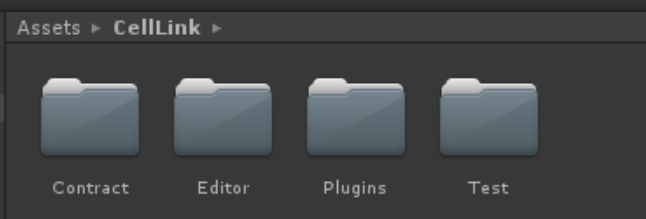
# CellLink SDKfor Unity3D v1.0

1. 简介

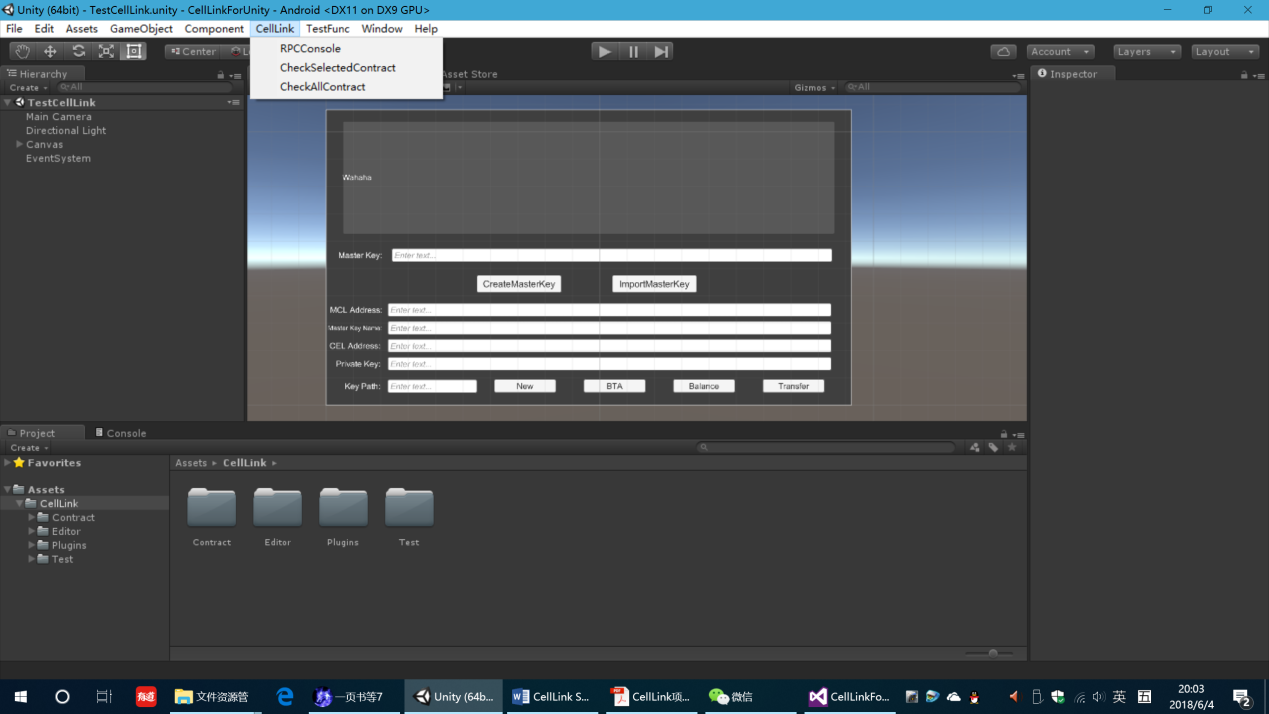
CellLink SDK For Unity3D 为广大Unity3D开发者提供便捷的区块链相关功能。通过SDK，开发可以直接查询CellLink主链及支链数据，转账，发布智能合约，调用智能合约中的函数等，甚至可以直接通过RPC调用各区块链的各功能。在游戏的任何开发阶段，开发者都可以方便快速的集成CellLinkSDK，游戏上链就是如此容易。

1. 下载安装

用户可以直接在Unity3D的资源商店下载CellLink SDK for Unity3D的插件，直接导入工程。目录结如下：

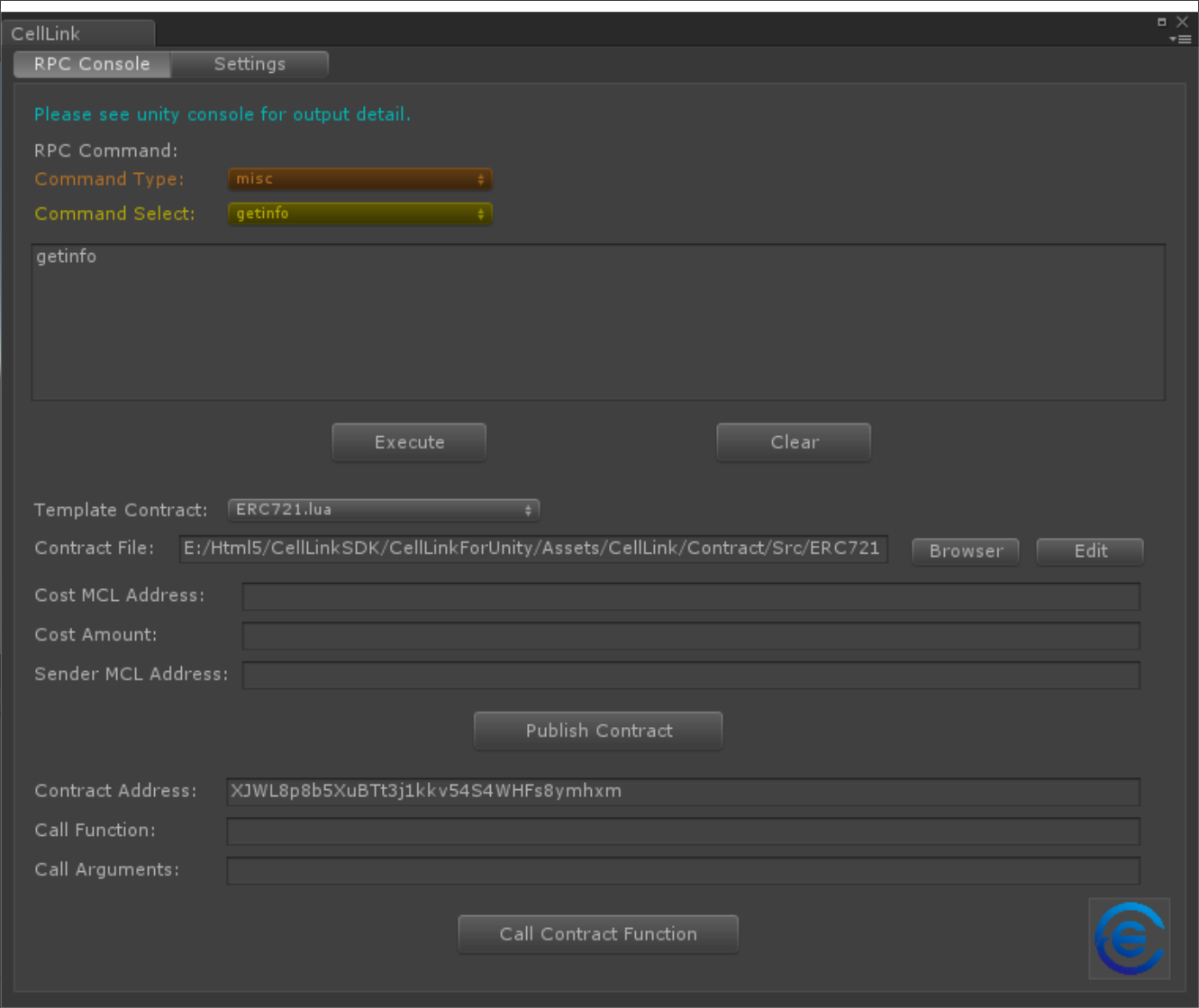


正确导入后，会出现CellLink SDK功能菜单：



1. RPC控制台说明

SDK提供一个CellLink区块链功能调试控制台，用户可以方便的直接在控制台中执行各RPC 命令，发布智能合约，调用智能合约中函数。执行菜单命令：CellLink/RPCConsole，打开控制台：

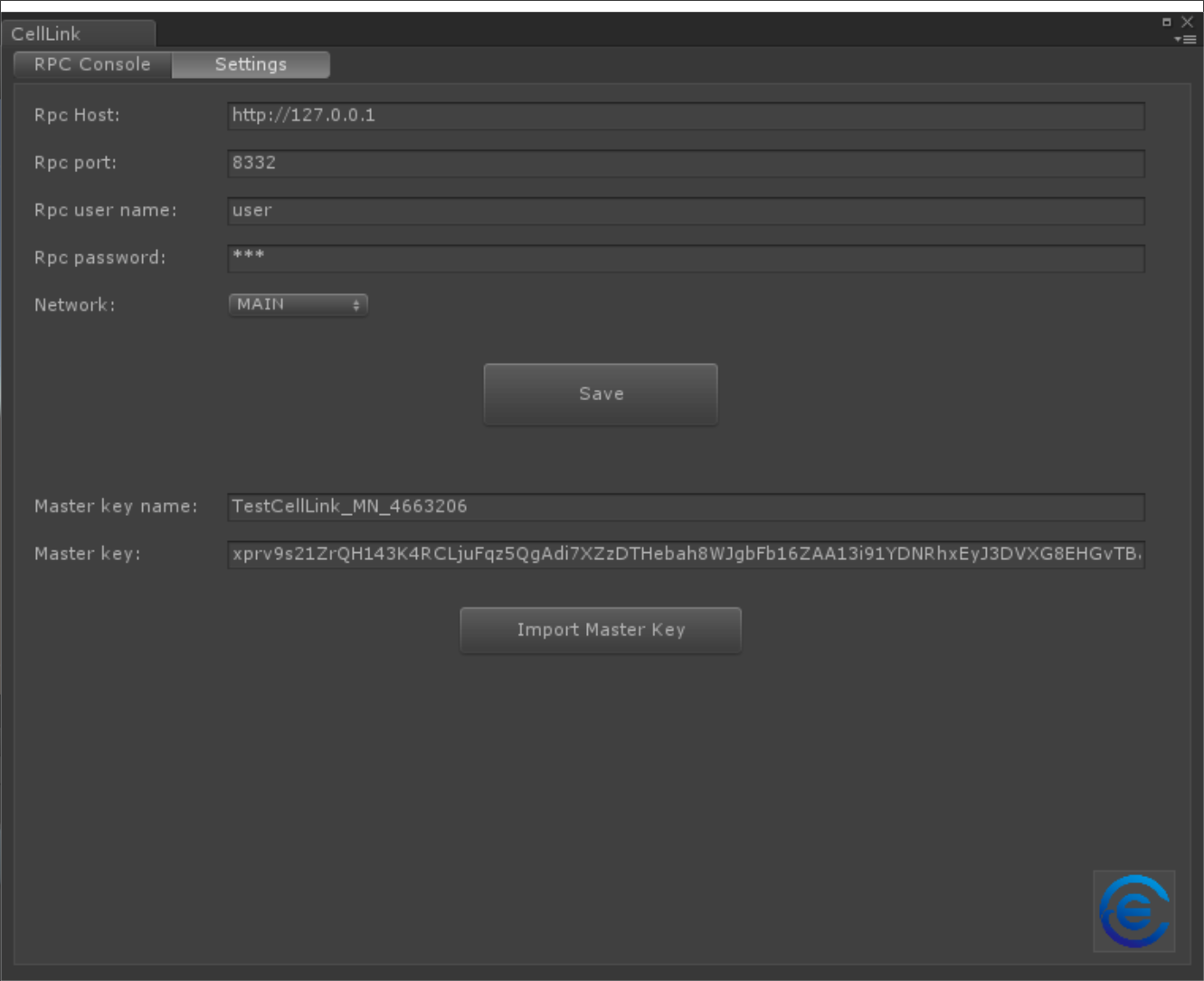


这里可以选择不同类型的命令，点Execute按扭执行。结果将显示在Unity3D的控制台中。

另外，用户可以直接在Template Contract中选预置的智能合约模板（文件位于Assets/CellLink/Contract/Src目录下），也可点Browser选择任意需要上传的智能合约文件(.lua文件)，设置好地址及消费直接将智能合约上传至区块链网络。也可以在上传前点Edit编辑智能合约，Save之后再行上传。菜单中提供了检查智能合约中是否有错误的命令：CellLink/CheckSelectedContract及CellLink/CheckAllContract，要求智能合约文件放到CellLink/Contract/Src目录。

最下面一部分可以测试调用智能合约中的函数。

在Settings分页中，需先设置远程CellLink链的节点地址，账号，密码等信息，及网格类型：



这里的Master key name 及 Master key 在后面第四节SDK集成中有说明。

1. SDK集成
2. **私钥及地址**

对于原生的CellLink地址，每一个都对应一个私钥，很多区块链操作，都需要私钥，如果我们对每个地址都保存私钥，势必会加大管理的难度及降低安全性。因此，SDK采用标准的BIP32 地址协议，只需要一个根私钥(Master Key)，就可以产生无数的地址及对应的私钥。每个地址对应一个PathKey，通过PathKey及根私钥，就可以计算出该地址对应的子私钥。为了便于开发者管理及使用，这里对原生的地址做了一层封装编码，每个原生的地址会生成一个MCL地址。对于每一个根私钥，在使用时我们可以给它指定一个在当前客户端环境中独一的名字（注意：这个名字只是为了SDK及用户方便管理，不会上传至网络，所以只需要当前客户端中唯一即可。）

MCL地址和原生地址的关系及使用如下示意图：

Master Key Name

Server

v

MCL Address

Client

CellLink SDK

Master Key

CellLink

Network

Raw Address

PathKey

可以看到，MCL Address由Raw Address, Path Key, Master Key Name编码组成，SDK提供相关接口可以编码及解码MCL Address。

1. **代码集成**
2. 初始化SDK及RPC。

创建IxCellLinkBridge对象后,须调用

Initialize(stringstrRPCAddress, intiRPCPort, stringstrCredUserName = "", stringstrCredPwd = "", NETWORK\_TYPEeNetType = NETWORK\_TYPE.TEST\_NET)

初始化RPC远程调用系统。

1. 创建或导入Master Key，生成新的MCL 地址。

每个客户端，至少应创建或导入一个Master Key，类似于钱包的私钥，该Master Key可以创建数量庞大的地址，只需要一个Master Key即可管理全部。相关接口：

CreateMasterExtKey(stringstrName)

ImportMasterExtKey(stringstrName, stringstrMasterWifExtKey, stringstrLastKeyPath = null)

这里strName为该Master Key 的名字，客户端指定，只需要保证当前客户端没有重复即可。创建或导入Master Key 之后，则可以用：

GenerateNewMCLAddress(stringstrMasterKeyName)

生成一个新的MCL地址。前文所说，一个MCL地址包括了Raw Address, Path Key, Master Key Name。通过：

EncodeMCLAddress(stringstrName, stringstrPath, stringstrCelAddress)

DecodeMCLAddress(stringstrMCLAddress, outstringstrName, outstringstrPath, outstringstrCelAddress)

可以编码/解码。注意，对于CellLink网络来说，其只有Raw Address，这里在客户端封装成 MCL Address，可以实现一个Master Key管理无数的子地址，因为MCL Address 中包括了Raw Address 和 Path Key，而通过Master Key和Path Key，可以计算出该Raw Address 对应的私钥，因为区块链很多操作必须要该地址对应的私钥。这样客户端只需要保证Master Key 的安全性，就可以保证所有MCL Address的安全性。

1. 查询余额，转账

相关接口：

GetBalance(stringstrMCLAddress)

GetBalanceAsync(stringstrMCLAddress, ON\_GET\_BALANCEfnGetBalance)

TransferMCLToMCLAddress(stringstrFromMCLAddress, stringstrToMCLAddress, floatfMoney, outstringstrTxid, floatfFee = -1.0f, stringstrChangeMCLAddress = null)

TransferMCLToMCLAddressAsync(stringstrFromMCLAddress, stringstrToMCLAddress, floatfMoney, ON\_TRANSFER\_DONEfnTransferDone, floatfFee = -1.0f, stringstrChangeMCLAddress = null)

带Async后缀的函数为异步版。

1. 智能合约发布及调用相关函数

相关接口：

PublishContract(outstringstrContractAddress, stringstrContractContent, stringstrCostFromMCLAddress, floatfCostAmount, stringstrSendMCLAddress, stringstrChargeMCLAdrres = null)

PublishContractAsync(ON\_PUBLISH\_CONTRACT\_DONEfnDone, stringstrContractContent, stringstrCostFromMCLAddress, floatfCostAmount, stringstrSendMCLAddress, stringstrChargeMCLAdrres = null, objectkArg = null)

CallContractFunction(stringstrContractAddress, stringstrCostMCLAddress, floatfCostAmount, stringstrSenderMCLAddress, stringstrFuncName, object[] arrArgs, stringstrChargeMCLAddress = null)

CallContractFunctionAsync(ON\_CALL\_CONTRACT\_FUNCTIONfnCallDone, stringstrContractAddress, stringstrCostMCLAddress, floatfCostAmount, stringstrSenderMCLAddress, stringstrFuncName, object[] arrArgs, stringstrChargeMCLAddress = null)

带Async后缀的为异步版函数。