TWiG Cheat Sheet

A list of functions you may find useful in your quest to issue and trade assets. A link to the docs is provided for more detail.

TWiG

getIssuerParty(serviceHub: ServiceHub): Party
A TWiG function that returns the "Issuer" Party (i.e. TWiG). You will need this for commissions.
BroadcastTransaction(stx: SignedTransaction): FlowLogic<Unit>
A TWiG subflow that sends a transaction and broadcasts it to all observing nodes (i.e. the regulator).
NotEnoughAssetsException(): FlowException()
A TWiG exception that is to be thrown in the Trade flow if there are insufficient assets.

Corda

LedgerTransaction
.inputsOfType(clazz: Class<T>): List<T>
Helper function to get input states of a certain type in the transaction.
.outputsOfType(clazz: Class<T>): List<T>
Helper function to get output states of a certain type in the transaction.

SignedTransaction
.id: SecureHash
The id of the WireTransaction

QueryCriteria
The base class of all criteria for states that you can query from.
.VaultQueryCriteria
Provides query attributes. Click through to read more about the different status available.

ContractState
The base class for on ledger states.
.participants: List<AbstractParty>
The parties for which this state is relevant.

OwnableState
.owner: AbstractParty
The owner of the state.
.withNewOwner: CommandAndState
Copies the underlying data structure, modifying the owner field and leaving everything else intact.

FungibleAsset
.amount: Amount<Issued<T>>
A Corda class that represents a positive quantity of an Asset on the ledger.
.withNewOwnerAndAmount: FungibleAsset<T>
Copies the underlying data structure, modifying the owner and amount field and leaving everything else intact.
.exitKeys: Collection<PublicKey>
The keys required to sign any transaction that will destroy the Asset.

Amount
.quantity: Long
The number of tokens.
.token: T
The type of token. This is usually a singleton.

ServiceHub
The starting point for most operations you need in a flow.
.vaultService
The vault service allows you to observe and soft lock states that involve you or are relevant to your node in some way.
.tryLockFungibleStatesForSpending()
Helper function to get spendable states and soft lock them. Works for all FungibleAsset states.
.queryBy(contractStateType: Class<out T>, criteria: QueryCriteria): Page<T>
Function that allows you to query for states from the Vault (but only the ones you have access to).
.networkMapCache
Provides access to the nodes on the network map (hint: this is how you get the notary)
.myInfo
Provides access to information on your own node.
.signInitialTransaction(builder: TransactionBuilder): SignedTransaction
Helper method to construct an initial partially signed transaction from a TransactionBuilder using keys stored inside the node.

FlowLogic
The base class for all flows.
.sleep(duration: Duration)
Suspends the flow until duration has passed (use this if insufficient states are found).
.waitForLedgerCommit(hash: SecureHash): SignedTransaction
Suspends the flow until a transaction with the given hash is received. This is useful for when you lock states as the state remains locked until the flow terminates.
.runId.uuid: UUID
A wrapper for a java UUID object that identifies the subflow.

Party
.name: CordaX500Name
The CordaX500Name of the Party.

TransactionBuilder
A mutable transaction class that is intended to be passed around nodes to be edited. Then once the states and commands are correct, this class can be the bucket that collects signatures.
.addInputState(stateAndRef: StateAndRef<*>): TransactionBuilder
Adds a single stateAndRef to the transaction.
.addOutputState(state: ContractState, contract: ContractClassName): TransactionBuilder
Adds an output state and respective contract to the transaction.
.addCommand(data: CommandData, keys: List<PublicKey>): TransactionBuilder
Adds a command to the transaction.

Flows
CollectSignaturesFlow
A subflow that allows you to automate the collection of signatures on a transaction.
FinalityFlow
A subflow that verifies the transaction, sends it to the notary for notarisation before storing the result in the vault.
SendStateAndRefFlow
A subflow that allows the sharing of a **StateAndRef** to another user that wishes to verify the state's integrity.
ReceiveStateAndRefFlow
A subflow that responds to the **SendStateAndRefFlow** and allows a user to check the integrity of a state that is being passed to it.
SignTransactionFlow
A subflow called in response to **CollectSignaturesFlow** which signs the transaction.

Exceptions
InsufficientBalanceException
An exception that is thrown when an insufficient amount is found. The amount that is missing is returned.

For more information about certain APIs, see here:
States
Contracts
Contract Constraints
Vault Query
Transactions
Flows
ServiceHub