

# サンプルコードについて

# サンプルコードの種類

サンプルコードは、DNCWARE Blockchain+™のアプリケーションを作成する際に参考になるプログラム(コード)を例示するものです。サンプルコードの種類として、以下の3種類があります。

## basicサンプル（基礎編）

基礎的な使い方を例示して説明するものです。

ウォレット作成、コントラクトのデプロイ、コントラクトAPI、グループ管理、アクセス権限、キーバリューコントラクト、UnifiedName、サブコントラクト、など

## advancedサンプル（応用編）

より高度な使い方を例示して説明するものです。

TX要求のオフライン作成、マルチシグ、コントラクトのライブラリ化、プロキシ環境対応、など

## practicalサンプル（実用編）

実用的なアプリケーションを想定したサンプルコードです。

「DNCWARE」、「DNCWARE Blockchain+」は、東芝デジタルソリューションズ株式会社の日本またはその他の国における登録商標または商標です。

# サンプルコードの見かた

HTML形式のファイルで提供され、ブラウザで開いて見ることができます。

右のようなノートブックとなっており、説明つきでサンプルコードとその実行結果が示されています。

実行は、上から下へと進みます。

コードセルの中にコードそのものが示されています。コードセルの前にその説明、後ろにその実行結果が示されています。

コードセル[1]の説明 変数xに数値1234を代入します。  
という説明は、下にあるコードセルの内容を説明しています。  
コードセルの実行結果は、コードセルの下側に続いて表示されます。

```
In [1]: //ここがコードセルです。  
var x;  
x = 1234;
```

コードセル[1]

Out[1]: 1234      コードセル[1]の実行結果

コードセル[2]の説明 コンソールへの出力は、コードセルの下側に表示されます。

```
In [2]: console.log('結果は', x+1);
```

コードセル[2]

結果は 1235      コードセル[2]の実行結果

コードセル[3]の説明 例外がスローされた場合、コードセルの下側に赤背景で表示されます。

```
In [3]: var y = null;  
y.a = 1; // ここでエラーが発生する。
```

コードセル[3]

TypeError: Cannot set properties of null (setting 'a')

コードセル[3]の実行結果

# サンプルコードの構成

構成はおおよそ下記の通りです。

- ・冒頭で概要を説明します。
- ・実行のための準備をします。
- ・一番目のトピックを説明します。
- ・二番目のトピックを説明します。

あとはこの繰り返しです。

準備の部分は、実行のために必要なセットアップです。概略を把握したいだけのときは、ここは読み飛ばして構いません。

その下に続くトピックがもともと説明したい内容となります。複数のトピックがある場合には、見出しを大きくして、トピックの境界が分かるようにしています。

## 凡例 2

このノートブックでは、ブロックチェーンにアクセスする 2 つの例を示します。

1. ダッシュボードの参照
2. スマートコントラクトの呼び出し

このノートブックの概要

## 準備

実行の準備を行います。事前に設定されたオブジェクトを読み込み、ブロックチェーンに接続します。

```
In [1]: var { api } = await import('../lib/load-blockchain-api.mjs');
var { adminWalletPassword, peerURL, chainID, domain } = await import('../lib/load-conf.mjs');
var { adminWalletJSON } = await import('../lib/load-wallet.mjs');

var adminWallet = await api.unlockWalletFile(await api.parseWalletFile(adminWalletJSON));
var rpc = new api.RPC(chainID);
rpc.connect(peerURL);
```

```
Out[1]: SocketHTTP {
  url: 'https://trial1.dncware-blockchain.biz',
  options: undefined
}
```

実行のための準備

## ダッシュボードの参照

ブロックチェーンに接続し、ダッシュボードの情報を読み出します。

```
In [2]: var resp = await rpc.call(adminWallet, 'c1query', { type: 'dashboard' });
console.log('c1query dashboard:', JSON.stringify(resp.value));

c1query dashboard: {"N":3,"F":1,"B":0,"num_transactions":535471,"num_blocks":42785,"num_
```

トピック 1

## スマートコントラクトの呼び出し

事前に設定されているスマートコントラクトを呼び出します。

```
In [3]: var resp = await rpc.call(adminWallet, 'test1@${domain}', { a: 1000 });
console.log(resp);

{
  txno: 535472,
  txid: 'xy7u9Tn7PEMRbBoqho2gHSLNQQd5cf8xqckm97Nnr4F2G8',
  status: 'denied',
  value: 'contract not found'
}
```

トピック 2

# サンプルコードの中で利用しているユーティリティ関数について

サンプルコードが煩雑になるのを避けるため、本題に関係のない処理はユーティリティ関数を利用して記述する場合があります。ここでは、代表的なユーティリティ関数を説明します。

(なお、ユーティリティ関数はサンプルコード専用であり、一般に利用することを想定したものではありません)

```
var { api } = await import('../lib/load-blockchain-api.mjs');
```

クライアントAPIのライブラリを読み込みます。

```
var { adminWalletPassword, peerURL, chainID, domain } = await import('../lib/load-config.mjs');
```

設定ファイル(samples/etc/config.json)から設定値を読み込みます。

```
var { loadWallet, adminWalletJSON } = await import('../lib/load-wallet.mjs');
```

設定フォルダ(samples/etc)に置かれているウォレットのJSONを読み込みます。

```
var { adminWallet, rpc, deploySmartContract } = await import('../lib/notebook-util.mjs');
```

基礎編1の準備と同様の処理を行います。

```
var cid = deploySmartContract([argtypes,] func, options);
```

スマートコントラクトをブロックチェーンにデプロイします。

**TOSHIBA**