

# CONTENTS

---

<b>サンプルの Experiment – AutoGraph Demo</b> .....	2
クイックノート .....	2
手順（デモを再現する手順） .....	2
詳細.....	3
プロシージャの内容.....	3
IndexedFile 関数のヘルプ .....	5
LoadWave 関数のヘルプ.....	7

# サンプルの Experiment – AutoGraph Demo

## クイックノート

### メニュー File → Example Experiments → Programming → AutoGraph

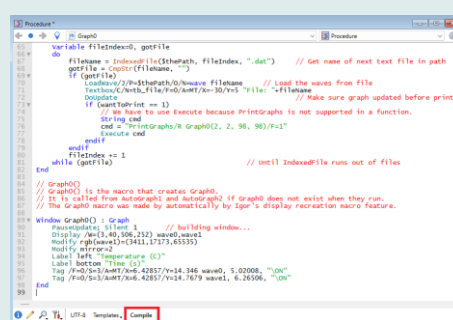
この Experiment では、データファイルを自動的に読み込み、データをグラフ化し、印刷する簡単なテクニックを紹介します。

### 手順 (デモを再現する手順)

新しい Experiment を作成したところからの手順で確認します。

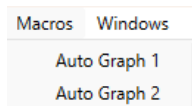
#### 1. メニュー Windows → Procedure Windows → Procedure Window を選択します。

デモ Experiment から、プロシージャ全体をコピーし、ウィンドウ下部の Compile ボタンをクリックします。



```
1 // Procedure
2
3 Variable fileIndex=0, getFile
4
5 do
6   filename = indexof(ikshpath, fileIndex, ".dat") // Get name of next text file in path
7   getFile = cursor(filename, "")
8   if (getFile)
9     loadmacro(C:\Igor\Examples\AutoGraph\AutoGraph1.mcr) // Load the waves from file
10    do
11      loadmacro(C:\Igor\Examples\AutoGraph\AutoGraph2.mcr) // Load the waves from file
12      do
13        // Make sure graph updated before print
14        // We have to use Execute because PrintGraphs is not supported in a function.
15        // Execute end
16        // PrintGraphs/R Graph0(2, 2, 98, 98)F=1"
17        // Execute end
18      end
19    end
20  while (getFile) // until Index#file runs out of files
21 end
22
23 // Graph0
24 // Graph0() is the macro that creates Graph0.
25 // It is called from AutoGraph1 and AutoGraph2 if Graph0 does not exist when they run.
26 // The Graph0 macro was made by automatically by Igor's display recreation macro feature.
27
28 Window Graph0 : Graph
29   // Building window...
30   // Create wave0, wave1
31   modify wave0=1:3411,17373,65533
32   modify wave1=1:3411,17373,65533
33   modify wave0=1:3411,17373,65533
34   modify wave1=1:3411,17373,65533
35   Label1 left "Temperature (C)"
36   Tag /F=0.5/3/AMT/A=6.42857/V=14.346 wave0, 5.02008, "Low"
37   Tag /F=0.5/3/AMT/A=6.42857/V=14.346 wave1, 6.26506, "Low"
38 end
39
40 // Compile
```

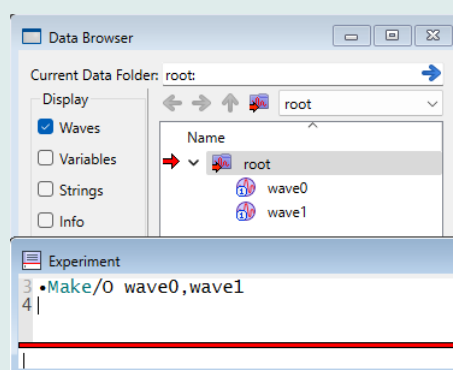
#### 2. Macros メニューに Auto Graph 1 と Auto Graph 2 という項目が追加されたことを確認してください。



#### 3. データを格納するための空のウェーブ (wave0, wave1) を作成します。

コマンドでも、メニュー Data → Make Waves を使っても構いません。

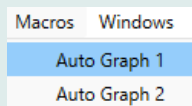
Make/O wave0, wave1



#### 4. サンプルのデータは Igor Pro フォルダ (Help → Igor Pro Folder で開くことができます) の Examples → Programming にある AutoGraph Data です。

わかりやすいようにこのフォルダを新しい Experiment と同じ場所にコピーしておきます。

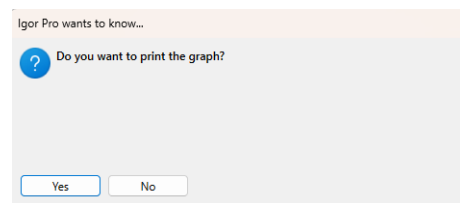
#### 5. メニュー Macros → Auto Graph 1 を選択します。



6. AutoGraph1 はテキストファイルを指定するためのダイアログを表示します。

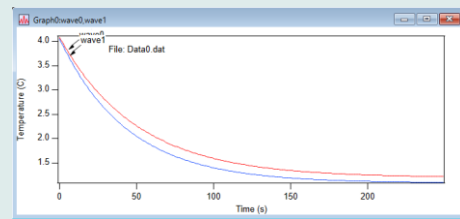
ここでは、Data0.dat を指定してみます。

テキストファイルから wave0 と wave1 にデータをロードし、その結果のグラフを印刷するかどうかを聞いてきます。



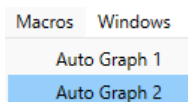
7. Yes をクリックすると、グラフが印刷されます。

画面にもグラフが表示されます。



8. Auto Graph 2 を選択したときの動きを確認します。

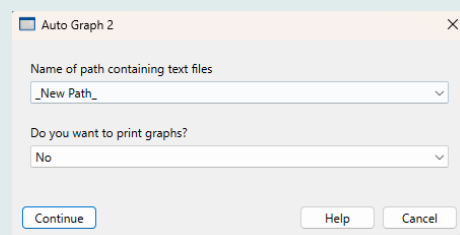
メニュー Macros → Auto Graph 2 を選択します。



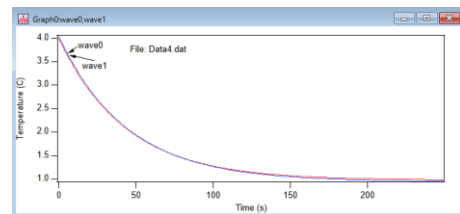
9. Auto Graph 2 では、データを読み込んでグラフ化するシンボリックパスを聞いてきます。

\_New Path\_ を選択し、Continue ボタンをクリックして、AutoGraph Data フォルダを指定します。

Auto Graph 2 は、このフォルダ内の各ファイルからデータを1つずつ読み込んでグラフ化し、印刷が指定されている場合はグラフを印刷します。



10. 画面には右のようなグラフが表示されます。



## 詳細

AutoGraph2 プロシージャは、シンプルでありながら便利なテクニックを多数含んでいます。

- シンボリックパスを使って、プロシージャをフォルダにポイントする。
- IndexedFile 関数を使って、フォルダ内の各データファイルを順に処理する。
- LoadWave コマンドの/N オプションを使って、グラフにすでに表示されているウェーブを上書きする。
- LoadWave によって設定された S\_fileName 変数を使って、読み込まれたファイルの名前を検索する。
- Textbox/C を使って、既存のテキストボックス内のテキストを変更する。

## プロシージャの内容

```
#pragma rtGlobals=1 // Use modern global access method.
```

```

Menu "Macros" // メニューに追加
    "Auto Graph 1", AutoGraph1()
    "Auto Graph 2", AutoGraph2()
End

// AutoGraph1()
// AutoGraph1 を実行するたびに、テキストファイルの入力を求め、テキストファイルから
// wave0 と wave1 にデータをロードし、その結果のグラフを印刷するかどうかを聞いてきます。

Function AutoGraph1()
    DoWindow /F Graph0 // Graph0 を前面のウィンドウにします。
    if (V_flag == 0) // Graph0 が存在しない？
        // 関数からマクロを呼び出すことはできないため、Execute を使う必要があります。
        Execute "Graph0()" // Graph0 を作成します。
    endif

    String S_fileName = "" // LoadWave によって設定されます。
    LoadWave/J/O/N=wave // ファイルのデータを wave0, wave1 . . . waveN に読み込みます。
    if (strlen(S_fileName) == 0)
        return -1 // ユーザーによるキャンセル。
    endif

    Textbox/C/N=tb_file/F=0/A=MT/X=-30/Y=5 "File: "+S_fileName
    DoAlert 1, "Do you want to print the graph?"
    if (V_Flag == 1) // DoAlert が V_flag を設定している？
        // PrintGraphs は関数でサポートされていないため、Execute を使う必要があります。
        String cmd
        cmd = "PrintGraphs/R Graph0(2, 2, 98, 98)/F=1"
        Execute cmd
    endif
End

// AutoGraph2()
// AutoGraph2 を実行するたびに、データを読み込んでグラフ化するシンボリックパスの名前を入力するよう
// 求められます。
// この Experiment では、AutoGraph Data フォルダを指す data というシンボリックパスを作成します。
// AutoGraph2 は、各ファイルからデータを 1 つずつ読み込み、グラフ化します。
// 印刷がリクエストされている場合、グラフが印刷されます。

Function AutoGraph2()
    String thePath="_New Path_"
    Prompt thePath, "Name of path containing text files", popup PathList(";", ";",
    ""))+ "_New Path_"
    Variable wantToPrint=2
    Prompt wantToPrint, "Do you want to print graphs?", popup, "Yes;No"
    DoPrompt "Auto Graph 2", thePath, wantToPrint
    if (V_flag != 0)
        return -1 // ユーザーによるキャンセル。
    endif

    if (CmpStr(thePath, "_New Path_") == 0) // ユーザーが _New Path_ を指定？
        NewPath/O data // ダイアログが表示され、パスが作成または上書きされます。
        if (V_flag != 0)
            return -1 // ユーザーによるキャンセル。
        endif
    endif
End

```

```

        endif
        thePath = "data"
    endif

    DoWindow /F Graph0 // Graph0 を前面のウィンドウにします。
    if (V_flag == 0) // Graph0 が存在しない?
        Execute "Graph0()" // Graph0 を作成します。
    endif

    String filename
    Variable fileIndex=0, gotFile
    do
        // パス内の次のテキストファイルの名前を取得します。
        fileName = IndexedFile($thePath, fileIndex, ".dat")
        gotFile = CmpStr(fileName, "")
        if (gotFile)
            // ファイルからウェーブを読み込みます。
            LoadWave/J/P=$thePath/O/N=wave fileName
            Textbox/C/N=tb_file/F=0/A=MT/X=-30/Y=5 "File: "+filename
            DoUpdate
            // 印刷の前にグラフが更新されていることを確実にします。
            if (wantToPrint == 1)
                // PrintGraphs は関数でサポートされていないため、Execute を使う必要があります。
                String cmd
                cmd = "PrintGraphs/R Graph0(2, 2, 98, 98)/F=1"
                Execute cmd
            endif
        endif
        fileIndex += 1
    while (gotFile) // ファイルがなくなるまで IndexedFile を繰り返します。

End

// Graph0()
// Graph0() は Graph0 を作成するマクロです。
// Graph0 が存在しない場合、AutoGraph1 と AutoGraph2 から呼び出されます。
// Graph0 マクロは、Igor の表示再現マクロ機能によって自動的に作成されたものです。

Window Graph0() : Graph
    PauseUpdate; Silent 1 // ウィンドウの構築中...
    Display /W=(3,40,506,252) wave0,wave1
    Modify rgb(wave1)=(3411,17173,65535)
    Modify mirror=2
    Label left "Temperature (C)"
    Label bottom "Time (s)"
    Tag /F=0/S=3/A=MT/X=6.42857/Y=14.346 wave0, 5.02008, "¥ON"
    Tag /F=0/S=3/A=MT/X=6.42857/Y=14.7679 wave1, 6.26506, "¥ON"

End

```

## IndexedFile 関数のヘルプ

**IndexedFile**(pathName, index, fileTypeOrExtStr [, creatorStr, separatorStr])

index がゼロ以上の場合、IndexedFile は、pathName で指定されたフォルダー内のファイルで、fileTypeOrExtStr で指定されたファイルタイプまたは拡張子に一致するインデックスの名前を含む文字列を返します。

index が -1 の場合、IndexedFile は、一致するすべてのファイルの文字列リストをセミコロンで区切って返します。

または、separatorStr が指定されている場合は、separatorStr で区切って返します。

IndexedFile は、一致するファイルがない場合は空文字列 ("") を返します。

## パラメーター

pathName は Igor のシンボリックパスの名前です。  
文字列ではありません。

インデックスは通常ゼロから始まります。

ただし、インデックスが -1 の場合、IndexedFile は、指定されたシンボリックパスに関連付けられたフォルダー内の、fileTypeOrExtStr に一致するすべてのファイル名のセミコロン区切りのリストを含む文字列を返します。

fileTypeOrExtStr は次のいずれかです：

- “.txt”、“.bwav”、“.c” のように、“.” で始まる文字列。  
ファイル名拡張子が一致するファイルのみがインデックス化されます。  
ファイル名が「.」で終わる場合、例えば「myFileNameEndsWithThisDot」のように、ファイル名をインデックス付けするには、fileTypeOrExtStr を“.” に設定します。
- “TEXT” や “IGBW” など、正確に 4 つの ASCII 文字を含む文字列。  
指定された MacOS ファイルタイプのファイルのみがインデックス化されます。  
fileTypeOrExtStr が “????” の場合、すべてのタイプのファイルがインデックス化されます。  
Windows では、Igor は拡張子が “.txt” のファイルを TEXT タイプと見なします。  
ファイルタイプを指定すると、他の拡張子についても同様のマッピングが行われます。

新しいバージョンでは creatorStr は廃止されているため、区切り文字を指定する必要がある場合は、creatorStr に “” を使ってください。

separatorStr は、index が -1 の場合に指定するオプションの文字列引数です。  
separatorStr を省略すると、ファイル名は「;」で区切られます。  
このパラメーターは Igor 9.01 で追加されました。

## IndexedFile が返すファイルの順序

IndexedFile によって返されるファイルの順序は、オペレーティングシステムによって決定されます。  
順序が重要である場合は、ファイル名を明示的にソートする必要があります。

例えば、“File1.txt”、“File2.txt”、“File10.txt” という名前のファイルがあるとします。

アルファベット順では、“File1.txt;File10.txt;File2.txt;” となります。

本当に必要なのは、アルファベット順と数字順の組み合わせで、“File1.txt;File2.txt;File10.txt;” と表示されるソート機能であることが多いと思います。

これを実行する関数は次のとおりです：

```
Function DemoIndexedFile()  
    String pathName = "Igor"           // "Igor Pro Folder" を参照。  
  
    // フォルダー内のコマで分割されたすべてのファイルのリストを取得。  
    String list = IndexedFile($pathName, -1, ".txt")  
  
    // アルファベットと数字を組み合わせたソート。  
    list = SortList(list, ";", 16)
```

```

// リストを処理。
Variable numItems = ItemsInList(list)
Variable i
for(i=0; i<numItems; i+=1)
    String fileName = StringFromList(i, list)
    Print i, filename
endfor
End

```

## LoadWave 関数のヘルプ

**LoadWave** [flags] *fileNameStr*

LoadWave コマンドは、指定された Igor バイナリ、Igor テキスト、区切り記号付きテキスト、固定フィールドテキスト、または一般テキストファイルからデータをウェーブに読み込みます。

LoadWave は、区切り記号付きテキスト、固定フィールドテキスト、一般テキストファイルから 1 次元および 2 次元データを読み込むことができます。

また、Igor バイナリファイルおよび Igor テキストファイルから任意の次元のデータを読み込むこともできます。

### パラメーター

ロードするファイルは、*fileNameStr* と */P=pathname* で指定され、*pathName* は Igor のシンボリックパスの名前です。

*fileNameStr* は、ファイルへのフルパス（この場合は */P* は不要）、*pathName* に関連付けられたフォルダーに対する相対的な部分パス、または *pathName* に関連付けられたフォルダー内のファイル名とすることができます。

LoadWave が *fileNameStr* と *pathname* からファイルの場所を特定できない場合、ファイルを指定するためのダイアログが表示されます。

*fileNameStr* にフルパスまたはパス名を使う場合、パスの作成方法の詳細は、ヘルプ Path Separators (Platform-Related Issues.ihf) を参照してください。

*fileNameStr* が "Clipboard" で、*/P=pathName* が省略されている場合、LoadWave はファイルではなくクリップボードからデータを取得します。

これは、バイナリのロードに対しては実装されていません。

*fileNameStr* が省略されたり、""（空文字列）であったり、または */I* フラグが使われた場合、LoadWave はファイルを開くためのダイアログを表示し、そこから読み込むファイルを選択することができます。

### フラグ

*/A* "Auto-name and go" オプション (*/G*, */F*, */J* と使う) です。

これにより、通常、ウェーブ名を入力するダイアログをスキップできます。

代わりに、*wave0*、*wave1* という形式の名前を自動的に割り当て、すでに使われている名前は選択しません。

*/W* フラグとともに使用すると、名前を自動的に割り当てる代わりにファイルからウェーブ名を読み込みます。

これに対して、*/A* はウェーブ名ダイアログをスキップするだけです。

*/B* フラグは、*/A* で指定された名前を上書きすることもできます。

詳細は、ヘルプ LoadWave Generation of Wave Names (Importing and Exporting Data.ihf) を参照してください。

*/A=baseName* /A と同じですが、baseName0、baseName1 という形式のウェーブ名を自動的に割り当てます。

*/B=columnInfoStr*

ファイル内の列の名前、フォーマット、数値タイプ、フィールド幅を指定します。  
詳細は、ヘルプ Specifying Characteristics of Individual Columns (Importing and Exporting Data.ihf) を参照してください。

*/C* これは、Igor によって生成された Experiment の再現コマンドで、ウェーブの読み込み時にエラーが発生した場合に、再現を強制的に継続させるために使用されます。

*/D* 倍精度ウェーブを作成します (*/G*、*/F*、*/J* と使う)。  
*/B* フラグは、*/D* フラグで指定された数値の精度を上書きすることができます。

*/E=editCmd* *editCmd* = 1 : 読み込んだウェーブを含む新しいテーブルを作成します。  
*editCmd* = 2 : 読み込んだウェーブを先頭のテーブルに追加します。  
テーブルが存在しない場合は、新しいテーブルが作成されます。  
*editCmd* = 0 : */E* が指定されていないかのように動作します  
(読み込まれたウェーブはどのテーブルにも配置されない)。

*/ENCG=textEncoding*

*/ENCG={textEncoding, tecOptions}*

読み込まれるプレーンテキストファイルのテキストエンコーディングを指定します。

このフラグは、Igor Pro 7.0 で追加されました。

Igor バイナリウェーブファイルをロードする場合には、このフラグは無視されます。

*textEncoding* の許容値の一覧については、ヘルプ Text Encoding Names and Codes (Text Encoding.ihf) を参照してください。

ほとんどの用途では、*tecOptions* のデフォルト値である 3 で問題ありません。

また、*/ENCG={textEncoding, tecOptions}* の代わりに、*/ENCG=textEncoding* を使うこともできます。

*tecOptions* は、テキストエンコーディングの変換をコントロールするオプションのビット単位パラメーターです。

ビット設定の詳細については、ヘルプ Setting Bit Parameters (Commands.ihf) を参照してください。

*tecOptions* は次のように定義されます :

Bit 0: クリアされている場合、Null バイトの存在により、LoadWave はすべてのバイト指向のテキストエンコーディングにおいてテキストを無効とみなします。  
これにより、LoadWave は通常 Null バイトを含む UTF-16 および UTF-32 テキストを識別しやすくなります。  
設定されている場合 (デフォルト)、バイト指向のテキストエンコーディングでは Null バイトが許可されます。

Bit 1: クリアされている場合、指定されたテキストのエンコーディングが UTF-8 の時、LoadWave はテキストを検証しません。  
設定されている場合 (デフォルト)、LoadWave はテキストエンコーディングが UTF-8 の場合でもテキストの検証を行います。

Bit 2: クリアされている場合 (デフォルト) は、LoadWave は指定されたテキストエンコーディングでテキストを検証します。



ただし、Bit 1 がクリアされている場合は、UTF-8 の検証はスキップされま  
す。

設定されている場合、LoadWave は指定のテキストエンコーディングでテキス  
トが有効であると想定し、検証を行いません。

このビットを設定すると、LoadWave は若干処理が早くなりますが、通常は無  
視できる程度です。

Bit 3: クリアされている場合 (デフォルト)、指定されたテキストエンコードがフ  
ァイル内のテキストに対して有効でない時、LoadWave は Choose Text  
Encoding ダイアログを表示します。

設定されている場合、指定されたテキストエンコーディングでテキストが有効  
でない時、LoadWave は Choose Text Encoding ダイアログを表示せず、代  
わりにエラーを返します。

自動化したプロシージャを実行していて、ダイアログによって中断されたくな  
い場合は、このビットを設定します。

詳細は、ヘルプ LoadWave Text Encoding Issues (Importing and  
Exporting Data.ihf) を参照してください。

*/F= {numColumns, defaultFieldWidth, flags}*

固定フィールドファイル形式を使用していることを示します。

ほとんどの FORTRAN プログラムは、この形式でファイルを生成します。

numColumns は、ファイル内のデータ列の数です。

defaultFieldWidth は、各列のデフォルトのバイト数です。

列のバイト数がすべて同じでない場合は、LoadWave に詳細情報を提供するために/B フ  
ラグを使う必要があります。

flags は、テキストの値への変換をコントロールするビット単位のパラメーターです。  
ビットは次のように定義されています：

Bit 0: 設定されている場合、数字 "9" のみで構成されるフィールドは空白とみなされ  
ます。

先頭の "+" または "-" を除き、すべて "9" の数字で構成されるフィールド  
も空白とみなされます。

その他のビットは予約済みであり、0 を指定してください。

*/G* ファイルが一般的なテキスト形式を使っていることを示します。

*/H* 現在の Experiment にウェーブをロードし、ウェーブとファイル間の接続を切断しま  
す。  
Experiment が保存されると、ウェーブのコピーも Experiment の一部として保存され  
ます。  
つまり、パッキングされた Experiment ファイル (.pxp) に保存されるということだ  
す。  
展開されていない Experiment の場合、Experiment のホームフォルダーに保存されま  
す。

ヘルプ Sharing Versus Copying Igor Binary Wave Files (Importing and  
Exporting Data.ihf) も参照してください。

*/I* /P と fileNameStr でファイルが完全に指定されている場合でも、LoadWave に Open  
File ダイアログを表示させます。

*/J* ファイルが区切り記号付きテキスト形式を使っていることを示します。

- /K=k ファイル内の列が数値かテキストかをどのように決定するかをコントロールします（区切り記号付きテキストファイルおよび固定フィールドテキストファイルのみ）。
- k=0: カラムの性質を自動的に推論します。  
k=1: すべての列を数値として扱います。  
k=2: すべての列をテキストとして扱います。
- LoadWave コマンドのデフォルトは /K=1 で、これは Igor がすべての列を数値として扱うことを意味します。  
テキストの列をテキストウェーブに読み込む場合は、/K=0 を使います。  
/K=2 は、テキスト処理アプリケーションで使用される場合があります。
- /L={nameLine, firstLine, numLines, firstColumn, numColumns}  
区切り記号付きテキスト、固定フィールドテキスト、および一般テキストファイルのみに影響します (/J、/F、/G)。  
/L はどのようなロード形式でも受け付けられますが、Igor バイナリおよび Igor テキストのロードでは無視されます。  
行番号と列番号は 0 から始まります。
- nameLine は、列名を含む行の番号です。  
一般的なテキスト読み込みの場合、0 は自動を意味します。
- firstLine は、ウェーブに読み込む最初の行の番号です。  
一般的なテキスト読み込みの場合、0 は自動を意味します。
- numLines は、データとして処理されるべき行の数です。  
0 は自動を意味し、ファイルの終わりまたは一般テキストファイル内のデータブロックの終わりまで読み込みます。  
numLines パラメーターは、非常に大きなファイルの読み込みをより効率的に行うためにも使用できます。  
詳細はヘルプ Loading Very Large Files (Importing and Exporting Data.ihf) を参照してください。
- firstColumn は、ウェーブに読み込む最初の列の番号です。  
これは、列をスキップする時に便利です。
- numColumns は、ウェーブに読み込む列の数です。  
0 は自動を意味し、すべての列が読み込まれます。
- /M 行列ウェーブとしてデータを読み込みます。  
/M が使われた場合、/W フラグ（ウェーブ名を読み込みます）は無視され、代わりに /U フラグに従います。  
/B フラグを使って特定のウェーブ名を指定しない限り、ウェーブは自動的に命名されます。  
ウェーブの種類（数値またはテキスト）は、/K フラグまたは /B フラグを使って上書きしない限り、最初に読み込まれた列の種類を Igor が判断した結果によって決定されます。
- 詳細はヘルプ The Load Waves Dialog for Delimited Text - 2D (Importing and Exporting Data.ihf) を参照してください。
- /N /A と同じですが、使われていない名前を選択するのではなく、既存のウェーブを上書きします。
- 詳細はヘルプ LoadWave Generation of Wave Names (Importing and Exporting Data.ihf) を参照してください。

`/N=baseName` `/N`と同じですが、Igor が `baseName0`, `baseName1` という形式のウェーブ名を自動的に割り当てる点が異なります。

`/NAME={namePrefix, nameSuffix, nameOptions}`

`/NAME` は Igor Pro 9.0 で追加されました。

`/NAME` フラグは、`LoadWave` で読み込まれたウェーブの名前に影響を与える他のフラグ (`/A`、`/N`、`/W`、`/B`) を無効化または組み合わせます。

これは、一般テキストの読み込み (`/G`)、区切り記号付きテキストの読み込み (`/J`)、固定フィールドテキストの読み込み (`/F`) に適用され、Igor バイナリファイルや Igor テキストファイルの読み込みには適用されません。

`/NAME` フラグは、主にファイル名をウェーブ名に簡単に組み込むためのものです。

例えば、`File.txt` という名前のファイルから1つのウェーブをロードする場合、1つのウェーブを `File` という名前で作成し、`File.txt` という名前のファイルから3つのウェーブをロードする場合、`File0`、`File1`、`File2` という名前で3つのウェーブを作成することができます。

`namePrefix` は "" または、最終的なウェーブ名の先頭に追加するテキストです。

`nameSuffix` は "" または、最終的なウェーブ名の後ろに追加するテキストです。

`namePrefix` と `nameSuffix` が両方とも "" の場合、`/NAME` フラグが完全に省略されたのと同じになります。

最終的なウェーブ名の生成において、`LoadWave` は、`namePrefix` または `nameSuffix` の `":filename:"` を、ロードされるファイルの名前 (拡張子なし) に置き換えます。

`nameOptions` はビット単位のパラメーターです。

ビットは以下のように定義されています。

- Bit 0 ( $2^0$ ) : 通常の名前を含めます。  
設定されている場合、「通常」のウェーブ名 (`/NAME` が省略された場合に用いられる名前) が、`namePrefix` と `nameSuffix` の間に含まれます。
- Bit 1 ( $2^1$ ) : 1つのウェーブをロードする時に、サフィックス番号を含めます。Bit 3 で無効化されていない限り、サフィックス番号が追加されます。
- Bit 2 ( $2^2$ ) : 複数のウェーブをロードする時に、サフィックス番号を含めます。Bit 3 で無効化されていない限り、サフィックス番号が追加されます。
- Bit 3 ( $2^3$ ) : サフィックス番号は、名前の競合がある場合のみ含めます。  
設定されていて名前の競合がない場合は、Bit 1 と Bit 2 が上書きされ、サフィックス番号の追加が防止されます。
- Bit 4 ( $2^4$ ) : 固有の名前を作成するために、サフィックス番号を選択します。  
設定されている場合、`LoadWave` は既存のウェーブとの競合を避けるためにサフィックス番号を選択します。  
クリアされている場合、サフィックス番号は0から始まり、ロードされたウェーブごとに1ずつ増加します。
- Bit 5 ( $2^5$ ) : 自由な名前を許可します (推奨しません)。

その他のビットはすべて予約済みであり、クリアする必要があります。

詳細と例はヘルプ `LoadWave Generation of Wave Names (Importing and Exporting Data.ihf)` を参照してください。

/O 名前が競合する場合は、既存のウェーブを上書きします。

/P=*pathName* ファイル名 (*fileNameStr*) を検索するフォルダーを指定します。  
パス名 (*pathName*) は、既存のシンボリックパスの名前です。

/Q 履歴エリアに表示される通常メッセージを非表示にします。

/R ={*languageName, yearFormat, monthFormat, dayOfMonthFormat, dayOfWeekFormat, layoutStr, pivotYear*}

ファイル内の日付の表示形式をカスタマイズします。

/R フラグが使用された場合、/V フラグの一部である日付設定が上書きされます。

*languageName* は、アルファベット表記の日付の構成要素、すなわち月と曜日で使用される言語をコントロールします。

*languageName* は以下のいずれかです。

Default / Chinese / ChineseSimplified / Danish / Dutch / English  
Finnish / French / German / Italian / Japanese / Korean  
Norwegian / Portuguese / Russian / Spanish / Swedish

デフォルトとは、Macintosh ではシステム言語、Windows ではユーザーのデフォルト言語を意味します。

*yearFormat* は以下のいずれかのコードです。

1 : 2桁の年号  
2 : 4桁の年号

*monthFormat* は以下のいずれかのコードです。

1 : 数字、先頭ゼロなし  
2 : 数字、先頭ゼロあり  
3 : 省略形のアルファベット (例 : Jan)  
4 : 完全なアルファベット (例 : January)

*dayOfMonthFormat* は以下のいずれかのコードです。

1 : 数字、先頭ゼロなし  
2 : 数字、先頭ゼロあり

*dayOfWeekFormat* は以下のいずれかのコードです。

1 : 省略形のアルファベット (例 : Mon)  
2 : 完全なアルファベット (例 : Monday)

*layoutStr* は、日付に表示されるコンポーネントと順序、使われる区切り記号を記述します。

*layoutStr* は以下のように構成されます (ただし、改行は使いません)。

```
"<component keyword><separator>  
<component keyword><separator>  
<component keyword><separator>  
<component keyword>"
```

ここで、<component keyword> は以下のいずれかです。

Year / Month / DayOfMonth / DayOfWeek

そして、<separator>は0から15バイトの文字列です。

最後に、使われていないレイアウト文字列の部分は省略しなければなりません。

layoutStr には余分なスペースを挿入することはできません。  
各セパレータは 15 バイト以内でなければなりません。  
コンポーネントは 1 度しか使用できません。  
一部のコンポーネントは 0 回使用できるものもあります。

pivotYear は、LoadWave が 2 桁の年をどのように解釈するかを決定します。  
年が 2 桁の yy で指定され、pivotYear より小さい場合、日付は 20yy とみなされます。

2 桁の年が pivotYear 以上の場合、年は 19yy と見なされます。  
pivotYear は 4 から 40 の間である必要があります。

日付のフォーマットに関する詳細はヘルプ Loading Custom Date Formats  
(Importing and Exporting Data.ihf) を参照してください。

/T ファイルが Igor のテキスト形式を使っていることを示しています。

LoadWave は一般的にスレッドセーフですが、Igor コマンド（例：「X  
<command>」）を含む Igor テキストファイルをロードする際にはスレッドセーフで  
はありません。

/U={readRowLabels, rowPositionAction, readColLabels, colPositionAction}

これらのパラメーターは、区切り記号付きテキスト (/J) または固定フィールドテキス  
ト (/F) ファイルからの行列 (/M) の読み込みに影響します。  
読み込みの種類に関わらず、これらのパラメーターは受け付けられますが、適用できな  
い場合は無視されます。

readRowLabels がゼロ以外の場合、ファイルの最初の列のデータを行列のウェーブの  
行ラベルとして読み取ります。

rowPositionAction は以下の値のいずれかを持ちます。

- 0 : ファイルには行位置の列がありません。
- 1 : 行位置の列を使って、行列ウェーブの行のスケーリングを設定します。
- 2 : 行の位置の列の値を含む 1D ウェーブを作成します。  
1D ウェーブの名前は、行列ウェーブと同じですが、プリフィックス "RP" が  
付きます。

readColLabels と colPositionAction パラメーターは同様の意味を持ちます。

列位置ウェーブに使用されるプリフィックスは "CP\_" です。

/V={delimsStr, skipCharsStr, numConversionFlags, loadFlags}

これらのパラメーターは、区切り記号付きテキスト (/J) および固定フィールドテキス  
ト (/F) のデータと列名の読み込みに影響します。  
一般テキスト (/G) の読み込みには影響しません。  
これらは、ロードの種類に関係なく受け入れられますが、適用されない場合は無視され  
ます。  
これらのパラメーターが必要になることはほとんどありません。

delimsStr は、区切り記号付きファイル読み込みの区切り記号として機能する文字を含  
む文字列式です（一般テキスト (/G) としてデータを読み込む場合、区切り記号は常に  
タブ、カンマ、スペースです）。

デフォルトはタブとカンマの "%t," です。

スペース文字を区切り文字として指定することはできますが、delimsStr に含まれる他  
の区切り文字よりも常に優先順位が低くなります。

優先順位が低いということは、テキスト行にスペース文字以外の区切り記号が含まれて  
いる場合、スペース文字ではなくその区切り記号が使用されることを意味します。

skipCharsStr は、常にゴミとして扱われ、数字の前に現れた場合はスキップされる文字  
を含む文字列式です。

デフォルトはスペースとドル記号 "\$" です。  
このパラメーターが必要になることはほとんどないはずです。

numConversionFlags は、テキストの数値への変換をコントロールするビット単位のパラメーターです。

ビットは以下のように定義されています。

- Bit 0 (2<sup>0</sup>) : 設定されている場合 : 日付は「dd/mm/yy」形式  
設定されていない場合 : 日付は「mm/dd/yy」形式
- Bit 1 (2<sup>1</sup>) : 設定されている場合 : 小数点記号はカンマ  
設定されていない場合 : ピリオド
- Bit 2 (2<sup>2</sup>) : 設定されている場合 : 区切り記号付きテキストのみを読み込む時  
(LoadWave/J)、数値の桁区切り記号は無視されます。

1,234 では、3 桁ごとの区切り記号はカンマです。  
カンマが小数点記号である場合は、1.234 ではドットが 3 桁ごとの区切り記号となります。

ほとんどの数値データファイルでは、桁区切り記号を使っていないため、その検索により読み込みが遅くなるので、通常は 0 にしておくべきです。

このビットは、delimsStr で指定されたように、桁区切り文字（例えばカンマ）が区切り文字の文字でもある場合には影響を与えません。

その他のビットはすべて予約済みであり、クリアする必要があります。

日付形式を指定するために /R フラグが使用された場合、ビット 0 の設定は無効になります。

その他のビットはすべて予約済みであり、0 を使う必要があります。  
両方のビットを設定するには、numConversionFlags=3 を使います。  
日付形式を指定するために /R フラグが使われた場合、ビット 0 の設定は無効になります。

loadFlags は、全体的な負荷を制御するビット単位のパラメーターです。  
ビットは以下のように定義されています。

- Bit 0 (2<sup>0</sup>) : 設定されている場合、列の末尾の空白を無視します。  
ファイルに長さが異なる列が含まれている場合に適切です。  
loadFlags=1 を設定すると、Bit 0 が設定されます。
- Bit 1 (2<sup>1</sup>) : 設定されている場合、/W フラグが指定され、列ラベルを含む行が 1 つ以上のスペース文字で始まっている場合、スペースは空白の列名であるとみなされます。  
結果として得られる列は "Blank" と名付けられます。  
スペース区切りファイルで、列ラベルを含む行とデータを含む行の両方が先頭スペースで始まる場合にこれを使用します。  
loadFlags=2 を設定して Bit 1 を設定します。
- Bit 2 (2<sup>2</sup>) : 設定されている場合 : データの行数の事前カウントを無効にします。
- Bit 3 (2<sup>3</sup>) : 設定されている場合 : テキストコラム内のバックスラッシュ文字のエスケープ解除を無効にします。
- Bit 4 (2<sup>4</sup>) : 設定されている場合 : 引用符で囲まれた文字列のサポートを無効にします。

このビットを設定すると、Igor7 との互換性が向上します。  
これは区切り記号付きテキストのみに適用され、ほとんど必要ないはず  
です。

その他のビットはすべて予約済みであり、0 を使用する必要があります。

/W ファイル内のウェーブ名を検索します (/G、/F、/J と使う)。

LoadWave は、列名を整理して、標準的なウェーブ名を作成します。

/W/A を使うと、ファイルからウェーブ名を読み込み、通常のウェーブ名ダイアログを  
表示せずに読み込みを続行します。

## 詳細

/G、/F、/J、または /T フラグがない場合、LoadWave は Igor のバイナリウェーブファイルを読み込みま  
す。

/P=pathName を使う場合は、NewPath で作成された Igor のシンボリックパスの名前であることに注意  
してください。

"hd:Folder1:" や "C:¥Folder1" のようなファイルシステム上のパスではありません。

一般的なテキストファイルを読み込む際、区切り文字は常にタブ、カンマ、スペースです。

## 出力変数

LoadWave は以下の変数を設定します。

V_flag	読み込まれたウェーブの数。
S_fileName	読み込まれているファイルの名前。
S_path	ファイルを含むフォルダーへのファイルシステムのパス。
S_waveNames	読み込まれたウェーブの名前のセミコロン区切りのリスト。

S\_path は、Windows 上でも Macintosh のパス構文 (例 : "hd:FolderA:FolderB:") を使います。  
末尾にコロンが付きます。

LoadWave がクリップボードから読み込んでいる場合、S\_path は "" に設定されます。

LoadWave が Open File ダイアログを表示し、ユーザーがキャンセルすると、V\_flag は 0 に、  
S\_fileName は "" に設定されます。