

# CONTENTS

---

<b>サンプルの Experiment – Multi-Experiment Process</b> .....	2
クイックノート .....	2
手順.....	2
自分のデータで処理する場合 .....	4
手順.....	4
Multi-Experiment Process.ipf プロシージャの内容.....	5

# サンプルの Experiment – Multi-Experiment Process

## クイックノート

### メニュー File → Example Experiments → Programming → Multi-Experiment Process

この Experiment では、「Multi-Experiment Process.ipf」プロシージャファイルについて説明します。このプロシージャファイルは、複数の Experiment にまたがるプロセスを示しています。

課題は、特定のフォルダーにある一連の圧縮された Experiment ファイル（拡張子「.pxp」）を開くことです。各 Experiment ファイルには、wave\_lif1 と wave\_lif2 という名前のウェーブが含まれています。これらのウェーブをグラフ化し、PNG ファイルとしてエクスポートし、次の実験のためにこのプロセスを繰り返すことができるようにします。

「Multi-Experiment Process.ipf」ファイルには、Experiment を順に実行し、各 Experiment に対して処理を実行するプロシージャが含まれています。

このファイルは、「Igor Pro Folder:Examples:Programming:Multi-Experiment Process」フォルダーにあります。

プロシージャファイルには、AfterFileOpenHook フック関数が含まれています。

ファイルが開かれると、Igor がこのフックを呼び出します。

フックは、開こうとしているファイルがパックされた Experiment ファイルであり、期待通りのウェーブを含んでいるかどうかを確認します。

ファイルが正しいければ、その中のウェーブをグラフ化し、PNG ファイルとしてグラフをエクスポートします。

次に、その次の Experiment ファイルを開き、処理を繰り返します。

プロシージャファイルは、AfterFileOpenHook の呼び出しの間に使われる情報を保存するために、Igor の LoadPackagePreferences と SavePackagePreferences を使います。

これは、次の Experiment の処理を開始するためのコマンドを Igor のコマンドキューに送信するために、Execute/P を使っています。

以下で説明するように、デモを試すには、「Multi-Experiment Process」フォルダーを Igor Pro フォルダーからディスク上の書き込み可能な場所にコピーする必要があります。

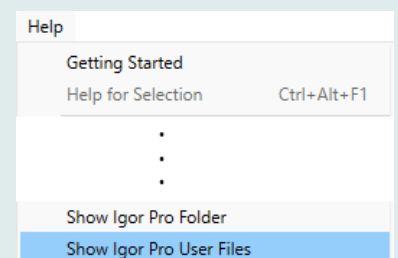
Windows では、通常、オペレーティングシステムが Igor Pro フォルダー、そのサブフォルダーへの書き込みを許可していません。

## 手順

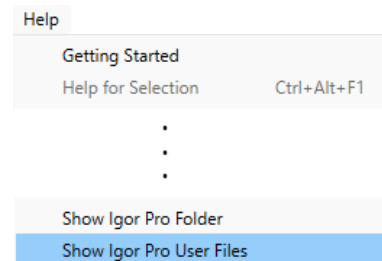
**1.** Igor を起動して、メニュー Help → Show Igor Pro User Files を選択すると、フォルダーの場所が表示されます（多くの場合、ドキュメントフォルダーにあります）。

**このフォルダーをデスクトップなど、アクセスが許可されている別の場所にコピーします。**

メニュー Misc → Miscellaneous Settings を選択して、Miscellaneous Settings ダイアログの左のメニューから Igor Pro User Files を選択して、Change Path ボタンを使って新しいフォルダーを指すようにします。



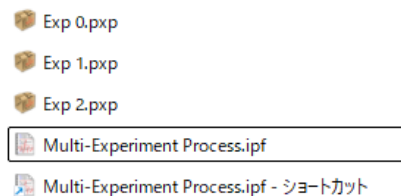
2. 次に、メニュー Help → Show Igor Pro Folder を選択し、Examples:Programming フォルダを Multi-Experiment Process フォルダが表示されるまで掘り下げてください。



3. Multi-Experiment Process フォルダを、コピーした Igor Pro User Files フォルダ内にコピーし、フォルダ名を「Multi-Experiment Process Test」とします。

4. Multi-Experiment Process Test フォルダを開きます。

フォルダ内の Multi-Experiment Process.ipf プロシージャファイルのショートカットを作成します。



5. ショートカットを、Igor Pro User Files フォルダ内の Igor Procedures フォルダに配置します。

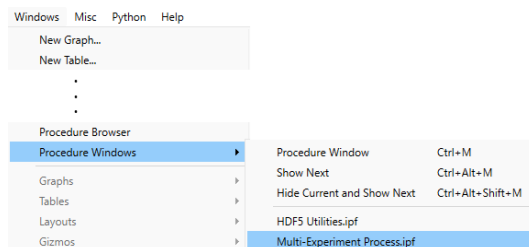
これは、Igor に、そのファイルがグローバルプロシージャファイルとして開かれ、Igor が終了するまで開いたままにしておくことを指示します。

6. Igor を再起動して、Multi-Experiment Process.ipf をグローバルプロシージャファイルとして読み込みます。

メニュー Windows → Procedure Windows を開きます。

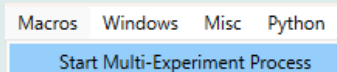
Multi-Experiment Process.ipf が表示されていることを確認してください。

これは、ファイルがグローバルプロシージャファイルとして Igor によって自動的に開かれたことを示しています。



7. サンプルの Experiment だけを開いている場合は、メニュー File → Start Another Igor Pro Instance を選択して、新しい Experiment を作成します。

Macros メニューから Start Multi-Experiment Process を選択します。



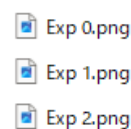
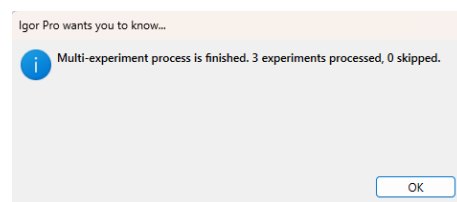
8. Igor は、処理する Experiment ファイルが含まれているフォルダを選択するダイアログを表示します。

前の手順で作成した Multi-Experiment Process Test フォルダを選択します。

このフォルダには、Exp 0.pxp、Exp 1.pxp、Exp 2.pxp の Experiment ファイルが含まれています。

その後、プロシージャがフォルダ内の各 Experiment ファイルを開いて処理が始まります。

開く順番は、オペレーティングシステムに依存します。



Experiment ファイルに期待されるウェーブが含まれている場合、プロセスはウェーブをグラフ化し、Experiment と同じベース名の .png ファイルにエクスポートします。

## 自分のデータで処理する場合

### 手順

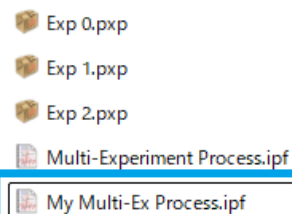
前の手順の続きから説明します。

#### 1. Igor を終了します。

#### 2. Multi-Experiment Process Test フォルダー内の Multi-Experiment Process.ipf プロシージャファイルの名前を変更し、任意の名前を付けてください。

ここでは、My Multi-Ex Process.ipf としました。

また、必要に応じて Multi-Experiment Process Test フォルダーの名前を変更することもできます。



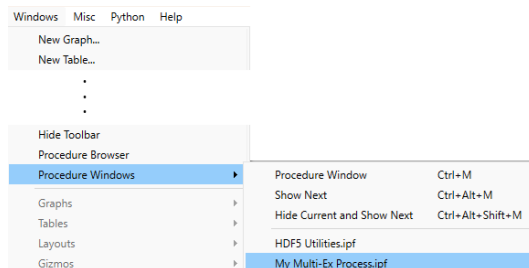
#### 3. Igor Procedures フォルダーに作成した古いショートカットを削除します。

名前を変更したプロシージャファイルのショートカットを Igor Procedures フォルダーに配置します。

#### 4. Igor を起動します。

Igor は、自動的に、名前を変更したプロシージャファイルをグローバルプロシージャファイルとして開きます。

メニュー **Windows** → **Procedure Windows** を選択し、名前を変更したプロシージャファイルを選択して表示します。



5. このステップは、プロシージャファイルで使用されているプレファレンスが、他のプレファレンスと衝突しないことを確認するためのものです。

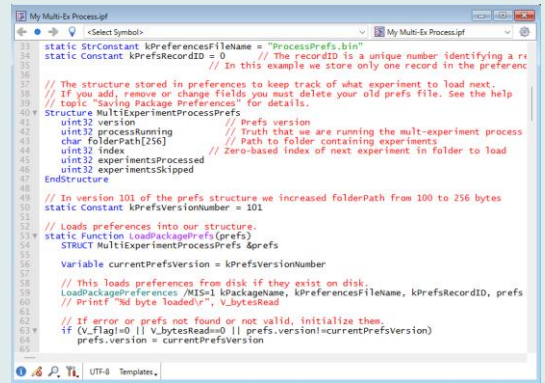
プロシージャファイル内の「パッケージ名」を、独自の識別名に変更してみます。

例えば、

```
static StrConstant kPackageName = "Multi-experiment  
Process"
```

を、次のように変更します：

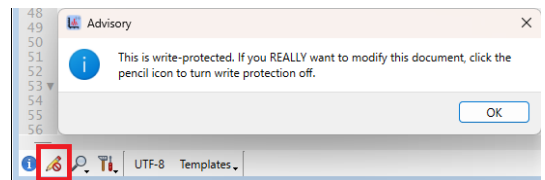
```
static StrConstant kPackageName = "WMHR Multi-experiment  
Process"
```



```
30 // NOTE: If you use these procedures for your own purposes, change the package name  
31 static StrConstant kPackageName = "WMHR Multi-experiment Process"  
32 static Constant kPrefsRecordID = 0 // The recordID is a unique number identifying  
33 // In this example we store only one record in the preferences
```

6. 編集しようとする時、右図のように書き込み保護されていると表示されることがあります。

その場合は、ウィンドウ右下の鉛筆アイコンをクリックして、保護を解除します。



7. 次に、プロシージャファイルを読んでいき、必要に応じて変更します。

変更のほとんどは、ProcessCurrentExperiment および IsAppropriateExperiment 関数に対して行うこととなります。

## Multi-Experiment Process.ipf プロシージャの内容

```
#pragma rtGlobals=1 // Use modern global access method.  
  
// MULTI-EXPERIMENT PROCESS について  
  
// このプロシージャファイルは、複数の Experiment にまたがるプロセスを示しています。  
  
// 課題は、特定のフォルダーにある一連の圧縮された Experiment ファイル（拡張子「.pxpl」）  
// を開くことです。  
// 各 Experiment ファイルには、wave_lif1 と wave_lif2 という名前のウェーブが含まれているとします。  
// これらのウェーブをグラフ化し、PNG ファイルとしてエクスポートし、次の Experiment に対して  
// この処理を繰り返すプロシージャを作成します。  
  
// このファイルには、Experiment を順に実行し、各 Experiment に対して処理を実行するプロシージャが含ま  
// れています。  
  
// ファイルが開かれると、Igor は、このファイル内の AfterFileOpenHook フックを呼び出します。  
// フックは、開こうとしているファイルがパックされた Experiment ファイルであり、期待通りのウェーブ  
// を含んでいるかどうかを確認します。  
// もしそうであれば、フック関数は、ウェーブをグラフ化し、PNG ファイルとしてグラフをエクスポートする  
// サブルーチンを呼び出します。  
// 次の Experiment ファイルを開き、処理を繰り返します。
```

```
// このプロシージャファイルでは、AfterFileOpenHook の呼び出しの間に使われる情報を保存するために、
// Igor の LoadPackagePreferences と SavePackagePreferences を使っています。
// 次の Experiment を開始するためのコマンドを Igor のコマンドキューに送信するために、
// Execute/P を使っています。
```

```
// デモの実行
```

```
// メニュー File -> Example Experiments -> Programming -> Multi-Experiment Process
// を選択してデモ Experiment を開き、その指示に従ってください（本ファイルの前半に記載）。
```

```
Menu "Macros"
```

```
    "Start Multi-Experiment Process", /Q, StartMultiExperimentProcess()
```

```
End
```

```
// 注記：これらのプロシージャを自分の目的で使う場合は、他のプレファレンスと衝突しないように、
// パッケージ名をわかりやすい名前に変更してください。
```

```
static StrConstant kPackageName = "Multi-experiment Process"
```

```
static StrConstant kPreferencesFileName = "ProcessPrefs.bin"
```

```
static Constant kPrefsRecordID = 0
```

```
    // recordID は、プレファレンスファイル内のレコードを特定する固有の ID です。
```

```
    // この例では、プレファレンスファイルには1つのレコードのみを保存します。
```

```
// 次にどの Experiment を読み込むかを追跡するために、構造はプレファレンスに保存されます。
```

```
// フィールドを追加、削除、変更した場合は、古い prefs ファイルを削除する必要があります。
```

```
// 詳細は、「Saving Package Preferences」のヘルプトピックを参照してください。
```

```
Structure MultiExperimentProcessPrefs
```

```
    uint32 version                                // Prefs のバージョン
```

```
    uint32 processRunning                        // 複数 Experiment 処理を実行中かどうか
```

```
    char folderPath[256]                        // Experiment を含むフォルダーへのパス
```

```
    uint32 index                                // 読み込みフォルダー内の次の Experiment のゼロベースインデックス
```

```
    uint32 experimentsProcessed
```

```
    uint32 experimentsSkipped
```

```
EndStructure
```

```
// このプロシージャのバージョン 101 のプリファレンス構造において、folderPath を 100 バイトから
```

```
// 256 バイトに増やしました。
```

```
static Constant kPrefsVersionNumber = 101
```

```
// 構造にプレファレンスを読み込みます。
```

```
static Function LoadPackagePrefs(prefs)
```

```
    STRUCT MultiExperimentProcessPrefs &prefs
```

```
    Variable currentPrefsVersion = kPrefsVersionNumber
```

```
    // ハードディスク上に存在する場合、そこからプレファレンスを読み込みます。
```

```
    LoadPackagePreferences /MIS=1 kPackageName, kPreferencesFileName,
```

```
kPrefsRecordID, prefs
```

```
    // エラーまたは設定が見つからない、または無効な場合は、初期化します。
```

```
    if (V_flag!=0 || V_bytesRead==0 || prefs.version!=currentPrefsVersion)
```

```
        prefs.version = currentPrefsVersion
```

```
        prefs.processRunning = 0
```

```
        prefs.folderPath = ""
```

```
        prefs.index = 0
```

```
        prefs.experimentsProcessed = 0
```

```
        prefs.experimentsSkipped = 0
```

```

        SavePackagePrefs (prefs)          // デフォルトの prefs ファイルを作成します。
    endif
End

// 構造をプレファレンスに保存します。
static Function SavePackagePrefs (prefs)
    STRUCT MultiExperimentProcessPrefs &prefs

    SavePackagePreferences kPackageName, kPreferencesFileName, kPrefsRecordID, prefs
End

// このプロシージャファイルを自分の目的で使うには、このルーチンを変更する必要があります。
static Function ProcessCurrentExperiment (prefs)
    STRUCT MultiExperimentProcessPrefs &prefs

    String folderPath = prefs.folderPath

    String experimentName = IgorInfo(1)
    String tmp = RemoveEnding(experimentName, ".pxp")
    String fullPath = folderPath + tmp + ".png"

    Display wave_lif1 wave_lif2
    ModifyGraph rgb(wave_lif1)=(0,0,65280), rgb(wave_lif2)=(0,52224,0)
    ModifyGraph mirror=2

    // 注釈を追加します。
    String text
    sprintf text, "Experiment %d, ¥"%s¥"", prefs.index, experimentName
    TextBox/C/N=text0/M/H=36/A=LT/X=5.00/Y=0.00 text

    SavePICT/E=-5/RES=600/I/W=(0,0,4,3)/O as fullPath
End

static Function IsAppropriateExperiment ()
    if (WaveExists(wave_lif1) && WaveExists(wave_lif2))
        return 1          // 処理したいデータと思われる場合。
    endif
    return 0              // 処理したいデータとは異なる場合。
End

// 次に読み込む Experiment ファイルのフルパスを返します。すべてが完了した場合は "" を返します。
static Function/S FindNextExperiment (prefs)
    STRUCT MultiExperimentProcessPrefs &prefs

    String folderPath = prefs.folderPath
    NewPath/O/Q MultiExperimentPath, folderPath

    String nextExpName = IndexedFile(MultiExperimentPath, prefs.index, ".pxp")

    if (strlen(nextExpName) == 0)
        return ""
    endif

    String fullPath = prefs.folderPath + nextExpName
    return fullPath
End

// 現在の Experiment を終了し、次の Experiment を開始するために、Igor のコマンドキューにコマンド
// を送信します。

```

// アイドル状態、つまり関数やコマンドを実行していないときに、コマンドキューのコマンドを実行します。

```
static Function PostLoadNextExperiment (nextExperimentFullPath)
    String nextExperimentFullPath

    Execute/P "NEWEXPERIMENT"           // Post コマンドでこの Experiment を閉じます。

    // Post コマンドで次の Experiment を開きます。
    String cmd
    sprintf cmd "Execute/P ¥"LOADFILE %s¥", nextExperimentFullPath
    Execute cmd

End
```

// これは、ファイルが開かれるたびに Igor が呼び出すフック関数です。

// Experiment の開始を検知し、ProcessCurrentExperiment 関数を呼び出すためにこれを使います。

```
static Function AfterFileOpenHook (refNum, file, pathName, type, creator, kind)
    Variable refNum, kind
    String file, pathName, type, creator

    STRUCT MultiExperimentProcessPrefs prefs

    LoadPackagePrefs (prefs)           // 構造に prefs を読み込みます。
    if (prefs.processRunning == 0)
        return 0                       // まだ処理が開始されていない場合。
    endif

    // ファイル形式をチェックします。
    if (CmpStr (type, "IGsU") != 0)
        return 0                       // パックされた Experiment ファイルではない場合。
    endif

    // 想定しているウェーブがあるかを確認します。
    if (IsAppropriateExperiment ())
        ProcessCurrentExperiment (prefs)
        prefs.index += 1               // 次の Experiment に進むためのインデックス更新。
        prefs.experimentsProcessed += 1
    else
        DoAlert 0, "This experiment is not suitable. Skipping to next
experiment."
        prefs.experimentsSkipped += 1
    endif

    // 次に処理する Experiment があるかを確認します。
    String nextExperimentFullPath = FindNextExperiment (prefs)
    if (strlen (nextExperimentFullPath) == 0)
        // 処理が完了した場合。
        prefs.processRunning = 0       // 処理が完了したことを示すフラグ。
        Execute/P "NEWEXPERIMENT "     // この Experiment を閉じるコマンド。
        String message
        sprintf message, "Multi-experiment process is finished. %d experiments
processed, %d skipped.", prefs.experimentsProcessed, prefs.experimentsSkipped
        DoAlert 0, message
    else
        // 次の Experiment がある場合には、所定のフォルダーにロードします。
        // 次の Experiment を読み込むために、コマンドをキューに入れます。
        PostLoadNextExperiment (nextExperimentFullPath)
    endif

endif
```



```

SavePackagePrefs (prefs)
return 0 // Igor に、デフォルトの方法でファイル进行处理するように指示します。
End

static Function PossiblySaveCurrentExperiment ()
// 現在の Experiment が保存できるかどうかを確認します。
DoIgorMenu/C "File", "Save Experiment"
if (V_flag == 0) // Experiment は修正されていません。
return 0
endif

DoAlert 2, "Save current experiment before starting?"
if (V_flag == 1) // Experiment を保存する。
SaveExperiment
endif
if (V_flag == 3) // キャンセル。
return -1
endif
return 0
End

// ユーザーに Experiment ファイルが保存されているフォルダーを選択させ、処理を開始します。
Function StartMultiExperimentProcess ()
STRUCT MultiExperimentProcessPrefs prefs

// まず、必要に応じて現在の Experiment を保存します。
if (PossiblySaveCurrentExperiment ())
return -1 // ユーザーがキャンセルした場合。
endif

// 処理する Experiment ファイルが保存されているフォルダーをユーザーに選択してもらいます。
String message = "Choose folder containing experiment files"
// フォルダーの選択を求めるダイアログを表示します。
NewPath/O/Q/M=message MultiExperimentPath
if (V_flag != 0)
return -1
// New Path ダイアログでユーザーがキャンセルした場合。
endif
PathInfo MultiExperimentPath

LoadPackagePrefs (prefs) // まだ存在しない場合は、prefs を初期化します。

prefs.processRunning = 1 // 処理が開始したことを示すフラグ。
prefs.folderPath = S_path // S_path は PathInfo によって設定されます。
prefs.index = 0
prefs.experimentsProcessed = 0
prefs.experimentsSkipped = 0

// 最初の Experiment をロードして処理を開始します。
String nextExperimentFullPath = FindNextExperiment (prefs)
PostLoadNextExperiment (nextExperimentFullPath) // 処理を開始。

SavePackagePrefs (prefs)
return 0
End

```