Lua Device Driver

プログラミングガイド（V1.0）

目次

1 概要 2
2 デバイスドライバーの概要 2
2.1 デバイスドライバーの種類 2
2.2 デバイスドライバーの構成要素 3
3 デバイスドライバーのインストール 3
3.1 公式ドライバー 3
3.2 カスタムドライバー 4
4 Lua Device Driver Builder 5
5 デバイスドライバーの開発 7
5.1 ドライバー基本情報 7
5.2 デバイス固有パラメーター 8
5.3 ドライバーの Lua スクリプト 9
5.4 ユーザーインターフェイスビルダー 18
6 ドライバーのテストとデバッグ 20
7 ドライバーのインポートとエクスポート 22
8 プロジェクトファイルとドライバー 23

1　概要

Lua Device Driver Builder ソフトウェアは Xilica Designer に統合されており、機器メーカー、システムインテグレーター、ディストリビューター、エンドユーザーが、Ethernet 制御に対応したあらゆるサードパーティー製デバイス向けに、Xilica Designer 内で制御インターフェイスを構築するためのツールを提供します。このツールには、通信プロトコルの処理と、コントロールパネルの UI 設計およびパッケージ化という、2 つの主要機能があります。メーカーが設計したドライバーは、Xilica Designer ソフトウェアにインストール可能なプラグインとして提供され、対象デバイスの制御可能なパラメーターにドラッグ＆ドロップでアクセスできます。十分にテストされたサードパーティー開発ドライバーのインストールパッケージは、www.xilica.com で一般公開される場合があります。

インストール済みのドライバーは、Xilica Designer ソフトウェアの Component Library に表示されます。デバイスをダブルクリックするとドライバーがダウンロードおよびインストールされ、その後、プロジェクトへドラッグ＆ドロップして使用できるようになります。プロジェクトウィンドウ内のデバイスアイコンをダブルクリックすると、そのモジュールの制御インターフェイスが表示されます。インターフェイス内の要素を選択し、XTouch パネル、XWP デバイス、または Project Controller ウィンドウへドラッグして配置できます。

公式ドライバーに加え、ユーザーは、ゼロから、または既存のドライバーをベースにして、独自のローカルカスタムドライバーを作成できます。こうしたカスタムドライバーは、エクスポートして共有することもできます。

2　デバイスドライバーの概要

各デバイスドライバーは、特定のメーカーおよび機種のハードウェア向けに作成してください。同一ドライバーの複数のインスタンスを 1 つの設計内で利用でき、各インスタンスに固有の IP アドレスを割り当てるだけで区別できます。

注：あるモジュールのスクリプト内のグローバル変数は、そのモジュールのインスタンス専用であり、同じモジュールの他のインスタンスからはアクセスできません。

同一モジュールの各インスタンスは、デバイス固有のパラメーターで個別にカスタマイズできます。たとえば、複数のインスタンスで共通するパラメーターに、それぞれ固有のデバイス ID を割り当てることができます。

2.1 デバイスドライバーの種類

Official Drivers（公式ドライバー）

この種類のドライバーは、サードパーティーメーカーまたは Xilica が開発し、公式にサポートされ、すべての Xilica Designer ユーザーに提供されます。新しいドライバーが一般公開されると、Xilica Designer がアプリケーションの起動時に新しいドライバーの有無を検出し、ドライバー情報を自動的にダウンロードします。新しいドライバーは、Xilica Designer 内の Component Library に一覧表示されます。

注：新しいドライバーが検出されると、Xilica Designer の再起動を促すメッセージが表示されます。再起動後に、新しいドライバーを使用できるようになります。

Custom Drivers(カスタムドライバー)

公式ドライバーに加え、ユーザーは、新規にゼロから作成するか、既存のドライバーを出発点として、独自のカスタムドライバーを定義できます。

カスタムドライバーを開発した後は、エクスポートして、他の Xilica Designer ユーザーが再インポートできます。

2.2 デバイスドライバーの構成要素

各デバイスドライバーは、次の 4 つの構成要素で構成されます。

Driver Basic Information(ドライバー基本情報)

名前や説明など、デバイスドライバーの一般情報です。ユーザーは、デバイスドライバーが持つ I/O ポイントを定義できます。これらの I/O ポイントは主にドキュメント化の目的で使用されます(プロジェクトビューへドラッグすると、デバイスアイコンに対応する I/O ポイントが表示され、プロジェクトの配線を文書化できます)。Dante の I/O ポイントについては、プロジェクトビューで Dante フローを設定でき、ハードウェアがオンラインになると、対応する Dante ルーティングが自動的に設定されます。

Device Specific Parameters(デバイス固有パラメーター)

1 つのドライバー定義は複数のインスタンス(同一プロジェクト内の複数のデバイス)に適用できるため、各デバイスには、それぞれ異なる情報が関連付けられる場合があります。代表的な例は、ログイン ID とパスワードです。同一ドライバーの各インスタンスが持つ、固有となり得るパラメーターを適切に扱えるよう、デバイス固有パラメーターの定義に対応しています。

Script Body(スクリプト本体)

Script Body は、実際の Lua スクリプトのプログラミングコードで構成されます。ドライバーの起動時にさまざまなコールバック関数が呼び出されるよう、各コールバック関数をスクリプトとして記述する必要があります。

UI Control Panels(UI コントロールパネル)

ドラッグ&ドロップ方式の GUI ビルダー(XTouch パネルのインターフェイス設計に類似)により、ドライバーのユーザーインターフェイスを開発できます。複数のインターフェイスページに、さまざまな UI オブジェクトを配置できます。各 UI オブジェクトには名前が付けられ、Lua スクリプトからその名前でアクセスできます。

3 デバイスドライバーのインストール

3.1 公式ドライバー

新しい公式ドライバーが Xilica Designer ソフトウェアに検出されると、まず、名前や説明などドライバーの基本情報が読み込まれます。この段階では、ドライバーは Component Library に一覧表示されますが、未インストールであることを示すためグレー表示になります。ドライバーをインストールするには、ダブルクリックします。コンピューターがインターネットに接続されていれば、自動的にダウンロードとインストールが行われま

す。ダウンロードとインストールが完了すると、ドライバーをプロジェクトウィンドウへ配置できるようになります。

次の例では、「Generic PELCO-D PTZ Camera (Lua)」をダブルクリックすると、ドライバーがダウンロードおよびインストールされます。

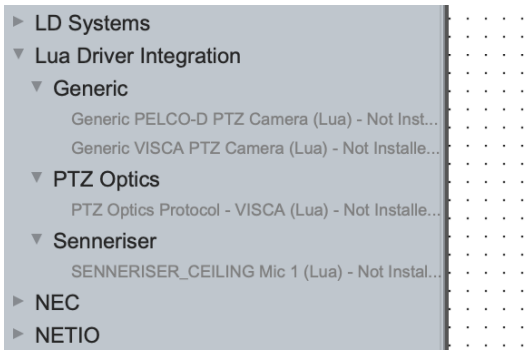


図:インストール前(“Not Installed” と表示)

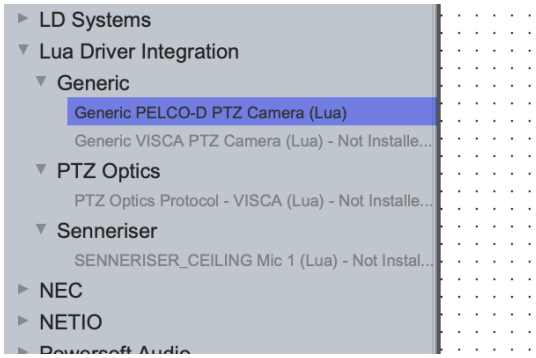
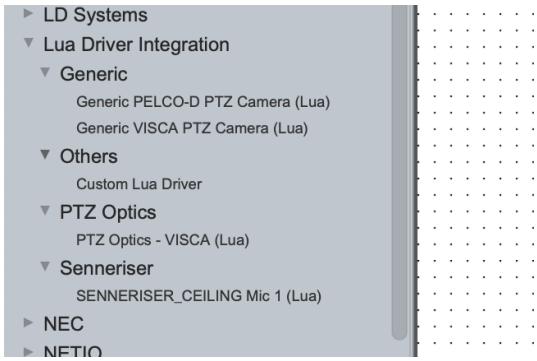


図:インストール後

3.2 カスタムドライバー

ユーザー定義のカスタムドライバー(またはインポートしたドライバー)は、ローカルで開発されたものであるため、再インストールは不要です。これらのドライバーは Component Library に自動的に一覧表示されます。

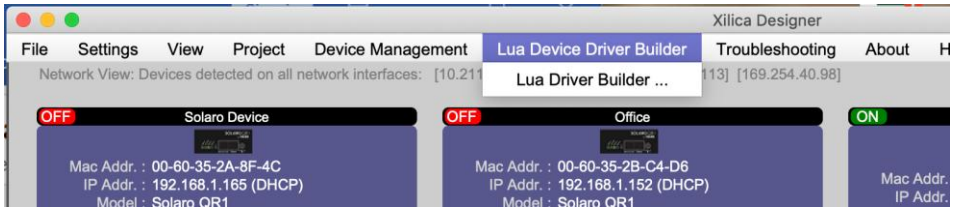
次の例では、「Custom Lua Driver」デバイスが自動的にインストールされ、利用可能になっています。



4 Lua Device Driver Builder

Lua Device Driver Builder は、プログラマーが新しいドライバーを作成したり、既存のドライバーを変更したりするために、Xilica Designer ソフトウェアに統合されています。Driver Builder を起動するには、上部メニューバーの「Lua Device Driver Builder」→「Lua Driver Builder ...」を選択します。Lua Device Driver List ダイアログが表示されます。

注：このオプションは、すべてのプロジェクトを閉じているときのみ使用できます。Lua Device Driver Builder を使用する前に、開いているプロジェクトを保存して終了する必要があります。



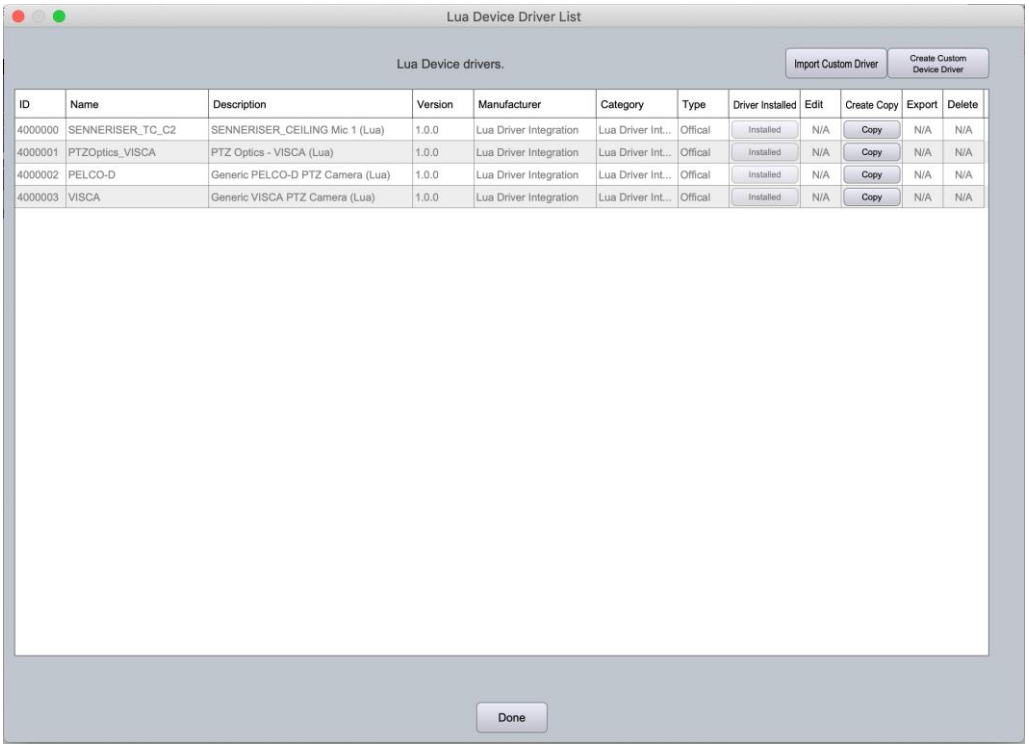
このダイアログには、すべてのドライバー（公式・カスタム）が表示されます。公式ドライバーが未インストールの場合は、「Install」ボタンをクリックしてダウンロードとインストールを行えます。ここでのインストールは、Component Library でダブルクリックしてインストールするのと同じです。

公式ドライバーでは、次の操作を選択できます。

・ドライバーのインストール（未インストールの場合）

・ドライバーのローカルコピーの作成。作成後は、必要に応じてドライバーを変更できます。

ローカルコピーを作成すると、そのコピーは元のドライバーから独立します。元の公式ドライバーは変更されません。

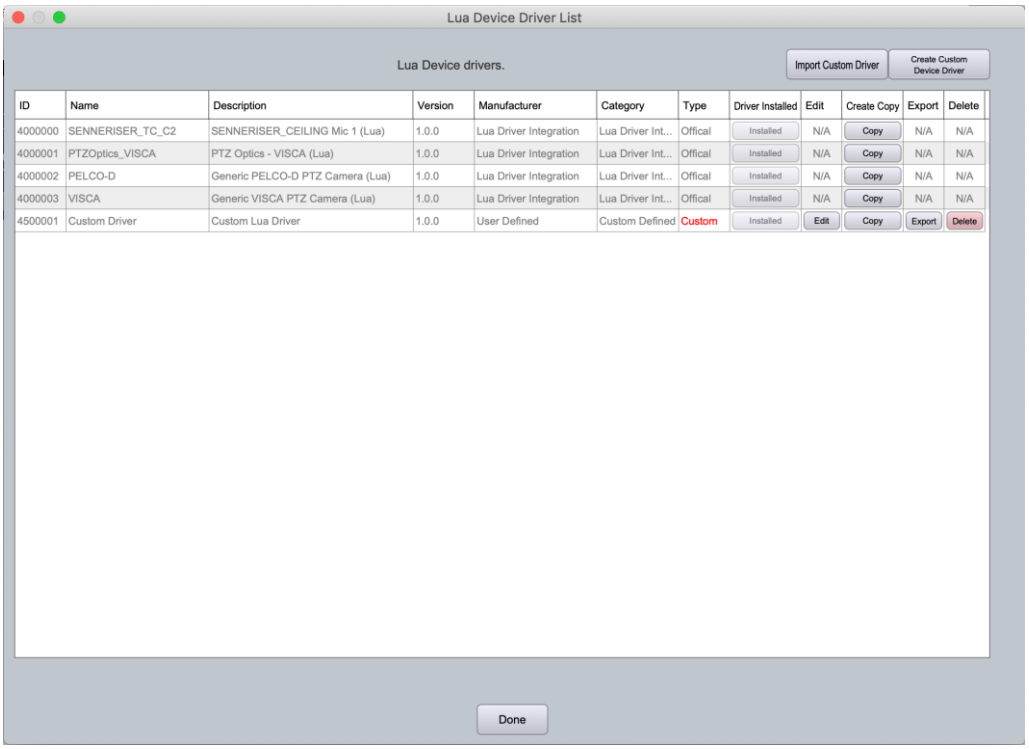


カスタムのローカルドライバーをゼロから作成するには、「Create Custom Device Driver」ボタンをクリックします。既定のコールバック関数を備えた新しいドライバーが作成されます。

注:Lua でのプログラミングが初めての場合は、公式ドライバーのコピーを作成して見本とし、プログラミング環境と構文に慣れることをお勧めします。

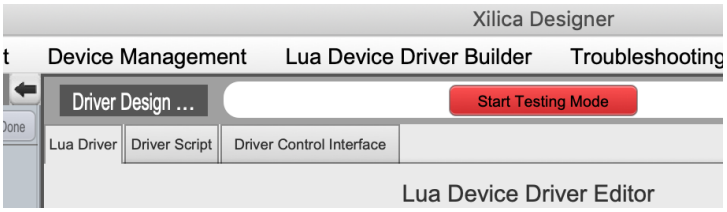
カスタムドライバーを作成すると、ダイアログ内に「Custom」として一覧表示され、次の操作を利用できます。

- ・Edit:ドライバーを編集・変更する
- ・Copy:ドライバーのコピーを作成する
- ・Export:他のユーザーが自分の Xilica Designer ソフトウェアにインポートできるよう、ドライバーをエクスポートする
- ・Delete:ドライバーを削除する



ドライバーの作成または編集を行うと、ビルダーエディターが表示されます。エディターは 3 つのタブで構成されます。

- ・「Lua Driver」タブ:ドライバーの基本情報を編集する
- ・「Driver Script」タブ:ドライバーの基になる Lua スクリプトを編集する
- ・「Driver Control Interface」タブ:ドライバーのユーザーインターフェイスを編集する



5　デバイスドライバーの開発

ここでは、デバイスドライバーの 4 つの構成要素ごとに、ドライバー開発の手順を説明します。

5.1 ドライバー基本情報

Driver ID は Xilica Designer が生成する固有の ID で、ユーザーは変更できません。ドライバーをエクスポートし、別の Xilica Designer にインポートすると、ID は再生成されます。

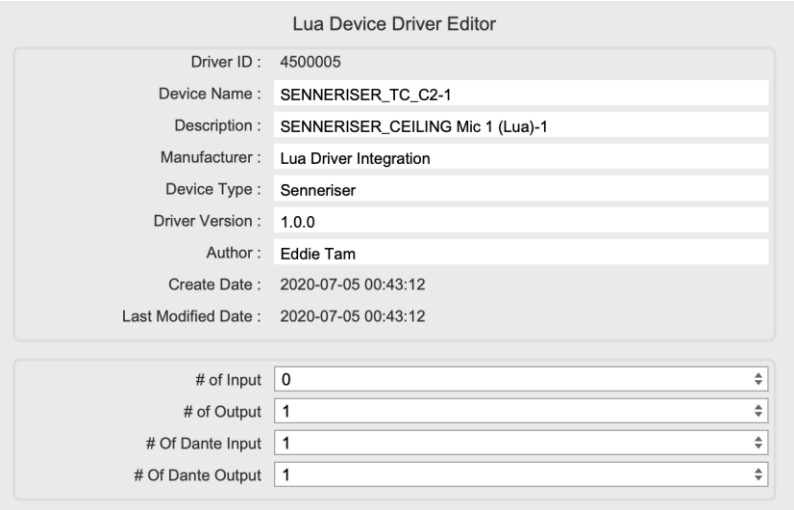
Device Name と Device Description は、Xilica Designer が検出したドライバーの中で重複がないか確認されます。名前が競合した場合は、使用可能な次の名前が提案されます。

ドライバー基本情報の項目の多くはドキュメント化の目的で使用されますが、Manufacturer と Device Type は、Xilica Designer の Component Library でドライバーを整理するために使用されます。Component Library ではデバイスがメーカー別、次にデバイスタイプ別に分類されるため、Manufacturer と Device Type の情報によって、ドライバーが Library 内のどこに配置されるかが決まります。

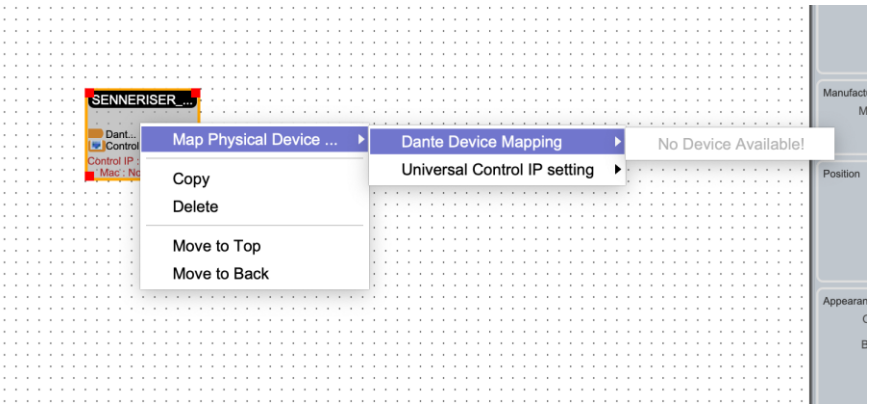
この画面の 2 番目のセクションでは、ドライバーのアナログ入出力数と Dante 入出力ポイント数を選択できます。ドライバーをプロジェクトビューへドラッグすると、定義した I/O ポイント数がアイコンに表示されます。アナログ I/O はドキュメント化の目的のみで使用されます。Dante I/O を定義すると、そのドライバーは Dante 対応になります。

注：ユーザー定義のカスタムドライバーでは、次の項目によってドライバーの識別情報と一意性が決まります。

- Driver Description（ドライバーの説明）
- Driver Manufacturer（ドライバーのメーカー）
- Author（作成者）
- Creation Date（作成日）



Dante 対応ドライバーをプロジェクトに配置する場合は、ネットワーク上で利用可能な対応する Dante デバイスにマッピングする必要があります（Xilica Designer の他の Dante 対応デバイスと同様）。また、次のように、デバイスの制御用 IP アドレスもマッピングする必要があります。

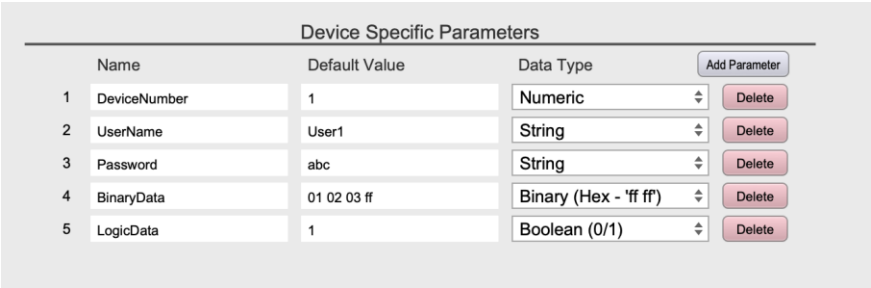


5.2 デバイス固有パラメーター

各デバイスについて、初期化時にドライバーへ渡すパラメーターを定義できます。次の 4 種類のデータ型に対応しています。

- ・Numeric
- ・Boolean
- ・String
- ・Binary

これらのパラメーターには既定値を設定できます。ただし、ドライバーを物理デバイスへマッピングする際に、これらの値を上書きできます。



ドライバーにこれらのパラメーターを定義した後、物理デバイスへのマッピング（すなわち、制御用の IP アドレスへのマッピング）を行うと、マッピングダイアログが表示されます。マッピングダイアログにはパラメーター一覧が表示され、このデバイス固有の値を必要に応じて編集できます。



適切なパラメーター値を入力し、プロジェクトをハードウェアにロードする(オンラインにする)と、パラメーター値が Lua の初期化スクリプト(LuaDriverInitialization)に渡されます。初期化関数については、後述の「ドライバーの Lua スクリプト」の LuaDriverInitialization の項で詳しく説明します。パラメーターはテーブルとして初期化スクリプトに渡され、各要素には名前でアクセスできます。

Lua スクリプトでは、テーブル「deviceParamTable」からパラメーター値を取得できます。次のサンプルコードは、テーブルからさまざまな型の値にアクセスする方法を示しています。

Lua Script Editor

Lua Script Body :

```
-- LuaDriverInitialization - Lua Initialization function that will be called when
--         we put the driver ONLINE
-------------------------------------------------------------------------------
function LuaDriverInitialization (id, ip, deviceParamTable)

G_driverID = id
G_deviceIP = ip
-- Setup additional device parameter into global variables

-- Example as below. You can access the table by the device parameter name defined in driver
G_deviceNumber = deviceParamTable['DeviceNumber']
G_username = deviceParamTable['UserName']
G_password = deviceParamTable['Password']
G_binaryData = deviceParamTable['BinaryData']
G_logicData = deviceParamTable['LogicData']

XConsoleLog(G_driverID, 'Starting Lua Script initialization for processor ID : ' ..
        string.format('%d', G_driverID) .. ' Device IP : ' .. G_deviceIP .. '\n')

XConsoleLog(G_driverID, 'Parameter DeviceNumber : ' .. string.format('%d', G_deviceNumber) .. '\n')
XConsoleLog(G_driverID, 'Parameter UserName : ' .. G_username .. '\n')
XConsoleLog(G_driverID, 'Parameter Password : ' .. G_password .. '\n')
XConsoleLog(G_driverID, 'Parameter BinaryData : ' .. string.format('%02x %02x %02x %02x',
        string.byte(G_binaryData, 1), string.byte(G_binaryData, 2),
        string.byte(G_binaryData, 3), string.byte(G_binaryData, 4)) .. '\n')
if (G_logicData) then
    XConsoleLog(G_driverID, 'Parameter LogicData TRUE\n')
else
    XConsoleLog(G_driverID, 'Parameter LogicData FALSE\n')
end

-- Start Network Server Initialization here

-- Start timer initialization here

return
end

```

この初期化スクリプトの実行結果は次のとおりです。

Lua Script Editor

Console Log : Clear

```
-------------------- Start of Lua Driver ---------------------
 Lua Processor ID : 1
Starting Lua Script initialization for processor ID : 1 Device IP : 192.168.1.200
Parameter DeviceNumber : 1
Parameter UserName : User1
Parameter Password : abc
Parameter BinaryData : 01 02 03 ff
Parameter LogicData TRUE
------ Lua Driver Initialization Complete Successfully -------

```

5.3 ドライバーの Lua スクリプト

ドライバーの Lua スクリプトでは、各ドライバーが、ドライバーのさまざまなイベントで実行される特定のコールバック関数を実装する必要があります。これらのイベントは、次のカテゴリーに分類できます。

Initialization(初期化)

```
LuaDriverInitialization (driverId, ip, deviceParamTable)
```

Network events(ネットワークイベント)

```
LuaTCPServerConnectionCreatedCallback (serverId, connectionId)
LuaTCPServerConnectionClosedCallback (serverId, connectionId)
LuaTCPServerMessageReceivedCallback (serverId, connectionId, receivedData)
LuaTCPClientMessageReceivedCallback (clientId, receivedData)
LuaUDPServerMessageReceivedCallback (serverId, receivedData)
```

User Interface events(ユーザーインターフェイスイベント)

```
LuaUIUpdatedCallback (objectName, strData)
```

Timer events(タイマーイベント)

```
LuaTimerCallback ()
```

次の関数は、ネットワークサーバー、UI の処理、およびタイマーの処理を管理するために使用します。
注:これらすべての呼び出しにおいて、第 1 引数は「driverId」です。この ID は、どのインスタンスに呼び出しをルーティングするかを示す固有の識別子です。この driverId は、LuaDriverInitialization 関数に渡されるパラメーターです。以降のすべての関数呼び出しでは、この ID を指定して、適切なインスタンスを対象にする必要があります。
注:driverId はグローバル変数として定義し、すべての呼び出しがこのグローバル変数を参照することをお勧めします。

Network service(ネットワークサービス)

serverId	XTCPServerSetup (driverId, serverPort, idleTimeout)
bool	XTCPServerClose (driverId, serverId)
bool	XTCPServerReplyMessage (driverId, serverId, clientId, sendData)
clientId	XTCPClientSetup (driverId, remoteIP, remotePort, idleTimeout)
bool	XTCPClientClose (driverId, clientId)
bool	XTCPClientIsConnected (driverId, clientId)
bool	XTCPClientSendMessage (driverId, clientId, sendData)
serverId	XUDPServerSetup (driverId, serverPort)
bool	XUDPServerClose (driverId, serverId)
bool	XUDPClientSendMessage (driverId, remoteIP, remotePort, localPort, sendData)

User Interface service(ユーザーインターフェイスサービス)

`bool`	XUIUpdateRequest (driverId, objectName, UIdata)

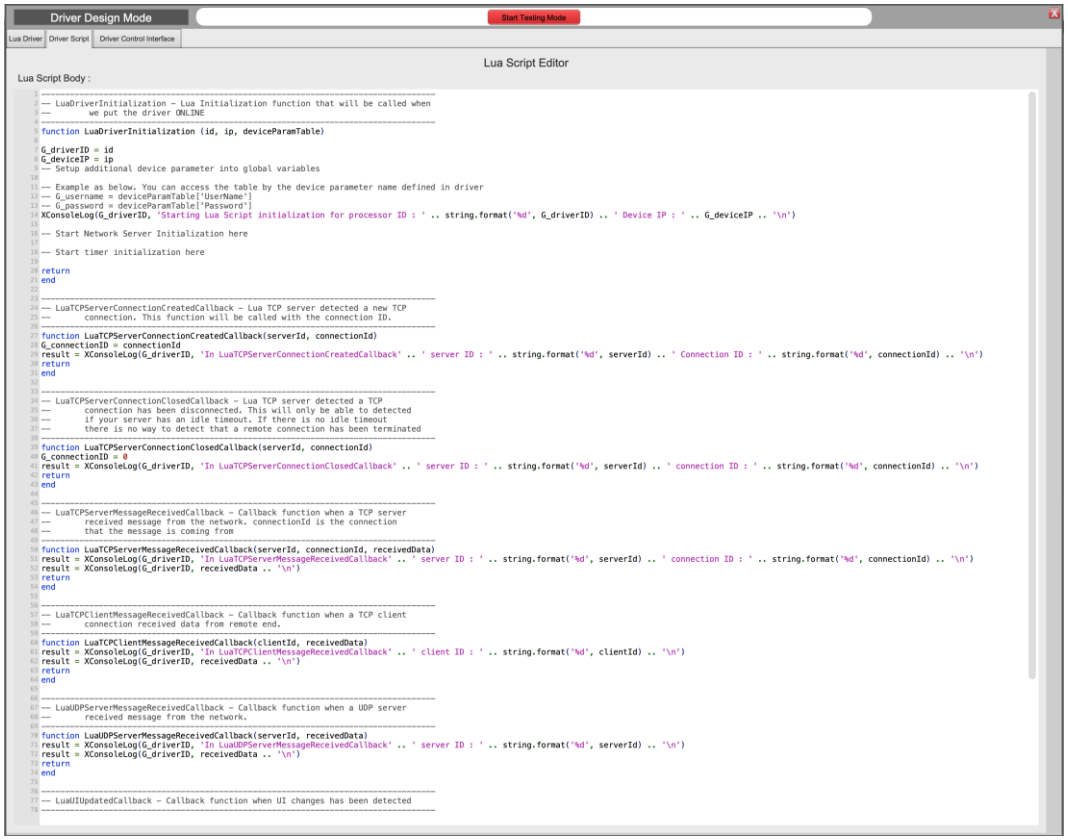
Timer service(タイマーサービス)

`void`	XStartTimer (driverId, timeoutInterval)
`void`	XStopTimer (driverId)

Console Log service(コンソールログサービス)

`void`	XConsoleLog (driverId, message)

「Driver Script」タブには、すべてのスクリプト関数呼び出し用のエディターがあります。ドライバーをゼロから新規作成すると、出発点として、関数呼び出しのサンプルシグネチャが提供されます。



新しいドライバーごとに、ドライバー開発者は、次のイベントに対応するスクリプトを実装する必要があります。

LuaDriverInitialization

```
LuaDriverInitialization (driverId, ip, deviceParamTable)
```

driverId	ドライバー環境を識別する、グローバルに一意なドライバーID
ip	このドライバーインスタンスの IP アドレス
deviceParamTable	ユーザーが入力したデバイス固有パラメーターをすべて含むテーブル

このスクリプトは、ドライバーの起動時に呼び出され、ドライバーセッションごとに 1 回だけ呼び出されます。この初期化スクリプトでは、次の処理を行うことをお勧めします。

- すべてのグローバル変数の設定。特に固有のドライバーID は、他のすべてのスクリプト関数呼び出しで参照する必要があるため、必ず設定します。
- deviceParamTable からのデバイス固有パラメーターの取得。値は、配列要素へのアクセスで取得できます(例:デバイスのユーザー名を取得するには deviceParamTable['UserName']を参照します。UserName などの変数は、ドライバーの作成時に定義できます)。
- 通信セッションの設定。すなわち、TCP サーバー、TCP クライアント、UDP サーバーの設定。
- デバイスから定期的にデータを送信またはポーリングする必要がある場合の、タイマーの設定。

LuaTCPServerConnectionCreatedCallback

```
LuaTCPServerConnectionCreatedCallback (serverId, connectionId)
```

serverId	着信 TCP 接続があったサーバーを示す
connectionId	特定のサーバー内で一意となる ID

このスクリプトは、事前に設定したサーバーが着信接続を受けたときに呼び出されます。TCP セッションが確立されると、この接続を識別するために固有の接続 ID が使用されます。

LuaTCPServerConnectionClosedCallback

```
LuaTCPServerConnectionClosedCallback (serverId, connectionId)
```

serverId	TCP 接続が切断されたサーバーを示す
connectionId	閉じられた特定の接続を示す

このスクリプトは、TCP 接続が切断されたときに呼び出されます。

LuaTCPServerMessageReceivedCallback

```
LuaTCPServerMessageReceivedCallback (serverId, connectionId, receivedData)
```

serverId	着信メッセージがあったサーバーを示す

connectionId	着信メッセージがあった接続を示す
receivedData	Lua の文字列データとしてのメッセージ

このスクリプトは、事前に設定したサーバーおよび特定の接続セッションがデータを受信したときに呼び出されます。着信した文字列データには、Lua においてバイナリーデータを格納できます。

LuaTCPClientMessageReceivedCallback

```
LuaTCPClientMessageReceivedCallback (clientId, receivedData)
```

clientId	着信メッセージがあった TCP クライアントを示す
receivedData	Lua の文字列データとしてのメッセージ

このスクリプトは、事前に設定した TCP クライアント接続がデータを受信したときに呼び出されます。

LuaUDPServerMessageReceivedCallback

```
LuaUDPServerMessageReceivedCallback (serverId, receivedData)
```

serverId	着信メッセージがあった UDP サーバーを示す
receivedData	Lua の文字列データとしてのメッセージ

このスクリプトは、事前に設定した UDP サーバーが着信メッセージを受信したときに呼び出されます。

LuaUIUpdatedCallback

```
LuaUIUpdatedCallback (objectName, strData)
```

objectName	更新された UI オブジェクトの名前（ASCII 形式）
strData	文字列形式の新しいデータ値

このスクリプトは、UI オブジェクトの値が変更されたとき（たとえば、フェーダーが動かされたとき）に呼び出されます。ユーザーは strData を取得し、それをリモートデバイスへ送信して、デバイスの状態を更新する必要があります。

LuaTimerCallback

```
LuaTimerCallback ()
```

このスクリプトは、タイマーが満了したときに呼び出されます。タイマーを設定すると、このスクリプトが定期的に呼び出されます。このスクリプトは、値のポーリングに利用できます。また、スクリプト内で XStopTimer 関数を呼び出して、タイマーが再びトリガーされないよう停止することもできます。

Lua スクリプト内では、標準的な Lua のプログラミング機能に加えて、通信やその他の機能のために、次の関数呼び出しが用意されています。これらの補助関数は、次のグループに分類できます。

・TCP/UDP 通信の設定

・UI の操作

・タイマーの操作

・デバッグ用のコンソールログメッセージ

> 注:これらの各関数呼び出しが、個々のデバイスインスタンスの適切な Lua 環境にアクセスするには、各関数呼び出しの第 1 引数として、固有のドライバーID(Lua 初期化スクリプトの呼び出しで提供されるもの)を指定する必要があります。

XTCPServerSetup

```
XTCPServerSetup (driverId, serverPort, idleTimeout)
```

driverId	個々のドライバーインスタンスを識別する ID
serverPort	TCP サーバーのサーバーポート
idleTimeout	サーバーがアイドル状態の接続を切断するための、アイドルタイムアウト値(ms)
return	サーバーの固有 ID である serverId

この関数呼び出しは、指定したポートで TCP サーバーを設定します。設定に成功すると、0 以外の serverId を返します。この ID は、設定したサーバーを識別するために、以降のスクリプトで使用します。設定したサーバーは、着信 TCP 接続を待ち受けます(設定に失敗した場合は 0 を返します)。

XTCPServerClose

```
XTCPServerClose (driverId, serverId)
```

driverId	個々のドライバーインスタンスを識別する ID
serverId	閉じるサーバーを示す
return	成功した場合は true、それ以外は false

この関数は、事前に作成した TCP サーバーを閉じます。

XTCPServerReplyMessage

```
XTCPServerReplyMessage (driverId, serverId, clientId, sendData)
```

driverId	個々のドライバーインスタンスを識別する ID
serverId	メッセージを送信するサーバーを示す
clientId	メッセージの送信先となるクライアント接続を示す

sendData	ネットワークへ送信する文字列(バイナリー)データ
return	成功した場合は true、それ以外は false

この関数は、serverId と clientId で示される TCP サーバー接続へメッセージを送信します。

XTCPClientSetup

```
XTCPClientSetup (driverId, remoteIP, remotePort, idleTimeout)
```

driverId	個々のドライバーインスタンスを識別する ID
remoteIP	接続先のリモートデバイスの IP アドレス
remotePort	接続先のリモート TCP ポート
idleTimeout	接続を切断するアイドルタイムアウト値(ms)(0 はタイムアウトなし)
return	スクリプト内でデータ送信に使用できる、固有の clientId

この関数呼び出しは、指定した IP アドレスとポートのリモートデバイスに対する TCP クライアント接続を設定します。idleTimeout を設定した場合、指定した時間アイドル状態が続くと接続が閉じられます。接続がタイムアウトした後でも、再度メッセージを送信するだけで、リモートサーバーへ再接続されます。
この設定呼び出しを行った時点では、このクライアントへメッセージを送信するまで、リモートへの TCP 接続は実際には作成されません。

XTCPClientClose

```
XTCPClientClose (driverId, clientId)
```

driverId	個々のドライバーインスタンスを識別する ID
clientId	閉じるクライアントを示す
return	成功した場合は true、それ以外は false

この関数は、事前に作成した TCP クライアント接続を閉じます。

> 訳注:原文では、パラメーターの記載が「serverId:閉じるサーバーを示す」、説明文が「事前に作成した UDP サーバーを閉じる」となっています。関数名およびシグネチャ(clientId)と一致しないため、原文の誤記と判断し、上記のとおり訳しています(原文の誤記かどうかは、メーカーへの確認が取れていません)。

XTCPClientIsConnected

```
XTCPClientIsConnected (driverId, clientId)
```

driverId	個々のドライバーインスタンスを識別する ID

clientId	接続を確認するクライアント
return	接続が有効な場合は true、それ以外は false

この関数呼び出しは、クライアント接続が現在接続されているかどうかを確認します。接続されていない場合は、クライアントへメッセージを送信するだけで再接続できます。このような確認機能が用意されているのは、TCP クライアント接続が確立されたときに、特定のメッセージを送信できるようにするためです。TCP 接続でログインが必要な場合に便利です。

XTCPClientSendMessage

```
XTCPClientSendMessage (driverId, clientId, sendData)
```

driverId	個々のドライバーインスタンスを識別する ID
clientId	メッセージの送信先となるクライアント接続を示す
sendData	ネットワークへ送信する文字列(バイナリー)データ
return	成功した場合は true、それ以外は false

この関数呼び出しは、事前に設定した特定の TCP クライアント接続へメッセージを送信します。デバイスがまだ接続されていない場合、この送信呼び出しは、メッセージを送信する前にリモートへ接続します。

XUDPServerSetup

```
XUDPServerSetup (driverId, serverPort)
```

driverId	個々のドライバーインスタンスを識別する ID
serverPort	UDP サーバーのサーバーポート
return	サーバーの固有 ID である serverId

この関数呼び出しは、指定したサーバーポートで UDP サーバーを設定します。

XUDPServerClose

```
XUDPServerClose (driverId, serverId)
```

driverId	個々のドライバーインスタンスを識別する ID
serverId	閉じるサーバーを示す
return	成功した場合は true、それ以外は false

この関数は、事前に作成した UDP サーバーを閉じます。

XUDPClientSendMessage

```
XUDPClientSendMessage (driverId, remoteIP, remotePort, localPort, sendData)
```

driverId	個々のドライバーインスタンスを識別する ID
remoteIP	接続先のリモートデバイスの IP アドレス
remotePort	接続先のリモート TCP ポート
localPort	使用するローカルポート(0 の場合は、OS がポートを選択します)
sendData	ネットワークへ送信する文字列(バイナリー)データ
return	成功した場合は true、それ以外は false

この関数呼び出しは、リモートの UDP サーバーへメッセージを送信します。UDP はコネクションレスであるため、リモートの UDP サーバーへメッセージを送信する前に、クライアントを設定する必要はありません。

XUIUpdateRequest

```
XUIUpdateRequest (driverId, objectName, UIdata)
```

driverId	個々のドライバーインスタンスを識別する ID
objectName	更新する UI オブジェクトの名前(ASCII)。同じ名前の UI オブジェクトが複数ある場合は、その名前を持つすべての UI オブジェクトが更新されます
UIdata	UI オブジェクトへ反映するデータ
return	成功した場合は true、それ以外は false

この関数呼び出しは、objectName で識別される UI オブジェクトを更新します。同じ名前の UI オブジェクトが複数存在する場合があり、この呼び出しは、その名前を持つすべてのオブジェクトを更新します。

XStartTimer

```
XStartTimer (driverId, timeoutInterval)
```

driverId	個々のドライバーインスタンスを識別する ID
timeoutInterval	タイマーの間隔(ミリ秒)
return	なし

この関数呼び出しは、定期タイマーを開始します。タイムアウトが発生すると、対応する Lua スクリプトを呼び出します。1 つの Lua 環境で設定できるタイマーは 1 つだけです。XStartTimer を再度実行すると、以前に設定した timeoutInterval が置き換えられます。

XStopTimer

```
XStopTimer (driverId)
```

driverId	個々のドライバーインスタンスを識別する ID
return	なし

この関数呼び出しは、事前に設定したタイマーを停止します。

XConsoleLog

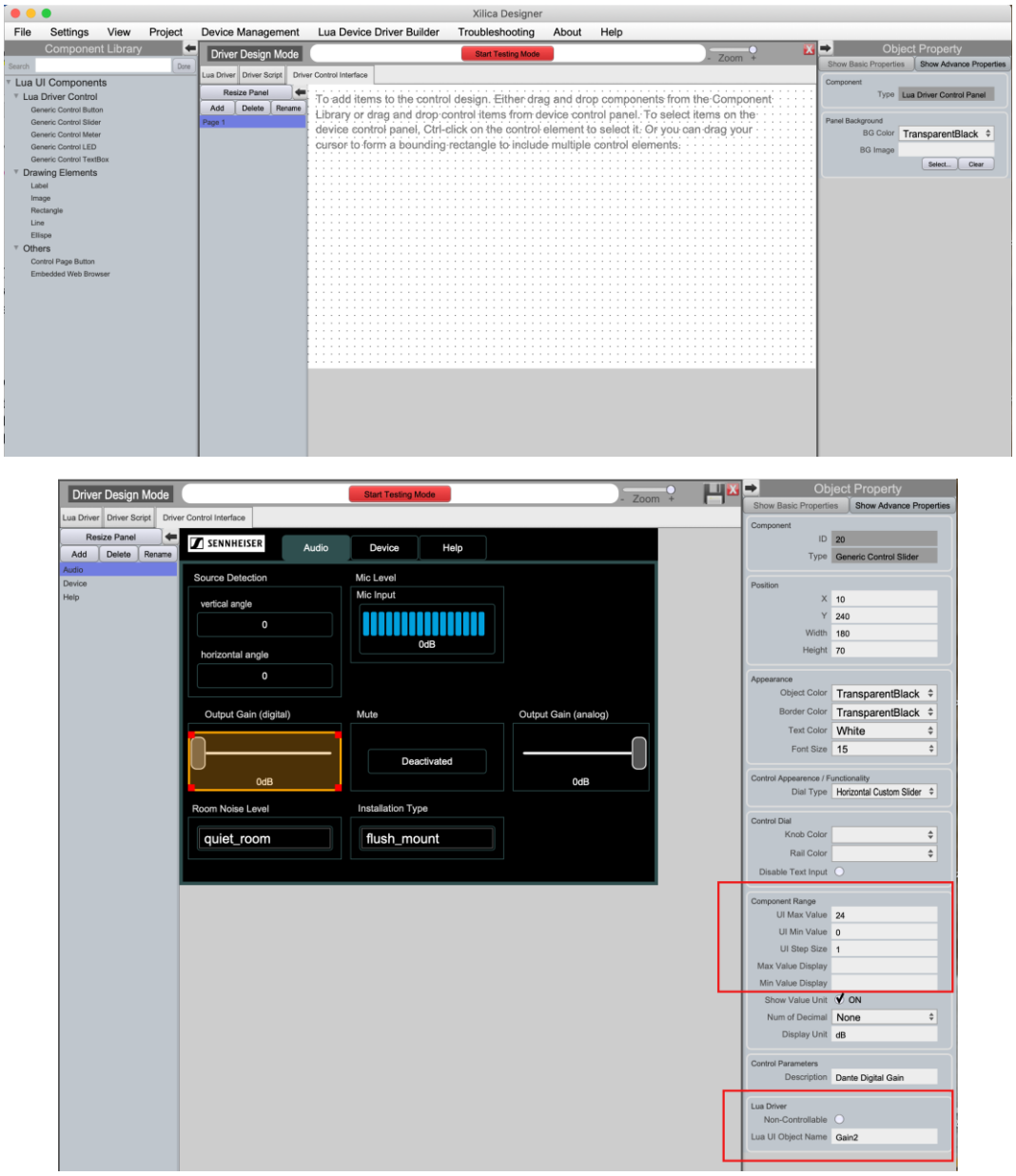
```
XConsoleLog (driverId, message)
```

driverId	個々のドライバーインスタンスを識別する ID
message	コンソールログへ送る文字列メッセージ
return	なし

この関数呼び出しは、メッセージをコンソールログへ送ります。コンソールログのメッセージは、ドライバーのデバッグセッション中に確認できます。通常の実行時には、この呼び出しは何の効果もありません。

5.4 ユーザーインターフェイスビルダー

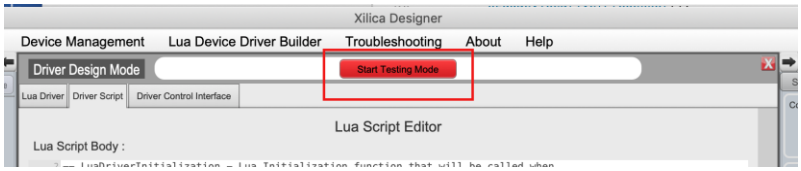
各ドライバーには、専用のユーザーインターフェイスが関連付けられます。ドライバーのユーザーインターフェイスの作成は、XTouch デバイスのユーザーインターフェイスの構築に似ています。フェーダー、ボタン、メーターなどを、インターフェイスエディターへドラッグして配置できます。
インターフェイスに配置した各 UI オブジェクトには名前を付けることができ、スクリプトからその名前で UI オブジェクトにアクセスできます。
また、複数のページを設定し、ページ切り替えボタンでこれらのページをリンクできます。
インターフェイスでは、複数のオブジェクトに同じ名前を付けることができます。この場合、UI 上でいずれかのオブジェクトの値が変更されると、Lua 側に UI 更新要求イベントが発生します。Lua スクリプトが名前付き UI オブジェクトの値を変更する場合は、同じ名前のすべてのオブジェクトが同時に更新されます。



図：UI オブジェクトの設定画面（赤枠：Component Range、および Lua Driver の Lua UI Object Name）

6　ドライバーのテストとデバッグ

ドライバーの情報、スクリプト本体、およびユーザーインターフェイスの設定が完了したら、ドライバーをテストできます。ドライバーをテストするには、メニューバー中央上部の「Start Testing Mode」ボタンをクリックします。

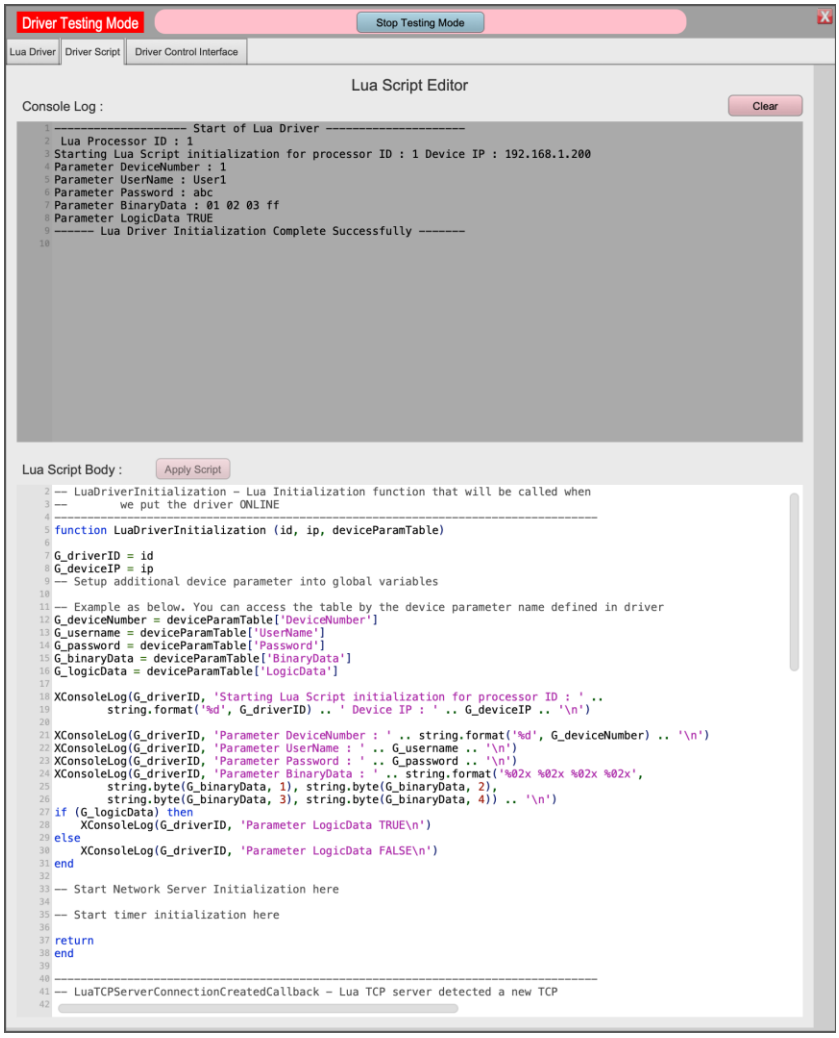


テストを開始する前に、ドライバーを保存しておく必要があります。テストを開始すると、物理デバイスへマッピングするためのダイアログが表示されるので、ターゲットの IP アドレスと、テストセッション用の関連パラメーターを入力します。



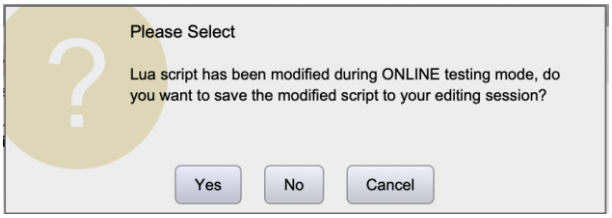
情報を入力して「Ok」をクリックすると、ドライバーが ONLINE になります。すべての編集パラメーターは読み取り専用となり、スクリプトエディターの上部にコンソールログ領域が表示されます。スクリプト内で生成されたすべてのコンソールメッセージが、このログ領域に表示されます。ログが長くなりすぎた場合は、「Clear」ボタンを押して古いログを消去できます。

スクリプト領域では、スクリプトをオンラインのまま直接変更できます。「Apply Script」ボタンを押すと、スクリプトの初期化が再開されます。開いているすべての接続はいったん閉じられ、新しい初期化スクリプトに従って再び開かれます。



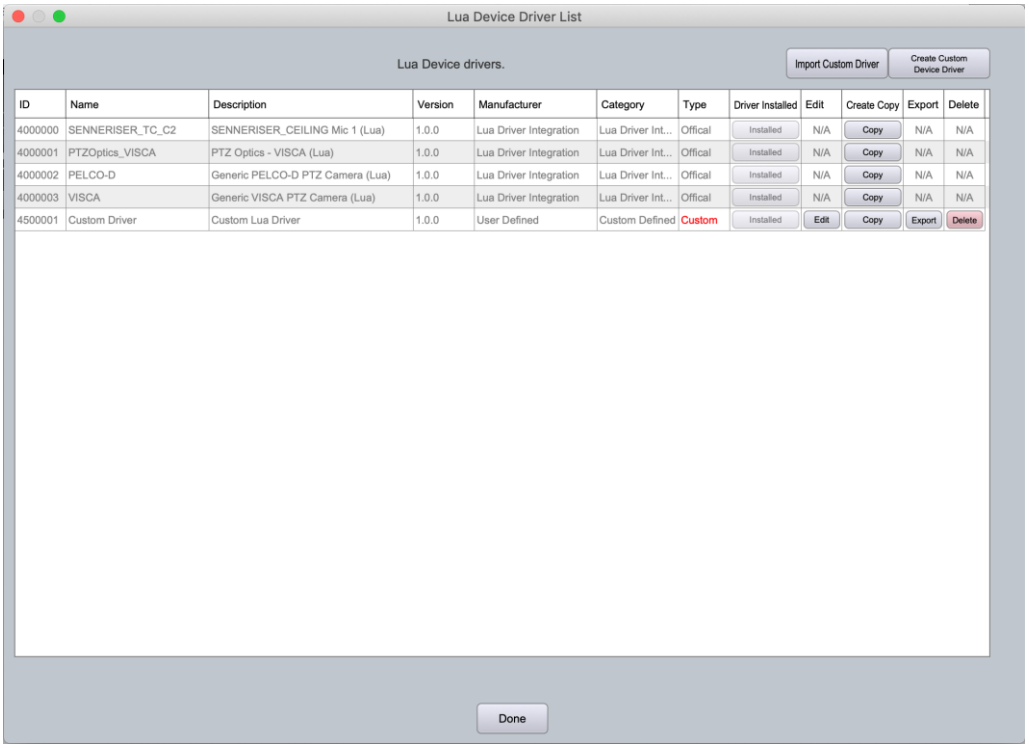
コントロールインターフェイスも ONLINE の制御モードになります。フェーダーやボタンの状態を変更すると、関連するスクリプトが実行され、それらのスクリプトからのログを確認できます。リモート側で変更があった場合は、Lua スクリプトから UI オブジェクトの状態を駆動することもできます。

テストセッションが終了したら、「Stop Testing Mode」を押してテストを停止し、編集モードに戻ります。テスト中にスクリプトを変更した場合は、スクリプトの変更を保存するか破棄するかを確認するダイアログが表示されます。



7　ドライバーのインポートとエクスポート

カスタムドライバーは、対応する「Export」ボタンをクリックしてエクスポートできます。エクスポートされるのは、すべての情報（ドライバー情報、スクリプト、コントロールインターフェイスを含む）を格納した XML ファイルです。このファイルを他のユーザーと共有できます。



他のユーザーがこのドライバーを使用するには、「Import Custom Driver」ボタンをクリックして、自分の Xilica Designer へドライバーをインポートします。インポートすると、デバイスが Component Library に自動的に一覧表示されます。

ドライバーを一度インストールすると、同じドライバーを再インストールすることはできません。再インストールするには、先に古いドライバーを削除する必要があります。同じドライバーを再インストールしようとすると、警告ダイアログが表示されます。

注：ユーザー定義のカスタムドライバーでは、次の項目がドライバーの識別情報を定義します。

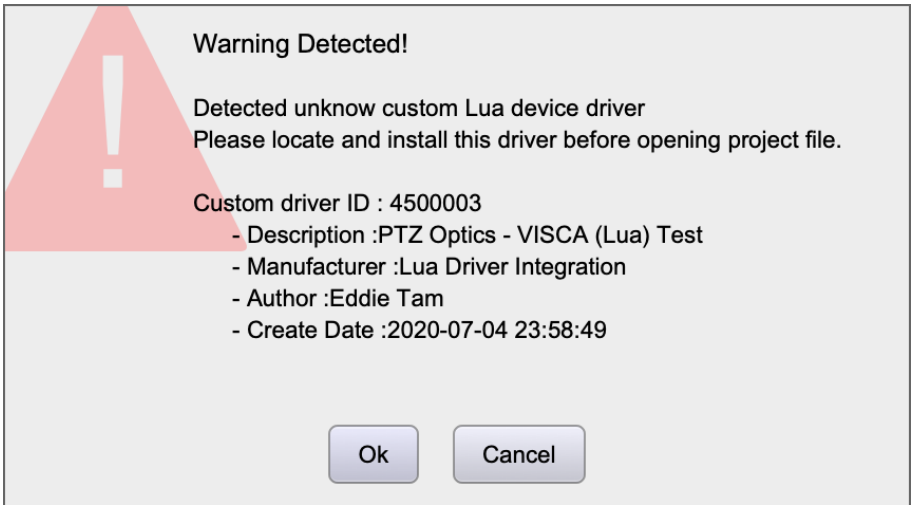
- ・Driver Description（ドライバーの説明）
- ・Driver Manufacturer（ドライバーのメーカー）
- ・Author（作成者）
- ・Creation Date（作成日）

これらのパラメーターのいずれかが変更されると、そのドライバーは別のドライバーとみなされます。

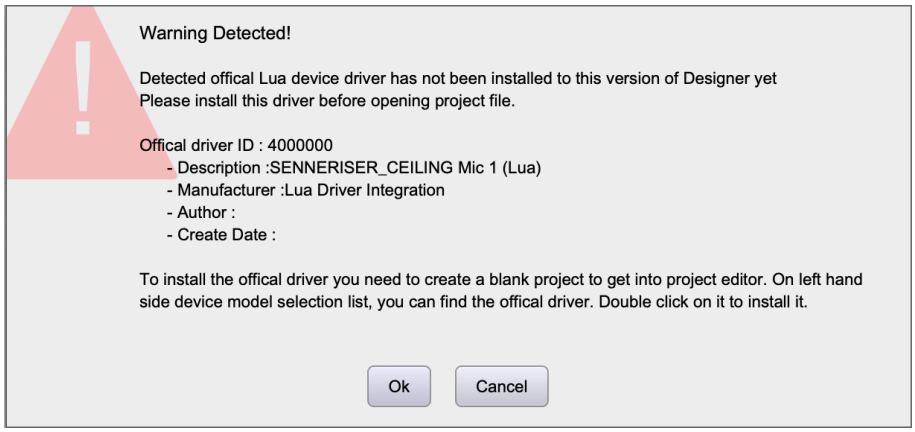


8 プロジェクトファイルとドライバー

ユーザー定義のカスタムドライバーを含むプロジェクトファイルを受け取った場合、そのプロジェクトファイルを開くことはできません。不足しているドライバーの詳細を示すダイアログが表示されます。プロジェクトファイルを開く前に、ドライバーの所有者に連絡して、カスタムドライバーをインストールする必要があります。
ドライバーのインポート時には、お使いの Xilica Designer のインストール環境内で、別のドライバーID がローカルに自動的に割り当てられます。



公式ドライバーを使用したデバイスを含むプロジェクトファイルの場合、Xilica Designer は、それらのドライバーがインストール済みかどうかを確認します。未インストールの場合は、Designer のプロジェクトエディターでドライバーをインストールしてください。すべての公式ドライバーは左側に一覧表示されます。ドライバーを見つけてダブルクリックするとインストールされます。インストールが完了すると、プロジェクトファイルを開けるようになります。


Warning Detected!
Detected offical Lua device driver has not been installed to this version of Designer yet
Please install this driver before opening project file.
Offical driver ID : 4000000
- Description :SENNERISER_CEILING Mic 1 (Lua)
- Manufacturer :Lua Driver Integration
- Author :
- Create Date :
To install the offical driver you need to create a blank project to get into project editor. On left hand side device model selection list, you can find the offical driver. Double click on it to install it.
Ok
Cancel