

Appendix 1: Reference Design of Instruction Opcodes

```
/*
 * @notice opcode enum is used to represent the instruction of the DARC protocol.
 */

enum EnumOpcode {

    /**
     * @notice Invalid Operation
     * ID: 0
     */
    UNDEFINED,

    /**
     * @notice Batch Mint Token Operation
     * @param ADDRESS_2DARRAY[0] address[] toAddressArray: the array of the address to mint
     new token to
     * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
     index to mint new token from
     * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount of the token
     to mint
     *
     * ID: 1
     */
    BATCH_MINT_TOKENS,

    /**
     * @notice Batch Create Token Class Operation
     * @param STRING_ARRAY[] string[] nameArray: the array of the name of the token class
     to create
     * @param UINT256_2DARRAY[0] uint256[] tokenIndexArray: the array of the token index
     of the token class to create
     * @param UINT256_2DARRAY[1] uint256[] votingWeightArray: the array of the voting weight
     of the token class to create
     * @param UINT256_2DARRAY[2] uint256[] dividendWeightArray: the array of the dividend
     weight of the token class to create
     *
     * ID:2
     */
    BATCH_CREATE_TOKEN_CLASSES,

    /**
     * @notice Batch Transfer Token Operation
     * @param ADDRESS_2DARRAY[0] address[] toAddressArray: the array of the address to transfer
     token to
     * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
     index to transfer token from
     * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount of the token
     to transfer
     *
     * ID:3
     */
    BATCH_TRANSFER_TOKENS,

    /**
     * @notice Batch Transfer Token From Addr A to Addr B Operation
```

```

    * @param ADDRESS_2DARRAY[0] address[] fromAddressArray: the array of the address to
transfer token from
    * @param ADDRESS_2DARRAY[1] address[] toAddressArray: the array of the address to transfer
token to
    * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
index to transfer token from
    * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount of the token
to transfer
    *
    * ID:4
    */
    BATCH_TRANSFER_TOKENS_FROM_TO,

/**
    * @notice Batch Burn Token Operation
    * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
index to burn token from
    * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount of the token
to burn
    *
    * ID:5
    */
    BATCH_BURN_TOKENS,

/**
    * @notice Batch Burn Token From Addr A Operation
    * @param ADDRESS_2DARRAY[0] address[] fromAddressArray: the array of the address to
burn token from
    * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
index to burn token from
    * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount of the token
to burn
    *
    * ID:6
    */
    BATCH_BURN_TOKENS_FROM,

/**
    * @notice Batch Add Member Operation
    * @param ADDRESS_2DARRAY[0] address[] memberAddressArray: the array of the address
to add as member
    * @param UINT256_2DARRAY[0] uint256[] memberRoleArray: the array of the role of the
member to add
    * @param STRING_ARRAY string[] memberNameArray: the array of the name of the member
to add
    *
    * ID:7
    */
    BATCH_ADD_MEMBERSHIP,

/**
    * @notice Batch Suspend Member Operation
    * @param ADDRESS_2DARRAY[0] address[] memberAddressArray: the array of the address
to suspend as member
    *
    * ID:8

```

```

    */
    BATCH_SUSPEND_MEMBERSHIP,

/**
 * @notice Batch Resume Member Operation
 * @param ADDRESS_2DARRAY[0] address[] memberAddressArray: the array of the address
to reinstate as member
 *
 * ID:9
 */
    BATCH_RESUME_MEMBERSHIP,

/**
 * @notice Batch Change Member Role Operation
 * @param ADDRESS_2DARRAY[0] address[] memberAddressArray: the array of the address
to change role of as member
 * @param UINT256_2DARRAY[0] uint256[] memberRoleArray: the array of the role of the
member to change
 *
 * ID:10
 */
    BATCH_CHANGE_MEMBER_ROLES,

/**
 * @notice Batch Change Member Name Operation
 * @param ADDRESS_2DARRAY[0] address[] memberAddressArray: the array of the address
to change name of as member
 * @param STRING_ARRAY string[] memberNameArray: the array of the name of the member
to change
 *
 * ID:11
 */
    BATCH_CHANGE_MEMBER_NAMES,

/**
 * @notice Batch Add Emergency Agent Operation
 * @param Plugin[] pluginList: the array of the plugins
 * ID:12
 */
    BATCH_ADD_PLUGINS,

/**
 * @notice Batch Enable Plugin Operation
 * @param UINT256_ARRAY[0] uint256[] pluginIndexArray: the array of the plugins index
to enable
 * @param BOOL_ARRAY bool[] isBeforeOperationArray: the array of the flag to indicate
if the plugin is before operation
 * ID:13
 */
    BATCH_ENABLE_PLUGINS,

/**
 * @notice Batch Disable Plugin Operation
 * @param UINT256_ARRAY[0] uint256[] pluginIndexArray: the array of the plugins index
to disable
 * @param BOOL_ARRAY bool[] isBeforeOperationArray: the array of the flag to indicate

```

```

if the plugin is before operation
    * ID:14
    */
    BATCH_DISABLE_PLUGINS,

/**
    * @notice Batch Add and Enable Plugin Operation
    * @param Plugin[] pluginList: the array of the plugins
    * ID:15
    */
    BATCH_ADD_AND_ENABLE_PLUGINS,

/**
    * @notice Batch Set Parameter Operation
    * @param MachineParameter[] parameterNameArray: the array of the parameter name
    * @param UINT256_2DARRAY[0] uint256[] parameterValueArray: the array of the parameter
value
    * ID:16
    */
    BATCH_SET_PARAMETERS,

/**
    * @notice Batch Add Withdrawable Balance Operation
    * @param address[] addressArray: the array of the address to add withdrawable balance
    * @param uint256[] amountArray: the array of the amount to add withdrawable balance
    * ID:17
    */
    BATCH_ADD_WITHDRAWABLE_BALANCES,

/**
    * @notice Batch Reduce Withdrawable Balance Operation
    * @param address[] addressArray: the array of the address to subtract withdrawable
balance
    * @param uint256[] amountArray: the array of the amount to subtract withdrawable balance
    * ID:18
    */
    BATCH_REDUCE_WITHDRAWABLE_BALANCES,

/**
    * @notice Batch Add Voting Rules
    * @param VotingRule[] votingRuleList: the array of the voting rules
    * ID:19
    */
    BATCH_ADD_VOTING_RULES,

/**
    * @notice Batch Pay to Mint Tokens Operation
    * @param ADDRESS_2DARRAY[0] address[] addressArray: the array of the address to mint
tokens
    * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
index to mint tokens
    * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount to mint
tokens
    * @param UINT256_2DARRAY[2] uint256[] priceArray: the price of each token class to
mint

```

```

    * @param UINT256_2DARRAY[3] uint256[1] dividendableFlag: the flag to indicate if the
payment is dividendable. 1 for yes (pay for purchase), 0 for no (pay for investment)
    * ID:20
    */
    BATCH_PAY_TO_MINT_TOKENS,

/**
    * @notice Pay some cash to transfer tokens (can be used as product coins)
    * @param ADDRESS_2DARRAY[0] address[] toAddressArray: the array of the address to transfer
token to
    * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: the array of the token class
index to transfer token from
    * @param UINT256_2DARRAY[1] uint256[] amountArray: the array of the amount of the token
to transfer
    * @param UINT256_2DARRAY[2] uint256[] priceArray: the price of each token class to
transfer
    * @ param UINT256_2DARRAY[3] uint256[1] dividendableFlag: the flag to indicate if the
payment is dividendable. 1 for yes (pay for purchase), 0 for no (pay for investment)
    * ID:21
    */
    BATCH_PAY_TO_TRANSFER_TOKENS,

/**
    * @notice Add an array of address as emergency agents
    * (can be used as product NFTs with a new unique token class)
    * @param ADDRESS_2DARRAY[0] address[] The array of the address to add as emergency
agents
    * ID:22
    */
    ADD_EMERGENCY,

/**
    * @notice withdraw cash from the contract's cash balance
    * @param address[] addressArray: the array of the address to withdraw cash to
    * @param uint256[] amountArray: the array of the amount of cash to withdraw
    * ID:23
    */
    WITHDRAW_CASH_TO,

/**
    * @notice Call emergency agents to handle emergency situations
    * @param UINT256_2DARRAY[0] address[] addressArray: the array of the emergency agents
index to call
    * ID:24
    */
    CALL_EMERGENCY,

/**
    * @notice Call a contract with the given abi
    * @param address contractAddress: the address of the contract to call
    * @param bytes abi: the abi of the function to call
    * ID:25
    */
    CALL_CONTRACT_ABI,

```

```

/**
 * @notice Pay some cash
 * @param uint256 amount: the amount of cash to pay
 * @param uint256 paymentType: the type of cash to pay, 0 for ethers/matic/original
tokens
 * 1 for USDT, 2 for USDC (right now only 0 is supported), 3 for DAI ...
 * @param uint256 dividendable: the flag to indicate if the payment is dividendable,
 * 0 for no (pay for investment), 1 for yes (pay for purchase)
 * ID:26
 */
PAY_CASH,

/**
 * @notice Calculate the dividends and offer to token holders
 * by adding the dividends to the withdrawable balance of each token holder
 *
 * ID:27
 */
OFFER_DIVIDENDS,

/**
 * @notice Withdraw dividends from the withdrawable dividends balance
 * @param address[] addressArray: the array of the address to withdraw dividends to
 * @param uint256[] amountArray: the array of the amount of dividends to withdraw
 * ID:28
 */
WITHDRAW_DIVIDENDS_TO,

/**
 * @notice Set the approval for all transfer operations by address
 * @param address: the address to set approval for all transfer operations
 * ID:29
 */
SET_APPROVAL_FOR_ALL_OPERATIONS,

/**
 * @notice Batch Burn tokens and Refund
 * @param UINT256_2D[0] uint256[] tokenClassArray: the array of the token class index
to burn tokens from
 * @param UINT256_2D[1] uint256[] amountArray: the array of the amount of the token
to burn
 * @param UINT256_2D[2] uint256[] priceArray: the price of each token class to burn
 * ID:30
 */
BATCH_BURN_TOKENS_AND_REFUND,

/**
 * @notice Add storage IPFS hash to the storage list permanently
 * @param STRING_2DARRAY[0] address: the address to set approval for all cash withdraw
operations
 * ID:31
 */
ADD_STORAGE_IPFS_HASH,

```

```

/**
 * Below are two operations than can be used during voting pending process
 */

/**
 * @notice Vote for a voting pending program
 * @param bool[] voteArray: the array of the vote for each program
 * ID:32
 */
VOTE,

/**
 * @notice Execute a program that has been voted and approved
 * ID:33
 */
EXECUTE_PROGRAM,

/**
 * @notice Emergency mode termination. Emergency agents cannot do anything after this
operation
 * ID:34
 */
END_EMERGENCY,

/**
 * @notice Upgrade the contract to a new contract address
 * @param ADDRESS_2DARRAY[0][0] The address of the new contract
 * ID:35
 */
UPGRADE_TO_ADDRESS,

/**
 * @notice Accepting current DARCs to be upgraded from the old contract address
 * @param ADDRESS_2DARRAY[0][0] The address of the old contract
 * ID:36
 */
CONFIRM_UPGRAED_FROM_ADDRESS,

/**
 * @notice Upgrade the contract to the latest version
 * ID:37
 */
UPGRADE_TO_THE_LATEST,

/**
 * @notice Batch Pay to Trasnfer Tokens Operation
 * ID:38
 */
op_BATCH_PAY_TO_TRANSFER_TOKENS
}

```

Appendix 2: Reference Design of Program and Operation

```

/**

```

```

    * The parameter(s) or operand(s) of the operation
    */
struct Param {
    uint256[] UINT256_ARRAY;
    address[] ADDRESS_ARRAY;
    string[] STRING_ARRAY;
    bool[] BOOL_ARRAY;
    VotingRule[] VOTING_RULE_ARRAY;
    Plugin[] PLUGIN_ARRAY;
    MachineParameter[] PARAMETER_ARRAY;
    uint256[][] UINT256_2DARRAY;
    address[][] ADDRESS_2DARRAY;
}

/**
 * The operation to be executed, including the operator address, the opcode and the parameters
 */
struct Operation {
    address operatorAddress;
    EnumOpcode opcode;
    Param param;
}

/**
 * The program to be executed, including the operator address and the operation array
 */
struct Program {
    address programOperatorAddress;

    /**
     * @notice operations: the array of the operations to be executed
     */
    Operation[] operations;
}

```

Appendix 3: Reference Design of Plugin

```

/**
 * The condition node types
 */
enum EnumConditionNodeType { UNDEFINED, EXPRESSION, LOGICAL_OPERATOR, BOOLEAN_TRUE, BOOLEAN_FALSE}

/**
 * The logical operator types
 */
enum EnumLogicalOperatorType {UNDEFINED, AND, OR, NOT }

enum EnumReturnType {

    /**
     * The default value. The plugin system will return UNDEFINED if no plugin is triggered.
     * Both BEFORE and AFTER operation plugin system may return UNDEFINED.
     */
    UNDEFINED,

```



```

/**
 * The operation is approved but must be executed in sandbox to check if the operation
 * is valid in the current machine state.
 * Only BEFORE operation plugin system may return SANDBOX_NEEDED.
 */
SANDBOX_NEEDED,

/**
 * The operation is disapproved and should be rejected at this level.
 * Both BEFORE and AFTER operation plugin system may return NO.
 */
NO,

/**
 * The decision is pending and a voting item should be created at this level.
 * Only AFTER operation plugin system may return VOTING_NEEDED.
 */
VOTING_NEEDED,

/**
 * The operation is approved and should skip the sandbox check.
 * Only BEFORE operation plugin system may return YES_AND_SKIP_SANDBOX.
 */
YES_AND_SKIP_SANDBOX,

/**
 * The operation is finally approved at this level.
 * Only AFTER operation plugin system may return YES.
 */
YES
}

/**
 * The condition node expression parameters
 */
struct NodeParam {
    uint256[] UINT256_ARRAY;
    address[] ADDRESS_ARRAY;
    string[] STRING_ARRAY;
    uint256[][] UINT256_2DARRAY;
    address[][] ADDRESS_2DARRAY;
    string[][] STRING_2DARRAY;
}

/**
 * The condition node struct
 */
struct ConditionNode {
    /**
     * current condition node index
     */
    uint256 id;

    /**

```

```

    * the type of current condition node
    */
EnumConditionNodeType nodeType;

/**
 * the logic operator of the current condition node
 */
EnumLogicalOperatorType logicalOperator;

/**
 * the condition expression of the current condition node
 */
EnumConditionExpression conditionExpression;

/**
 * a list of the child nodes of the current condition node
 */
uint256[] childList;

/**
 * The array of the EXPRESSION node parameters
 */
NodeParam param;
}

/**
 * The struct of the plugin
 */
struct Plugin {
    /**
     * the return type of the current condition node
     */
    EnumReturnType returnType;

    /**
     * the level of restriction, from 0 to the maximum value of uint256
     */
    uint256 level;

    /**
     * condition binary expression tree vector
     */
    ConditionNode[] conditionNodes;

    /**
     * the voting rule id of the current plugin if the return type is VOTING_NEEDED
     */
    uint256 votingRuleIndex;

    /**
     * the plugin note
     */
    string note;

    /**
     * the boolean that indicates whether the plugin is enabled or not

```

```
    */
    bool bIsEnabled;

    /**
     * the boolean that indicates whether the plugin is deleted or not
     */
    bool bIsInitialized;

    /**
     * the boolean that indicates whether the plugin is a before operation
     * plugin or after operation plugin
     */
    bool bIsBeforeOperation;
}
```