**HTTPサーバマクロ  
ソースコード構成説明書**

**目次**

[1. 目的 4](#_Toc507090410)

[2. 適用範囲 4](#_Toc507090411)

[3. ソリューション構成 5](#_Toc507090412)

[4. NISからの起動ルート 6](#_Toc507090413)

[5. APIの追加方法 7](#_Toc507090414)

[5.1. APIクラスの追加 7](#_Toc507090415)

[5.2. NisAPIBaseクラスの継承 8](#_Toc507090416)

[5.3. パラメータの定義（パラメータ有りのAPIの場合） 9](#_Toc507090417)

[5.3.1. パラメータの順序定義 9](#_Toc507090418)

[5.3.2. パラメータの内容定義 10](#_Toc507090419)

[5.4. API処理の実装 12](#_Toc507090420)

[5.4.1. NISマクロの実行 12](#_Toc507090421)

[5.4.2. その他処理 12](#_Toc507090422)

[5.4.3. HTTPレスポンスの設定 13](#_Toc507090423)

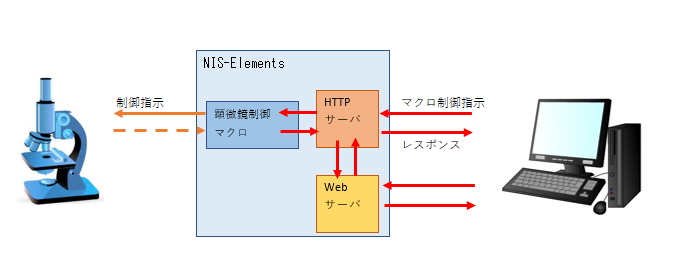
# 目的

本ソフトウェアのソースコードの構成を記載し、その後の運用における一助となる事目的とする。

# 適用範囲

リクエスト発信から顕微鏡制御までの概要全体図は下記のとおりである。

本インターフェース仕様の適用箇所を赤矢印にて示した。



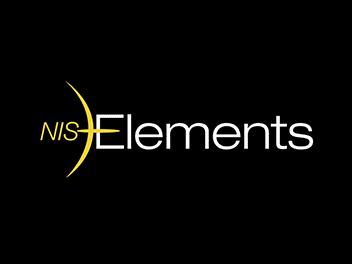
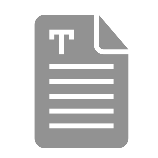
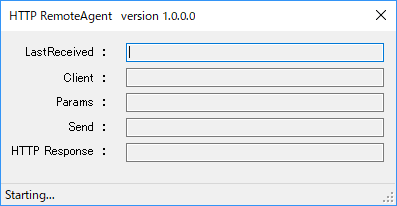
# ソリューション構成

|  |  |  |  |
| --- | --- | --- | --- |
| 項目(注記ない場合はフォルダ名) | | | 概要 |
| HttpConnectプロジェクト | | | 本ソフトウェアの本体となる、exeファイルを生成するプロジェクト。 |
|  | API | | API実装を格納するフォルダ  装置毎に装置名のフォルダを作成し、APIクラスを格納する。 |
|  |  | NIS\_Elements | NIS-ElementsのAPIを格納するフォルダ |
|  |  | APIValue | APIのパラメータに関するクラスを格納するフォルダ |
|  | APIFactory | | 実装したAPIと、HTTPで受信したリクエストを紐付ける為のクラスを格納するフォルダ。 |
|  |  | NIS-Elements | NIS-Elements向けの実装を格納するフォルダ |
|  | Common | | 本ソフト共通で使用するクラスなどについて格納するフォルダ |
|  | EventArg | | 本ソフトのイベントで使用するイベント引数クラスについて格納するフォルダ |
|  | Forms | | 画面クラスを格納するフォルダ |
|  | HttpUtil | | Http通信に関連するクラス格納するフォルダ |
|  | InternalData | | 内部データを格納するフォルダ |
|  | Log | | ログ機能について格納するフォルダ |
|  | UserUtil | | 汎用的に使用できるクラスを格納するフォルダ |

# NISからの起動ルート

NIS-Elementsから起動される場合の処理の流れについて記載する。

先ずは以下に概要図を示す。



Main()

StartHttpConnect()

Run macro form file

１．NIS-Elementsにて起動マクロファイルを実行する

２.　起動マクロファイルは、起動DLLファイル(HttpConnect.Core.dll)の開始メソッドをCallする。

３．起動Dllファイルは、サーバーアプリのMainメソッドを実行する。

即ち、最終的に起動されるのはサーバーアプリ(HttpConnect.exe)であり、起動DLLファイル（HttpConnect.Core.dll）

はサーバーアプリを呼び出す為のDLLに過ぎない。

C#にて制作したアプリケーションを起動マクロファイルから直接呼び出すことが出来ないため、C++にて開発した

起動Dllファイルを経由している。

# APIの追加方法

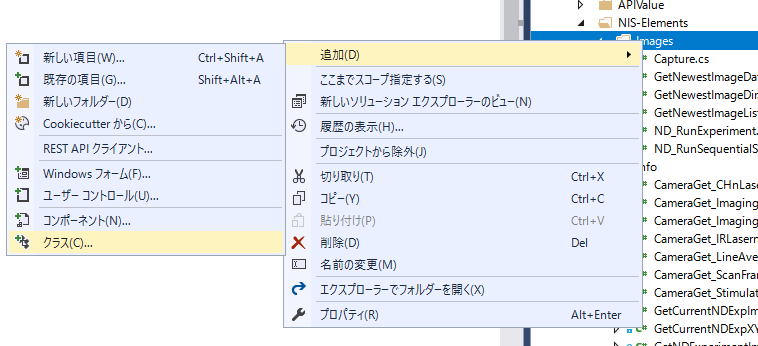
APIの設計概要の詳細は「HTTPサーバーマクロ\_API設計概要.doc」に記載する。

ここでは、ソースコードに即した形でのNIS-ElementsのAPI追加方法を記載する。

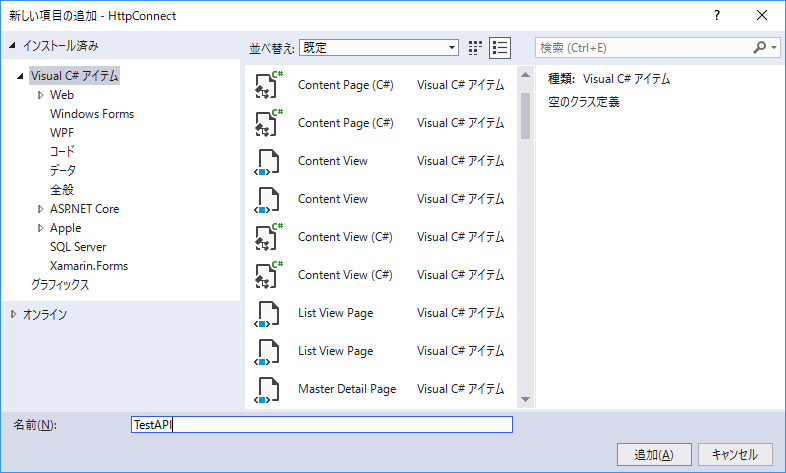
## APIクラスの追加

APIは、１API＝１ファイルとして作成する。

先ずは指定のフォルダにクラスを新規追加する。以下にVisualStudio上でのクラスの追加方法について記載する。



任意のフォルダにて右クリックをし、「追加」＞「クラス」を選択する。



上図の画面が表示されるので、「名前」欄に任意のクラス名を入力し、「追加」をクリックする。

ここで、入力したクラス名がそのままAPI名となることに留意する。

　→※ソースコード上で任意に変更する事も可能である。

## NisAPIBaseクラスの継承

全てのAPIは「IAPIBase」インターフェースを実装したクラスとする。

各APIの実行処理の詳細は対応する顕微鏡ソフトウェアなどにより異なることが想定されるた為、各顕微鏡ソフトウェア毎の

API用にこの「IAPIBase」インターフェースを実装した基底クラスを用意することが望ましい。

NIS-Elementsにおいては「NisAPIBase」クラスが上記の基底クラスとなっている為、このクラスを継承する。

namespace HttpConnect.API.NIS\_Elements.Images

{

public class TestAPI : NisAPIBase

{

}

}

この「NisAPIBase」には、基本的に必要な機能が実装されている。

以下に新しく追加したAPIにて必要な実装を記載する。

## パラメータの定義（パラメータ有りのAPIの場合）

各APIにはパラメータを必要とするケースが想定される。

パラメータは「APIValueBase」を継承したクラスによって表され、受信したリクエストの解析と値の格納・取得を

これら各クラスで行う。

現状では「文字列型」を表す「StringValue」クラス、「Double型」を表す「DoubleValue」クラス、「Int型」を表す

「IntValue」クラスを実装している。

パラメータが存在しないAPIの場合は、本章における実装は不要である。

以下に実装例のソースコード記載し、その次に各処理について順を追って説明する。

private enum eQueryIndex

{

IntParam = 0,

DoubleParam,

StringParam

}

protected override void DefineAPIParams()

{

ParamValues.Add(eQueryIndex.IntParam,

new IntValue()

{

MinValue = 0,

MaxValue = 100

});

ParamValues.Add(eQueryIndex.DoubleParam,

new DoubleValue()

{

// 今後変化する可能性を考慮しあえて範囲を設けない

});

ParamValues.Add(eQueryIndex.StringParam,

new IntValue()

{

MaxLength = 1024

});

CreateParamList(typeof(eQueryIndex));

}

パラメータの順序定義

パラメータの内容定義

### パラメータの順序定義

本ソフトウェアに於けるHTTP通信のパラメータは引き渡される順序が意味を成す為、パラメータを必要とするAPIについては

パラメータの順序を定義する必要がある。

パラメータ順序はEnumで定義をする。Enumであれば形は問わず、名前やEnumの数値等は自由である。

※但し数値に重複なきこと。

上記の例では「IntParam」が「０」番目に渡されるパラメータで、「DoubleParam」「StringParam」が続いて

渡されるパラメータを意味する。

### パラメータの内容定義

次にパラメータの内容を定義する。

パラメータが持つ要素として、以下の４つが挙げられる。

・パラメータの順序

・型

・値の範囲

・文字列の最大数

以下に「5.3パラメータの定義（パラメータ有りのAPIの場合）」にて示した例からパラメータの内容定義部を抜粋して記載する。

protected override void DefineAPIParams()

{

ParamValues.Add(eQueryIndex.IntParam,

new IntValue()

{

MinValue = 0,

MaxValue = 100

});

ParamValues.Add(eQueryIndex.DoubleParam,

new DoubleValue()

{

//範囲を設けない

});

ParamValues.Add(eQueryIndex.StringParam,

new IntValue()

{

MaxLength = 1024

});

CreateParamList(typeof(eQueryIndex));

}

パラメータの順序

値の範囲

型

文字列の最大数

#### パラメータの順序

パラメータの定義は、連想記憶配列（Dictionary）にて、「5.3.1パラメータの順序定義」にて定義したEnumの値をKey、

本節にて定義するパラメータの内容をValueとして保持する。示した「ParamValues」は「NisAPIBase」クラスにて宣言されている

Dictionary型の変数となっているため、ここにパラメータの定義を追加していく。

#### 型

パラメータの型は「APIValueBase」を継承した各型のクラスによって表される。この為、各クラスのインスタンスを格納することに

より、各パラメータの型が定義される。

#### 値の範囲

パラメータには実行時に値の範囲を制限する必要がある場合が想定される。例えば、システム上取りうる値が決まっている

もの（レーザーパワ等）である。

このように値の範囲を制限したい場合は「MinValue」及び「MaxValue」にて上下限を設定する。

範囲を設定する必要がない場合は、本節の実装は不要である。

この場合、値の範囲は当該型の取りうる最大・最小の値となる。

また、この範囲は「IntValue」および「DoubleValue」の場合に有効となり、文字列型のパラメタの場合は本設定

は考慮されない。

#### 文字列の最大数

値の範囲同様、文字列型のパラメータについて文字数を制限する必要がある場合が想定される。

文字数を制限する場合は「MaxLength」にて上限を設定する。

上限を設定する必要がない場合は、本節の実装は不要である。

この場合、文字数の上限は「Int型」の取りうる最大の値(2,147,483,647)となる。

## API処理の実装

API処理は「NisAPIBase」の「RunAPIImpl」メソッドをオーバライドして実装する。

以下に実装例を示す。

protected override bool RunAPIImpl()

{

try

{

NisMacro.Net.Macro.Macros.ROI.ChangeROIType macro

= new NisMacro.Net.Macro.Macros.ROI.ChangeROIType()

{

RoiId = ParamValues[eQueryIndex.ID].ToInt,

RoiType = ParamValues[eQueryIndex.ROIType].ToInt

};

AddMacroToQueSync(macro);

HttpResponse.ResponseParam.Add("test");

HttpResponse.ResultStatusCode = CommonDefine.eHttpStatusCode.OK;

return true;

}

catch (Exception e)

{

HttpResponse.ResultStatusCode = CommonDefine.eHttpStatusCode.InternalServerError;

OutputLog("", e);

return false;

}

}

### NISマクロの実行

NISマクロを実行する際は、NISマクロフレームワークの「NisMacro.Net.Macro.dll」にて実装したマクロクラスの

インスタンスを生成し、値を設定する。

生成したインスタンスは以下の「NisAPIBase」にて実装されている以下の２メソッドにて実行可能である。

・AddMacroToSync：同期的にNISマクロを実行する。

・AddMacroToAsync：NISマクロの実行キューに実行マクロを追加し、追加したマクロの処理終了を待たずに

　　　 　　　　　　　　　　　　　　　　処理をすすめる。

### その他処理

本メソッド内ではNISマクロを実行せず、別の処理を実装することも可能である。

### HTTPレスポンスの設定

API処理の実装では、かならずHTTPのレスポンスデータを格納し、処理を終了することとする。

Httpのレスポンスデータは「NisAPIBase」にて保持している「HttpResponse」に格納する。

#### レスポンスステータスの格納

「HttpResponse」の「ResultStatusCode」に、クライアントへ返却するHTTPステータスコードを格納する。

#### HTTPレスポンスデータの格納

　クライアントへ返却するデータが存在する場合「HttpResponse」の「ResponseParam」へデータを追加する。

本変数はObject型のListとなっている為、任意の型を格納できる。

但し、最終的に全て文字列に変換した後にクライアントへ送信されることに留意する。

以上