CONTENTS

ビジュアルヘルプ – デバッグ

デバッグの方法

Igor には、プロシージャのデバッグを行うための２つのテクニックがあります。

- Print コマンドで出力させる方法
- シンボリックなデバッガーを使う方法

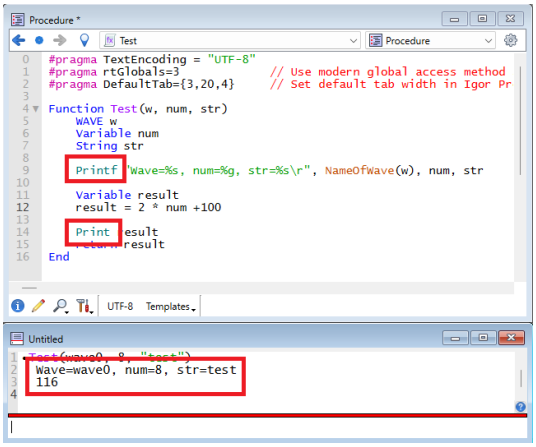
ほとんどの状況では、シンボリックデバッガーが最も効果的なツールです。
場合によっては、戦略的に配置された Print コマンドで十分です。

Print コマンドで出力させてデバッグ

このテクニックは、プロシージャの特定のポイントに Print 文を配置し、Igor の履歴エリアにデバッグメッセージを表示するものです。
下の例では、Printf を使ってパラメーターの値を関数に表示し、その後、Print を使って関数の結果を表示します。

プロシージャウィンドウに次のプログラムを記述します。

```
Function Test(w, num, str)
        WAVE w
        Variable num
        String str

        Printf "Wave=%s, num=%g, str=%s¥r",
NameOfWave(w), num, str

        // ここでは簡単な式としました。
        Variable result
        result = 2 * num +100

        Print result
        return result
End
```



コマンドウィンドウで次を実行すると履歴エリアに出力されます。

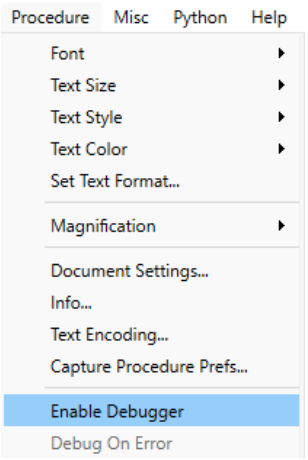
```
Test(wave0, 8, "test")
```

Printf コマンドの詳細については、ヘルプ Creating Formatted Text を参照してください。

デバッガー

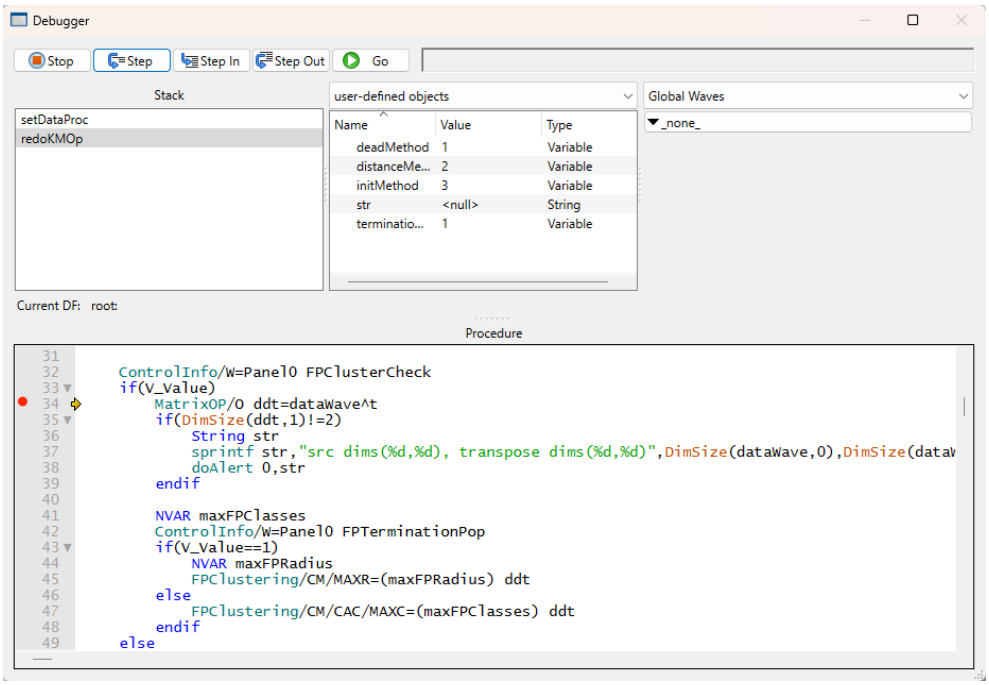
プロシージャが期待通りの結果を生成しない場合、Igor に組み込まれているデバッガーを使って、コードの行を１つずつ確認しながら、マクロやユーザー定義関数の実行を観察することができます。

デバッガーは通常無効になっています。
Procedure メニュー（プロシージャウィンドウが表示されているときに現れる）または任意のプロシージャウィンドウで右クリックして表示されるコンテキストメニューで、Enable Debugger を選択します。



Igor は、以下のいずれかのイベントが発生すると、デバッガーウィンドウを表示します。

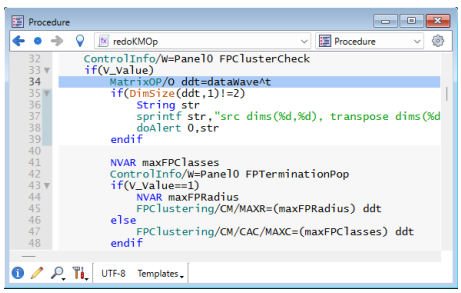
1. 事前に設定したブレークポイントがヒットした
2. エラーが発生し、その種のエラーのデバッグを有効にしている場合
3. エラーダイアログが表示され、Debug ボタンをクリックした
4. Debugger コマンドが実行された

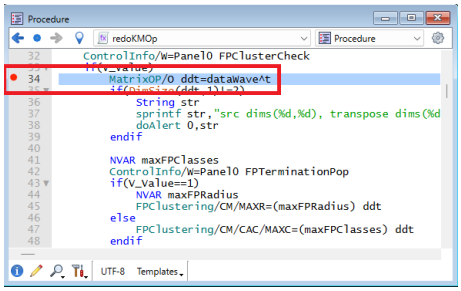


注記： デバッガーはスレッドセーフなコードでは使用できません。
詳細は、「スレッドセーフなコードのデバッグ」のセクションを参照してください。

ブレークポイントの設定

特定のルーチンが実行されている様子を観察したい場合は、デバッガーを表示させたい行にブレークポイントを設定します。
これを行うには、ルーチンを含むプロシージャウィンドウを開き、左側の「ブレークポイントマージン」をクリックします。
ブレークポイントのマージンは、デバッガーが有効になっている場合にのみ表示されます。
下の図は、デバッガーが無効（左）および有効（右）のプロシージャウィンドウを示しています。





赤い点が、設定したブレークポイントを示しています。

ブレークポイントが設定されたコード行が実行されようとすると、Igor がデバッガーウィンドウを表示します。

赤い点を再度クリックすると、ブレークポイントがクリアされます。
右クリックしてポップアップメニューを使うと、すべてのブレークポイントをクリアしたり、現在選択されているプロシージャウィンドウの行のブレークポイントを無効にしたりすることができます。

エラー発生時のデバッグ

エラーが発生したときにデバッガーウィンドウを自動的に開くよう Igor に指示することができます。
いくつかのエラーのカテゴリから選択できます。

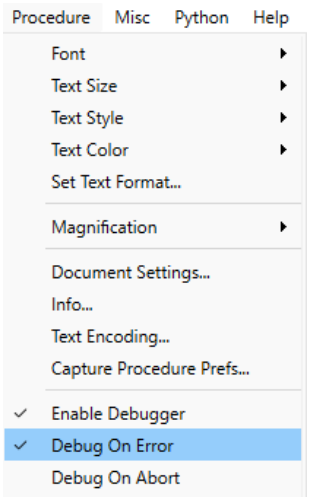
Debug On Error NVAR、SVAR、または WAVE 参照のエラーを除く、すべての実行時エラーです。

Debug On User Abort ステータスバーの Abort ボタンは Debug に名称変更されました。
Debug をクリックすると、現在実行中の行が終了した後にデバッガーに入ります。

NVAR SVAR WAVE Checking NVAR、SVAR、または WAVE 参照の失敗です。

Igor のプログラマーは、エラーに関するタイムリーな情報を取得するために、これらのオプションのすべてをオンにすることをお勧めします。

Procedure メニューまたはコンテキストメニューを使って、これらのエラーカテゴリを有効または無効にします。
選択したエラーが発生した場合、Igor はステータスエリアにエラーメッセージを表示したデバッガーを表示します。



プロシージャウィンドウで次をコンパイルします。

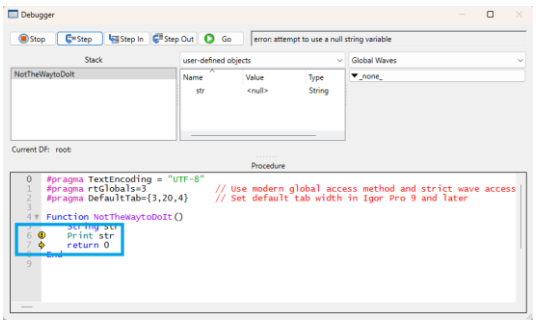
```
Function NotTheWaytoDoIt()
	String str
	Print str
	return 0
End
```

Procedure メニューで
	Enable Debugger
	Debug On Error
をオンにします。

コマンドウィンドウで次を実行します。

```
NotTheWaytoDoIt()
```

エラーメッセージは、この例では「Print str」コマンドと示された黄色い丸いアイコンで示されたコマンドによって生成されました。



デバッガーを回避するためのコード記述

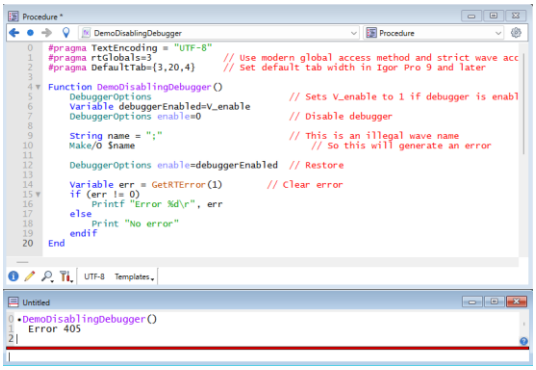
エラーを引き起こす可能性があることを承知で何かを行い、デバッガーを中断することなく自分でエラーを処理したいと思うことがあります。
そのようなケースの１つは、存在するかどうかわからないウェーブや変数にアクセスしようとすることです。
デバッガーに入らずに、その存在をテストしたいような場合です。

/Z フラグを使うと、NVAR、SVAR、またはウェーブ参照が失敗した時に Debug on Error 機能が作動するのを防ぐことができます。
例えば、

```
WAVE/Z w = <path to possibly missing wave>
if (WaveExists(w))
	<do something with w>
endif
```

エラーが発生する可能性があり、それを自分で処理したいその他のケースでは、デバッガーを一時的に無効化し、GetRTError を使ってエラーを取得、クリアする必要があります。
例えば、

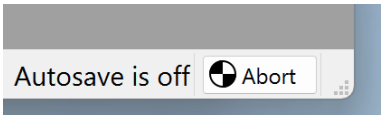
```
Function DemoDisablingDebugger()
	// デバッガーが有効なとき、V_enable を 1 に設定
	DebuggerOptions
	Variable debuggerEnabled=V_enable
	// デバッガーを無効にする
	DebuggerOptions enable=0

	// 不正なウェーブ名を指定
	String name = ";"
	// エラーが発生する文
	Make/O $name

	// リストア（デバッガーを有効に）
	DebuggerOptions enable=debuggerEnabled

	// エラーをクリア
	Variable err = GetRTError(1)
	if (err != 0)
		Printf "Error %d¥r", err
	else
		Print "No error"
	endif
End
```



コマンドウィンドウで
`DemoDisablingDebugger()`
を実行すると
`Error 405`
が返される

Abort 時のデバッグ

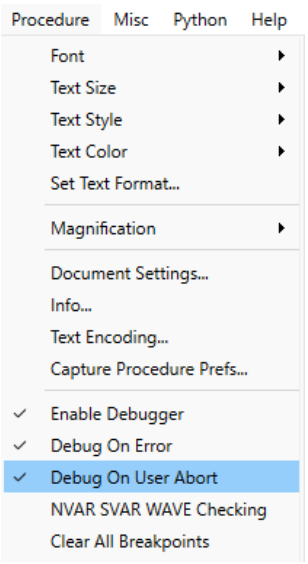
Igor の MDI フレームの右下隅にある中止ボタンを押すことで、ユーザーコードの実行を停止できます：



これは、予想よりはるかに長い時間を要しているコードを停止するのに役立ちます。
例えば、長時間かかるウェーブの代入文を中止できます。

通常、Abort ボタンは単にユーザーコードの実行を停止し、通常の対話操作にコントロールを戻します。
ユーザーが実行を中止した時点で Igor が何をしていたかは、デバッガーでコードをゆっくりステップ実行するなどして、手動で調べる必要があります。

コードが実行時に何を行っているかを判断するより有用な方法は、Debug on User Abort を有効にすることです。
この機能を有効または無効にするには、Procedure→Debug On User Abort を選択するか、Procedure ウィンドウ内で右クリックし、ポップアップメニューから Debug On User Abort を選択します。



これは、Abort ボタンを押した時に単に実行を停止するのではなく、代わりに実行中のコードの位置でデバッガーウィンドウを自動的に開くよう Igor に指示します。
これにより実行が一時停止され、デバッガーが開くため、何が起きていたのかを確認できます。

Debug On User Abort が有効な場合、ステータスバーの Abort ボタンは Abort ではなく Debug と表示されます。
デバッグをクリックすると、現在実行中の行が終了した後にデバッガーに入ります。

Debug on User Abort が有効な場合、コマンドが途中で中断されたときにデバッガーに入るのを避けるため、Debug ボタンをクリックすると、Igor は現在のコマンドが完了するまで待機してからデバッガーに入ります。
非常に長い操作や代入文が実行中の場合、デバッガーが起動するまで長時間待つ必要があるかもしれません。

現在実行中のコマンドを中断し、デバッグボタンをクリックする代わりにユーザー中断キーの組み合わせを使うことで、より迅速にデバッガーに入ることができます。
これにより、代入が部分的にしか完了していない時点でデバッガーに入ることができます。

Debug On User Abort は、Abort、AbortOnRTE、または AbortOnValue コマンドを使ったプログラムによる中断には影響しません。

一部の Igor コードはデバッグできません。
これには、ほとんどのスレッドセーフ関数コード、非公開コード、および独立モジュールコードが含まれます（ヘルプ SetIgorOption IndependentModuleDev=1 を参照）。
デバッグできないコードの実行中に Debug ボタンをクリックするか、Abort キーの組み合わせを使うと、ユーザーによる中断時の Debug On Abort が設定されていない場合と同様に、コードの実行が停止します。
例えば、次のコード：

```
ThreadSafe Function Mistake(Variable loopCount)
	// compute 2^loopCount (badly)
	Variable n=1/2
	do
		n *= 2
		// forgot to decrement loopCount
	while (loopCount > 0)
	return n
End

Function MultithreadedCaller()
	Make/O/N=10 outWave
	MultiThread outWave = Mistake(10)
	Print "back from threads"
End
```

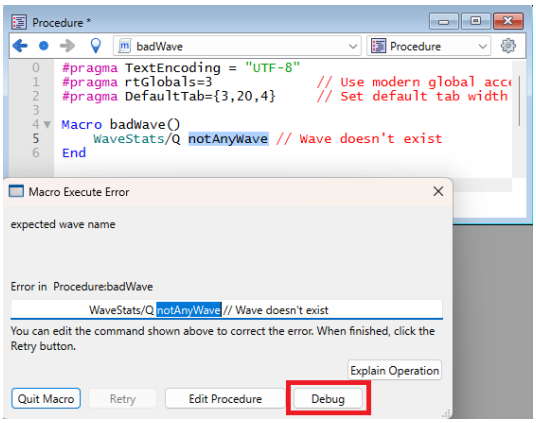
は、Debug on User Abort が有効な状態で MultithreadedCaller が呼び出されると、容易にデバッグできません。

ステータスバーの Debug ボタンをクリックすると、Igor はマルチスレッドコードが戻ってくるまで（この例では戻りませんが）実行の中断を遅延させます。

その状況では、実行を中断するために Abort キーの組み合わせ（Shift+Escape または Ctrl+Break）を使う必要があります。

このコードのデバッグについては、「スレッドセーフなコードのデバッグ」のセクションを参照してください。

Macro Execute Error: Debug ボタン



デバッガーが有効になっており、マクロでエラーが発生すると、Igor はほとんどの場合、Debug ボタン付きのエラーダイアログを表示します。
Debug ボタンをクリックすると、デバッガーウィンドウが開きます。

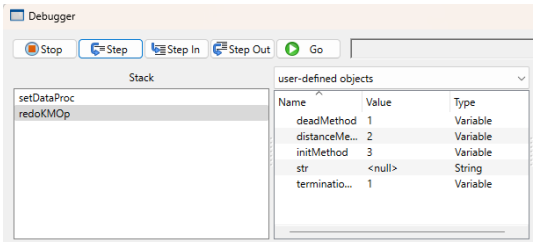
マクロやプロシージャのエラーは、発生後すぐに報告されます。

ユーザー定義関数でエラーが発生した場合、Igor は実際にはエラーが発生してからかなり時間が経ってからエラーダイアログを表示します。
Debug On Error オプションはプログラマー向けであり、エラー発生直後にデバッガーを起動することで、関数内のエラーを発生時に表示します。

コードのステップ実行

コードを１ステップずつ実行することは、コードがどのような処理を行っているか、また変数がどのようにしてその値を持つようになったかがわからない場合に便利です。

デバッガーを有効にし、関心のあるコード行にブレークポイントを設定することから始めます。
または、エラーが発生したためにデバッガーが自動的に開いたところから始めます。
デバッガーウィンドウの上部にあるボタンを使って、コードをステップ実行します。



Stop ボタン

Stop ボタンは、実行中の関数またはマクロの実行を完了前に停止します。
これは、プロシージャが実行中のときに Igor の Abort ボタンをクリックするのと同じです。
Debug On Abort を有効にしている場合でも、Stop ボタンをクリックすると実行が停止します。

キーボードショートカット：Ctrl+Break

Step ボタン

Step（ステップオーバー）ボタンは次の行を実行します。
その行に１つ以上のサブルーチンへの呼び出しが含まれている場合、サブルーチンが戻ってくるか、エラーまたはブレークポイントが発生するまで実行が継続されます。
返ってくると、別のボタンをクリックするまで実行が停止します。

キーボードショートカット：Enter、F10

Step Into ボタン

Step Into ボタンは次の行を実行します。
その行に１つ以上のサブルーチンへの呼び出しが含まれている場合、最初のサブルーチンに入ると実行が停止します。
現在実行中のルーチンのスタックリストでは、最後に実行されたルーチンがリストの最後の項目として表示されます。

キーボードショートカット：+、=、F11

Step Out ボタン

Step Out ボタンは、現在のサブルーチンが終了するか、エラーまたはブレークポイントに遭遇するまで実行を継続します。

キーボードショートカット：-（マイナス）、_（アンダースコア）、Shift+F11

Go ボタン

Go ボタンはプログラムの実行を再開します。
デバッガーウィンドウは、実行が完了するか、エラーまたはブレークポイントが発生するまで開いたままになります。

Go ボタンをクリックしながら Alt キーを押すと、実行が完了するか、エラーまたはブレークポイントに到達するまでデバッガーウィンドウが閉じられます。

キーボードショートカット：Esc、F5

スタックと変数リスト

Stack リストには、現在実行中のルーチンと、それを呼び出したルーチンの連鎖が表示されます。
リストの一番上の項目は実行を開始したルーチンであり、一番下の項目は現在実行中のルーチンです。

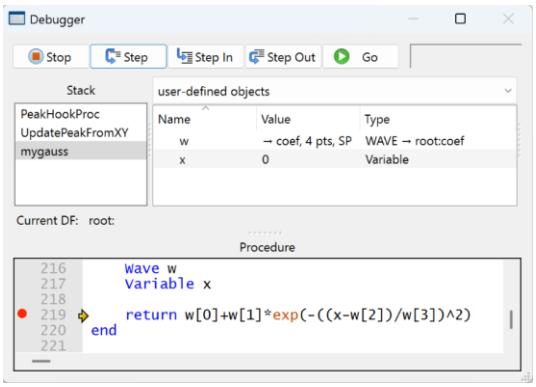
右の例では、実行を開始したルーチンは PeakHookProc であり、これが直近で UpdatePeakFromXY を呼び出し、その後現在実行中の mygauss ユーザー関数を呼び出しました。

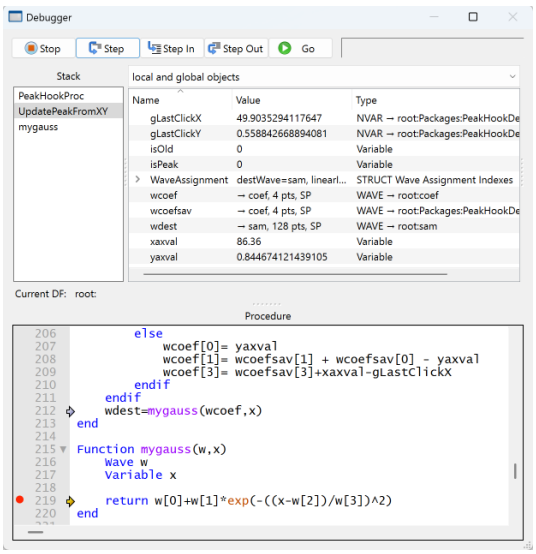
スタックリストの右側にあるオブジェクトリストは、関数パラメーター w と x の値が coef（ウェーブ）と 0（数値）であることを示しています。
ポップアップメニューはリストに表示されるオブジェクトをコントロールします。
この例では、変数、文字列、WAVE 参照などのユーザー定義ローカルオブジェクトのみが表示されています。

スタックリストで任意のルーチンに関連するオブジェクトを調べるには、単にそのルーチンを選択するだけで済みます。

ここでは、mygauss を呼び出すルーチン UpdatePeakFromXY を選択しました。
オブジェクトリストには UpdatePeakFromXY で使われるオブジェクトが表示されていることに注意してください。

説明のため、オブジェクトリストのポップアップメニューはローカルオブジェクトとグローバルオブジェクト、および型情報を表示するように設定しています。





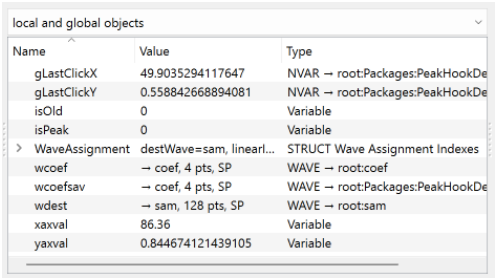
オブジェクトリストの列

オブジェクトリストは、オブジェクトポップアップメニューの show variables types がチェックされているかどうかによって、２列または３列で表示されます。

列のヘッダーをダブルクリックすると、その列の幅が内容に合わせて変更されます。
もう一度ダブルクリックすると、列の幅がデフォルトの幅に変更されます。
列ヘッダーをシングルクリックすると、クリックした列の内容でリストが並べ替えられます。
再度クリックすると、並べ替え順序が逆になります。

最初の列はオブジェクトのローカルの名前です。
NVAR、SVAR、または WAVE 参照の名前は、データフォルダー内のグローバルオブジェクトを参照するマクロまたは関数にローカルな名前です。

２番目の列はローカルオブジェクトの値です。
２番目の列をダブルクリックすると、その場で数値を編集できます。



また、行の任意の場所をダブルクリックすると、該当するインスペクターで構造体のウェーブ、文字列、SVAR、または文字配列を「確認」できます。

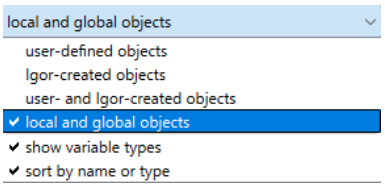
ウェーブの場合、そのサイズと精度がここに表示されます。
「->」文字は「参照する」という意味です。
この例では、wcoef は coef という名前の（グローバルな）ウェーブを参照するローカル名です。
このウェーブは一次元で、４つのポイントを持ち、単精度浮動小数点です。

特定のウェーブ要素の値を決定するには、「ウェーブの検査」のセクションで説明されているインスペクターを使います。
または、式インスペクターで wcoef[2] のような式を記述します。

オプションの３番目の列には、変数のタイプが「変数」、「文字列」、「NVAR」、「SVAR」、「WAVE」など、何であるかが示されます。
グローバル参照の場合、グローバルへの完全なデータフォルダーパスが表示されます。

オブジェクトポップアップメニュー

オブジェクトポップアップメニューでは、オブジェクトリストに表示する情報をコントロールします。
関数をデバッグする時は、右のような表示になります。

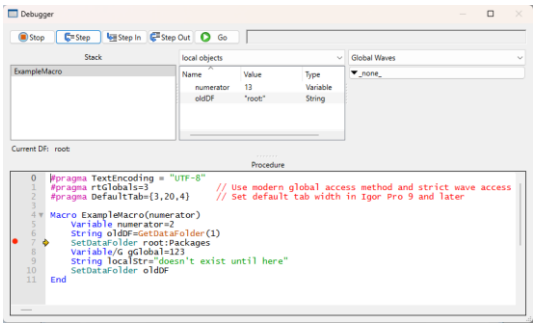


マクロ、プロシージャ、またはウィンドウマクロをデバッグしているときは、ポップアップメニューの最初の２つの項目は使用できません。

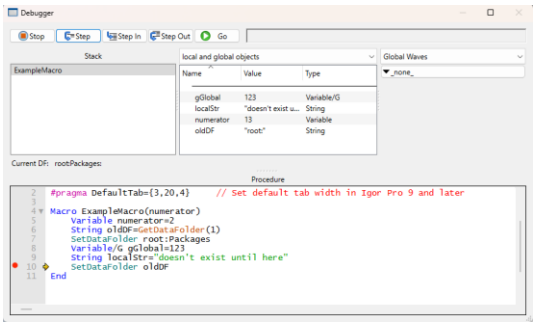
マクロオブジェクト

次の ExampleMacro は、マクロ、プロシージャ、ウィンドウプロシージャ内の変数がローカルまたはグローバルとして分類される方法を示しています。

マクロのローカルオブジェクトには、パラメーターとして渡されたすべての項目（この例では numerator）と、定義が実行されたローカル変数およびローカル文字列（oldDF）、およびコマンドが実行された後に WaveStats などのコマンドによって作成された Igor 作成のローカル変数と文字列が含まれます。
コマンドがまだ実行されていないため、localStr はリストに表示されていません。



マクロ内のグローバルオブジェクトには、マクロで使われているかどうかに関わらず、現在のデータフォルダー内のすべての項目が含まれます。
SetDataFolder コマンドによりデータフォルダーが変更された場合、グローバル変数のリストも変更されます。
マクロには、NVAR、SVAR、WAVE、STRUCT の参照がないことに注意してください。



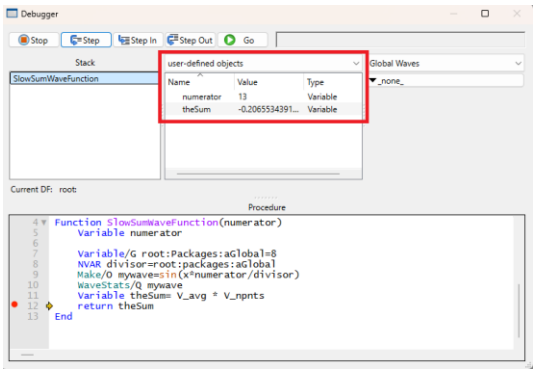
関数オブジェクト

次の SlowSumWaveFunction の例は、関数内の異なる種類の変数がどのように分類されるかを示しています。

関数内のユーザー定義オブジェクトには、パラメーターとして渡されたすべての項目（この例では numerator）と、ローカル文字列および変数（theSum など）が含まれます。

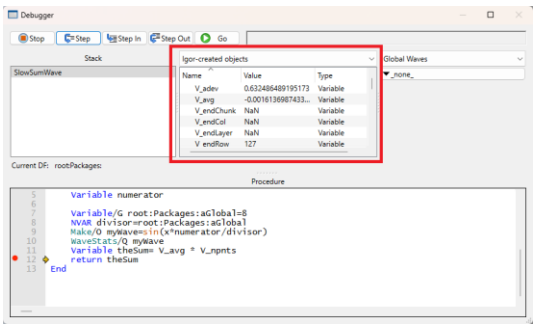
ローカル変数はプロシージャが実行されている間は存在しますが、プロシージャから戻る（return）と同時に消滅します。
ローカル変数は、ウェーブのようなグローバル変数のようにデータフォルダーに存在することはありません。

NVAR、SVAR、WAVE、Variable/G、String/G の参照はグローバル変数を指しているため、ユーザー定義（ローカル）変数としてリストアップされません。

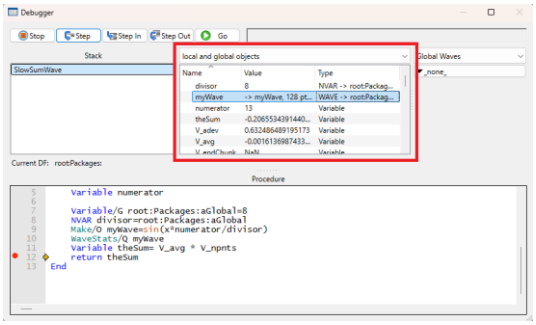


Igor-created objects を使って、結果を返すコマンドや関数を呼び出す時に、Igor が関数用に作成したローカル変数を表示します。
例えば、WaveStats コマンドでは、統計結果を格納する変数として V_adev、V_avg、その他の変数を定義しています。

user- and Igor-created objects メニュー項目には、両方の種類のローカル変数が表示されます。



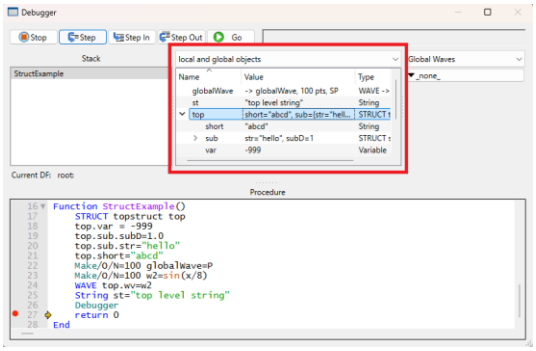
local and global variables objects メニュー項目には、ユーザー作成のローカルオブジェクト、Igor が作成したほとんどのローカルオブジェクト、NVAR、SVAR、WAVE 参照によるグローバル変数、文字列とウェーブへの参照が表示されます。



関数の構造

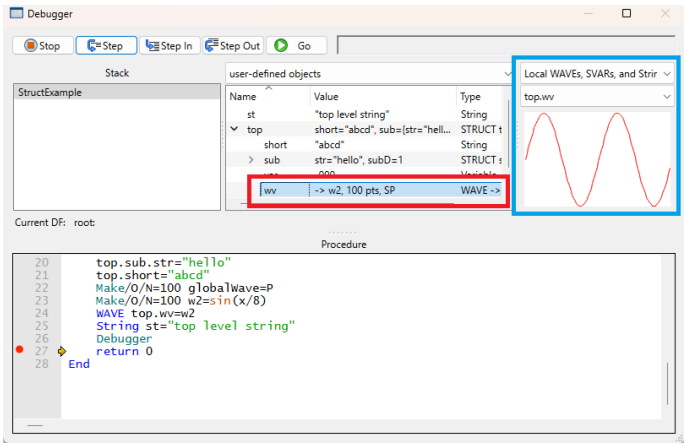
構造の要素（ヘルプ Structures in Functions を参照）は、変数リストにツリー形式で表示されます。
三角アイコンをクリックして構造内のノードを展開または折りたたむか、行をダブルクリックします。

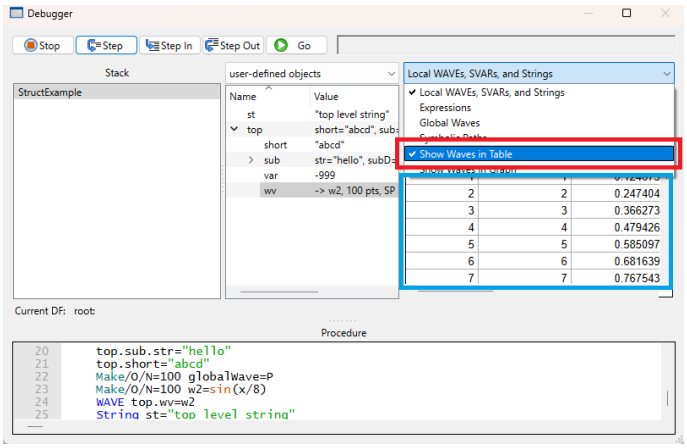
右図は次のコードを表示したものです。



```
Structure substruct
        Variable subD
        String str
EndStructure

Structure topstruct
        Variable var
        STRUCT substruct sub
        String short
        WAVE wv
EndStructure

Function StructExample()
        STRUCT topstruct top
        top.var = -999
        top.sub.subD=1.0
        top.sub.str="hello"
        top.short="abcd"
        Make/O/N=100 globalWave=P
        Make/O/N=100 w2=sin(x/8)
        WAVE top.wv=w2
        String st="top level string"
        Debugger
        return 0
End
```

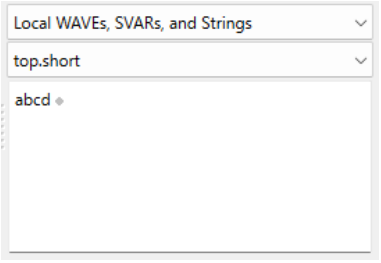
WAVE 要素（top.wv など）をダブルクリックすると、その要素がウェーブインスペクター（インスペクターポップアップでチェックされている内容に応じて、テーブルまたはグラフ）に送信されます。





（グラフまたはテーブルのインスペクタが表示されていない場合、非表示になっている可能性があります。デバッガーウィンドウの右端にある非表示位置から、インスペクタの仕切りを左にドラッグしてください。）

String 要素または文字配列（top.short など）をダブルクリックすると、それが String インスペクターに表示されます。



ウェーブ代入のデバッグ

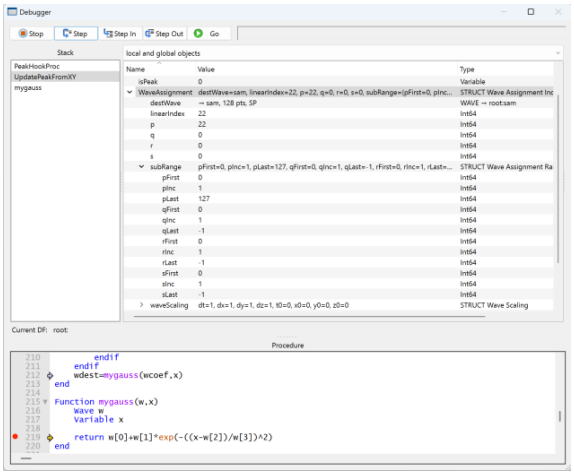
ウェーブフォームへの代入の実行状態は、代入が行われるルーチン内でデバッガーが生成する WaveAssignment 構造体で確認できます。

この例では、次の行：

```
wdest=mygauss(wcoef,x)
```

が、UpdatePeakFromXY 関数内で現在実行中のウェーブフォーム代入ステートメントです（矢印で示されている箇所）。

UpdatePeakFromXY は、mygauss ユーザー定義関数を呼び出し、目的のウェーブ（グローバルウェーブ「sam」を参照するウェーブ wdest）の 128 個の値それぞれを設定します。



例として示されている WaveAssignment 構造体は、sam[22] が代入されていることを示しています。

WaveAssignment 構造体は以下を示します。

destWave　　代入先のウェーブ

linearIndex　　ウェーブを 1 次元ウェーブとして見た場合、代入されるウェーブのポイント。

p, q, r, s　　ウェーブ内のポイントが代入される現在の行、列、レイヤー、チャンクのインデックス（多次元ウェーブとして見た場合）。

subRange　　行、列、レイヤー、チャンクの次元の開始インデックス、増加量、終了インデックス。

最後のインデックスが -1 の場合、対応する次元が代入に関連しないことを示す。
例として示したウェーブは 1 次元であるため、qLast = -1 は列次元が関連しないことを示す。

waveScaling　　ウェーブの各次元のウェーブスケーリングで各次元の開始値と増分値として表現される。

x0, dx = 行スケーリング、DimOffset(destWave,0)、DimDelta(destWave,0)

y0, dy = 列スケーリング、DimOffset(destWave,1)、DimDelta(destWave,1)

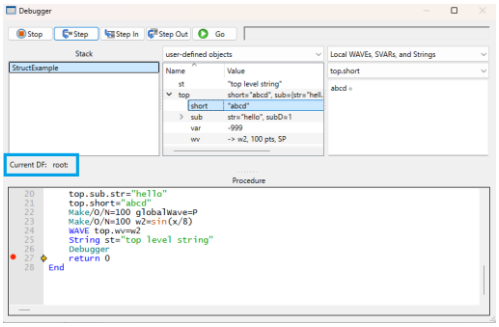
z0, dz = レイヤースケーリング等

t0, dt = チャンクスケーリング

詳細については、ヘルプ Waveform Arithmetic and Assignment を参照してください。

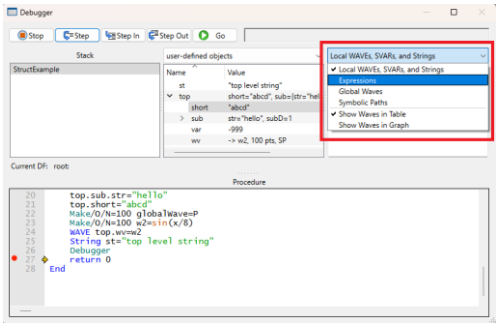
カレントのデータフォルダー

Current DF は、現在のデータフォルダーへのパスを表示します。
データフォルダーの値を選択してコピーすることができます。
データフォルダーの詳細については、ヘルプ Data Folders を参照してください。



グラフ、テーブル、文字列、式インスペクター

オブジェクトリストの右側に、ウェーブ、文字列、または式インスペクターのいずれかが表示されます。
このペインは、ペインとオブジェクトリストの間の仕切りを右端までドラッグすると非表示になります。
仕切りを左にドラッグすると、ペインが表示されます。
スペースを確保するためにウィンドウを広げる必要があるかもしれません。

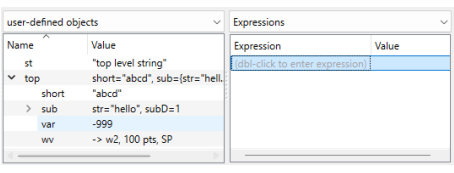


ポップアップを使って、希望するインスペクターを選択します。

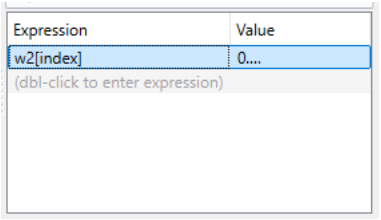
メニュー項目	**表示内容**
Local WAVEs, SVARs, and Strings	選択した関数におけるこれらのもののリスト。 自由ウェーブへの参照もここに記載されています。 また、「検査可能」な STRUCT の要素も記載されており、文字配列は文字列として表示することができます。
Local Strings	選択した関数、マクロ、またはプロシージャにローカルな文字列の一覧。
Expressions	選択された関数またはマクロのコンテキストで評価される数値または文字列式。
Global Waves	グローバルデータフォルダー内の任意のウェーブを表示するポップアップウェーブセレクター。 フリーウェーブはここにリストアップされていません。
Show Waves in Table	ウェーブは、テーブルまたはグラフで表示されます。
Show Waves in Graph	グラフにウェーブを表示するは、この 2 つのうちどちらがチェックされているかによって異なります。

式の検査

インスペクターのポップアップから Expressions を選択すると、Expressions とそれらの値のリストが表示されます。



(dbl-click to enter expression) という項目をダブルクリックして、数値または文字列式を入力してリターンキーを押して置き換えます。

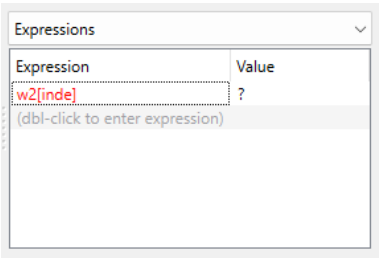


式を追加すると、リストの最後に行が追加され、その行をダブルクリックして別の式を入力することができます。
ダブルクリックして入力することで、どの式も編集することができます。

式は、選択して Delete または Backspace キーを押すことで削除できます。

プロシージャをステップ実行する時に、式の結果が再計算されます。
式は現在選択されているプロシージャのコンテキストで評価されます。

グローバルの式は現在のデータフォルダーのコンテキストで評価されますが、データフォルダーを明示的に指定することもできます。



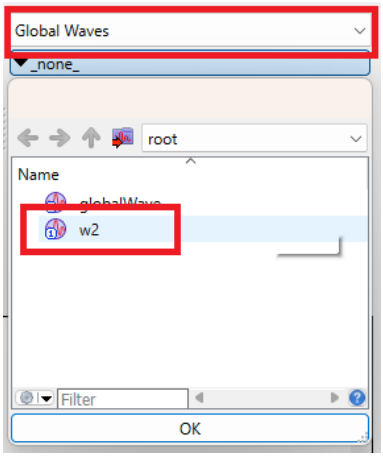
式が不正な場合、結果は「？」と表示され、式は赤色に変わります。

新しい Igor エクスペリメントが開かれた場合、または Igor が終了した場合、これらの式は破棄されます。

ウェーブの検査

ウェーブの内容は、テーブルまたはグラフで「検査」（表示）できます。
これらは、サポート用のダイアログがないため、フル機能のテーブルやグラフではありません。
コンテキストメニューを使って、これらのプロパティを変更することができます。

検査するウェーブを３つの方法のうちの１つで選択します。



1. Global Waves を選択し、ポップアップのウェーブブラウザーからウェーブを選択します。
2. Local WAVEs, SVARs, and Strings を選択し、表示されたオブジェクトの中からウェーブを選択します。
3. 変数リスト内の任意の WAVE 参照をダブルクリックします。

テーブル内のウェーブを検査

通常のテーブルと同様に、テーブルを使ってウェーブ内の値を編集することができます。
コンテキストメニューを使うと、列のフォーマットを変更することができます。

グラフ内のウェーブを検査

ウェーブをグラフで表示できます。
コンテキストメニューで、軸の表示/非表示を選択したり、表示モードを変更したりできます。

２次元ウェーブは画像プロットとして表示されます。

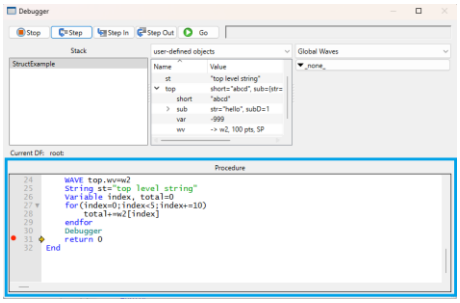
文字列の検査

２つの方法で検査する文字列または文字配列を選択します。

1. Local WAVEs, SVARs, and Strings を選択し、リストされたオブジェクトの中から、String、SVAR、または文字配列を選択します。

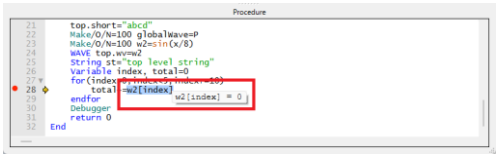
2. 変数リスト内の文字列、SVAR、または文字配列をダブルクリックします。

プロシージャペイン



プロシージャペインには、Stack リストで選択したルーチンのプロシージャウィンドウのコピーが表示されます。
このペインでは、ブレークポイントマージンと右クリックメニューを使って、プロシージャウィンドウと同様にブレークポイントの設定とクリアが可能です。

デバッガーの非常に便利な機能として、カーソル下の変数または式の値を表示する自動テキスト式評価機能があります。
値はツールチップとして表示されます。
これは、変数、ウェーブ、または構造体メンバー参照の値を決定するために、変数リストをスクロールしたり、式リストで式を入力したりするよりも早い場合が多いです。



変数の値は、変数名が選択されているかどうかに関わらず表示することができます。
wave[ii]+3 のような式を評価するには、その式を選択し、カーソルをその選択部分に置く必要があります。

デバッガーは、ユーザー定義関数の呼び出しを含む式を評価しません。
これにより、予期せぬ副作用（例えば、関数がウェーブの内容を上書きする可能性）を防止します。
この制限は、グローバル変数 root:V_debugDangerously を作成し、それを 1 に設定することで解除できます。

バグを発見した後は

デバッガーウィンドウでの編集は、コードが現在実行中であるため無効になっています。
デバッガーを終了した後にルーチンを追跡するには、以下の手順に従うと簡単です。

1. プロシージャコードからルーチンの名前をクリップボードにコピーします。

2. デバッガーを終了します。

3. ルーチンの名前をコマンドラインに貼り付けます。ここでは実行しないでください。

4. 名前を右クリックし、ポップアップメニューから「Go to <ルーチン名>」を選択します。

選択したルーチンがトップのプロシージャウィンドウに表示され、そこで編集することができます。

スレッドセーフなコードのデバッグ

Igor デバッガーは、スレッドセーフな関数がメインスレッドで実行されていない限り、使うことができません。

スレッドセーフな関数内でブレークポイントを設定しても、メインスレッドで実行されていない限り、デバッガーはブレークしません。

デバッガーは、スレッドセーフな関数をステップ実行することができません。
ただし、メインスレッドで実行されている場合はこの限りではありません。

スレッドセーフではない関数からスレッドセーフな関数を呼び出した場合でも、これらの制限は適用されます。

優先スレッドで実行されるスレッドセーフなコードのデバッグの主なテクニックは、Print 文を使うことです。
「Print 文を使ったデバッグ」のセクションを参照してください。

以下のコマンドを実行して、一時的にマルチスレッド処理を無効化することで、通常、プリエンプティブスレッドで実行される関数に対してデバッガーを使うことができます。

```
SetIgorOption DisableThreadSafe = 1        // マルチスレッドを無効にする
```

これにより、Igor はすべてのプロシージャを再コンパイルし、Threadsafe および MultiThread キーワードを無視します。
その後、デバッガーを使ってプロシージャのデバッグを行うことができます。

```
SetIgorOption DisableThreadSafe = 0        // マルチスレッドを有効にする
```

デバッグのショートカット

（本セクションはヘルプ Debugger Shortcuts より流用）

デバッガーを有効にする

・Procedure メニューから Enable Debugger を選択します。
・プロシージャウィンドウで右クリックし、コンテキストメニューから Enable Debugger を選択します。

エラー発生時に自動的にデバッガーに入る

・Procedure メニューから Debug on Error を選択します。
・プロシージャウィンドウで右クリックし、コンテキストメニューから Enable Debugger を選択します。

ブレークポイントを設定・クリアする

・プロシージャウィンドウの左のマージンをクリックします。
・ブレークポイントを設定または解除したいプロシージャウィンドウの行の任意の場所をクリックし、右クリックして、コンテキストメニューから Set Breakpoint または Clear Breakpoint を選択します。

ブレークポイントを有効・無効にする

・プロシージャウィンドウの左マージンにあるブレークポイントをシフトクリックします。
・ブレークポイントを有効または無効にしたいプロシージャウィンドウの行の任意の場所をクリックし、右クリックしてコンテキストメニューから Enable Breakpoint または Disable Breakpoint を選択します。

次のコマンドを実行する

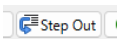
・Enter キーを押します。
・フォーカスが当たっているボタンがない場合は、Ente キーを押します。
そうでない場合は、黄色い矢印ボタンをクリックします。

サブルーチンにステップインする

・「+」「=」キーを押すか、Step In ボタンをクリックするか、青い下向きの矢印ボタンをクリックします。

サブルーチンから呼び出し元のルーチンに戻る

・「-」「_」キーを押すか、Step Out ボタンをクリックするか、青い上向きの矢印ボタンをクリックします。



通常の実行を再開する

・Esc キーを押すか、緑の矢印ボタンをクリックします。

実行をキャンセルする

・赤い Stop ボタンをクリックします。

マクロ変数または関数変数の値を編集する

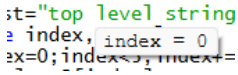
・変数リストの２番目の列をダブルクリックし、値を編集して、Enter キーを押します。

関数の文字列の値を null に設定する

変数リストの２番目の列をダブルクリックし、<null> と入力し、Enter キーを押します。

マクロ変数または関数変数の現在の値を表示する

カーソルを変数名のプロシージャテキストに移動し、少し待ちます。
値がツールチップウィンドウに表示されます。

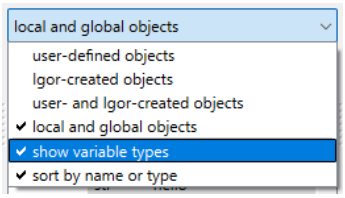


式の現在の値を表示する

カーソルでテキストを選択し、選択部分にカーソルを移動して、少し待ちます。
値がツールチップウィンドウに表示されます。
（ユーザー定義関数を含む式は、V_debugDangerously が 1 に設定されていない限り評価されません）

現在のデータフォルダー内のグローバル値を表示する

デバッガーのポップアップメニューから local and global objects を選択します。



変数の型情報を表示する

デバッガーのポップアップメニューから show variable types を選択します。

変数リスト内の列のサイズを変更する

リスト内の仕切りを左または右にドラッグします。

ウェーブ、構造体、式ペインを表示または非表示にする

変数リストの右側にある仕切りを左または右にドラッグします。