SIMBA Chain’s Smart Contract Designer (SCD) interface allows users to quickly and easily create logical entities and relationships which are used to generate smart contracts.
OVERVIEW ......................................................... 3
  ASSETS .......................................................... 3
  TRANSACTIONS .................................................. 3
  RELATIONSHIPS .................................................. 4

CONTROLLING WHO CAN EXECUTE METHODS ............. 5

USING THE GRAPHICAL INTERFACE ......................... 5
  ADD ASSETS ...................................................... 5
  ADD TRANSACTIONS ............................................. 6
  ADD RELATIONS (TRANSACTION-TO-ASSET, ASSET-TO-ASSET) ........ 7
  ADD PERMISSION ............................................... 7

SOLIDITY MAPPING .............................................. 8

SOLIDITY TYPES .................................................. 9
OVERVIEW

The purpose of the Smart Contract Designer (SCD) graphical interface is to allow users to quickly and easily create logical entities and relationships which are used to generate smart contracts specifically for the purpose of recording entities, events and state transitions in business processes. Currently Ethereum Solidity contracts are supported.

The graphical interface builds up a graph of relationships using a model we call an “Asset and Transaction” model. This model is then converted to Solidity where each asset and transaction becomes a Solidity Smart Contract method that can be executed on the blockchain. The designer is not a generic graphical code editor. Rather, it provides a logical model related to use cases around supply chain, audit and provenance.

The model provides the means to create a contract structure that echoes your business processes. As well as easing contract creation, this structure provides attributes that SIMBA can use to facilitate easier and more complex search and query of assets and transactions on the blockchain that have been created with the asset and transaction structure.

The designer lets you create contracts purely from the graphical interface. Alternatively you can create a skeleton contract in the graphical interface and then move to Expert Mode where you or your development team work directly with the generated code to flesh out any specific rules you want to apply to the overall business model you have designed in the graphical interface.

ASSETS

Assets are business domain entities. They are the nouns of your business workflow. For example, if you are a car manufacturer, assets may be cars or designs. If you are a cattle farmer, assets may be livestock.

When creating an asset in the SCD, you are creating a type of asset (Car, Cattle). When you use assets on the blockchain, or associate transactions with assets, you use an instance of an asset. The instance identifier (aka primary key) is determined by your business workflow and existing systems. This allows you to reference assets on the blockchain that map back to your operational databases and documents.

So, for example, given an asset type of Car, you may choose that the instance identifier of all cars is their serial number. In this case, you use the serial number when creating or referring to Car instances on the blockchain. If you are working with Cattle, you may want to use the ear tag, or an RFID identifier. As a rule, asset instance identifiers should be globally unique to your system.

TRANSACTIONS

Transactions are business domain processes or states that use or impact assets. They are actions or asset states that result from actions.
When you create a transaction in SCD you associate it with one or more assets. When a transaction is invoked, it takes the instance identifier or associated assets along with any other contextual parameters you define, such as timestamps, state identifiers or associated files.

The figure below shows how business domain objects and actions map to smart contract design. Actors performing actions in the business domain translate to entities that invoke smart contract methods which describe the assets, states and context of the business process.

The graphical designer and the generated Solidity code represent this mapping from business process to executable smart contract code.

At runtime entities (users and systems) invoke smart contract methods that describe the business processes taking place for specific assets in time and space. These invocations result in transactions being written to the blockchain, as shown below.

Entities digitally sign the transactions they execute, meaning the chain contains not just the process event, but the actor that asserted the event.

**RELATIONSHIPS**

The asset and Transaction Model supports two types of relationships between entities:

1. **Asset to Transaction Relationship.** This represents the association between an asset type and a transaction. In Smart Contract terms, this means an asset is referenced within a smart contract method and when executed, results in a blockchain transaction with the asset referenced by its instance identifier. A single transaction can be associated with multiple asset types.

2. **Asset to Asset Relationship.** This represents a dependency of one asset type on another. For example, a Component may have a dependency on a Design asset. In this case, it is not possible for a Component to exist without referring to a Design instance. This relationship allows you to model that type of association.
CONTROLLING WHO CAN EXECUTE METHODS

In the graphical interface, it is possible to add known addresses that are permitted to execute contract methods. This allows you to restrict execution to addresses of your choice. When an address that is not in your whitelist attempts to execute a method on the contract, the transaction will fail. We recommend using this feature if you are deploying a contract to a main net. See more details in the following section. In Expert Mode you can edit the code directly and add any other security features you require.

USING THE GRAPHICAL INTERFACE

ADD ASSETS

An asset can be added to the canvas by clicking the + ADD ASSET button, and filling out with the asset’s name, then hitting enter/return key or anywhere on the canvas. An asset is represented as a rectangle with its name on the canvas.

If an asset selected by single clicking, three options are provided for modification.

Parameters can be added to an asset by hitting ADD PARAMETER when an asset is selected or simply double clicking the asset. A dialog contains the selected asset’s parameters will pop up.

A parameter can be added via this window including the `_bundleHash` parameter, which allows large files get uploaded. The external file system can be enabled/disabled by toggling the button.
**ADD TRANSACTIONS**

A transaction can be added to the canvas by clicking the `➕ ADD TRANSACTION` button, and filling out with the transaction's name, then hitting enter/return key or anywhere on the canvas. A transaction is represented as an oval with its name on the canvas.

If a transaction selected by single clicking, three options are provided for modification.

Parameters can be added to a transaction by hitting the `➕ ADD PARAMETER` when a transaction is selected or simply double clicking the transaction. A dialog contains the selected transaction's parameters will pop up.

A parameter can be added via this window including the `_bundleHash` parameter, which allows large files get uploaded. The external file system can be enabled/disabled by toggling the `Off-Chain Storage` button.
ADD RELATIONS (TRANSACTION-TO-ASSET, ASSET-TO-ASSET)

One can enter relation-adding model by clicking button. Dragging a directed line from a transaction to an asset or an asset to an asset will build a relation between those.

The default parameters of a transaction or an asset are used to denote their relations with others, which can be found at the same place as user-defined parameters. The default parameters cannot be modified directly since they are generated based on the graph structure. As a transaction, it has the names of the assets that it points to, following two underscores `__` as the default parameters. As an asset, it has the names of the assets including itself and the ones it points to, following two underscores `__` as the default parameters.

NOTE: Solidity will struggle to compile methods with more than around 10 parameters. There is no exact figure as it depends on the types.

ADD PERMISSION

The Ethereum accounts/wallets used to invoke method call for data posting to blockchain can be limited to a certain group by adding a list of Ethereum addresses/public keys to the permission list. Notice that this only limits the accounts/wallets that can make the method call transaction not the accounts/wallets that can deploy the application.

To add an address to the permission list, click on the toolbar at the bottom to expand the option dropdown, then click Permission to invoke the configuration dialog.

An address can be added by manually typing or selecting from your local wallet, if any.
SOLIDITY MAPPING

The model that is created in the graphical interface is converted to a smart contract. Currently Ethereum Solidity is supported. When working with the graphical model, you can view the code that is generated. The code that is generated uses the graphical model to create specific functions and parameter names.

If you like the asset and Transaction model, but want to manually create contracts or extend the generated contract in expert model, this is also possible. Manually creating a contract that is compatible with the graphical model is easy, and has benefits beyond the graphical interface as it will enable Simba to perform enhanced queries over your contracts.

The mapping uses a convention over configuration approach. Assets are simply logical named entities such as Car or Component or Design.

To associate a solidity method with an asset type, simply include a string parameter in the method which is named after the asset preceded by two underscores, e.g., __Car, __Component or __Design. This will be understood to be a reference to an asset using the instance identifier. In the example below, a delivered method is associated with the Component asset type.

```solidity
function delivered (string memory __Component, uint timestamp, string memory location)
public {
}
```

Additionally, you can create what we call asset constructor methods. A constructor’s parameters include a slot for the asset’s own instance identifier, as well as identifiers for any other assets it depends on. The constructor is always named after the asset. For example:

```solidity
function Component (string memory __Component, string memory __Design, uint timestamp, string memory _bundleHash)
public {
}
```

This is a constructor for a Component asset and declares that the asset also depends on a Design asset instance. Purely for example purposes, there are two additional parameters - an unsigned integer representing a timestamp as well as a _bundleHash parameter. The _bundleHash parameter is a special parameter that is a flag declaring that the HTTP API to this method will also allow file upload and off chain file storage. When files are stored off chain, SIMBA creates a bundle representing the files uploaded as well as a manifest containing file hashes and other metadata. The manifest itself is associated with a hash which is used to populate this parameter once the files have been stored.

**NOTE:** As a rule, avoid parameter names that start with an underscore as SIMBA uses such parameters for blockchain and transaction related metadata. Avoiding underscores at the start avoids naming clashes with the parameters you define.
SOLIDITY TYPES

The graphical UI supports the following types:

1. bool
2. string
3. address
4. int8 - int32
5. uint8 - uint32
6. Bytes8, bytes16, bytes32 and bytes256

The code view supports any types supported by solidity. In terms of generating HTTP APIs for the contracts designed in expert mode, the primitive types, bool, string, address, int, uint, bytes, as well as arrays of these, and structs are supported. Note that constructor args are not currently supported. Additionally, we only support a single contract to be compiled.
SIMBA Chain enables seamless utilization and integration of blockchain technology to bolster trust, security, and risk mitigation for enterprise and government.

simbachain.com
574-914-4446
info@simbachain.com