# MasterMinter contract

The MasterMinter is a governance contract. It delegates the functionality of the masterMinter role in the TypeX NATGX contract to multiple addresses. (The masterMinter role can add and remove minters from a NATGXToken and set their allowances.) The MasterMinter contract delegates the minter management capability to controllers. Each controller manages exactly one minter, and a single minter may be managed by multiple controllers. This allows separation of duties (off-line key management) and simplifies nonce management for warm transactions.

Minters and NATGXToken holders are not affected by replacing a masterMinter user address with a MasterMinter contract.

# Roles

The MasterMinter contract has the following roles:

* owner - adds and removes controllers, sets the address of the minterManager, and sets the owner.
* minterManager - address of a contract (e.g. NATGX) with a MinterManagementInterface. The minterManager contract stores information about minter allowances and which minters are enabled/disabled.
* controller - each controller manages exactly one minter. A controller can enable/disable its minter, and modify the minting allowance by calling functions on the MasterMinter contract, and MasterMinter will call the appropriate functions on the minterManager.
* minter - each minter is managed by one or more controller. The minter cannot perform any actions on the MasterMinter contract. It interacts only with the NATGXToken contract.

# Interaction with NATGXToken contract

The owner of the NATGXToken contract can set the masterMinter role to point to the address of the MasterMinter contract. This enables the MasterMinter contract to call minter management functions on the NATGXToken contract:

* configureMinter(minter, allowance) - Enables the minter and sets its minting allowance.
* removeMinter(minter) - Disables the minter and sets its minting allowance to 0.
* isMinter(minter) - Returns true if the minter is enabled, and false otherwise.
* minterAllowance(minter) - Returns the minting allowance of the minter.

Together, these four functions are defined as the MinterManagementInterface. The MasterMinter contains the address of a minterManager that implements the MinterManagementInterface. The MasterMinter interacts with the NATGX token via the minterManager.

When a controller calls a function on MasterMinter, the MasterMinter will call the appropriate function on the NATGXToken contract on its behalf. Both the MasterMinter and the NATGXToken do their own access control.

# Function Summary

* configureController(controller, minter) - The owner assigns the controller to manage the minter. This allows the controller to call configureMinter, incrementMinterAllowance and removeMinter. Note: configureController(controller, 0x00) is forbidden because it has the effect of removing the controller.
* removeController(controller) - The owner disables the controller by setting its minter to 0x00.
* setMinterManager(minterManager) - The owner sets a new contract to the minterManager address. This has no effect on the old minterManager contract. If the new minterManager does not implement the MinterManagementInterface or does not give this instance of the MasterMinter contract permission to call minter management functions then the controller calls to configureMinter, incrementMinterAllowance, and removeMinter will throw.
* configureMinter(allowance) - A controller enables its minter and sets its allowance. The MasterMinter contract will call the minterManager contract on the controller’s behalf.
* incrementMinterAllowance - A controller increments the allowance of an enabled minter (incrementMinterAllowance will throw if the minter is disabled). The MasterMinter contract will call the minterManager contract on the controller’s behalf.
* removeMinter - A controller disables a minter. The MasterMinter contract will call the minterManager contract on the controller’s behalf.

# Deployment

The MasterMinter may be deployed independently of the NATGXToken contract (e.g. NATGX).

* NATGXToken then MasterMinter. Deploy MasterMinter and set the minterManager to point to the NATGXToken in the constructor. Then use the MasterMinter owner role to configure at least one controller for each existing minter in the NATGXToken. Once the MasterMinter is fully configured, use the NATGXToken owner role to broadcast an updateMasterMinter transaction setting masterMinter role to the MasterMinter contract address.
* MasterMinter then NATGXToken. Deploy MasterMinter and set the minterManager to point to address 0x00 in the constructor. Then deploy the NATGXToken and set the masterMinter to be the address of the MasterMinter contract in the constructor. Next, use the MasterMinter owner to set the minterManager and configure controllers.

# Configuring the MasterMinter

We recommend assigning at least two controllers to each minter.

* AllowanceController. Use this controller to enable the minter with a single configureMinter transaction, and then use it exclusively to sign incrementMinterAllowance transactions. There may be multiple AllowanceControllers that sign different size allowance increment transactions.
* SecurityController. Use this controller to sign a single removeMinter transaction and store it for emergencies.

The private keys to the AllowanceController and SecurityController should stay in cold storage. This configuration lets the Controller keep multiple warm incrementMinterAllowance transactions on hand, as well as the removeMinter transaction in case of a problem. Broadcasting the removeMinter transaction will cause all future incrementMinterAllowance transactions to throw. Since the two types of transactions are managed by different addresses, there is no need to worry about nonce management.

# MasterMinter vs. MintController

Creating a MasterMinter contract that *inherits* from a MintController contract with no changes may seem like a curious design choice. This leaves open the possibility of creating other contracts that inherit from MintController without creating naming confusion due to their different functionality.