~
5. ~~Line 158: the check `.add(1).div(PRECISE_UNIT_INT).sub(1)` could be simplified as `.add(PRECISE_UNIT_INT - 1).div(PRECISE_UNIT_INT)`.~~
6. ~~Line 187: the check `.sub(1).div(b).add(1)` could be simplified as `.add(b - 1).div(b)` In case we already required that `b > 0`.~~

10.4 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision

Line 50, 61, 72, 83: the `preciseUnit` and the `maxUint256` functions duplicate the `PRECISE_UNIT` and the `MAX_UINT_256` constant accordingly. The same for the `maxInt256` and the `minInt256` (they duplicate `MAX_UINT_256` and the `minInt256` constant). Is it really necessary to have both, the constant and the function?

10.5 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 143: the name of the function `preciseMulCeil` is confusing, as usually, `ceil` is used for rounding up, rather than rounding away from zero. Consider choosing a different name.~~

11. AddressArrayUtils.sol

In this section we describe issues found in the AddressArrayUtils.sol.

11.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. ~~Line 33: it is a common practice for similar functions to return a special value in case the value was not found. In this line uint(-1) would be a good candidate. This value could not be a valid array index in Solidity, as it would require the array to have at least 2^256 elements, while the maximum array length is 2^256 - 1.~~
2. ~~Line 42: returning zero in the `return (0, false)` is error-prone. Zero is most probably a valid array index. Consider returning uint(-1) in this line.~~
3. ~~Line 52: the `isIn` variable is redundant. Just given this name to the returned value like this: returns (bool isIn).~~
4. ~~Line 63: the `append` function has O(n) complexity, as it copies all the existing addresses to a new location. This could lead to overconsumption of gas.~~
5. Line 75: the `remove` function traverses an array two times: once to find element index, and another time to copy the elements into a new location. Consider rewriting it to do both things in one pass.
6. Line 94: it is actually possible to remove array elements in place, without allocating a new array. However this would require using some assembly code. Also, there is no check for the `index` value, while only the values less than `A.length` seem to be valid. Consider adding an explicit range check.

11.2 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

Line 33: the code does not provide a situation when the `index` element is in the array.

11.3 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 100: the underflow is possible in `--` operation.~~
3. ~~Line 35, 63, 75: it is uncommon to start parameter names with upper case letters like the `memory A`. Consider renaming. You can use `array` as it is not a keyword in Solidity.~~
4. ~~Line 73: the `@return Returns the new array` doesn't actually describe what the function does. Consider adding more details.~~
5. ~~Line 91: the comment `Resulting ordering is not guaranteed` is incorrect. The function actually preserves the ordering of the elements.~~

12. SetTokenCreator

In this section we describe issues found in the SetTokenCreator.sol.

12.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. ~~Line 36: the `_manager` parameter should probably be indexed.~~
2. ~~Line 65-66: passing a single array of structs with two fields would be more efficient and would make length check unnecessary.~~
3. ~~Line 81: it is not checked that `_components[i]` are unique. Consider adding the check.~~

12.2 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision

~~Line 77-78: it is unclear why the tokens with no modules forbidden `_modules.length > 0`. Similar for the `_manager != address(0)`.It is anyway possible to create a Set token with a manager address owned by nobody.~~

12.3 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 36: there is no good reason to prefix the event parameter `_setToken` name with underscore, as these names may not clash with state variable names or any other identifiers. Consider removing underscores. Also, events are usually named via nouns, such as `NewSetToken` or `SetTokenCreation`.~~

13. ISetToken.sol

In this section we describe issues found in the ISetToken.sol.

13.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 30: the `ISetToken` interface combines several independent things: modular contract API, Set token API, and lockable contract API. There could be Set tokens not based on modular design, and there could be non-lockable Set tokens. Consider moving all module-related stuff into IModular interface, and all lock-related stuff into ILockable interface.
2. ~~Line 45: it would probably be more efficient to mark default state by assigning zero address to module field.~~
3. ~~Line 54: using bytes8 instead of uint does not save any storage space, but limits the abilities to encode some useful information into the state. If the set of possible states is not predefined and modules may introduce more states, then consider changing this type to uint.~~
4. ~~Line 62, 63: the `pushPosition` function should probably return the index of just pushed position. The `popPosition` function should probably return just the popped position.~~
5. ~~Line 64: there is no function that:~~
   - ~~defines the number of positions~~
   - ~~defines the index of just pushed position~~
   - ~~receives position by index~~

   ~~This makes it hard to refer the positions by indexes. In case there could not be several positions with the same components. Consider referring positions by component address. Alternatively, consider adding missing functions listed above.~~
6. ~~Line 85: the `Position[] memory` returns all the details of all the positions, while the caller may only need some details for a single position. Consider implementing a function to get details for a particular position identified by some unique attribute or surrogate index.~~
7. ~~Line 88: the semantics of the `isPendingModule` function is unclear. Consider adding a documentation comment. Also, this function should probably be renamed to `getModuleState` and should return `ModuleState` value. This would make the `isModule` function redundant.~~

13.2 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

~~Line 60, 62, 63, 71, 72, 74, 75, 79: it is unclear from functions name (`invoke`, `pushPosition`, `popPosition`, `mint`, `burn`, `lock`, `unlock`, `initializeModule`) and signature what functions actually do. Consider adding a documentation comment.~~

13.3 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 47: some additional comment is required: possible values for the `@param positionState`.~~

14. ExplicitERC20

In this section we describe issues found in the ExplicitERC20.sol.

14.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. ~~Line 52: the check `_quantity > 0` skips zero-quantitiy transfer, but doesn't skip trivial transfer whose `_from == _to`. Consider skipping trivial transfers as well, as the code below is not able to handle them properly.~~
2. ~~Line 66: the check `newBalance == existingBalance.add(_quantity)` will most probably fail in case `_from == _to`. Also, it ensures that the recipient got exactly `_quantity` tokens, but doesn't check that the sender lost exactly `_quantity` tokens. This could be an issue for token smart contracts that charge transfer fees from the sender.~~

14.2 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

~~Line 36: some additional comment is needed. Perhaps, there should be a check for the quantity sender spent.~~

14.3 Other Issues

This section lists other minor issues which were found in the token smart contract.

~~Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~

15. ModuleBase

In this section we describe issues found in the ModuleBase.sol.

15.1 Other Issues

This section lists other minor issues which were found in the token smart contract

1. Line 19: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 51: there should be the `initializations` instead of the `initiazations`.

16. IModule

In this section we describe issues found in the IModule.sol.

16.1 Other Issues

This section lists other minor issues which were found in the token smart contract

1. Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 29: it is unclear who calls the `SetToken` function. Probably a better description would be: `Called by a SetToken to notify that this module was removed from the Set token.`

17. IOracle

In this section we describe issues found in the IOracle.sol.

17.1 Other Issues

This section lists other minor issues which were found in the token smart contract

1. Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.
2. Line 28, 32: the name of the `IOracle` function is too generic. This interface is not for any kind of oracle, but only for those oracles that return asset prices. So the name `IPriceOracle` would be more relevant. It is also true for the `read` function. Consider renaming to something like `urrentPrice` or `readCurrentPrice`.

3. ~~Line 30: from the comment `Current price of asset represented in uint256` it is unclear how the price is encoded. Consider adding more details about this.~~

18. IOracleAdapter

In this section we describe issues found in the IOracleAdapter.sol.

18.1 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. Line 34: the purpose of the value `Boolean indicating if oracle exists` is unclear. What price will be returned in case oracle doesn't exist?
2. Line 35: it is unclear what is the price encoding. Consider adding more details.

18.2 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. Line 25: the description `Interface for calling an oracle adapter.` is confusing. It is unclear what is an oracle adapter, and why a separate interface is needed to call it.
3. Line 37: consider renaming the `address _assetOne` and the `address _assetTwo` to `_baseAsset` and `_quoteAsset` or some other names that emphasize the roles of the assets.

19. IController

In this section we describe issues found in the IController.sol.

19.1 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 21: there should be `ISetToken` instead of `address`.
2. Line 27: the `address` should be IERC20.

19.2 Other Issues

This section lists other minor issues which were found in the token smart contract

~~Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~

20. IDepositor

In this section we describe issues found in the IDepositor.sol.

20.1 Other Issues

This section lists other minor issues which were found in the token smart contract

1. ~~Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~
2. ~~Line 23: from the function name `executeIssuanceDeposit` and the signature it is unclear what the function does. Consider adding a documentation comment.~~

21. IManagerIssuanceHook

In this section we describe issues found in the IManagerIssuanceHook.sol.

21.1 Other Issues

This section lists other minor issues which were found in the token smart contract

~~Line 18: the `pragma solidity` version should be `0.6.0` according to the common best practice, unless there is something special with this particular version.~~

22. Summary

Based on our findings, we also recommend the following:

1. Fix critical issues which could lead to asset loss.
2. Pay attention to major and moderate issues.
3. Check issues marked “unclear behavior” against functional requirements.
4. Refactor the code to remove suboptimal parts.
5. Fix other (minor) issues.

```
/*
    Copyright 2021 Set Labs Inc.

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

    SPDX-License-Identifier: Apache License, Version 2.0
*/

pragma solidity 0.6.10;
pragma experimental "ABIEncoderV2";

import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";
import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

import { Compound } from "../integration/lib/Compound.sol";
import { ICErc20 } from "../../interfaces/external/ICErc20.sol";
import { IComptroller } from "../../interfaces/external/IComptroller.sol";
import { IController } from "../../interfaces/IController.sol";
import { IDebtIssuanceModule } from "../../interfaces/IDebtIssuanceModule.sol";
import { IExchangeAdapter } from "../../interfaces/IExchangeAdapter.sol";
import { ISetToken } from "../../interfaces/ISetToken.sol";
import { ModuleBase } from "../lib/ModuleBase.sol";
```

```
/**
 * @title CompoundLeverageModule
 * @author Set Protocol
 *
 * Smart contract that enables leverage trading using Compound as the lending protocol. This module is paired with a
debt issuance module that will call
 * functions on this module to keep interest accrual and liquidation state updated. This does not allow borrowing of
assets from Compound alone. Each
 * asset is leveraged when using this module.
 *
 * Note: Do not use this module in conjunction with other debt modules that allow Compound debt positions as it
could lead to double counting of
 * debt when borrowed assets are the same.
 *
 */
contract CompoundLeverageModule is ModuleBase, ReentrancyGuard, Ownable {
    using Compound for ISetToken;

    /* ============ Structs ============ */

    struct EnabledAssets {
        address[] collateralCTokens;             // Array of enabled cToken collateral assets for a SetToken
        address[] borrowCTokens;                 // Array of enabled cToken borrow assets for a SetToken
        address[] borrowAssets;                  // Array of underlying borrow assets that map to the array of
enabled cToken borrow assets
    }

    struct ActionInfo {
        ISetToken setToken;                      // SetToken instance
        IExchangeAdapter exchangeAdapter;        // Exchange adapter instance
        uint256 setTotalSupply;                  // Total supply of SetToken
        uint256 notionalSendQuantity;            // Total notional quantity sent to exchange
        uint256 minNotionalReceiveQuantity;      // Min total notional received from exchange
        ICErc20 collateralCTokenAsset;           // Address of cToken collateral asset
```

```
    ICErc20 borrowCTokenAsset;               // Address of cToken borrow asset
    uint256 preTradeReceiveTokenBalance;     // Balance of pre-trade receive token balance
}

/* ============ Events ============ */

event LeverageIncreased(
    ISetToken indexed _setToken,
    IERC20 indexed _borrowAsset,
    IERC20 indexed _collateralAsset,
    IExchangeAdapter _exchangeAdapter,
    uint256 _totalBorrowAmount,
    uint256 _totalReceiveAmount,
    uint256 _protocolFee
);

event LeverageDecreased(
    ISetToken indexed _setToken,
    IERC20 indexed _collateralAsset,
    IERC20 indexed _repayAsset,
    IExchangeAdapter _exchangeAdapter,
    uint256 _totalRedeemAmount,
    uint256 _totalRepayAmount,
    uint256 _protocolFee
);

event CollateralAssetsUpdated(
    ISetToken indexed _setToken,
    bool indexed _added,
    IERC20[] _assets
);

event BorrowAssetsUpdated(
    ISetToken indexed _setToken,
```

```
        bool indexed _added,
        IERC20[] _assets
    );

    event SetTokenStatusUpdated(
        ISetToken indexed _setToken,
        bool indexed _added
    );

    event AnySetAllowedUpdated(
        bool indexed _anySetAllowed
    );

    /* ============ Constants ============ */

    // String identifying the DebtIssuanceModule in the IntegrationRegistry. Note: Governance must add
DefaultIssuanceModule as
    // the string as the integration name
    string constant internal DEFAULT_ISSUANCE_MODULE_NAME = "DefaultIssuanceModule";

    // 0 index stores protocol fee % on the controller, charged in the trade function
    uint256 constant internal PROTOCOL_TRADE_FEE_INDEX = 0;

    /* ============ State Variables ============ */

    // Mapping of underlying to CToken. If ETH, then map WETH to cETH
    mapping(IERC20 => ICErc20) public underlyingToCToken;

    // Wrapped Ether address
    IERC20 internal weth;

    // Compound cEther address
    ICErc20 internal cEther;
```

```
// Compound Comptroller contract
IComptroller internal comptroller;

// COMP token address
IERC20 internal compToken;

// Mapping to efficiently check if cToken market for collateral asset is valid in SetToken
mapping(ISetToken => mapping(ICErc20 => bool)) public collateralCTokenEnabled;

// Mapping to efficiently check if cToken market for borrow asset is valid in SetToken
mapping(ISetToken => mapping(ICErc20 => bool)) public borrowCTokenEnabled;

// Mapping of enabled collateral and borrow cTokens for syncing positions
mapping(ISetToken => EnabledAssets) internal enabledAssets;

// Mapping of SetToken to boolean indicating if SetToken is on allow list. Updateable by governance
mapping(ISetToken => bool) public allowedSetTokens;

// Boolean that returns if any SetToken can initialize this module. If false, then subject to allow list
bool public anySetAllowed;


/* ============ Constructor ============ */

/**
 * Instantiate addresses. Underlying to cToken mapping is created.
 *
 * @param _controller               Address of controller contract
 * @param _compToken                Address of COMP token
 * @param _comptroller              Address of Compound Comptroller
 * @param _cEther                   Address of cEther contract
 * @param _weth                     Address of WETH contract
 */
constructor(
```

```
        IController _controller,
        IERC20 _compToken,
        IComptroller _comptroller,
        ICErc20 _cEther,
        IERC20 _weth
    )
        public
        ModuleBase(_controller)
    {
        compToken = _compToken;
        comptroller = _comptroller;
        cEther = _cEther;
        weth = _weth;

        ICErc20[] memory cTokens = comptroller.getAllMarkets();

        for(uint256 i = 0; i < cTokens.length; i++) {
            ICErc20 cToken = cTokens[i];
            underlyingToCToken[
                cToken == _cEther ? _weth : IERC20(cTokens[i].underlying())
            ] = cToken;
        }
    }

    /* ============ External Functions ============ */

    /**
     * MANAGER ONLY: Increases leverage for a given collateral position using an enabled borrow asset that is
enabled.
     * Performs a DEX trade, exchanging the borrow asset for collateral asset.
     *
     * @param _setToken             Instance of the SetToken
     * @param _borrowAsset          Address of asset being borrowed for leverage
     * @param _collateralAsset      Address of collateral asset (underlying of cToken)
```

```
     * @param _borrowQuantity       Borrow quantity of asset in position units
     * @param _minReceiveQuantity   Min receive quantity of collateral asset to receive post-trade in position units
     * @param _tradeAdapterName     Name of trade adapter
     * @param _tradeData            Arbitrary data for trade
     */
    function lever(
        ISetToken _setToken,
        IERC20 _borrowAsset,
        IERC20 _collateralAsset,
        uint256 _borrowQuantity,
        uint256 _minReceiveQuantity,
        string memory _tradeAdapterName,
        bytes memory _tradeData
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        // For levering up, send quantity is derived from borrow asset and receive quantity is derived from
        // collateral asset
        ActionInfo memory leverInfo = _createAndValidateActionInfo(
            _setToken,
            _borrowAsset,
            _collateralAsset,
            _borrowQuantity,
            _minReceiveQuantity,
            _tradeAdapterName,
            true
        );

        _borrow(leverInfo.setToken, leverInfo.borrowCTokenAsset, leverInfo.notionalSendQuantity);

        uint256 postTradeReceiveQuantity = _executeTrade(leverInfo, _borrowAsset, _collateralAsset, _tradeData);
```

```
        uint256 protocolFee = _accrueProtocolFee(_setToken, _collateralAsset, postTradeReceiveQuantity);

        uint256 postTradeCollateralQuantity = postTradeReceiveQuantity.sub(protocolFee);

        _mintCToken(leverInfo.setToken, leverInfo.collateralCTokenAsset, _collateralAsset,
postTradeCollateralQuantity);

        _updateLeverPositions(leverInfo, _borrowAsset);

        emit LeverageIncreased(
            _setToken,
            _borrowAsset,
            _collateralAsset,
            leverInfo.exchangeAdapter,
            leverInfo.notionalSendQuantity,
            postTradeCollateralQuantity,
            protocolFee
        );
    }

    /**
     * MANAGER ONLY: Decrease leverage for a given collateral position using an enabled borrow asset that is enabled
     *
     * @param _setToken             Instance of the SetToken
     * @param _collateralAsset      Address of collateral asset (underlying of cToken)
     * @param _repayAsset           Address of asset being repaid
     * @param _redeemQuantity       Quantity of collateral asset to delever
     * @param _minRepayQuantity     Minimum amount of repay asset to receive post trade
     * @param _tradeAdapterName     Name of trade adapter
     * @param _tradeData            Arbitrary data for trade
     */
    function delever(
        ISetToken _setToken,
        IERC20 _collateralAsset,
```

```
        IERC20 _repayAsset,
        uint256 _redeemQuantity,
        uint256 _minRepayQuantity,
        string memory _tradeAdapterName,
        bytes memory _tradeData
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        // Note: for delevering, send quantity is derived from collateral asset and receive quantity is derived from
        // repay asset
        ActionInfo memory deleverInfo = _createAndValidateActionInfo(
            _setToken,
            _collateralAsset,
            _repayAsset,
            _redeemQuantity,
            _minRepayQuantity,
            _tradeAdapterName,
            false
        );

        _redeemUnderlying(deleverInfo.setToken, deleverInfo.collateralCTokenAsset,
deleverInfo.notionalSendQuantity);

        uint256 postTradeReceiveQuantity = _executeTrade(deleverInfo, _collateralAsset, _repayAsset, _tradeData);

        uint256 protocolFee = _accrueProtocolFee(_setToken, _repayAsset, postTradeReceiveQuantity);

        uint256 repayQuantity = postTradeReceiveQuantity.sub(protocolFee);

        _repayBorrow(deleverInfo.setToken, deleverInfo.borrowCTokenAsset, _repayAsset, repayQuantity);

        _updateLeverPositions(deleverInfo, _repayAsset);
```

```
        emit LeverageDecreased(
            _setToken,
            _collateralAsset,
            _repayAsset,
            deleverInfo.exchangeAdapter,
            deleverInfo.notionalSendQuantity,
            repayQuantity,
            protocolFee
        );
    }

    /**
     * MANAGER ONLY: Pays down the borrow asset to 0 selling off a given collateral asset. Any extra received
     * borrow asset is updated as equity. No protocol fee is charged.
     *
     * @param _setToken              Instance of the SetToken
     * @param _collateralAsset       Address of collateral asset (underlying of cToken)
     * @param _repayAsset            Address of asset being repaid (underlying asset e.g. DAI)
     * @param _redeemQuantity        Quantity of collateral asset to delever
     * @param _tradeAdapterName      Name of trade adapter
     * @param _tradeData             Arbitrary data for trade
     */
    function deleverToZeroBorrowBalance(
        ISetToken _setToken,
        IERC20 _collateralAsset,
        IERC20 _repayAsset,
        uint256 _redeemQuantity,
        string memory _tradeAdapterName,
        bytes memory _tradeData
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
```

```
    {
        uint256 notionalRedeemQuantity = _redeemQuantity.preciseMul(_setToken.totalSupply());

        require(borrowCTokenEnabled[_setToken][underlyingToCToken[_repayAsset]], "Borrow not enabled");
        uint256 notionalRepayQuantity = underlyingToCToken[_repayAsset].borrowBalanceCurrent(address(_setToken));

        ActionInfo memory deleverInfo = _createAndValidateActionInfoNotional(
            _setToken,
            _collateralAsset,
            _repayAsset,
            notionalRedeemQuantity,
            notionalRepayQuantity,
            _tradeAdapterName,
            false
        );

        _redeemUnderlying(deleverInfo.setToken, deleverInfo.collateralCTokenAsset,
deleverInfo.notionalSendQuantity);

        uint256 postTradeReceiveQuantity = _executeTrade(deleverInfo, _collateralAsset, _repayAsset, _tradeData);

        // We use notionalRepayQuantity vs. Compound's max value uint256(-1) to handle WETH properly
        _repayBorrow(deleverInfo.setToken, deleverInfo.borrowCTokenAsset, _repayAsset, notionalRepayQuantity);

        // Update default position first to save gas on editing borrow position
        _setToken.calculateAndEditDefaultPosition(
            address(_repayAsset),
            deleverInfo.setTotalSupply,
            deleverInfo.preTradeReceiveTokenBalance
        );

        _updateLeverPositions(deleverInfo, _repayAsset);

        emit LeverageDecreased(
```

```
            _setToken,
            _collateralAsset,
            _repayAsset,
            deleverInfo.exchangeAdapter,
            deleverInfo.notionalSendQuantity,
            notionalRepayQuantity,
            0 // No protocol fee
        );
    }

    /**
     * CALLABLE BY ANYBODY: Sync Set positions with enabled Compound collateral and borrow positions. For collateral
     * assets, update cToken default position. For borrow assets, update external borrow position.
     * - Collateral assets may come out of sync when a position is liquidated
     * - Borrow assets may come out of sync when interest is accrued or position is liquidated and borrow is repaid
     *
     * @param _setToken               Instance of the SetToken
     * @param _shouldAccrueInterest   Boolean indicating whether use current block interest rate value or stored
value
     */
    function sync(ISetToken _setToken, bool _shouldAccrueInterest) public nonReentrant
onlyValidAndInitializedSet(_setToken) {
        uint256 setTotalSupply = _setToken.totalSupply();

        // Only sync positions when Set supply is not 0. This preserves debt and collateral positions on issuance /
redemption
        if (setTotalSupply > 0) {
            // Loop through collateral assets
            address[] memory collateralCTokens = enabledAssets[_setToken].collateralCTokens;
            for(uint256 i = 0; i < collateralCTokens.length; i++) {
                ICErc20 collateralCToken = ICErc20(collateralCTokens[i]);
                uint256 previousPositionUnit =
_setToken.getDefaultPositionRealUnit(address(collateralCToken)).toUint256();
                uint256 newPositionUnit = _getCollateralPosition(_setToken, collateralCToken, setTotalSupply);
```

```
                // Note: Accounts for if position does not exist on SetToken but is tracked in enabledAssets
                if (previousPositionUnit != newPositionUnit) {
                  _updateCollateralPosition(_setToken, collateralCToken, newPositionUnit);
                }
            }

            // Loop through borrow assets
            address[] memory borrowCTokens = enabledAssets[_setToken].borrowCTokens;
            address[] memory borrowAssets = enabledAssets[_setToken].borrowAssets;
            for(uint256 i = 0; i < borrowCTokens.length; i++) {
                ICErc20 borrowCToken = ICErc20(borrowCTokens[i]);
                IERC20 borrowAsset = IERC20(borrowAssets[i]);

                int256 previousPositionUnit = _setToken.getExternalPositionRealUnit(address(borrowAsset),
address(this));

                int256 newPositionUnit = _getBorrowPosition(
                    _setToken,
                    borrowCToken,
                    setTotalSupply,
                    _shouldAccrueInterest
                );

                // Note: Accounts for if position does not exist on SetToken but is tracked in enabledAssets
                if (newPositionUnit != previousPositionUnit) {
                    _updateBorrowPosition(_setToken, borrowAsset, newPositionUnit);
                }
            }
        }
    }


    /**
```

```
     * MANAGER ONLY: Initializes this module to the SetToken. Only callable by the SetToken's manager. Note:
managers can enable
     * collateral and borrow assets that don't exist as positions on the SetToken
     *
     * @param _setToken             Instance of the SetToken to initialize
     * @param _collateralAssets     Underlying tokens to be enabled as collateral in the SetToken
     * @param _borrowAssets         Underlying tokens to be enabled as borrow in the SetToken
     */
    function initialize(
        ISetToken _setToken,
        IERC20[] memory _collateralAssets,
        IERC20[] memory _borrowAssets
    )
        external
        onlySetManager(_setToken, msg.sender)
        onlyValidAndPendingSet(_setToken)
    {
        if (!anySetAllowed) {
            require(allowedSetTokens[_setToken], "Not allowed SetToken");
        }

        // Initialize module before trying register
        _setToken.initializeModule();

        // Get debt issuance module registered to this module and require that it is initialized
        require(_setToken.isInitializedModule(getAndValidateAdapter(DEFAULT_ISSUANCE_MODULE_NAME)), "Issuance not
initialized");

        // Try if register exists on any of the modules including the debt issuance module
        address[] memory modules = _setToken.getModules();
        for(uint256 i = 0; i < modules.length; i++) {
            try IDebtIssuanceModule(modules[i]).registerToIssuanceModule(_setToken) {} catch {}
        }
```

```
        // Enable collateral and borrow assets on Compound
        addCollateralAssets(_setToken, _collateralAssets);

        addBorrowAssets(_setToken, _borrowAssets);
    }

    /**
     * MANAGER ONLY: Removes this module from the SetToken, via call by the SetToken. Compound Settings and manager
enabled
     * cTokens are deleted. Markets are exited on Comptroller (only valid if borrow balances are zero)
     */
    function removeModule() external override onlyValidAndInitializedSet(ISetToken(msg.sender)) {
        ISetToken setToken = ISetToken(msg.sender);

        // Sync Compound and SetToken positions prior to any removal action
        sync(setToken, true);

        address[] memory borrowCTokens = enabledAssets[setToken].borrowCTokens;
        for (uint256 i = 0; i < borrowCTokens.length; i++) {
            ICErc20 cToken = ICErc20(borrowCTokens[i]);

            // Will exit only if token isn't also being used as collateral
            if(!collateralCTokenEnabled[setToken][cToken]) {
                // Note: if there is an existing borrow balance, will revert and market cannot be exited on Compound
                setToken.invokeExitMarket(cToken, comptroller);
            }

            delete borrowCTokenEnabled[setToken][cToken];
        }

        address[] memory collateralCTokens = enabledAssets[setToken].collateralCTokens;
        for (uint256 i = 0; i < collateralCTokens.length; i++) {
            ICErc20 cToken = ICErc20(collateralCTokens[i]);
```

```
            setToken.invokeExitMarket(cToken, comptroller);

            delete collateralCTokenEnabled[setToken][cToken];
        }

        delete enabledAssets[setToken];

        // Try if unregister exists on any of the modules
        address[] memory modules = setToken.getModules();
        for(uint256 i = 0; i < modules.length; i++) {
            try IDebtIssuanceModule(modules[i]).unregisterFromIssuanceModule(setToken) {} catch {}
        }
    }

    /**
     * MANAGER ONLY: Add registration of this module on debt issuance module for the SetToken. Note: if the debt
issuance module is not added to SetToken
     * before this module is initialized, then this function needs to be called if the debt issuance module is later
added and initialized to prevent state
     * inconsistencies
     *
     * @param _setToken             Instance of the SetToken
     * @param _debtIssuanceModule   Debt issuance module address to register
     */
    function registerToModule(ISetToken _setToken, IDebtIssuanceModule _debtIssuanceModule) external
onlyManagerAndValidSet(_setToken) {
        require(_setToken.isInitializedModule(address(_debtIssuanceModule)), "Issuance not initialized");

        _debtIssuanceModule.registerToIssuanceModule(_setToken);
    }

    /**
     * MANAGER ONLY: Add enabled collateral assets. Collateral assets are tracked for syncing positions and entered
in Compound markets
```

```
     *
     * @param _setToken             Instance of the SetToken
     * @param _newCollateralAssets  Addresses of new collateral underlying assets
     */
    function addCollateralAssets(ISetToken _setToken, IERC20[] memory _newCollateralAssets) public
onlyManagerAndValidSet(_setToken) {
        for(uint256 i = 0; i < _newCollateralAssets.length; i++) {
            ICErc20 cToken = underlyingToCToken[_newCollateralAssets[i]];
            require(address(cToken) != address(0), "cToken must exist");
            require(!collateralCTokenEnabled[_setToken][cToken], "Collateral enabled");

            // Note: Will only enter market if cToken is not enabled as a borrow asset as well
            if (!borrowCTokenEnabled[_setToken][cToken]) {
                _setToken.invokeEnterMarkets(cToken, comptroller);
            }

            collateralCTokenEnabled[_setToken][cToken] = true;
            enabledAssets[_setToken].collateralCTokens.push(address(cToken));
        }

        emit CollateralAssetsUpdated(_setToken, true, _newCollateralAssets);
    }

    /**
     * MANAGER ONLY: Remove collateral asset. Collateral asset exited in Compound markets
     * If there is a borrow balance, collateral asset cannot be removed
     *
     * @param _setToken             Instance of the SetToken
     * @param _collateralAssets     Addresses of collateral underlying assets to remove
     */
    function removeCollateralAssets(ISetToken _setToken, IERC20[] memory _collateralAssets) external
onlyManagerAndValidSet(_setToken) {
        // Sync Compound and SetToken positions prior to any removal action
        sync(_setToken, true);
```

```
        for(uint256 i = 0; i < _collateralAssets.length; i++) {
            ICErc20 cToken = underlyingToCToken[_collateralAssets[i]];
            require(collateralCTokenEnabled[_setToken][cToken], "Collateral not enabled");

            // Note: Will only exit market if cToken is not enabled as a borrow asset as well
            // If there is an existing borrow balance, will revert and market cannot be exited on Compound
            if (!borrowCTokenEnabled[_setToken][cToken]) {
                _setToken.invokeExitMarket(cToken, comptroller);
            }

            delete collateralCTokenEnabled[_setToken][cToken];
            enabledAssets[_setToken].collateralCTokens.removeStorage(address(cToken));
        }

        emit CollateralAssetsUpdated(_setToken, false, _collateralAssets);
    }

    /**
     * MANAGER ONLY: Add borrow asset. Borrow asset is tracked for syncing positions and entered in Compound markets
     *
     * @param _setToken             Instance of the SetToken
     * @param _newBorrowAssets      Addresses of borrow underlying assets to add
     */
    function addBorrowAssets(ISetToken _setToken, IERC20[] memory _newBorrowAssets) public
onlyManagerAndValidSet(_setToken) {
        for(uint256 i = 0; i < _newBorrowAssets.length; i++) {
            IERC20 newBorrowAsset = _newBorrowAssets[i];
            ICErc20 cToken = underlyingToCToken[newBorrowAsset];
            require(address(cToken) != address(0), "cToken must exist");
            require(!borrowCTokenEnabled[_setToken][cToken], "Borrow enabled");

            // Note: Will only enter market if cToken is not enabled as a borrow asset as well
            if (!collateralCTokenEnabled[_setToken][cToken]) {
```

```
                _setToken.invokeEnterMarkets(cToken, comptroller);
            }

            borrowCTokenEnabled[_setToken][cToken] = true;
            enabledAssets[_setToken].borrowCTokens.push(address(cToken));
            enabledAssets[_setToken].borrowAssets.push(address(newBorrowAsset));
        }

        emit BorrowAssetsUpdated(_setToken, true, _newBorrowAssets);
    }

    /**
     * MANAGER ONLY: Remove borrow asset. Borrow asset is exited in Compound markets
     * If there is a borrow balance, borrow asset cannot be removed
     *
     * @param _setToken             Instance of the SetToken
     * @param _borrowAssets         Addresses of borrow underlying assets to remove
     */
    function removeBorrowAssets(ISetToken _setToken, IERC20[] memory _borrowAssets) external
onlyManagerAndValidSet(_setToken) {
        // Sync Compound and SetToken positions prior to any removal action
        sync(_setToken, true);

        for(uint256 i = 0; i < _borrowAssets.length; i++) {
            ICErc20 cToken = underlyingToCToken[_borrowAssets[i]];
            require(borrowCTokenEnabled[_setToken][cToken], "Borrow not enabled");

            // Note: Will only exit market if cToken is not enabled as a collateral asset as well
            // If there is an existing borrow balance, will revert and market cannot be exited on Compound
            if (!collateralCTokenEnabled[_setToken][cToken]) {
                _setToken.invokeExitMarket(cToken, comptroller);
            }

            delete borrowCTokenEnabled[_setToken][cToken];
```

```
            enabledAssets[_setToken].borrowCTokens.removeStorage(address(cToken));
            enabledAssets[_setToken].borrowAssets.removeStorage(address(_borrowAssets[i]));
        }

        emit BorrowAssetsUpdated(_setToken, false, _borrowAssets);
    }

    /**
     * GOVERNANCE ONLY: Add or remove allowed SetToken to initialize this module. Only callable by governance.
     *
     * @param _setToken             Instance of the SetToken
     */
    function updateAllowedSetToken(ISetToken _setToken, bool _status) external onlyOwner {
        allowedSetTokens[_setToken] = _status;
        emit SetTokenStatusUpdated(_setToken, _status);
    }

    /**
     * GOVERNANCE ONLY: Toggle whether any SetToken is allowed to initialize this module. Only callable by
governance.
     *
     * @param _anySetAllowed            Bool indicating whether allowedSetTokens is enabled
     */
    function updateAnySetAllowed(bool _anySetAllowed) external onlyOwner {
        anySetAllowed = _anySetAllowed;
        emit AnySetAllowedUpdated(_anySetAllowed);
    }

    /**
     * GOVERNANCE ONLY: Add Compound market to module with stored underlying to cToken mapping in case of market
additions to Compound.
     *
     * IMPORTANT: Validations are skipped in order to get contract under bytecode limit
     *
```

```
     * @param _cToken                   Address of cToken to add
     * @param _underlying               Address of underlying token that maps to cToken
     */
    function addCompoundMarket(ICErc20 _cToken, IERC20 _underlying) external onlyOwner {
        require(address(underlyingToCToken[_underlying]) == address(0), "Already added");
        underlyingToCToken[_underlying] = _cToken;
    }

    /**
     * GOVERNANCE ONLY: Remove Compound market on stored underlying to cToken mapping in case of market removals
     *
     * IMPORTANT: Validations are skipped in order to get contract under bytecode limit
     *
     * @param _underlying               Address of underlying token to remove
     */
    function removeCompoundMarket(IERC20 _underlying) external onlyOwner {
        require(address(underlyingToCToken[_underlying]) != address(0), "Not added");
        delete underlyingToCToken[_underlying];
    }

    /**
     * MODULE ONLY: Hook called prior to issuance to sync positions on SetToken. Only callable by valid module.
     *
     * @param _setToken             Instance of the SetToken
     */
    function moduleIssueHook(ISetToken _setToken, uint256 /* _setTokenQuantity */) external onlyModule(_setToken) {
        sync(_setToken, false);
    }

    /**
     * MODULE ONLY: Hook called prior to redemption to sync positions on SetToken. For redemption, always use
current borrowed balance after interest accrual.
     * Only callable by valid module.
     *
```

```
     * @param _setToken             Instance of the SetToken
     */
    function moduleRedeemHook(ISetToken _setToken, uint256 /* _setTokenQuantity */) external onlyModule(_setToken) {
        sync(_setToken, true);
    }

    /**
     * MODULE ONLY: Hook called prior to looping through each component on issuance. Invokes borrow in order for
module to return debt to issuer. Only callable by valid module.
     *
     * @param _setToken             Instance of the SetToken
     * @param _setTokenQuantity     Quantity of SetToken
     * @param _component            Address of component
     */
    function componentIssueHook(ISetToken _setToken, uint256 _setTokenQuantity, IERC20 _component, bool /* _isEquity
*/) external onlyModule(_setToken) {
        int256 componentDebt = _setToken.getExternalPositionRealUnit(address(_component), address(this));

        require(componentDebt < 0, "Component must be negative");

        uint256 notionalDebt = componentDebt.mul(-1).toUint256().preciseMul(_setTokenQuantity);

        _borrow(_setToken, underlyingToCToken[_component], notionalDebt);
    }

    /**
     * MODULE ONLY: Hook called prior to looping through each component on redemption. Invokes repay after issuance
module transfers debt from issuer. Only callable by valid module.
     *
     * @param _setToken             Instance of the SetToken
     * @param _setTokenQuantity     Quantity of SetToken
     * @param _component            Address of component
     */
```

```
    function componentRedeemHook(ISetToken _setToken, uint256 _setTokenQuantity, IERC20 _component, bool /*
_isEquity */) external onlyModule(_setToken) {
        int256 componentDebt = _setToken.getExternalPositionRealUnit(address(_component), address(this));

        require(componentDebt < 0, "Component must be negative");

        uint256 notionalDebt = componentDebt.mul(-1).toUint256().preciseMulCeil(_setTokenQuantity);

        _repayBorrow(_setToken, underlyingToCToken[_component], _component, notionalDebt);
    }


    /* ============ External Getter Functions ============ */

    /**
     * Get enabled assets for SetToken. Returns an array of enabled cTokens that are collateral assets and an
     * array of underlying that are borrow assets.
     *
     * @return                    Collateral cToken assets that are enabled
     * @return                    Underlying borrowed assets that are enabled.
     */
    function getEnabledAssets(ISetToken _setToken) external view returns(address[] memory, address[] memory) {
        return (
            enabledAssets[_setToken].collateralCTokens,
            enabledAssets[_setToken].borrowAssets
        );
    }

    /* ============ Internal Functions ============ */

    /**
     * Mints the specified cToken from the underlying of the specified notional quantity. If cEther, the WETH must
be
     * unwrapped as it only accepts the underlying ETH.
```

```
     */
    function _mintCToken(ISetToken _setToken, ICErc20 _cToken, IERC20 _underlyingToken, uint256 _mintNotional)
internal {
        if (_cToken == cEther) {
            _setToken.invokeUnwrapWETH(address(weth), _mintNotional);

            _setToken.invokeMintCEther(_cToken, _mintNotional);
        } else {
            _setToken.invokeApprove(address(_underlyingToken), address(_cToken), _mintNotional);

            _setToken.invokeMintCToken(_cToken, _mintNotional);
        }
    }

    /**
     * Invoke redeem from SetToken. If cEther, then also wrap ETH into WETH.
     */
    function _redeemUnderlying(ISetToken _setToken, ICErc20 _cToken, uint256 _redeemNotional) internal {
        _setToken.invokeRedeemUnderlying(_cToken, _redeemNotional);

        if (_cToken == cEther) {
            _setToken.invokeWrapWETH(address(weth), _redeemNotional);
        }
    }

    /**
     * Invoke repay from SetToken. If cEther then unwrap WETH into ETH.
     */
    function _repayBorrow(ISetToken _setToken, ICErc20 _cToken, IERC20 _underlyingToken, uint256 _repayNotional)
internal {
        if (_cToken == cEther) {
            _setToken.invokeUnwrapWETH(address(weth), _repayNotional);

            _setToken.invokeRepayBorrowCEther(_cToken, _repayNotional);
```

```
        } else {
            // Approve to cToken
            _setToken.invokeApprove(address(_underlyingToken), address(_cToken), _repayNotional);
            _setToken.invokeRepayBorrowCToken(_cToken, _repayNotional);
        }
    }

    /**
     * Invoke the SetToken to interact with the specified cToken to borrow the cToken's underlying of the specified
borrowQuantity.
     */
    function _borrow(ISetToken _setToken, ICErc20 _cToken, uint256 _notionalBorrowQuantity) internal {
        _setToken.invokeBorrow(_cToken, _notionalBorrowQuantity);
        if (_cToken == cEther) {
            _setToken.invokeWrapWETH(address(weth), _notionalBorrowQuantity);
        }
    }

    /**
     * Invokes approvals, gets trade call data from exchange adapter and invokes trade from SetToken
     */
    function _executeTrade(
        ActionInfo memory _actionInfo,
        IERC20 _sendToken,
        IERC20 _receiveToken,
        bytes memory _data
    )
        internal
        returns (uint256)
    {
         ISetToken setToken = _actionInfo.setToken;
         uint256 notionalSendQuantity = _actionInfo.notionalSendQuantity;

         setToken.invokeApprove(
```

```
            address(_sendToken),
            _actionInfo.exchangeAdapter.getSpender(),
            notionalSendQuantity
        );

        (
            address targetExchange,
            uint256 callValue,
            bytes memory methodData
        ) = _actionInfo.exchangeAdapter.getTradeCalldata(
            address(_sendToken),
            address(_receiveToken),
            address(setToken),
            notionalSendQuantity,
            _actionInfo.minNotionalReceiveQuantity,
            _data
        );

        setToken.invoke(targetExchange, callValue, methodData);

        uint256 receiveTokenQuantity =
_receiveToken.balanceOf(address(setToken)).sub(_actionInfo.preTradeReceiveTokenBalance);
        require(
            receiveTokenQuantity >= _actionInfo.minNotionalReceiveQuantity,
            "Slippage too high"
        );

        return receiveTokenQuantity;
    }

    /**
     * Calculates protocol fee on module and pays protocol fee from SetToken
     */
```

```
    function _accrueProtocolFee(ISetToken _setToken, IERC20 _receiveToken, uint256 _exchangedQuantity) internal
returns(uint256) {
        uint256 protocolFeeTotal = getModuleFee(PROTOCOL_TRADE_FEE_INDEX, _exchangedQuantity);

        payProtocolFeeFromSetToken(_setToken, address(_receiveToken), protocolFeeTotal);

        return protocolFeeTotal;
    }

    function _updateLeverPositions(ActionInfo memory actionInfo, IERC20 _borrowAsset) internal {
        _updateCollateralPosition(
            actionInfo.setToken,
            actionInfo.collateralCTokenAsset,
            _getCollateralPosition(
                actionInfo.setToken,
                actionInfo.collateralCTokenAsset,
                actionInfo.setTotalSupply
            )
        );

        _updateBorrowPosition(
            actionInfo.setToken,
            _borrowAsset,
            _getBorrowPosition(
                actionInfo.setToken,
                actionInfo.borrowCTokenAsset,
                actionInfo.setTotalSupply,
                false // Do not accrue interest
            )
        );
    }

    function _updateCollateralPosition(ISetToken _setToken, ICErc20 _cToken, uint256 _newPositionUnit) internal {
        _setToken.editDefaultPosition(address(_cToken), _newPositionUnit);
```

```
}

function _updateBorrowPosition(ISetToken _setToken, IERC20 _underlyingToken, int256 _newPositionUnit) internal {
    _setToken.editExternalPosition(address(_underlyingToken), address(this), _newPositionUnit, "");
}

/**
 * Construct the ActionInfo struct for lever and delever
 */
function _createAndValidateActionInfo(
    ISetToken _setToken,
    IERC20 _sendToken,
    IERC20 _receiveToken,
    uint256 _sendQuantityUnits,
    uint256 _minReceiveQuantityUnits,
    string memory _tradeAdapterName,
    bool _isLever
)
    internal
    view
    returns(ActionInfo memory)
{
    uint256 totalSupply = _setToken.totalSupply();

    return _createAndValidateActionInfoNotional(
        _setToken,
        _sendToken,
        _receiveToken,
        _sendQuantityUnits.preciseMul(totalSupply),
        _minReceiveQuantityUnits.preciseMul(totalSupply),
        _tradeAdapterName,
        _isLever
    );
}
```

```
    /**
     * Construct the ActionInfo struct for lever and delever accepting notional units
     */
    function _createAndValidateActionInfoNotional(
        ISetToken _setToken,
        IERC20 _sendToken,
        IERC20 _receiveToken,
        uint256 _notionalSendQuantity,
        uint256 _minNotionalReceiveQuantity,
        string memory _tradeAdapterName,
        bool _isLever
    )
        internal
        view
        returns(ActionInfo memory)
    {
        uint256 totalSupply = _setToken.totalSupply();
        ActionInfo memory actionInfo = ActionInfo ({
            exchangeAdapter: IExchangeAdapter(getAndValidateAdapter(_tradeAdapterName)),
            setToken: _setToken,
            collateralCTokenAsset: _isLever ? underlyingToCToken[_receiveToken] : underlyingToCToken[_sendToken],
            borrowCTokenAsset: _isLever ? underlyingToCToken[_sendToken] : underlyingToCToken[_receiveToken],
            setTotalSupply: totalSupply,
            notionalSendQuantity: _notionalSendQuantity,
            minNotionalReceiveQuantity: _minNotionalReceiveQuantity,
            preTradeReceiveTokenBalance: IERC20(_receiveToken).balanceOf(address(_setToken))
        });

        _validateCommon(actionInfo);

        return actionInfo;
    }
```

```
    function _validateCommon(ActionInfo memory _actionInfo) internal view {
        require(collateralCTokenEnabled[_actionInfo.setToken][_actionInfo.collateralCTokenAsset], "Collateral not
enabled");
        require(borrowCTokenEnabled[_actionInfo.setToken][_actionInfo.borrowCTokenAsset], "Borrow not enabled");
        require(_actionInfo.collateralCTokenAsset != _actionInfo.borrowCTokenAsset, "Must be different");
        require(_actionInfo.notionalSendQuantity > 0, "Quantity is 0");
    }

    function _getCollateralPosition(ISetToken _setToken, ICErc20 _cToken, uint256 _setTotalSupply) internal view
returns (uint256) {
        uint256 collateralNotionalBalance = _cToken.balanceOf(address(_setToken));
        return collateralNotionalBalance.preciseDiv(_setTotalSupply);
    }

    /**
     * Get borrow position. If should accrue interest is true, then accrue interest on Compound and use current
borrow balance, else use the stored value to save gas.
     * Use the current value for debt redemption, when we need to calculate the exact units of debt that needs to be
repaid.
     */
    function _getBorrowPosition(ISetToken _setToken, ICErc20 _cToken, uint256 _setTotalSupply, bool
_shouldAccrueInterest) internal returns (int256) {
        uint256 borrowNotionalBalance = _shouldAccrueInterest ? _cToken.borrowBalanceCurrent(address(_setToken)) :
_cToken.borrowBalanceStored(address(_setToken));
        // Round negative away from 0
        int256 borrowPositionUnit = borrowNotionalBalance.preciseDivCeil(_setTotalSupply).toInt256().mul(-1);

        return borrowPositionUnit;
    }
}
```

```
/*
    Copyright 2020 Set Labs Inc.

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

    SPDX-License-Identifier: Apache License, Version 2.0
*/

pragma solidity 0.6.10;
pragma experimental "ABIEncoderV2";

import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import { SafeMath } from "@openzeppelin/contracts/math/SafeMath.sol";
import { SafeCast } from "@openzeppelin/contracts/utils/SafeCast.sol";

import { IController } from "../../interfaces/IController.sol";
import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import { IExchangeAdapter } from "../../interfaces/IExchangeAdapter.sol";
import { IIntegrationRegistry } from "../../interfaces/IIntegrationRegistry.sol";
import { Invoke } from "../lib/Invoke.sol";
import { ISetToken } from "../../interfaces/ISetToken.sol";
import { ModuleBase } from "../lib/ModuleBase.sol";
import { Position } from "../lib/Position.sol";
import { PreciseUnitMath } from "../../lib/PreciseUnitMath.sol";
```

```
/**
 * @title TradeModule
 * @author Set Protocol
 *
 * Module that enables SetTokens to perform atomic trades using Decentralized Exchanges
 * such as 1inch or Kyber. Integrations mappings are stored on the IntegrationRegistry contract.
 */
contract TradeModule is ModuleBase, ReentrancyGuard {
    using SafeCast for int256;
    using SafeMath for uint256;

    using Invoke for ISetToken;
    using Position for ISetToken;
    using PreciseUnitMath for uint256;

    /* ============ Struct ============ */

    struct TradeInfo {
        ISetToken setToken;                             // Instance of SetToken
        IExchangeAdapter exchangeAdapter;               // Instance of exchange adapter contract
        address sendToken;                              // Address of token being sold
        address receiveToken;                           // Address of token being bought
        uint256 setTotalSupply;                         // Total supply of SetToken in Precise Units (10^18)
        uint256 totalSendQuantity;                      // Total quantity of sold token (position unit x total
supply)
        uint256 totalMinReceiveQuantity;                // Total minimum quantity of token to receive back
        uint256 preTradeSendTokenBalance;               // Total initial balance of token being sold
        uint256 preTradeReceiveTokenBalance;            // Total initial balance of token being bought
    }

    /* ============ Events ============ */

    event ComponentExchanged(
```

```
    address _setToken,
    address _sendToken,
    address _receiveToken,
    address _exchangeAdapter,
    uint256 _totalSendAmount,
    uint256 _totalReceiveAmount,
    uint256 _protocolFee
);

/* ============ Constants ============ */

// 0 index stores the fee % charged in the trade function
uint256 constant internal TRADE_MODULE_PROTOCOL_FEE_INDEX = 0;

/* ============ Constructor ============ */

constructor(IController _controller) public ModuleBase(_controller) {}

/* ============ External Functions ============ */

/**
 * Initializes this module to the SetToken. Only callable by the SetToken's manager.
 *
 * @param _setToken                Instance of the SetToken to initialize
 */
function initialize(
    ISetToken _setToken
)
    external
    onlyValidAndPendingSet(_setToken)
    onlySetManager(_setToken, msg.sender)
{
    _setToken.initializeModule();
}
```

```
/**
 * Executes a trade on a supported DEX. Only callable by the SetToken's manager.
 * @dev Although the SetToken units are passed in for the send and receive quantities, the total quantity
 * sent and received is the quantity of SetToken units multiplied by the SetToken totalSupply.
 *
 * @param _setToken             Instance of the SetToken to trade
 * @param _exchangeName         Human readable name of the exchange in the integrations registry
 * @param _sendToken            Address of the token to be sent to the exchange
 * @param _sendQuantity         Units of token in SetToken sent to the exchange
 * @param _receiveToken         Address of the token that will be received from the exchange
 * @param _minReceiveQuantity   Min units of token in SetToken to be received from the exchange
 * @param _data                 Arbitrary bytes to be used to construct trade call data
 */
function trade(
    ISetToken _setToken,
    string memory _exchangeName,
    address _sendToken,
    uint256 _sendQuantity,
    address _receiveToken,
    uint256 _minReceiveQuantity,
    bytes memory _data
)
    external
    nonReentrant
    onlyManagerAndValidSet(_setToken)
{
    TradeInfo memory tradeInfo = _getTradeInfo(
        _setToken,
        _exchangeName,
        _sendToken,
        _receiveToken,
        _sendQuantity,
        _minReceiveQuantity
```

```
    );

    _validatePreTradeData(tradeInfo, _sendQuantity);

    _executeTrade(tradeInfo, _data);

    uint256 exchangedQuantity = _validatePostTrade(tradeInfo);

    uint256 protocolFee = _accrueProtocolFee(tradeInfo, exchangedQuantity);

    (
        uint256 netSendAmount,
        uint256 netReceiveAmount
    ) = _updateSetTokenPositions(tradeInfo);

    emit ComponentExchanged(
        address(_setToken),
        _sendToken,
        _receiveToken,
        address(tradeInfo.exchangeAdapter),
        netSendAmount,
        netReceiveAmount,
        protocolFee
    );
}

/**
 * Removes this module from the SetToken, via call by the SetToken. Left with empty logic
 * here because there are no check needed to verify removal.
 */
function removeModule() external override {}

/* ============ Internal Functions ============ */
```

```
/**
 * Create and return TradeInfo struct
 *
 * @param _setToken             Instance of the SetToken to trade
 * @param _exchangeName         Human readable name of the exchange in the integrations registry
 * @param _sendToken            Address of the token to be sent to the exchange
 * @param _receiveToken         Address of the token that will be received from the exchange
 * @param _sendQuantity         Units of token in SetToken sent to the exchange
 * @param _minReceiveQuantity   Min units of token in SetToken to be received from the exchange
 *
 * return TradeInfo             Struct containing data for trade
 */
function _getTradeInfo(
    ISetToken _setToken,
    string memory _exchangeName,
    address _sendToken,
    address _receiveToken,
    uint256 _sendQuantity,
    uint256 _minReceiveQuantity
)
    internal
    view
    returns (TradeInfo memory)
{
    TradeInfo memory tradeInfo;

    tradeInfo.setToken = _setToken;

    tradeInfo.exchangeAdapter = IExchangeAdapter(getAndValidateAdapter(_exchangeName));

    tradeInfo.sendToken = _sendToken;
    tradeInfo.receiveToken = _receiveToken;

    tradeInfo.setTotalSupply = _setToken.totalSupply();
```

```
        tradeInfo.totalSendQuantity = Position.getDefaultTotalNotional(tradeInfo.setTotalSupply, _sendQuantity);

        tradeInfo.totalMinReceiveQuantity = Position.getDefaultTotalNotional(tradeInfo.setTotalSupply,
_minReceiveQuantity);

        tradeInfo.preTradeSendTokenBalance = IERC20(_sendToken).balanceOf(address(_setToken));
        tradeInfo.preTradeReceiveTokenBalance = IERC20(_receiveToken).balanceOf(address(_setToken));

        return tradeInfo;
    }

    /**
     * Validate pre trade data. Check exchange is valid, token quantity is valid.
     *
     * @param _tradeInfo            Struct containing trade information used in internal functions
     * @param _sendQuantity         Units of token in SetToken sent to the exchange
     */
    function _validatePreTradeData(TradeInfo memory _tradeInfo, uint256 _sendQuantity) internal view {
        require(_tradeInfo.totalSendQuantity > 0, "Token to sell must be nonzero");

        require(
            _tradeInfo.setToken.hasSufficientDefaultUnits(_tradeInfo.sendToken, _sendQuantity),
            "Unit cant be greater than existing"
        );
    }

    /**
     * Invoke approve for send token, get method data and invoke trade in the context of the SetToken.
     *
     * @param _tradeInfo            Struct containing trade information used in internal functions
     * @param _data                 Arbitrary bytes to be used to construct trade call data
     */
    function _executeTrade(
```

```
    TradeInfo memory _tradeInfo,
    bytes memory _data
)
    internal
{
    // Get spender address from exchange adapter and invoke approve for exact amount on SetToken
    _tradeInfo.setToken.invokeApprove(
        _tradeInfo.sendToken,
        _tradeInfo.exchangeAdapter.getSpender(),
        _tradeInfo.totalSendQuantity
    );

    (
        address targetExchange,
        uint256 callValue,
        bytes memory methodData
    ) = _tradeInfo.exchangeAdapter.getTradeCalldata(
        _tradeInfo.sendToken,
        _tradeInfo.receiveToken,
        address(_tradeInfo.setToken),
        _tradeInfo.totalSendQuantity,
        _tradeInfo.totalMinReceiveQuantity,
        _data
    );

    _tradeInfo.setToken.invoke(targetExchange, callValue, methodData);
}

/**
 * Validate post trade data.
 *
 * @param _tradeInfo                Struct containing trade information used in internal functions
 * @return uint256                  Total quantity of receive token that was exchanged
 */
```

```
    function _validatePostTrade(TradeInfo memory _tradeInfo) internal view returns (uint256) {
        uint256 exchangedQuantity = IERC20(_tradeInfo.receiveToken)
            .balanceOf(address(_tradeInfo.setToken))
            .sub(_tradeInfo.preTradeReceiveTokenBalance);

        require(
            exchangedQuantity >= _tradeInfo.totalMinReceiveQuantity,
            "Slippage greater than allowed"
        );

        return exchangedQuantity;
    }

    /**
     * Retrieve fee from controller and calculate total protocol fee and send from SetToken to protocol recipient
     *
     * @param _tradeInfo                Struct containing trade information used in internal functions
     * @return uint256                  Amount of receive token taken as protocol fee
     */
    function _accrueProtocolFee(TradeInfo memory _tradeInfo, uint256 _exchangedQuantity) internal returns (uint256)
{
        uint256 protocolFeeTotal = getModuleFee(TRADE_MODULE_PROTOCOL_FEE_INDEX, _exchangedQuantity);

        payProtocolFeeFromSetToken(_tradeInfo.setToken, _tradeInfo.receiveToken, protocolFeeTotal);

        return protocolFeeTotal;
    }

    /**
     * Update SetToken positions
     *
     * @param _tradeInfo                Struct containing trade information used in internal functions
     * @return uint256                  Amount of sendTokens used in the trade
     * @return uint256                  Amount of receiveTokens used in the trade (net of fees)
```

```
     */
    function _updateSetTokenPositions(TradeInfo memory _tradeInfo) internal returns (uint256, uint256) {
        (uint256 currentSendTokenBalance,,) = _tradeInfo.setToken.calculateAndEditDefaultPosition(
            _tradeInfo.sendToken,
            _tradeInfo.setTotalSupply,
            _tradeInfo.preTradeSendTokenBalance
        );

        (uint256 currentReceiveTokenBalance,,) = _tradeInfo.setToken.calculateAndEditDefaultPosition(
            _tradeInfo.receiveToken,
            _tradeInfo.setTotalSupply,
            _tradeInfo.preTradeReceiveTokenBalance
        );

        return (
            _tradeInfo.preTradeSendTokenBalance.sub(currentSendTokenBalance),
            currentReceiveTokenBalance.sub(_tradeInfo.preTradeReceiveTokenBalance)
        );
    }
}
```

```
/*
    Copyright 2020 Set Labs Inc.

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

    SPDX-License-Identifier: Apache License, Version 2.0
*/

pragma solidity 0.6.10;
pragma experimental "ABIEncoderV2";

import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import { SafeCast } from "@openzeppelin/contracts/utils/SafeCast.sol";
import { SafeMath } from "@openzeppelin/contracts/math/SafeMath.sol";

import { IController } from "../../interfaces/IController.sol";
import { IIntegrationRegistry } from "../../interfaces/IIntegrationRegistry.sol";
import { Invoke } from "../lib/Invoke.sol";
import { ISetToken } from "../../interfaces/ISetToken.sol";
import { IWETH } from "../../interfaces/external/IWETH.sol";
import { IWrapAdapter } from "../../interfaces/IWrapAdapter.sol";
import { ModuleBase } from "../lib/ModuleBase.sol";
import { Position } from "../lib/Position.sol";
```

```
import { PreciseUnitMath } from "../../lib/PreciseUnitMath.sol";

/**
 * @title WrapModule
 * @author Set Protocol
 *
 * Module that enables the wrapping of ERC20 and Ether positions via third party protocols. The WrapModule
 * works in conjunction with WrapAdapters, in which the wrapAdapterID / integrationNames are stored on the
 * integration registry.
 *
 * Some examples of wrap actions include wrapping, DAI to cDAI (Compound) or Dai to aDai (AAVE).
 */
contract WrapModule is ModuleBase, ReentrancyGuard {
    using SafeCast for int256;
    using PreciseUnitMath for uint256;
    using Position for uint256;
    using SafeMath for uint256;

    using Invoke for ISetToken;
    using Position for ISetToken.Position;
    using Position for ISetToken;

    /* ============ Events ============ */

    event ComponentWrapped(
        address indexed _setToken,
        address _underlyingToken,
        address _wrappedToken,
        uint256 _underlyingQuantity,
        string _integrationName
    );

    event ComponentUnwrapped(
        address indexed _setToken,
```

```
    address _underlyingToken,
    address _wrappedToken,
    uint256 _wrappedQuantity,
    string _integrationName
);

/* ============ State Variables ============ */

// Wrapped ETH address
IWETH public weth;

/* ============ Constructor ============ */

/**
 * @param _controller               Address of controller contract
 * @param _weth                     Address of wrapped eth
 */
constructor(IController _controller, IWETH _weth) public ModuleBase(_controller) {
    weth = _weth;
}

/* ============ External Functions ============ */

/**
 * MANAGER-ONLY: Instructs the SetToken to wrap an underlying asset into a wrappedToken via a specified adapter.
 *
 * @param _setToken             Instance of the SetToken
 * @param _underlyingToken      Address of the component to be wrapped
 * @param _wrappedToken         Address of the desired wrapped token
 * @param _underlyingUnits      Quantity of underlying units in Position units
 * @param _integrationName      Name of wrap module integration (mapping on integration registry)
 */
function wrap(
    ISetToken _setToken,
```

```
        address _underlyingToken,
        address _wrappedToken,
        uint256 _underlyingUnits,
        string calldata _integrationName
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        uint256 notionalUnderlyingWrapped = _validateWrapAndUpdate(
            _integrationName,
            _setToken,
            _underlyingToken,
            _wrappedToken,
            _underlyingUnits,
            false // does not use Ether
        );

        emit ComponentWrapped(
            address(_setToken),
            _underlyingToken,
            _wrappedToken,
            notionalUnderlyingWrapped,
            _integrationName
        );
    }

    /**
     * MANAGER-ONLY: Instructs the SetToken to wrap Ether into a wrappedToken via a specified adapter. Since
SetTokens
     * only hold WETH, in order to support protocols that collateralize with Ether the SetToken's WETH must be
unwrapped
     * first before sending to the external protocol.
     *
```

```
     * @param _setToken             Instance of the SetToken
     * @param _wrappedToken         Address of the desired wrapped token
     * @param _underlyingUnits      Quantity of underlying units in Position units
     * @param _integrationName      Name of wrap module integration (mapping on integration registry)
     */
    function wrapWithEther(
        ISetToken _setToken,
        address _wrappedToken,
        uint256 _underlyingUnits,
        string calldata _integrationName
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        uint256 notionalUnderlyingWrapped = _validateWrapAndUpdate(
            _integrationName,
            _setToken,
            address(weth),
            _wrappedToken,
            _underlyingUnits,
            true // uses Ether
        );

        emit ComponentWrapped(
            address(_setToken),
            address(weth),
            _wrappedToken,
            notionalUnderlyingWrapped,
            _integrationName
        );
    }

    /**
```

```
 * MANAGER-ONLY: Instructs the SetToken to unwrap a wrapped asset into its underlying via a specified adapter.
 *
 * @param _setToken             Instance of the SetToken
 * @param _underlyingToken      Address of the underlying asset
 * @param _wrappedToken         Address of the component to be unwrapped
 * @param _wrappedUnits         Quantity of wrapped tokens in Position units
 * @param _integrationName      ID of wrap module integration (mapping on integration registry)
 */
function unwrap(
    ISetToken _setToken,
    address _underlyingToken,
    address _wrappedToken,
    uint256 _wrappedUnits,
    string calldata _integrationName
)
    external
    nonReentrant
    onlyManagerAndValidSet(_setToken)
{
    uint256 notionalUnderlyingUnwrapped = _validateUnwrapAndUpdate(
        _integrationName,
        _setToken,
        _underlyingToken,
        _wrappedToken,
        _wrappedUnits,
        false // uses Ether
    );

    emit ComponentUnwrapped(
        address(_setToken),
        _underlyingToken,
        _wrappedToken,
        notionalUnderlyingUnwrapped,
        _integrationName
```

```
        );
    }

    /**
     * MANAGER-ONLY: Instructs the SetToken to unwrap a wrapped asset collateralized by Ether into Wrapped Ether.
Since
     * external protocol will send back Ether that Ether must be Wrapped into WETH in order to be accounted for by
SetToken.
     *
     * @param _setToken                 Instance of the SetToken
     * @param _wrappedToken             Address of the component to be unwrapped
     * @param _wrappedUnits             Quantity of wrapped tokens in Position units
     * @param _integrationName          ID of wrap module integration (mapping on integration registry)
     */
    function unwrapWithEther(
        ISetToken _setToken,
        address _wrappedToken,
        uint256 _wrappedUnits,
        string calldata _integrationName
    )
        external
        nonReentrant
        onlyManagerAndValidSet(_setToken)
    {
        uint256 notionalUnderlyingUnwrapped = _validateUnwrapAndUpdate(
            _integrationName,
            _setToken,
            address(weth),
            _wrappedToken,
            _wrappedUnits,
            true // uses Ether
        );

        emit ComponentUnwrapped(
```

```
        address(_setToken),
        address(weth),
        _wrappedToken,
        notionalUnderlyingUnwrapped,
        _integrationName
    );
}

/**
 * Initializes this module to the SetToken. Only callable by the SetToken's manager.
 *
 * @param _setToken             Instance of the SetToken to issue
 */
function initialize(ISetToken _setToken) external onlySetManager(_setToken, msg.sender) {
    require(controller.isSet(address(_setToken)), "Must be controller-enabled SetToken");
    require(isSetPendingInitialization(_setToken), "Must be pending initialization");
    _setToken.initializeModule();
}

/**
 * Removes this module from the SetToken, via call by the SetToken.
 */
function removeModule() external override {}


/* ============ Internal Functions ============ */

/**
 * Validates the wrap operation is valid. In particular, the following checks are made:
 * - The position is Default
 * - The position has sufficient units given the transact quantity
 * - The transact quantity > 0
 *
 * It is expected that the adapter will check if wrappedToken/underlyingToken are a valid pair for the given
```

```
     * integration.
     */
    function _validateInputs(
        ISetToken _setToken,
        address _transactPosition,
        uint256 _transactPositionUnits
    )
        internal
        view
    {
        require(_setToken.hasDefaultPosition(_transactPosition), "Target default position must be component");
        require(
            _setToken.hasSufficientDefaultUnits(_transactPosition, _transactPositionUnits),
            "Unit cant be greater than existing"
        );

        require(_transactPositionUnits > 0, "Target position units must be >0");
    }

    /**
     * The WrapModule calculates the total notional underlying to wrap, approves the underlying to the 3rd party
     * integration contract, then invokes the SetToken to call wrap by passing its calldata along. When raw ETH
     * is being used (_usesEther = true) WETH position must first be unwrapped and underlyingAddress sent to
     * adapter must be external protocol's ETH representative address.
     *
     * Returns notional amount of underlying tokens wrapped.
     */
    function _validateWrapAndUpdate(
        string calldata _integrationName,
        ISetToken _setToken,
        address _underlyingToken,
        address _wrappedToken,
        uint256 _underlyingUnits,
        bool _usesEther
```

```
)
    internal
    returns (uint256)
{
    _validateInputs(_setToken, _underlyingToken, _underlyingUnits);

    // Snapshot pre wrap balances
    (
        uint256 preActionUnderlyingNotional,
        uint256 preActionWrapNotional
    ) = _snapshotTargetAssetsBalance(_setToken, _underlyingToken, _wrappedToken);

    uint256 notionalUnderlying = _setToken.totalSupply().getDefaultTotalNotional(_underlyingUnits);
    IWrapAdapter wrapAdapter = IWrapAdapter(getAndValidateAdapter(_integrationName));

    // Execute any pre-wrap actions depending on if using raw ETH or not
    if (_usesEther) {
        _setToken.invokeUnwrapWETH(address(weth), notionalUnderlying);
    } else {
        address spender = wrapAdapter.getSpenderAddress(_underlyingToken, _wrappedToken);

        _setToken.invokeApprove(_underlyingToken, spender, notionalUnderlying);
    }

    // Get function call data and invoke on SetToken
    _createWrapDataAndInvoke(
        _setToken,
        wrapAdapter,
        _usesEther ? wrapAdapter.ETH_TOKEN_ADDRESS() : _underlyingToken,
        _wrappedToken,
        notionalUnderlying
    );

    // Snapshot post wrap balances
```

```
    (
        uint256 postActionUnderlyingNotional,
        uint256 postActionWrapNotional
    ) = _snapshotTargetAssetsBalance(_setToken, _underlyingToken, _wrappedToken);

    _updatePosition(_setToken, _underlyingToken, preActionUnderlyingNotional, postActionUnderlyingNotional);
    _updatePosition(_setToken, _wrappedToken, preActionWrapNotional, postActionWrapNotional);

    return preActionUnderlyingNotional.sub(postActionUnderlyingNotional);
}

/**
 * The WrapModule calculates the total notional wrap token to unwrap, then invokes the SetToken to call
 * unwrap by passing its calldata along. When raw ETH is being used (_usesEther = true) underlyingAddress
 * sent to adapter must be set to external protocol's ETH representative address and ETH returned from
 * external protocol is wrapped.
 *
 * Returns notional amount of underlying tokens unwrapped.
 */
function _validateUnwrapAndUpdate(
    string calldata _integrationName,
    ISetToken _setToken,
    address _underlyingToken,
    address _wrappedToken,
    uint256 _wrappedTokenUnits,
    bool _usesEther
)
    internal
    returns (uint256)
{
    _validateInputs(_setToken, _wrappedToken, _wrappedTokenUnits);

    (
        uint256 preActionUnderlyingNotional,
```

```
        uint256 preActionWrapNotional
    ) = _snapshotTargetAssetsBalance(_setToken, _underlyingToken, _wrappedToken);

    uint256 notionalWrappedToken = _setToken.totalSupply().getDefaultTotalNotional(_wrappedTokenUnits);
    IWrapAdapter wrapAdapter = IWrapAdapter(getAndValidateAdapter(_integrationName));

    // Get function call data and invoke on SetToken
    _createUnwrapDataAndInvoke(
        _setToken,
        wrapAdapter,
        _usesEther ? wrapAdapter.ETH_TOKEN_ADDRESS() : _underlyingToken,
        _wrappedToken,
        notionalWrappedToken
    );

    if (_usesEther) {
        _setToken.invokeWrapWETH(address(weth), address(_setToken).balance);
    }

    (
        uint256 postActionUnderlyingNotional,
        uint256 postActionWrapNotional
    ) = _snapshotTargetAssetsBalance(_setToken, _underlyingToken, _wrappedToken);

    _updatePosition(_setToken, _underlyingToken, preActionUnderlyingNotional, postActionUnderlyingNotional);
    _updatePosition(_setToken, _wrappedToken, preActionWrapNotional, postActionWrapNotional);

    return postActionUnderlyingNotional.sub(preActionUnderlyingNotional);
}

/**
 * Create the calldata for wrap and then invoke the call on the SetToken.
 */
function _createWrapDataAndInvoke(
```

```
    ISetToken _setToken,
    IWrapAdapter _wrapAdapter,
    address _underlyingToken,
    address _wrappedToken,
    uint256 _notionalUnderlying
) internal {
    (
        address callTarget,
        uint256 callValue,
        bytes memory callByteData
    ) = _wrapAdapter.getWrapCallData(
        _underlyingToken,
        _wrappedToken,
        _notionalUnderlying
    );

    _setToken.invoke(callTarget, callValue, callByteData);
}

/**
 * Create the calldata for unwrap and then invoke the call on the SetToken.
 */
function _createUnwrapDataAndInvoke(
    ISetToken _setToken,
    IWrapAdapter _wrapAdapter,
    address _underlyingToken,
    address _wrappedToken,
    uint256 _notionalUnderlying
) internal {
    (
        address callTarget,
        uint256 callValue,
        bytes memory callByteData
    ) = _wrapAdapter.getUnwrapCallData(
```

```
        _underlyingToken,
        _wrappedToken,
        _notionalUnderlying
    );

    _setToken.invoke(callTarget, callValue, callByteData);
}

/**
 * After a wrap/unwrap operation, check the underlying and wrap token quantities and recalculate
 * the units ((total tokens - airdrop)/ total supply). Then update the position on the SetToken.
 */
function _updatePosition(
    ISetToken _setToken,
    address _token,
    uint256 _preActionTokenBalance,
    uint256 _postActionTokenBalance
) internal {
    uint256 newUnit = _setToken.totalSupply().calculateDefaultEditPositionUnit(
        _preActionTokenBalance,
        _postActionTokenBalance,
        _setToken.getDefaultPositionRealUnit(_token).toUint256()
    );

    _setToken.editDefaultPosition(_token, newUnit);
}

/**
 * Take snapshot of SetToken's balance of underlying and wrapped tokens.
 */
function _snapshotTargetAssetsBalance(
    ISetToken _setToken,
    address _underlyingToken,
    address _wrappedToken
```

```
    ) internal view returns(uint256, uint256) {
        uint256 underlyingTokenBalance = IERC20(_underlyingToken).balanceOf(address(_setToken));
        uint256 wrapTokenBalance = IERC20(_wrappedToken).balanceOf(address(_setToken));

        return (
            underlyingTokenBalance,
            wrapTokenBalance
        );
    }
}
```