**TESTING CORE IDENTITY**

Note that by running COIDTesting.js, all the tests are conducted, and if they pass/fail is noted. This file can be used when testing COID in a new environment by running the command,

**node COIDTesting.js**

Here are some of the parameters that we start off with. To see all parameters, please see COIDTesting.js. This document is made just for a quick overview of testing of core identity.

|  |  |  |  |
| --- | --- | --- | --- |
| **Owner Hash** | “A1” | “A2” | “A3” |
| **Owner Value** | 3 | 3 | 3 |
| **Controller Hash** | “A1” | “A2” | “C1” |
| **Controller Value** | 5 | 5 | 5 |

As seen above, at instantiation, there exists a controller “A3” who is not specified as a controller. Additionally, there is a controller “C1” who is not an Owner. It is verified that the contract by default adds the owner “A3” as a controller by StartCoid().

*Note, the internal functions for calculating control token ID and ownership token ID have been verified to work*

1. The javascript calls the functions in the following order, which have been verified to work via temporary debugging functions:

*setUniqueID(theUniqueID, theUniqueIDAttributes,isHumanValue)*

*setOwnership(theOwnerIDList,theOwnershipStakes)*

*setControl(theControlTokens,theControlIDList)*

*setRecovery(theIdentityRecoveryList,theRecoveryCondition)*

*StartCoid()*

1. We call the function: **delegate**(“A1”,”D1”,2) to have the Controller “A1” to delegate “D1” two tokens. This function is verified by the response of true. We then call the function **myAmount**(“D1”), which further verifies delegate as it returns 2.
2. We can the function **spendMyTokens(**“D1”,1). Subsequently, we call **myAmount**(“D1”), which returns one token as expected.
3. We then call the function **amountDelegated**(“A1”), which returns 1 as expected. This is because “A1” initially delegated “D1” two tokens, but “D1” spent one token, which is returned to “A1”.
4. We then call the function **revokeDelegation**(“A1”, “D1”,1), which revokes the delegation of one token to “D1”. This means “D1” should now have no tokens delegated to them, which is verified by the result of zero returned from the function **myAmount**(“D1”). Additionally, revokeDelegation returned true.
5. We then call the function **addController**(“C4”). The result is true. Subsequently, we call **changeTokenController**(“C1”,”C4”,1) which gives “C4” one token from “C1”. The result is also true.
6. Next, we have “C4” delegate one token to “D1” using the function **delegate**(“C4”,”D1”,1). It returns true, so this adds additional trust in (6.) that “C4” had a token to delegate.
7. After that, we call **removeController**(“C4”), which returns true. Subsequently, calling **myAmount**(“D1”) returns one as desired, meaning that “D1” still has one token after removal of “C4”.
8. We now check the amount of tokens possessed by everyone using the temporary function **getList**. We see that the token amounts are as desired, and “A3” who initially had no tokens now has one token absorbed from the removal of “C4”. This is expected—when a controller is removed, their tokens go to the controller with the least amount of tokens.
9. We call the function **addOwner**(“A5”,5), which returns true, and then check the amount of tokens “A5” has, which is successfully returned as 5 by calling the function **myTokenAmount**(“A5”). Then we call the function **myTokenAmount**(“A3”) and see that “A3” has three tokens as expected.
10. We now call the function **giveTokens**(“A5”,”A1”,5), and it returns true. Calling the function **myTokenAmount**(“A5”) verifies that “A5” gave all of their tokens, as the result of the function is true. Then we call **removeOwner**(“A5”) which returns true.