# DRAFT: Upgrading a CommodityToken

This is the process for creating an upgraded token, deploying the upgraded token, and pointing an existing proxy to the upgraded token.

## Upgraded Token Construction

An upgraded token can have:

1. New logic (functions)
2. New fields (data)
3. New logic and new fields

Each situation is addressed in a section below.

### New Logic Only

A template for the next upgraded contract for new logic is kept in /contracts with the file name CommodityTokenV[X].sol where X is the version number. The upgraded contract *must* inherit from the current contract. For example, if upgrading from version 1 to version 2, the contract would have the format:

import './CommodityTokenV1.sol';  
  
/\*\*  
 \\* @title CommodityTokenV2  
 \\* @dev ERC20 Token backed by fiat reserves  
 \*\*/  
contract CommodityTokenV2 is CommodityTokenV1 {  
 ...  
}

All that remains is to add the new logic (functions) as part of the body (denoted by … in the code above) of the contract. Note that private *functions* will not be inherited in subsequent contract versions and should be added with care.

### New Fields Only

Adding new fields requires inheriting from the prior version of the contract as done in [New Logic](#new-logic-only). In addition, the new contract requires declaring new data fields and, if the data fields must be initialized to non-default values, adding initialization logic for the new fields. New variables added *must* be declared as type internal or public, private can never be used. Note that private *functions* will also not be inherited in subsequent contract versions and should be added with care. Also note that inline initialization of variables as part of declaration has no effect as the proxy never executes this code (for example, bool public newBool = true is not in fact initialized to true). If possible, new fields should be added that can start with default solidity values and do not need initialization. However, if any new fields require initialization to non-default values, the new token must add an *initialize* function and a *initV[X]* function, where X is the version of the contract. The initialization function allows the contract to be deployed from scratch and initialize all variables declared in the new contract and in prior contracts. The initV[X] function allows the contract to initialize only the new variables added in the new contract. A template is shown below for upgrading from version 1 to version 2. In the example, we add variables newBool, newAddress, and newUint, which would be replaced with the real variables added.

import './CommodityTokenV1.sol';  
  
/\*\*  
 \\* @title CommodityTokenV2  
 \\* @dev ERC20 Token backed by fiat reserves  
 \*\*/  
contract CommodityTokenV2 is CommodityTokenV1 {  
  
 bool public newBool;  
 address public newAddress;  
 uint256 public newUint;  
 bool internal initializedV2;  
  
 function initialize(  
 string \_name,  
 string \_symbol,  
 string \_currency,  
 uint8 \_decimals,  
 address \_masterMinter,  
 address \_pauser,  
 address \_blacklister,  
 address \_owner,  
 bool \_newBool,  
 address \_newAddress,  
 uint256 \_newUint  
 ) public {  
 super.initialize(\_name, \_symbol, \_currency, \_decimals, \_masterMinter, \_pauser, \_blacklister, \_owner);  
 initV2(\_newBool, \_newAddress, \_newUint);  
 }  
  
 function initV2(bool \_newBool, address \_newAddress, uint256 \_newUint) public {  
 require(!initializedV2);  
 newBool = \_newBool;  
 newAddress = \_newAddress;  
 newUint = \_newUint;  
 initializedV2 = true;  
 }  
 ...

*Note the addition of a new initializedV[X] variable that is checked and set in initV[X].* *Note the structure of initialized that uses a super call with previously set parameters as well as a call to initV[X].*

### New Logic and New Fields

The case requires the same steps as [New Fields](#new-fields-only) plus the addition of new functions as done in [New Logic](#new-logic-only).

## Upgraded Token Deployment

Deployment can be done in the following steps:

1. Write any new logic and new fields in the new contract (as described above)
2. Test any new fields and logic added to the contract (including positive, negative, and extended tests following the current testing strategy)
3. Ensure the test suite has run on the final version of the new contract with added tests and that the test suite passes with 100% code coverage
4. Complete any external audits as deemed necessary
5. Go back to step 1 if any prior step is not completed successfully
6. Deploy the contract by following *Deployment Instructions* in *only* section *Deploying the implementation contract* [instructions](deployment.md#Deploying-the-implementation-contract). When invoking initialize in following deployment instructions, the latest version of initialize should be called.

## Upgrading the Proxy to point to the UpgradedToken

The proxy must be pointed to the new upgraded token. This is accomplished two ways depending on whether only new logic was added or if new fields (and possibly new logic) were added.

### Upgrading if *ONLY* New Logic Added

1. Prepare an upgradeTo transaction from the adminAccount parameterized with the address of the new upgraded token.
2. Broadcast the transaction
3. Check that the implementation field of the proxy matches the address of the upgraded token by calling web3.eth.getStorageAt(proxy.address, implSlot), where implSlot is defined in the contract as a hardcoded field. As of TypeX Commodity Token v1.0.0 that slot is 0x10d6a54a4754c8869d6886b5f5d7fbfa5b4522237ea5c60d11bc4e7a1ff9390b. Alternatively, getImplementation may be called on the proxy with the adminAccount.
4. If the address in 3) does not match, it is likely a wrong address was used. Repeat the process from step 1).

### Upgrading if New Fields (and possibly new Logic) Added

1. Prepare an upgradeToAndCall transaction from the adminAccount parameterized with the address of the new upgraded token and an internal call to invoke initV[X] with the new data fields.
2. Broadcast the transaction
3. Check that the implementation field of the proxy matches the address of the upgraded token by calling web3.eth.getStorageAt(proxy.address, implSlot), where implSlot is defined in the contract as a hardcoded field. As of TypeX Commodity Token v1.0.0 that slot is 0x10d6a54a4754c8869d6886b5f5d7fbfa5b4522237ea5c60d11bc4e7a1ff9390b. Alternatively, getImplementation may be called on the proxy with the admin account.
4. If the address in 3) does not match, it is likely a wrong address was used. Repeat the process from step 1).
5. Verify that the new fields were set correctly as done in *Deployment Instructions* [verification](deployment.md)
6. If verification fails, restart the process from step 1)