KitSprocket (キットスプロケット)
コマンドリファレンス(3.2.6)

内容

コマンド書式

■コマンドの基本書式

スクリプトを構成するそれぞれの行は、コマンドと引数(値 1 値 2...)の間をタブ文字で区切ります。

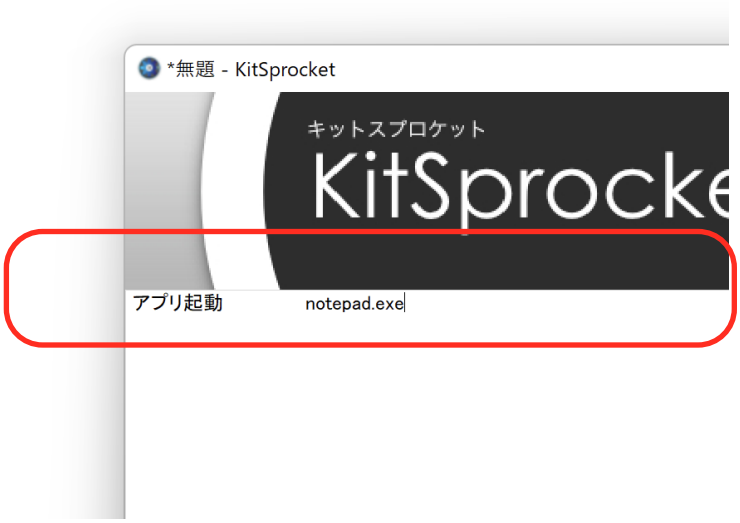
基本書式：　コマンド名　　値 1　　値 2　　...
変数名=データ　　// 変数名にデータを格納します
変数=[コマンド]　　値 1　　値 2　　...　　//コマンドの実行結果を変数に入れるには[]で囲みます。でないと コマンド という文字列が変数に入るだけになる
変数="アイウエオ カ""キ""クケコ"　　// <- 変数に　アイウエオ<改行>カ"キ"クケコ が入る
コマンド[2]　　値 1　　//コマンドで指定した画面上のアイテムが同名だったときに 2 番目を選択
コマンド A.grp　　//グループ名がついた最初のコマンドの実行結果があとのコマンドに影響します コマンド B.grp　　//コマンド A が失敗したらコマンド B は実行されません
コマンド A.grp コマンド B.!grp　　//コマンド A が失敗したらコマンド B は実行されます (※グループ名を!で始めると否定になる)
// コメント　　コメント行になります。コマンドの引数の後ろには入れないでください。（//が入力が必要な場合もあるため引数として扱われます）

※行の先頭のタブは無視され、区切りに複数タブが入力されている場合は 1 つのタブ文字とみなします。
※それぞれのコマンドでは最低限必要な引数があり、間違っていると何も実行されません。
※編集エリアで右クリックすると、コマンドのサンプルを入力できます。
※タブは複数あっても無視されます。先頭にタブがあっても最初の文字列をコマンドとして認識するので、スクリプトをタブで整形することができます。
※変数名=[コマンド名]　　とするとで、コマンドの実行結果を代入することができます(結果を返すコマンドのみ)
※変数名の指定がない場合、@result@ という変数に結果が保存されます
※変数に格納したいデータを""で囲むと、改行やタブを含めることができます。(""で囲まれているなかで"を入力したい場合は、""と 2 つ入れます)
※コマンド名に[2]のように数値を追加すると、同名のウインドウの 2 番目というような指定ができます。(対応しているコマンドのみ)

※コマンド名に.hoge のように 任意の拡張子を付けると、同じ拡張子がついたメンバーをグループにできます。同じグループのいずれかで失敗したら実行しないようにしたり、その逆で失敗した時だけ実行するというような使い方ができます。

入力例 1：

```
アプリ起動　　notepad.exe
```



※途中に入れるタブ文字は複数あっても 1 つとして扱うので、スクリプトを見やすく整形することができます。また行の先頭に // を入れるとコメント行になります。

注意：マニュアル PDF 上の記述例をコピー&ペーストしてもそのまま使えません。空白部分をタブに書き換えてください ¥ 記号も書き直す必要があります。コピー＆ペーストしたい場合は、サンプルスクリプトフォルダにある「コマンドリファレンス.txt」を開いてください。

コマンドリファレンス

以下に、KitSprocket で利用可能なコマンドの詳細情報を記載します。
必須ではない引数は(())で囲まれています。
概要に <> で記載されている文字は マウス右クリックで表示されるショートカットメニューでの表記です。

なお、ウインドウ名やアプリ名の表記については、*を前後に使うことで特殊な意味を持ちます。

【ウインドウの名前、アプリの名前の指定方法】
　　*　　・・・すべての名前が対象
　ABC*　　・・・ 名前が ABC から始まるもの
　*ABC　　・・・ 名前が ABC で終わるもの
　ABC　・・・ 名前に ABC が含まれるもの

$変数=[コマンド]　引数　… のように記述すると、コマンドの実行結果を $変数に 格納します。 $変数= と書かない場合は [] は必要ありません。また引数がある場合も []は必須ではありません。

OK　$変数=文字入力　あいうえお
NG　$変数=OS バージョン　　（→　$変数=[OS バージョン]）

■■ アプリ、ウインドウ操作 ■■

<アプリ起動> コマンド

書き方：

変数名=[アプリ起動]　(値 1:アプリ実行ファイルのパス)　((値 2:引数)　(値 3:最小化/最大化))

概要　：値 1 に指定されたアプリを起動します。基本フルパスを指定しますが、パスが通った Windows のアクセサリ(cmd.exe , notepad.exe)などについては、アプリ名だけで起動できます。

値 2 にはアプリに対する起動オプションを指定できます。ファイルを同時に開きたい場合はファイルパスを指定します。値 3 には、起動時の状態を指定します。
値 3 に 最小化 または 最大化 といれておくと、アプリが起動時に最小化した状態で起動する

か、最大化した状態で起動するかを選択できます。（アプリによっては対応していないため動作しない場合があります）

変数名=[アプリ起動] … とすると、起動したアプリのプロセス ID(PID)が取得できます。ただし、アプリによっては正確な PID が取得できない場合があります。
プロセスが起動しなかった場合は、変数名に<-1> というデータが入ります。

例 1)アプリ起動　notepad.exe

例 2)アプリ起動　notepad.exe　c:¥file.txt

例 3)アプリ起動　notepad.exe　最小化

例 4)pid=[アプリ起動]　　notepad.exe

※起動時の状態 指定は Windows アプリ側にも対応が必要であるため、動作しない場合があります。

<アプリの起動を待機> コマンド

書き方：

変数名=[アプリ待機]（値 1：アプリ名/PID) ((値 2：待機する秒数:初期値 30))

概要　：値 1 で指定したアプリが出現するまで処理を中断します。
(何も指定しない場合は 30 秒待つ)
アプリの名前ではなく、アプリのファイル名で指定してください。うまく機能しないアプリもあるので注意してください。
変数名= とすると、変数名にアプリのプロセス ID(PID)が入ります。見つからない場合は変数名に<-1>が入ります。

例 1)アプリ待機　notepad.exe　10　　//notepad が起動するまで 10 秒待機します。

例 2)PID=[アプリ待機]　notepad.exe　//最初に見つかった notepad.exe の PID が変数に入ります

<アプリの存在を確認> コマンド

書き方：

アプリ存在確認　(値 1：アプリ名/PID)

概要　：値 1 で指定したアプリが起動しているかどうか確認します。

例) アプリ存在確認.chk　notepad.exe

メッセージ.?chk　　アプリは存在します！

メッセージ.!chk　　アプリは存在しません！

<アプリ強制終了> コマンド

書き方：

アプリ強制終了　(値 1：アプリ名/PID)　　((値 2：すべて終了する))

概要　：値 1 で指定したアプリを強制的に終了します。値 2 になにか文字が入っていると、見つかったアプリをすべて強制終了します。保存してないデータが失われるので注意してください。

例)アプリ強制終了　　notepad.exe

<アプリ終了待機> コマンド

書き方：

アプリ終了待機　(値 1：アプリ名/PID)　　((値 2：待機する秒数：初期値 30)

概要　：値 1 で指定したアプリが終了するまで処理を中断します(初期値では最大 30 秒待つ)名前またはプログラム名またはプロセス ID(PID)で指定してください。

例)アプリ終了待機　　notepad.exe　10　　//notepad が終了するまで 10 秒待機します。

<起動中のアプリ情報取得> コマンド

書き方：

変数名=[起動アプリ情報]　　(値 1：アプリ名/PID)((値 2：PID,名前,使用メモリ,最大メモリ,パス))

概要　：値 1 で指定したアプリの情報を取得して変数名に格納します。
取得できる情報は、値 2 に PID(プロセス ID)、名前(プロセスのファイル名)、使用メモリ(プロセスが現在使用しているメモリサイズ)、最大メモリ(プロセスが使用した最大のメモリサイズ)、パス(起動アプリのファイルパス)を指定することで取得できます。

例)$情報$=起動アプリ情報　　notepad.exe　PID,名前

<起動しているアプリのリストを取得> コマンド

書き方：

変数名=[起動アプリリスト] (値 1：アプリファイル名/PID) ((値 2：PID,名前))

概要　：値 1 で指定したアプリファイル名に一致するリストを取得します。アプリファイル名を*とすると、すべてのプロセスを取得します。値 2 は取得する形を選択します。PID かファイル名かを選択できます。

例 1)
$リスト$=[起動アプリリスト]　*　　名前,PID
リスト選択ダイアログ　$リスト$

<ウインドウが出現するまで待機> コマンド

書き方：

変数名=[ウインドウ待機] (値 1：ウインドウ名/ID) ((値 2：アプリ名/PID) (値 3：待機する秒数))

概要　：値 1 で指定したウインドウが出現するのを 30 秒待ちます。
値 2 にアプリ名、値 3 にウインドウが出現するのを待つ時間を設定できます。
値 3 が指定されていない場合、値 2 に数値が入っていれば待つ時間の指定、それ以外だとアプリ名として処理します。
変数名= とすると、変数名にウインドウ ID が入ります。ID を使うことで同じタイトルのウインドウでも区別することができます。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ待機　　メモ帳

例 2)ウインドウ待機　　メモ帳　10

例 3)ウインドウ待機　　*　　NotePad.exe　10

<ウインドウの存在確認> コマンド

書き方：

変数名=[ウインドウ存在確認]　(値 1：ウインドウ名/ID)　　((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウの存在を確認します。ウインドウ名を * とすると、最前面のウインドウが対象になります。値 2 の指定は必須ではありません
変数名= とすると、変数名にウインドウ ID が入ります。ID を使うことで同じタイトルのウインドウでも区別することができます。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ存在確認 メモ帳

例 2)ウインドウ存在確認.chk　　*　　NotePad.exe

メッセージ.?chk　　ウインドウは存在します

メッセージ.!chk　　ウインドウは存在しません

<ウインドウが最前面かどうか> コマンド

書き方：

変数名=[ウインドウ前面]　　(値 1：ウインドウ名/ID)　　((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウが存在し、かつ最前面にいるかどうかを確認します。ウインドウ名を * とすると、最前面のウインドウが対象になります。値 2 の指定は必須ではありません
変数名= とすると、変数名にウインドウ ID が入ります。ID を使うことで同じタイトルのウインドウでも区別することができます。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ前面　　メモ帳

例 2)ウインドウ前面.chk　*　　NotePad.exe

メッセージ.?chk　　ウインドウは最前面です

メッセージ.!chk　　ウインドウは最前面ではありません

<ウインドウの切り替え> コマンド

書き方：

変数名=[ウインドウ] (値 1：ウインドウ名/ID) ((値 2：アプリ名/PID) (値 3：待機する秒数))

概要 ： 値 1 で指定したウインドウに切り返えます。待機する秒数を指定しない場合、完全に切り替わるまで、30 秒待機します。
値 1 を指定しない場合、各ウインドウのタイトルをツールチップで表示します。
ウインドウの順番の指定する数値指定に対応しています。
ウインドウ名を * とすると、最前面のウインドウが対象になります。待機する秒数を指定しない場合は 5 秒待機します。
値 3 を指定しない場合、値 2 に数字を入れると待機時間の指定、文字列だとアプリの指定として扱います。
変数名= とすると、変数名にウインドウ ID が入ります。ID を使うことで同じタイトルのウインドウでも区別することができます。ウインドウが見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ切り替え メモ帳 //メモ帳というウインドウに切り替えます
例 2)ウインドウ // 引数を指定しないとウインドウのタイトルとそれぞれを指定する番号が表示されます。
例 3)ウインドウ[3] //3 番目のウインドウに切り替えます。
例 4)ウインドウ * NotePad.exe 5 //NotePad.exe (メモ帳)の 最初のウインドウに切り替えます。5 秒以内に終わらないと失敗となります

<ウインドウ最小化> コマンド

書き方：

変数名=[ウインドウ最小化] (値 1：ウインドウ名/ID) ((値 2：アプリ名/PID))

概要 ：値 1 で指定したウインドウを最小化します。ウインドウ名を * とすると、最前面のウインドウが対象になります。値 2 の指定は必須ではありません。
変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ最小化 メモ帳
例 2)ウインドウ最小化 * NotePad.exe

<ウインドウ最大化>コマンド

書き方：

変数名=[ウインドウ最大化]　　(値 1：ウインドウ名/ID) ((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウを最大化します。ウインドウ名を * とすると、最前面のウインドウが対象になります。値 2 の指定は必須ではありません。
変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ最大化　　メモ帳
例 2)ウインドウ最大化　　*　　NotePad.exe

<ウインドウ復元>コマンド

書き方：

変数名=[ウインドウ復元]　　(値 1：ウインドウ名/ID) ((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウを復元します。(最小化または最大化状態からフリーサイズに戻す動き)ウインドウ名を * とすると、最前面のウインドウが対象になります。
変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ復元　　メモ帳
例 2)ウインドウ復元　　*　　NotePad.exe

<ウインドウを隠す>コマンド

書き方：

変数名=[ウインドウ非表示]　　(値 1：ウインドウ名/ID)　((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウを非表示にします。最小化ではなく、完全に見えなくなりますので、「ウインドウを表示する」コマンドで元に戻さないと閉じることができなくなりますので注意してください。

変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ非表示　　メモ帳

例 2)ウインドウ非表示　　*　　NotePad.exe

<ウインドウを表示する> コマンド

書き方：

変数名=[ウインドウ表示]　　(値 1：ウインドウ名/ID) ((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウが、非表示のウインドウの場合、表示します。変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ表示　　メモ帳

例 2)ウインドウ表示　　*　　NotePad.exe

<ウインドウ閉じる> コマンド

書き方：

変数名=[ウインドウ閉じる]　　(値 1：ウインドウ名/ID) ((値 2：アプリ名/PID))

概要　：値 1 で指定したウインドウを閉じます。|で区切って複数のウインドウを閉じることができます。

値 2 にアプリのファイル名(メモ帳ならば Notepad.exe)を指定すると、そのアプリのウインドウのみ対象とします。

変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ閉じる　　メモ帳

例 2)ウインドウ閉じる　　ドキュメント|ダウンロード

例 3)ウインドウ閉じる　*　　Notepad.exe　//Notepad.exe の 一番先頭のウインドウを閉じる

<ウインドウ移動> コマンド

書き方：

変数名=[ウインドウ移動]　(値 1：ウインドウ名/ID)　(値 2：アプリ名/PID)
(値 3：移動位置または移動位置とウインドウサイズ)

概要　：値 1 で指定したウインドウを値 2(または値 2)の位置に移動します。位置の指定は[x 座標,y 座標]または[x 座標,y 座標,ウインドウ幅,ウインドウ高さ]をピクセル単位で指定します。

ウインドウ名を * とすると、最前面のウインドウが対象になります。
値 2 にアプリのファイル名(メモ帳ならば Notepad.exe)を指定すると、そのアプリのウインドウのみ対象とします。
変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウ移動　メモ帳　[10,10,600,400]
例 2)ウインドウ移動　*　Notepad.exe　[10,10,600,400]

<ウインドウ閉じるのを待つ> コマンド

書き方：

変数名=[ウインドウ閉じるのを待つ]　(値 1：ウインドウ名/ID) (値 2：アプリ名/PID) (値 3：待機する秒数)

概要　：値 1 で指定したウインドウが閉じるの指定した秒数待ち続けます。たとえばなにか処理中のダイアログが閉じたらシャットダウンしたいときなどに使います。
ウインドウ名を * とすると、最前面のウインドウが対象になります。
値 3 を指定しない場合、値 2 に数字を入れると待機時間の指定、文字列だとアプリの指定として扱います。
変数名= とすると、変数名にウインドウ ID が入ります。見つからない場合は変数名に<0>が入ります。

例 1)ウインドウを閉じるのを待つ　メモ帳　1800
例 2)ウインドウを閉じるのを待つ　*　NotePad.exe　1800

<ウインドウの現在の位置とサイズを取得> コマンド

書き方：

変数名=[ウインドウサイズ]　　(値 1：ウインドウ名/ID)　(値 2：アプリ名/PID)　　(値 3：全体位置/左上位置/幅/高さ)

概要　：値 1 で指定したウインドウの位置とサイズを取得して変数名に格納します。値 2(または値 3)に 全体位置/左上位置/幅/高さ サイズ のいずれかを指定して、その数値を取得することができます。

例 1)ウインドウサイズ　メモ帳　$サイズ$

例 2)ウインドウサイズ　*　NotePad.exe　高さ　$高さ$

<ウインドウの情報を取得> コマンド

書き方：

変数名=[ウインドウ情報]　　(値 1：ウインドウ名/ID)　(値 2：アプリ名/PID)　　(値 3：タイトル,位置,幅,高さ,存在,可視,有効,最前面,最小化,最大化,アプリ名,ハンドル) (値 4：表示/非表示)

概要　：ウインドウに含まれる情報を取得します。タイトル,位置(の座標),高さ,幅,存在(するかどうか),可視(見えているか、隠れているか),有効(グレーアウトしているかどうか),最前面(最前面かどうか),最小化,最大化,アプリ名(ウインドウを表示しているアプリのプロセス名),ハンドル(ウインドウ固有の ID)

例 1)ウインドウ情報　メモ帳　notepad.exe　タイトル　$情報$

例 2)ウインドウ情報　メモ帳　ハンドル　$ハンドル$

<ウインドウのリストを取得> コマンド

書き方：

変数名=[ウインドウリスト]　(値 1：ウインドウ名/ID)　((値 2：アプリ名/PID)　　(値 3：タイトル,ハンドル,非表示))

概要　：ウインドウのリストを取得します。値 1 のウインドウ名に*をつけると、その文字列を含むウインドウが取得できます。アプリのファイル名を値 2 に設定すると、さらにアプリケーションを絞り込むことができます。

値 3 にタイトル,ハンドル,非表示 のいずれかまたはすべての記述をすると取得できる情報が変化します。

例 1) $リスト$=ウインドウリスト　メモ帳　explorer.exe

例 2) $リスト$=[ウインドウリスト]

例 3) $リスト$=[ウインドウリスト]　*　explorer.exe　タイトル,ハンドル

リスト選択ダイアログ　$リスト$

<ウインドウ数が変化するのを待つ> コマンド

書き方：

変数名=[ウインドウ数変化待機] (値 1：ウインドウ名/ID)　((値 2：アプリ名/PID)　(値 3：秒数))

概要　：ウインドウ数が変化するのを待ちます。どんなタイトルが開いてくるか不明だが、ウインドウが新しく開くのを待ちたい場合に使います。
使い方は、秒数に -1 を入れると、現在の状況をメモリ上に記憶し、もう一度コマンドを実行することでメモリに記憶したウインドウ数と比較して、ウインドウが増えるまで待つという流れです。
変数名= とすると、変数名に最前面のウインドウの ID が入ります。
なお、コマンドの実行毎にメモリ上には前回の値が残り続けるので、連続して 1 つずつ増えるのを確認するような処理でもつかえます。

例 1)
ウインドウ数変化待機　*　explorer.exe　-1
ファイル開く　c:¥windows
ウインドウ数変化待機　*　explorer.exe　10
メッセージ　ウインドウ数が変化しました。

例 2)
ウインドウ数変化待機　*　explorer.exe　-1
ファイル開く　c:¥windows
ウインドウ数変化待機　*　explorer.exe
メッセージ　ウインドウ数が変化しました。
ファイル開く　c:¥windows¥system32
ウインドウ数変化待機　*　explorer.exe
メッセージ　ウインドウ数が変化しました。

■■ メニュー操作 ■■

<メニュー> コマンド

書き方：

メニュー　　(値 1：メニュー階層 1)　　(値 2：メニュー階層 2)　　....

概要　：値 1～値 4 までの指定でメニューを選択します。| で区切ることでも階層を指定できます。※リボンや特殊なアプリでは動作しない場合があります。

例 1)メニュー　ファイル　　開く

例 2)メニュー　ファイル|開く

※アプリケーションによってはこの命令を受け付けません。その場合は、キー入力 [alt+f].. のようにキー操作で代用してください

<貼り付け> コマンド

書き方：

貼り付け　　(値 1：貼り付けたい文字列)

概要　：値 1 で指定した文字列をクリップボード経由で貼り付けます。実際はテキストをコピーして ctrl + v するだけなので、テキスト入力エリアが最前面にあるようにスクリプトを工夫する必要があります。

例)貼り付け　　自動で文字を貼り付け

■■ ダイアログ操作 ■■

<ボタン> コマンド

書き方：

ボタン　　(値 1：ボタン名)

概要　：値 1 で指定したボタン名のボタンをクリックします。値 1 を指定しない場合、ボタン番号がツールチップで表示されます。

例 1)ボタン　　OK

例 2)ボタン[2]

// 2 つめのボタンを押します。ボタンの番号を知りたい場合は「ボタン」とだけ書いて実行します。

ラジオボタン　　(値 1：ラジオボタン項目名)

概要　：値 1 で指定した項目名のラジオボタンをクリックします。値 1 を指定しない場合、ラジオボタン番号がツールチップで表示されます。

例 1)ラジオボタン　　最近使ったフォルダ

例 2)ラジオボタン[2]

// 2 つめのラジオボタンを押します

<チェックボックス> コマンド

書き方：

チェックボックス　　(値 1：チェックボックスのラジオボタン) (値 2：ON または OFF)

概要　：値 1 で指定したチェックボックスの ON/OFF を値 2 で指定します。

例)チェックボックス　　次回から表示しない　　ON　　//オフにしたい場合は ON を OFF としてください

<エディット> コマンド

書き方：

エディット　　(値 1：置き換えたい文字列)

概要　：値 1 で指定した文字列を直接 ダイアログの文字枠(エディット)に書き込みます。クリップボードを経由しません。また、値 1 を指定しない場合、エディットの位置番号をツールチップで表示します。

例 1)エディット　こんにちは

　//すでにエディットに「あいうえお」がはいっていると「あいうえお」が消えて「こんにちは」に置き換わります

例 2)エディット[2]　　ハロー

　//ダイアログの 2 番目のエディットにハローを代入します

<エディット追加> コマンド

書き方：

エディット追加　(値 1：追加したい文字列)

概要　：値 1 で指定した文字列を現在のエディットフィールドの文字列に追加します。

例 1)エディット追加　　たろうさん　　//すでにエディットに「こんにちわ」が入っているばあい「こんにちはたろうさん」になります

例 2)エディット追加[2]　パンダ　//ダイアログの 2 番目のエディットにパンダを代入します

<リストボックス> コマンド

書き方：

リストボックス　(値 1:リストボックスの選択肢)

概要　：値 1 で指定したリストボックスの項目を選択します。値 1 を指定しないとリストボックスの番号をツールチップで表示します。 ※コンボボックス、リストビュー というコマンドでも動作します

例 1)リストボックス　　///リストボックスの番号を確認

例 2)リストボックス[2]　MS ゴシック　// 2 つめのリストボックスの選択を MS ゴシックにします

<タブの切り替え> コマンド

書き方：

タブ　　(値 1:切り替えたいタブ名の一部)

概要　：値 1 で指定したタブに切り替えます。値 1 を指定しないとタブの番号をツールチップで表示します。 ※Explorer 、Google Chrome などブラウザのタブには対応しません

例 1)タブ　　詳細　//エクスプローラーのファイルのプロパティダイアログなど

<ラベルの検出> コマンド

書き方：

ラベル　　(値 1:ダイアログ上に見つけたい文字列の一部)　(値 2:ウインドウ名)　(値 3:アプリ名)

概要　：値 1 で指定した文字列を含むダイアログを見つけたら成立します。値 1 を指定しないとラベルの番号をツールチップで表示します。 値 2 でウインドウ、値 3 でアプリケーションを指定できます。※ダイアログによってはラベルが取得できない場合があります。

例 1)ラベル.chk　　保存しますか？

メッセージ.chk　　保存するかどうか確認中

■■ マウス操作 ■■

<マウスクリック> コマンド

書き方：

マウスクリック　(値 1：クリックしたい座標) ((値 2：ウインドウ名/ID))

概要　：値 1 に指定されている座標をクリックします。座標は[x,y]という書式で書きます。値 2 に基準となるウインドウ名またはウインドウ ID を指定すると、座標指定はウインドウからの相対位置になります。なお、このコマンドのみ[]で数字しているると、マウスボタンを指定できます。(例:マウスクリック[2]　[10,10] で右クリック)

例 1)マウスクリック　　[100,200]

///デスクトップ左上から、100 ドット横、200 ドット下の位置をクリックします。

例 2)マウスクリック　　[10,20]　　無題

// 無題というウインドウ左上から 10 ドット横、20 ドット下の位置をクリックします。

例 3)マウスクリック[3]　[10,20]

// 左上から 10 ドット横、20 ドット下の位置を中央ボタンでクリックします。

<マウスダブルクリック> コマンド

書き方：

マウスダブルクリック　(値 1：ダブルクリックしたい座標) ((値 2：ウインドウ名/ID))

概要　：値 1 に指定されている座標をダブルクリックします。座標は[x,y]という書式で書きます。値 2 に基準となるウインドウ名またはウインドウ ID を指定すると、座標指定はウインドウからの相対位置になります。

記述例：「マウスダブルクリック[100,200]」

※デスクトップ左上から、100 ドット横、200 ドット下の位置をダブルクリックします。

記述例：「マウスダブルクリック[10,20]　　無題」

※無題というウインドウ左上から 10 ドット横、20 ドット下の位置をダブルクリックします。

<マウストリプルクリック> コマンド

書き方：

マウストリプルクリック (値 1：クリックしたい座標) ((値 2：ウインドウ名/ID))

概要　：値 1 に指定されている座標を 3 回(トリプル)クリックします。座標は[x,y]という書式で書きます。値 2 に基準となるウインドウ名またはウインドウ ID を指定すると、座標指定はウインドウからの相対位置になります。

例 1)マウストリプルクリック　[100,200]

//デスクトップ左上から、100 ドット横、200 ドット下の位置をトリプルクリックします。

例 2)マウストリプルクリック　[10,20]　　無題

//無題というウインドウ左上から 10 ドット横、20 ドット下の位置をトリプルクリックします。

<マウス右クリック> コマンド

書き方：

マウス右クリック　(値 1：右クリックしたい座標)　((値 2：ウインドウ名/ID))

概要　：値 1 に指定されている座標を右クリック(第 2 ボタンのクリック)します。座標は[x,y]という書式で書きます。値 2 に基準となるウインドウ名またはウインドウ ID を指定すると、座標指定はウインドウからの相対位置になります。

例 1)マウス右クリック　[100,200]

//デスクトップ左上から、100 ドット横、200 ドット下の位置を右クリックします

例 2)マウス右クリック　[10,20]　　無題

```
//無題というウインドウ左上から 10 ドット横、20 ドット下の位置を右クリックします
```

<マウス移動> コマンド

書き方：

```
マウス移動　　(値 1：移動先の座標)
```

概要　：値 1 に指定されている座標をにマウスポインタを移動します。座標は[x,y]という書式で書きます。

```
例)マウス移動　[100,200]
 //デスクトップ左上から、100 ドット横、200 ドット下の位置に移動します。
```

<マウスドラッグ(ボタン 1)> コマンド

書き方：

```
マウスドラッグ　(値 1：開始位置の座標)　　(値 2：移動先の座標)
```

概要　：値 1 から値 2 までドラッグします。座標は[x,y]という書式で書きます。

```
例)マウスドラッグ　　[10,10]　　[100,200]
 //デスクトップ左上 10 ドット横、10 ドット下から、100 ドット横、200 ドット下の位置にドラッグします。
```

<マウスドラッグ(ボタン 2)> コマンド

書き方：

```
右マウスドラッグ (値 1：開始位置の座標)　　(値 2：移動先の座標)
```

概要　：値 1 から値 2 まで右ボタン(第 2 ボタン)ドラッグします。座標は[x,y]という書式で書きます。

```
例)右マウスドラッグ　　[10,10]　　[100,200]
 //デスクトップ左上 10 ドット横、10 ドット下から、100 ドット横、200 ドット下の位置にドラッグします。
```

<マウスホイール> コマンド

書き方：

マウスホイール　(値 1：上 または 下) (値 2：まわす回数)

概要　：値 1 に方向にマウスホイールを値 2 の数値ぶん回転します。回す回数は 1～10 までの範囲で指定します

例 1)マウスホイール　　上　　10

//マウスの現在位置でホイールを回す操作をします。このためウインドウに重なっていないとなにも起きません

例 2)

ウインドウ　　ドキュメント

マウス移動　　ドキュメント

マウスホイール　　下　　2

<マウス押しっぱなし> コマンド

書き方：

マウスダウン　　(値 1：押さえるボタン　left/right/middle)

概要　：値 1 で指定されたボタンを押さえます。left が左ボタン、right が右ボタン、middle が中央ボタンです

例)マウスダウン left

<マウス押しっぱなしから離す> コマンド

書き方：

マウスアップ　　(値 1：離すボタン　left/right/middle)

概要　：マウスダウンで押さえっぱなしになっているボタン(値 1 で指定)を押さえっぱなしから解除します。

例)マウスアップ left

<マウスを最後に移動した前の場所に戻す> コマンド

書き方：

マウスをもどす

概要　：マウスクリックで移動したマウスポインタをコマンド実行直前の位置に戻します。画像判定でクリックした後、クリックすべきボタンにマウスが重なった状態になるため、画像判定で一致しなくなるのを防ぎます

例)マウスをもどす

<マウスの現在の位置を取得> コマンド

書き方：

変数名=[マウスの現在位置]　(値 1：クリック操作のタイミングで取得)

概要　：マウスの現在位置を取得します。値 2 が何か指定されていると、マウスクリックしたタイミングでマウスの位置を取得します

例1)　$point=[マウスの現在位置]

例2)　$point=[マウスの現在位置]　　クリックを待つ

■■ キー操作 、特殊キー入力■■

<キー、文字列の入力> コマンド

書き方：

キー入力	(値 1：入力したい文字、または特殊キー) ((値 2:対象のウインドウ名/ID))

概要　：文字を入力します。日本語を入れることで日本語も入力できます。キーボードショートカットのような組み合わせも送信できます。値 2 にウインドウ名またはウインドウ ID を入れることで、そのウインドウに切り替えて入力を行うようにできます。

例 1)キー入力　あいうえお
例 2)キー入力　さしすせそ　　メモ帳

例)

キー入力　　[control+c]
キー入力　　ここにひらがなまたは Alphabet
キー入力　　{!}+!#メモ帳
キー入力　　[無変換]　　メモ帳

◎ control や shift などのキーを入力したい場合は[]で囲んでください

キー入力 [shift]

※どのような特殊キーが使えるかは、「キー入力　表記法一覧」を参照

キー入力　表記法一覧

書き方	押される(入力される)キー	書き方例
ctrl または control または　rctrl ※	Ctrl キー	キー入力　[ctrl+n]
alt　または ralt ※	Alt キー	キー入力　[alt+f]
shift または rshift ※	Shift キー	キー入力　[shift+a]
win または rwin ※	Windows キー	キー入力　[win+f]
bs または backspace	BS キー	キー入力　[bs]

書き方	押される(入力される)キー	書き方例	
delete または del	DEL キー	キー入力	[del]
tab	TAB キー	キー入力	[tab]
ins または insert	INS キー	キー入力	[ins]
f1、f2、f3、... f12	F1 から F12 まで	キー入力	[f5]
up、down、left、right	矢印キー	キー入力	[up]
pageup	PAGE UP	キー入力	[pageup]
pagedown	PAGE DOWN	キー入力	[pagedown]
kanjikey または 半角/全角	半角/全角 キー	キー入力	[半角/全角]
無変換、変換、かな、英数	無変換、変換、かな、英数	キー入力 キー入力 キー入力 キー入力	[無変換] [変換] [かな] [英数]
enter または return	enter キー	キー入力	[enter]
home	home キー	キー入力	[home]
end	end キー	キー入力	[end]
appskey	appskey キー(通常、ショートカットメニューが表示される)	キー入力	[appskey]
space	スペースキー	キー入力	[space]
esc	esc	キー入力	[esc]
home	home	キー入力	[home]
printscreen	printscreen	キー入力	[printscreen]

※r から始まるキー表記(rshift など)はキーボード右側の特殊キー(shift キー)ということになります。アプリによっては左と右で機能が異なる場合があります。

また、以下のコマンドで ctrl,alt,shift,win のキーを押しっぱなし(down)にしたり、押しっぱなしの解除(up)が可能です。操作に失敗すると、KtiSprocket を終了してもキーが押されっぱなしになるので、扱いには十分注意してください。

書き方	押される(入力される)キー	書き方例	
ctrldown	Ctrl キー	キー入力	[ctrldown]
ctrlup	Ctrl キー	キー入力	[ctrlup]
altdown	Alt キー	キー入力	[altdown]
altup	Alt キー	キー入力	[altup]
shiftdown	SHIFT キー	キー入力	[shiftdown]
shiftup	SHIFT キー	キー入力	[shiftup]
windown	Windows キー	キー入力	[windown]
winup	Windows キー	キー入力	[winup]

[特殊機能]

キー入力 abcd "" 100 50 とすると「abcd」の 1 文字ずつの入力間隔が 1000 分の 100 秒になります。また キーを押さえ続ける時間が 1000 分の 50 になります。

キー入力の速度は、環境設定でも変更できます。

※ 右クリックメニューの表記と入力されるコマンド

キー操作：

(上矢印キー)キー入力 [up]

(下矢印キー)キー入力 [down]

(左矢印キー)キー入力 [left]

(右矢印キー)キー入力 [right]

(Shift キー押しっぱなし)キー入力 [shiftdown]

(Shift キー押しっぱなしからはなす)キー入力 [shiftup]

(Ctrl キー押しっぱなし)キー入力 [ctrldown]

(Ctrl キー押しっぱなしからはなす)キー入力 [ctrlup]

特殊キー入力：

(Home キー)キー入力　[home]

(End キー)キー入力　[end]

(Enter キー)キー入力　[enter]

(Tab キー)キー入力　[tab]

(Esc キー)キー入力　[esc]

(半角/全角キー)キー入力　[半角/全角]

(Ins キー)キー入力　[ins]

(Del キー)キー入力　[del]

(BS キー)キー入力　[bs]

(Printscreen キー)キー入力　[printscreen]

(Ctrl と Home を同時に押す)キー入力[ctrl+home]

(Alt と f を同時に押す)キー入力　[alt+f]

(Shift と Ctrl と c を同時に押す)キー入力　[shift+ctrl+c]

■■ 文字列、数値の操作 ■■

<文字の入力ダイアログを表示> コマンド

書き方：

変数名=[文字入力ダイアログ]　　(値 1:キャプション) (値 2:初期値) (値 3:伏せ字として使う文字)

概要　：文字を入力するダイアログを表示します。初期値を指定できます。値 3 に文字を入れるとその文字を伏せ字として利用できます

例 1) $文字$=文字入力ダイアログ　　文字を入れてください

例 2) $文字$=文字入力ダイアログ　　文字を入れてください　http://

例 3）$文字$=文字入力ダイアログ　　パスワードを入力してください ""　　*

メッセージ　　$文字$

<文字数を数える> コマンド

書き方：

変数名=[文字数] (値 1:数えたい文字列)　　(値 2:検索する文字)

概要　：文字数を数えます。特定の文字数だけを数えたい場合は、値 2 に文字列を入れます

例 1) $文字数$=文字数　　あいうえお

メッセージ $文字数$

例 2) $要素数$=文字数　　A|B|C　　|

メッセージ　$要素数$

<文字列を比較> コマンド

書き方：

文字を比較　　(値 1:文字列) (値 2:探す文字列)　(値 3:比較条件)

概要　：値 1 の文字列に値 2 の文字列が含まれていれば成立します。

値 1 より値 2 が多い文字列の場合、値 2 に値 1 が含まれていると成立します。（この動作が不要な場合、後述の「文字の位置」コマンドを使ってください

値 3 はオプションで = で完全一致、< で、先頭一致　> で末尾一致、<> または！で不一致となります。? の指定ではどこかに文字が含まれていれば成立となります。

※変数を比較 コマンド と異なるのは引数の書き方と、< や > の意味が変わります。

例 1) $テストファイル$=c:¥test.text
　　文字を比較.chk　$テストファイル$　c:¥test.text
　　メッセージ.?chk　文字列は同じです。
　　メッセージ.!chk　文字列は異なります。
例 2)文字を比較.chk　ows　Windows　>　//成立
例 3)文字を比較.chk　Win　Windows　<　//成立
例 4)文字を比較.chk　Windows　Windows　=　//成立(大文字小文字区別します)
例 5)文字を比較.chk　Windows　Win　<　//成立

<文字列のどの位置に文字列が含まれるか> コマンド

書き方：

変数名=[文字の位置]　(値 1:文字列) (値 2:探す文字列)

概要　：値 1 の文字列に値 2 の文字列が含まれていれば成立し、その位置を返します。動作としては「文字を比較」で、比較条件を？とした場合と同じですが、こちらは、値 1 の中に値 2 が含まれなければ成立しません（値 2 の中に値 1 があっても不成立）

例 1) $文字の位置$=[文字を含むかどうか]　あいうえお　うえ
メッセージ　$文字の位置$　//3 が返ります

<文字列を置換> コマンド

書き方：

変数名=[文字を置換]　(値 1:文字列)(値 2:探す文字列)　(値 3:置換文字列)　(値 4:検索方向)

概要　：文字列を置換します。値 4 に> を入れると、先頭から検索して最初にヒットする文字だけ置き換えます。または 値 4 に <　を入れると、末尾から検索してヒットする最初の文字だけを置換します（値 4 になにも入れないとすべて置き換えます）
値 2 を[abc][def]のようにすると、abc から始まって def で終わる範囲を対象として置き換えます。

例 1) $置換後$=文字を置換　c:¥test.txt　c:¥　c:¥temp¥
　　メッセージ　$置換後$

例 2) $置換後$=文字を置換　　c:¥test¥test.txt　　test　　work　<

　　メッセージ　　$置換後$

例 3）$置換後$=文字を置換　　c:¥test¥test.txt　　[:¥te][st.]　　work

　　メッセージ　　$置換後$

＜文字列を切り取り＞ コマンド

書き方：

変数名=[文字を切り取り]　　(値 1:切り取りしたい文字列)　　(値 2:開始位置)　(値 3:終了位置)　　(値 4:オプション)

概要　：文字列を指定範囲で切り取ります。（バージョン 3.2.6 より仕様変更）

値 2 および値 3 に、[<]、[>]の文字列 を使った特殊な書き方をすることで、切り取る範囲の微調整を行うことができます。

「あいう AAAA えがお BBB かきくけこ」をから AAAA 切り取りたい場合、

値 2 を [あいう[>]] 、値 3 を[[<]えがお]とします。値 2 を[[>]AAAA]、値 3 を[AAAA[<]]としても同じ結果になります。[>]は切り取り文字の位置を示しています。

さらに、値 2 を[[>]]とすることで、開始位置からデータの最後まで取得するという意味になります。([[>]]はデータの先頭となります)

値 4 に <または>を入れることで、文字の検索を前から行うか、後方から行うかを指定できます。「aaaabbbccccdddaaaaeeee」とあるばあい、後半の aaaa を取り出したい場合は、<のオプションを指定します。開始位置の検索方向と終了位置の検索方向を変えたい場合 <<のように 2 つオプションに指定します。

例 1) $結果$=文字を切り取り　　あいうえお　　い　　え

　　メッセージ $結果$　　//「いうえ」

例 2) $結果$=文字を切り取り　あいうえお　　[い[>]][[<]お]

　メッセージ　$結果$　//「うえ」

例 3) $結果$=文字を切り取り　あいうえお　　[[>]い][お[<]]

　メッセージ　$結果$ //「いうえお」

例 5) $結果$=文字を切り取り　c:¥folder¥data.txt　　[¥[>]] .txt　　<

　メッセージ　$結果$ //「data.txt」

例 6）$改行入り$="

文字切り取りのテスト　その 1

です"

```
$結果$=文字を切り取り  $改行入り$      [その[>]]       [[>]@改行@]
    メッセージ    $結果$ //「1」
```

<文字列を切り取りした残り> コマンド

書き方：

変数名=[文字を切り取りした残り]　(値 1:切り取りしたい文字列)　(値 2:開始位置)　(値 3:終了位置)　(値 4:オプション)

概要　：「文字を切り取り」の逆の部分の文字列を取得します。
引数などは「文字を切り取り」と同じで、切り取った文字を除く残りのデータを変数名に格納します

```
例 1) $改行入り$="
文字切り取りのテスト　その 1
です"
$結果$=文字を切り取りした残り $改行入り$      [その[>]]       [[>]@改行@]
    メッセージ    $結果$ //「1」だけが消える
```

<文字列を並び替える> コマンド

書き方：

変数名=[文字並べ替え]　(値 1:並べ替えしたい文字列)　(値 2:並べ替えの設定)　(値 3:区切り文字)

概要　：文字列を並べ替えの設定に従って、並び替えます。区切り文字を利用することで、例えば「100,10,110,5」のような数値や「あああ|いいい|あ|えええ」の文字列を並び替えることができます。

設定には、昇順か降順か、数字かを指定できます。「降順,数字」のように書きます。設定がない場合や矛盾する要素が設定された場合はは、文字数とコード順(比較する文字数が同じ場合)の昇順となります。

```
例 1) $並替後$=文字並べ替え　　ああああ|いいい|え|ううう　　|
      メッセージ　　$並替後$
例 2) $並替後$=文字並べ替え　　ああああ|いいい|え|ううう　　降順　|
      メッセージ　　$並替後$
```

例 3) $並替後$=文字並べ替え　　100,10,500,0001　　数字　,
　　メッセージ　　$並替後$
例 4) $並替後$=文字並べ替え　　100,10,500,0001　　降順,数字　　,
　　メッセージ　　$並替後$

<文字コードを取得> コマンド

書き方：

変数名=[文字コード]　　(値 1:コードを得たい文字列)

概要　：値 1 に指定した文字列を|区切りの文字コード(ユニコードで 10 進数表記)に変換し、値 2 の変数名に格納します。

例 1) $aiueo$=文字コード　　あいうえお
　　メッセージ　　$aiueo$

<コードを文字に変換> コマンド

書き方：

変数名=[コードから文字]　　(値 1:文字コード)

概要　：値 1 に指定した文字コード(|区切りの文字コード(ユニコードで 10 進数表記))から、文字列に変換し、値 2 の変数名に格納します。

例 1) $aiueo$=文字コード　　12354|12356|12358|12360|12362
　　メッセージ　　$aiueo$

<16 進数に変換> コマンド

書き方：

変数名=[16 進数に変換] (値 1:数値)　(値 2:桁数)

概要　：値 1 に指定した 10 進数の数値を 16 進数に変換します。値 2 に桁数、値 3 に変数名とすると、出力される 16 進数が指定した桁数で表記されます。
　　桁数の指定は省略可能

例 1) $aiueo$=16 進数に変換　100
　　メッセージ　　$aiueo$
例 2) $aiueo$=16 進数に変換　100　3
　　メッセージ　　$aiueo$

<10 進数に変換> コマンド

書き方：

変数名=[10 進数に変換] (値 1:16 進数の文字列)

概要　：値 1 に指定した 16 進数の文字列を 10 進数に変換します。(表記法は 数値のみまたは 0x から始まる文字列に対応)

例 1) $aiueo$=10 進数に変換　064
　　メッセージ　　$aiueo$
例 2) $aiueo$=10 進数に変換　0x064
　　メッセージ　　$aiueo$

■■ ファイル操作 ■■

<ファイルを開く> コマンド

書き方：

ファイル開く　　(値 1：開くファイルのパス) ((値 2：開きたいアプリ))

概要　：値 1 で指定したパスのファイルを開きます。値 2 でアプリを指定すると、アプリを指定してファイルを開くことができます。

例)ファイル開く c:¥sample¥readme.txt　　notepad.exe

<ファイルを移動> コマンド

書き方：

ファイル移動　　(値 1：移動するファイルのパス)　(値 2：移動先のファイル、またはフォルダパス) (値 3：上書き)

概要　：値 1 で指定したパスのファイルを値 2 のパスに移動します。フォルダにも対応します。値 3 になにか文字を入れておくと、既存のファイルを上書きします。

例 1)ファイル移動　　c:¥sample¥readme.txt　　d:¥

例 2)ファイル移動　　c:¥test.doc　c:¥Users¥test¥Desktop¥test.doc　　上書き

※値 3 に書く文字列はどのようなものでも OK

<ファイルをコピー> コマンド

書き方：

ファイルコピー　(値 1：コピーしたいファイルのパス)　(値 2：コピー先のファイル、またはフォルダパス)　((値 3：上書き))

概要　：値 1 で指定したパスのファイルを値 2 のパスにコピーします。値 3 に任意の文字列があると、コピー先に同名のファイルがある場合は上書きします。フォルダにも対応

例 1) ファイルコピー　　c:¥sample¥readme.txt　　d:¥

//↑d:¥に readme.txt があると上書きしない

例 2）ファイルコピー　　c:¥sample¥readme.txt　　d:¥　　上書き

//↑d:¥に readme.txt があると上書きする

※値 3 に書く文字列はどのようなものでも OK

<ファイルを削除> コマンド

書き方：

ファイル削除　　(値 1：削除するファイルまたはフォルダのパス)　((値 2：完全削除))

概要　：値 1 で指定したパスのファイルを削除します。値 2 に指定がないとファイル、フォルダをゴミ箱に移動します。値 2 に任意の文字列があると、ファイルを完全に削除するため復活できませんので注意。フォルダの場合はフォルダを削除します。

例 1)ファイル削除　　c:¥sample¥readme.txt　　完全に削除する　　// readme.txt は復元不可能になります

例 2)ファイル削除　　c:¥sample¥readme.txt　　//readme.txt はゴミ箱に移動します。後でゴミ箱から復活させることができます

<ファイルを選択> コマンド

書き方：

変数名=[ファイル選択]　((値 1：キャプション)　(値 2：拡張子指定)　(値 3: 初期フォルダ) (値 4:初期ファイル名))

概要　：ファイル選択ダイアログを表示して、新規ファイルの指定や既存ファイルを選択します。(値 1：キャプション)にファイル選択ダイアログに表示する文字列、(値 2：拡張子指定)には、絞り込みたい拡張子を|区切りで指定します。

例 1) $パス$=ファイル選択　ファイルを指定してください　txt|jpg|gif

例 2) $パス$=ファイル選択　ファイルを指定してください　txt　c:¥test testfile.txt

<ファイル名を取得>コマンド

書き方：

変数名=[ファイル名取得]　(値 1：ファイルのパス)　((値 2：拡張子不要))

概要　：値 1 で指定したファイルパスのファイル名だけ取得します。値 2 に任意の文字列と配置すると、拡張子なしのファイル名が取得できます。

例 1) $filename=ファイル名取得　c:¥Users¥hoge¥text.txt

//→　$filename に text.txt が格納されます

例 2) $filename=ファイル名取得　c:¥Users¥hoge¥text.txt　no 拡張子

//→　$filename に text が格納されます

<親フォルダのパスを取得> コマンド

書き方：

変数名=[親フォルダ取得]　(値 1：ファイルのパス)　(値 2：存在を無視)

概要　：値 1 で指定したファイルパスの直前のフォルダパスを取得します。値 2 になにか文字を入れると、パスが実際に存在していなくても親フォルダのパスを生成します。

例) $pfolder=親フォルダ取得　c:¥Users¥hoge¥text.txt

//→　$pfolder に c:¥Users¥hoge が格納されます

<ファイル名の変更> コマンド

書き方：

変更後のファイル名=[ファイル名変更] (値 1：ファイルのパス)　　(値 2 :変更後の名前)

概要　：値 1 で指定したファイルパスのファイル名(またはフォルダ名)を値 2 で指定した名前に置き換えます
(場所を移動したい場合は ファイル移動 コマンドを使ってください)

例)ファイル名変更　　c:¥Users¥hoge¥text.txt　　result.txt
//→　c:¥Users¥hoge¥text.txt　が　c:¥Users¥hoge¥result.txt に変更される

<ファイルの情報取得> コマンド

書き方：

変数名=[ファイル情報]　　(値 1：ファイルのパス)　　(値 2：情報の種類)

概要　：値 1 で指定したファイルパスのファイル名(またはフォルダ名)の更新日時などを取得します。
値 2 を指定しない場合は、更新日時が返ります。

値 2 に指定できる文字は、「更新日時」「作成日時」「アクセス日時」「サイズ」「バージョン」「リンク」「属性」となります。
同時に取得したい場合、「,」で区切って指定します。(結果はそれぞれが改行で区切られて返ります)
(例：更新日時,サイズ,バージョン)

属性については、ファイルまたはフォルダの属性が以下のアルファベットが組み合わされて返ります

R … 読み取り専用
A … アーカイブ
S … システム属性
H … 不可視
N(I) … 非インデックス対象ファイル属性
D … フォルダである
O … オフライン属性
C … 圧縮属性
T … 一時ファイル属性
X … スクラブファイルなし属性

例 1) $info$=ファイル情報　　c:¥Users¥hoge¥text.txt　　更新日時,サイズ

//

//2021/8/29 11:00:12|16

//の文字列が$info$に格納されます。

例 2)$attrib$=ファイル情報　　c:¥Users　　属性

//→ $attrib$ に D が入る

※「リンク」はファイルパスがショートカットだった場合のみ有効。オリジナルのパスを返します

<ファイルの存在確認> コマンド

書き方：

ファイル確認　　(値 1：存在を確認するファイルまたはフォルダのパス)

概要　：値 1 で指定したパスのファイルの存在を確認します。ファイルが存在しないことを取得するにはグループ機能を使います。

記述例：ファイル存在確認.fchk　c:¥macos¥bbb.txt

メッセージ.!fchk　　ファイルが存在しません

<ファイルを読み込む> コマンド

書き方：

変数名=[ファイル読み込み]　　(値 1：テキストファイルのフルパス)

概要　：値 1 で指定したパスにあるテキストファイルの中身を読み込んで、値 2 で指定した変数名に格納します。

例) $test の中身$=ファイル読み込み　　c:¥test.txt

<ファイルに書き込み> コマンド

書き方：

ファイル書き込み (値１：書き込む文字列または変数名)　　(値２：書き込むファイルのフルパス)　　((値３：上書きするかどうか))

概要　：値１で指定したデータを値２のパスに書き込みます。値３になにか文字列を入力していると既存のファイルを上書きします。
値３を指定しない場合、追記型として動作します。（中身が追加される）
別名保存のような動作をしたい場合、ファイル確認 コマンドにより確認して処理を分岐させてください

例)ファイル書き込み　　書き込みデータ@改行@あいうえお　　c:¥text.txt

<ファイルの中身を消去> コマンド

書き方：

ファイル消去　　(値１：テキストファイルのパス)

概要　：値１で指定したテキストファイルの中身だけ消去します。ファイル自体は消えません。確認メッセージなどは表示されないため十分注意して使用してください。

例)ファイル消去　　c:¥text.txt

<ファイルリスト> コマンド

書き方：

変数名=[ファイルリスト]　(値１：ファイルリストを取得したいフォルダのパス)　((値２：ファイルリストの区切り文字(初期値は|) (値３：オプション))

概要　：値１で指定したフォルダの中身のファイル名のリストを、|で区切られた文字列で変数に格納します。値２で区切り文字を指定して区切り文字は変更することができます。値３でサブフォルダ以下まで検索するかどうかを指定します。また、値３には , を区切って、ファイルリストに付加したい情報を指定できます。
サブフォルダ,フルパス,日付時刻,ファイルサイズ,ファイルバージョン,リンク先のパスに展開,空のフォルダを含む,フォルダを含める
を指定可能。
“サブフォルダ”はサブフォルダ以下のファイルも対象にするかどうか
“フルパス”はリストの結果をフルパスで表示するかどうか
“日付時刻”はファイルパスの後ろに日付時刻が追加
“ファイルサイズ”はファイルパスの後ろにファイルサイズが追加
“ファイルバージョン”はファイルパスの後ろにファイルバージョン（あれば）を追加

“リンク先のパスの展開”は、ショートカット（.lnk 拡張子のファイル）自身のパスではなく、そのショートカット先のパスを取得するようにします
“空のフォルダを含む” は　空のフォルダはファイルではないので、リストに含めるかどうかを指定
“フォルダを含める” は　ファイルリストにフォルダだけのパスを含めるかどうか
それぞれ サブ,フル,日付,サイズ,バージョン,リンク,空,フォルダ　の文字を持つ任意の文字列で指定可能です。

例 1) $winfilelist$=ファイルリスト　　c:¥windows
例 2) $winfilelist$=ファイルリスト　　c:¥windows　@改行@
例 3) $winfilelist$=ファイルリスト　　c:¥windows　|　　サブフォルダ
例 4) $winfilelist$=ファイルリスト　　c:¥windows　@改行@　　日付,サイズ,バージョン,リンク

<フォルダリスト> コマンド

書き方：

変数名=[フォルダリスト]　　(値 1：フォルダリストを取得したいフォルダのパス)
((値 2：ファイルリストの区切り文字(初期値は|))

概要　：値 1 で指定したフォルダの中身のフォルダリストを取得し、|で区切られた文字列で、変数に格納します。値 2 で区切り文字を指定して区切り文字は変更することができます。

例 1) $winfolderlist$=フォルダリスト　c:¥windows
例 2) $winfolderlist$=フォルダリスト　c:¥windows　@改行@

<フォルダを作成> コマンド

書き方：

フォルダ作成　　(値 1：作成したいフォルダのフルパス)

概要　：値 1 で指定したフォルダパスを作成します。

例)フォルダ作成　%userprofile%¥Downloads¥test

※%userprofile% は Windows で定義されているユーザーフォルダとなります。ユーザー名が kanji だったら c:¥Users¥kanji　となります。

<フォルダを選択> コマンド

書き方：

変数名=[フォルダ選択]　　((値 1：キャプション)　　(値 2：初期フォルダのパス))

概要　：フォルダ選択ダイアログを表示して、フォルダを選択します。値 1 にフォルダ選択ダイアログに表示する文字列、値 2 には、最初に開きたいフォルダの位置を指定します。

例 1)$選択フォルダ$=フォルダ選択　　結果を保存するフォルダを指定してください　　c:¥test

<URL を開く> コマンド

書き方：

URL を開く　　(値 1：開きたい URL)

概要　：値 1 で指定した URL をデフォルトブラウザで開きます。

例)URL を開く　http://kitworksinc.com

<インターネットのコンテンツを読み込む> コマンド

書き方：

変数名=[インターネット読み込み]　　(値 1：読み込みたい URL)

概要　：値 1 で指定した URL の HTML ソースを取得します。

例) $google$=インターネット読み込み　https://www.google.com

<インターネットのファイルをダウンロード> コマンド

書き方：

インターネットダウンロード　(値 1：http から始まるアドレス)　(値 2：書き込むファイルへのパス)　(値 3：既存ファイルを上書きするかどうか)

概要　：値 1 で指定した URL を値 2 のパスにダウンロードします。値 2 は完全なパスを指定してください。例えば画像ファイルをデスクトップにダウンロードする場合、記述例 1 のようになります。

例 1)インターネットダウンロード　　https://www.google.com/images/nav_logo229.png を%USERPROFILE%¥Desktop¥nav_logo2291.png

// %USERPROFILE%はログインしているユーザーのルートフォルダに変換されます。

<ファイルサーバーに接続> コマンド

書き方：

サーバーに接続　(値 1：¥¥から始まるアドレス)　(値 2：ユーザー名)　(値 3：パスワード)　(値 4:接続するドライブ文字)

概要　：値 1 で指定したファイルサーバーを値 4 で指定したドライブにマップします。ドメインアカウントで接続するときはユーザー名を「ドメイン名¥アカウント」にします。

例)サーバーに接続　¥¥server1¥share　user　password　x:

<ショートカット作成> コマンド

書き方：

ショートカットの作成　(値 1：プログラム/ファイルへのパス)　(値 2：作成するショートカットのパス)　(値 3：プログラムへの引数)

概要　：値 1 で指定したアプリ or ファイルのショートカットを値 2 のリンクに作成します。値 3 には値 1 がアプリだった場合の引数を指定できます。

例)ショートカット作成　c:¥App.exe　c:¥Users¥test¥Desktop¥App.lnk

<専用ショートカット作成> コマンド

書き方：

専用ショートカットの作成　(値 1：スクリプトファイルのパス)　(値 2：作成するショートカットのパス)　(値 3：アイコン No)

概要　：値 1 で指定したスクリプトのパスを簡単に実行するためのショートカットを作成します。アイコン No は 4-319 までの範囲で指定可能です。

例)専用ショートカット作成　c:¥script.txt　c:¥Users¥test¥Desktop¥test.lnk　5

■■ 画像検出と操作 ■■

<イメージチェック> コマンド

書き方：

イメージチェック (値 1:画像ファイルへのパス)　　(値 2:領域またはウインドウ名/ID)

概要　：画像ファイル(PNG など)の内容と一致する箇所が画面に存在するかどうかをチェックします。。値 2 にウインドウ名かウインドウ ID、[]で囲った画面領域([10,100,200,500])指定か、検出したい画像ファイルのパスを入れておくとその領域に限定してイメージを検索します。グループ機能を使って判定などに利用します。

例 1)イメージチェック.test　　c:¥test.png

例 2)イメージチェック.test　　c:¥test.png　　[10,100,50,500]

例 3)イメージチェック.test　　c:¥test.png　　Facebook

例 4)イメージチェック.test　　c:¥test.png　　[c:¥bigtest.png]　　//<-bigtest.png を画面上に見つけたら、その矩形ないに test.png があるかどうかを見つけます。

メッセージ.test　　画像が見つかりました

メッセージ.!test　画像が見つかりませんでした

<イメージへ移動> コマンド

書き方：

イメージへ移動　(値 1:画像ファイルへのパス)　　(値 2:領域またはウインドウ名/ID)

概要　：画像ファイル(PNG など)の内容と一致する箇所にマウスポインタを移動します。値 2 にウインドウ名かウインドウ ID、[]で囲った画面領域([10,100,200,500])指定か、検出したい画像ファイルのパスを入れておくとその領域に限定してイメージを検索します。

例 1)イメージへ移動　　c:¥test.png

<イメージクリック> コマンド

書き方：

イメージクリック (値 1:画像ファイルへのパス)　　(値 2:領域またはウインドウ名/ID)

概要　：画像ファイル(PNG など)の内容と一致する箇所を画面から検索してクリックします。値 2 にウインドウ名かウインドウ ID、[]で囲った画面領域([10,100,200,500])指定か、検出したい画像ファイルのパスを入れておくとその領域に限定してイメージを検索します。

例 1)イメージクリック　c:¥test.png

例 2)イメージクリック　c:¥test.png　[10,100,50,500]

例 3)イメージクリック　c:¥test.png　Facebook

例 4)イメージクリック　c:¥test.png　[c:¥bigtest.png]　//<-bigtest.png を画面上に見つけたら、その矩形ないに test.png があるかどうかを見つけます。

<イメージダブルクリック> コマンド

書き方：

イメージダブルクリック (値 1:画像ファイルへのパス)　(値 2:領域またはウインドウ名/ID)

概要　：画像ファイル(PNG など)の内容と一致する箇所を画面から検索してダブルクリックします。
値 2 にウインドウ名かウインドウ ID、[]で囲った画面領域([10,100,200,500])指定か、検出したい画像ファイルのパスを入れておくとその領域に限定してイメージを検索します。

例 1)イメージダブルクリック　c:¥test.png

<イメージ右クリック> コマンド

書き方：

イメージ右クリック　(値 1:画像ファイルへのパス)　(値 2:領域またはウインドウ名/ID)

概要　：画像ファイル(PNG など)の内容と一致する箇所を画面から検索して右クリックします。値 2 にウインドウ名かウインドウ ID、[]で囲った画面領域([10,100,200,500])指定か、検出したい画像ファイルのパスを入れておくとその領域に限定してイメージを検索します。

例 1)イメージ右クリック c:¥test.png

<イメージ待機> コマンド

書き方：

イメージ待機　(値 1:画像ファイルへのパス)　(値 2:待機する時間(秒))　(値 3:領域またはウインドウ名/ID)

概要　：画像ファイル(PNG など)の内容と一致する箇所が出現するまで待機します。値 2 が省略されている場合、デフォルトの 30 秒待機します。値 3 にウインドウ名かウインドウ ID、[]で囲った画面領域([10,100,200,500])指定か、検出したい画像ファイルのパスを入れておくとその領域に限定してイメージを検索します。

例 1)イメージ待機　c:¥test.png　10

<イメージファイルの比較> コマンド

書き方：

$比較結果$=イメージファイル比較　(値 1:画像ファイル 1 へのパス)　(値 2:画像ファイル 2 へのパス)

概要　：２つの画像ファイルを比べて同じかどうかを判定します。大きさが異なる場合、大きい画像に対して小さい画像が一致する箇所があれば、その領域を返します。一致する箇所がなければ、[]が結果として返ります。
２つのイメージのサイズがまったく同じ場合はイメージ比較レートの設定は無視されます。

例 1) $比較結果$=イメージファイル比較　c:¥test1.png　c:¥test2.png
計算.img　$比較結果$<>"[]"
メッセージ.img　画像は一致します。一致する領域は$比較結果$です。

メッセージ.!img　画像は一致しません

■■ 変数と配列■■

<変数を登録> コマンド

書き方：

変数　(値 1：変数名)　　(値 2: 変数名に格納する文字列)

変数名=(値 2：変数名に格納する文字列)

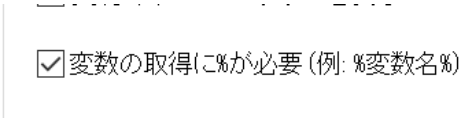
概要　：値 1 のテキストを置き換えたい文字列を値 2 に指定します。
例えば
「変数　$テストファイル$　c:¥test.txt」
と実行してから、
「ファイル開く　　$テストファイル$」
を実行すれば、c:¥test.txt が開きます。

全てのコマンドの引数となる文字列を置き換えてしまうので、変数名には $hogehoge$ のようにあまり使われない文字を使います(逆に文字列を置換する方法として使えます)
変数に空白(ヌル)を登録したい場合は以下のように""に変換するよう定義してください。

例 1)変数$テストファイル$　　c:¥test.txt
例 2)変数$テストファイル$　　""

なお、変数名には[]<>などの括弧、タブ文字は使えません。
$変数名$=文字列　のように書くこともできます。

また、KitSprocket の環境設定では「変数の取得に%が必要」というオプションがあり、データ上の文字列が予期せず変換されてしまいにくくすることができます



※なお、KitSprocket では、初期状態から定義済みの変数があります。定義済みの変数については、この章の最後に記載しています。

<変数を取得> コマンド

書き方：

変数名=[変数取得]　　(値 1:中身を取得したい変数名)　　((値 2:文字列に変数が含まれている場合変換する))

概要　：変数名の中身の文字列を別の変数名に格納します。変数展開　しない　設定のとき、値がなにか指定されていると、文字列の中にある変数名が実際の値に変換されます。

※　$テストデータ$　に　「これは$test$です」と含まれており $test$ には マクロ という言葉が格納されている場合

変数展開 しない

$test$=マクロ

$テストデータ$=これは$test$です

例 1) ABC=変数取得　$テストデータ$　　変換をする

→　ABC　に　「これはマクロです」が格納されます

例 2) ABC=変数取得　$テストデータ$

→　ABC　に　「これは$test$です」が格納されます

<変数を削除> コマンド

書き方：

変数削除　　(値 1:変数名)

概要　：登録されている変数を削除します。例 1 では $テストファイル$という変数を削除します。使用済みの変数が登録されているままだと意図せず文字列が置き換わります。
文字列置換の代わりに変数を定義し、置換が終わったらすぐに削除するという使い方ができます。

例 1)変数削除　$テストファイル$

<変数を比較> コマンド

書き方：

比較　　(値 1:変数名) (値 2:比較記号)　　(値 3:変数名)

概要　：変数またはデータを比較します。符号は < 、>、 <=、 >=、 ！、 <>を使います。文字の比較と異なるのは、数値を比較した場合、その大きさで比較結果を返すという点です。
文字を比較した場合、同じ文字数である場合にも＝が成立。<や> は文字数で比較します。
！では値 1 と値 3 が異なることで成立します。このコマンドの返り値は 1 または 0 です。

例 1)比較.chk　100　=　200
メッセージ.?chk　同じ
メッセージ.!chk　異なる
例 2)比較.chk　あい　<　あいうえお
メッセージ.?chk　右に左が含まれる
メッセージ.!chk　右に左が含まれない
例 3)変数名=比較　100　!　20
メッセージ　変数名　//不一致することが期待値なので、1(成立)が返ります。

<配列を数える> コマンド

書き方：

変数名=[配列数カウント]　　(値 1:配列変数名)

概要　：配列の項目数を数えます。

例 1)
配列=aaa|bbb|ccc
$配列数$=配列数カウント　　配列
メッセージ　　$配列数$

<配列を登録> コマンド

書き方：

変数名=[配列]　(値 1：|で区切られた文字列)

概要　：配列を定義します。|で区切った文字列が配列として扱われます。多次元配列にしたい場合は、||のように区切り文字を 2 倍にしていきます。なお、空白を入れたい場合は、半角スペースを 1 つ入れてください。半角スペース 1 つを扱いたい場合は、半角スペースを 2 個にします。
（例：次は空白| |次は半角スペース 1 つ|　|最後)

例 1)

```
$配列$=配列　　aaa|bbb|ccc
メッセージ　　$配列$[1]
メッセージ　　$配列$[2]
メッセージ　　$配列$[3]
```

<配列に追加> コマンド

書き方：

```
変数名=[配列追加]　　(値 1：追加する文字列)　　(値 2：追加する位置)
```

概要　：配列の任意の位置にデータを追加します。

```
例 1)
$配列$=配列　　aaa|bbb|ccc
配列追加 $配列$ ddd　　2
メッセージ　　$配列$
```

<配列を編集> コマンド

書き方：

```
変数名=[配列編集]　　(値 1：置き換える文字列)　(値 2：置き換える位置)
```

概要　：配列の任意の位置のデータを置き換えます。

```
例 1)
$配列$=配列　　aaa|bbb|ccc
配列編集 $配列$ ddd　　2
メッセージ　　$配列$
```

<配列から取り出し> コマンド

書き方：

```
変数名=[配列取得]　　(値 1:配列変数名)　(値 2：取り出し位置)
```

概要　：配列の任意の位置のデータを取り出します。

```
例 1)
$配列$=配列　　aaa|bbb|ccc
$アイテム$=配列取得　　$配列$ 2
```

```
メッセージ　　$アイテム$
```

<配列を削除> コマンド

書き方：

変数名=[配列削除]　　(値 1:配列変数名)　(値 2：削除する位置)

概要　：配列の任意の位置のデータを削除します。

```
例 1)
$配列$=配列　　aaa|bbb|ccc
$配列$=配列削除　　$配列$ 2
メッセージ　　$配列$
```

<結果を取得> コマンド

書き方：

変数名=[結果取得]　　((値 1:@result@から取得した文字列に変数が含まれている場合変換する))

概要　：KitSprocket ではコマンドの実行結果をすべて@result@という変数に格納しています。この変数の中身の文字列を別の変数名に格納します。このとき、値 2 がなにか指定されていると、文字列の中にある変数名が実際の値に変換されます。

```
※　@result@　に　「これは$test$です」と含まれており $test$ には マクロ という言葉が格納されている場合
変数展開 しない
$test$=マクロ
@result@=これは$test$です
//例 1)
$ABC$=[結果取得]　　変換
　//→　$ABC$　に　「これはマクロです」が格納されます
d=メッセージ　　$ABC$
//例 2)
$DEF$=[結果取得]
　//→　$DEF$　に　「これは$test$です」が格納されます
d=メッセージ　　$DEF$
```

<変数利用時の変換をするかどうか> コマンド

書き方：

変数展開	(値 1:する/しない　または 数字)

概要　：変数を定義したり、利用するときに変数の中身に展開するかどうかを設定します。しないの時は、変数が展開されなくなります。ただし、変数名単体で使う場合はこの設定の影響をうけません→例 1)
値 1 は数字でも指定可能で、0、1 の場合は、変数名そのものの場合は展開しますが、展開後の文字列に含まれる変数は展開しません。2 だと展開後の文字列も展開されます（する の設定は 2 に相当）

設定は初期値として設定されず、スクリプト実行毎に初期値にリセットされます。
※Ver 3.0.7 で仕様を若干変更

```
例 1)
$データ$=あいうえお
$データ 2$=かきくけこ
メッセージ　　$データ$　　// あいうえお　が表示される
メッセージ　　変数は「$データ$」です。　//変数は「あいうえお」です。と表示される
変数展開 しない
メッセージ　　$データ$　　// あいうえお　が表示される
メッセージ　　変数は「$データ$」です。　//変数は「$データ$」です。と表示される
特殊データ=てすと$データ$おわり
メッセージ　　特殊データ　　//てすと$データ$おわり　と表示される
変数展開 する
メッセージ　　特殊データ　　//てすとあいうえおおわり　と表示される
変数展開 0　　//しないと同じ
$テスト$=（$データ$_$データ 2$）
メッセージ　　$テスト$　　//（$データ$_$データ 2$）と表示される
変数展開 1
メッセージ　　$テスト$　　//（$データ$_$データ 2$）と表示される
変数展開 2
メッセージ　　$テスト$　　//（あいうえお_かきくけこ）と表示される
```

<グループの有効化> コマンド

書き方：

グループ有効化　(値 1:有効にしたいグループ名)

概要　：エラー判定されたグループ識別子を再び有効にします。グループ識別子のついたコマンドが再び実行されるようになります。

値 1 で指定したグループをエラー判定されたグループから除外します。ループ処理などで、グルーピングをリセットしたい場合に使います。コロン(,)または縦線(|)で区切ることで複数指定が可能です。

例 1)計算.test1　hogehoge

グループ有効化　test1

メッセージ.test1 hogehoge という計算はできないので test1 はエラー判定となりますが、test1 を有効化したことでこのメッセージは表示されます。

例 2)グループ有効化　　test1,test2,test3

例 3)グループ　　test1

<グループ履歴のリセット> コマンド

書き方：

グループ履歴リセット

概要　：グループ判定結果の履歴をすべて消去します。

例 1)グループ履歴リセット

<グループ指定なしで設定されるグループ> コマンド

書き方：

グループ継続　　(値 1：グループ名)

概要　：グループ指定の初期値を設定します。値 1 を指定しないと、初期化します。
特定のグループ判定がエラーとなった場合に、複数行の処理を実施したくない場合に使います。

例 1)
グループ継続　　doit
メッセージ　　(m1)ここをキャンセルすると doit が無効化される
メッセージ.!doit (m1)をキャンセルされたときに表示される
メッセージ　　(m1)を OK したときにのみ表示される
グループ継続
メッセージ　　このメッセージは必ず表示されます。

■■ 制御や変換(繰り返しなど)■■

<ウエイト> コマンド

書き方：

ウエイト　　(値 1：秒数)

概要　：値で指定し秒数動作を停止します。アプリ起動したあと、実際に操作可能になるまで待ちたい場合などで利用します。

例)ウエイト　　10　　//10 秒停止します

<計算と判定> コマンド

書き方：

変数名=[計算]　(値 1:数式または比較式)　(値 2:結果 TRUE のときの値|結果 FALSE のときの値)

※　計算　の代わりに　判定　というコマンド表記にしても同じ動作をします。

概要　：計算結果を値 2 に定義した変数に記憶します。あるいは、論理式を使うこともできます。論理式(比較とか)の場合は、結果によって変数に入れる値を切り替えることができます(例 3)

四則演算の符号は 足し算が + 、引き算は - 、割り算が / 、掛け算 * 、割り算の「余り」が欲しいときは mod (a,b)、「商」は int(a /b) を使います。

例)
$答$=[計算]　　1+1
計算.calc $答$=2
　　メッセージ.calc　　答は 2 と一致します

```
        メッセージ.!calc    答は 2 と一致しません
例 2)$test$=あいうえお
        $結果$=[計算]      """$test$""=""あいうえお"""    同じ|同じではない
        メッセージ          $結果$
例 4)//割った結果のあまりは mod(割られる数,割る数)とします
       $余り$=mod(5,2)
       メッセージ  $余り$
例 5)//int()でかっこ内の数字を切り捨てで整数にします
$商$=int(5/2)
        メッセージ  $商$
例 6)//round()で四捨五入します
$商$=round(9/2)
        メッセージ  $商$
```

<現在日時の取得> コマンド

書き方：

変数名=[現在日時]　　(値 1:フォーマット)

概要　：現在の日付または時刻を取得できます。値 1 に""と入力するとデフォルトのフォーマットで出力されます。フォーマットは 年 で 4 桁の西暦、月、日、時、分、秒　がそれぞれ実行時の数値に変換されます。
「曜日」または「曜」をフォーマットに入れると、日、月、火、水、木、金、土　のいずれかが返されます。

```
例 1) $日時$=現在日時   年/月/日 時/分/秒/曜日
        メッセージ          $日時$
例 2) $日時$=現在日時   ""
        メッセージ          $日時$
例 3) $日付$=現在日時   年年月月日日
        メッセージ          $日付$          // 2022 年 1 月 2 日　というようになります
```

<日時の比較> コマンド

書き方：

変数名=[日付比較]　　(値 1:日付 1) (値 2:日付 2) (値 3：比較方法)　(値 4:比較の単位)

概要　：値 1 と値 2 の日付を比較します。値 3 を指定しない場合は、=(等しいかどうか)をチェックし、等しくなければエラーとなります。値 4 に 秒、分、時間、日、月、年 のどれか一つ指定すると比較の単位を変更できます。
また、値 2 には 現在日時 や -10 日のように現在日時をベースに指定した日付を指定することができます。

例 1) $秒数$=日付比較.date　　2021/1/1　　2021/1/1
メッセージ.date　同じ日付です。差の秒数:$秒数$
メッセージ.!date　異なる日付です。差の秒数:$秒数$
//「同じ日付です。差の秒数：0」とダイアログが表示されます。

例 2)日付比較.date　　2021/2/1　　2021/1/1　　>
メッセージ.date　1 つめの日付がより未来です。 差の秒数:$秒数$
メッセージ.!date　2 つめの日付が同じか、またはより未来です。差の秒数:$秒数$

<ランダムな数値> コマンド

書き方：

変数名=ランダム (値 1:ランダム数値の範囲)

概要　：値 1 で指定した範囲でランダムな数を生成します。0-100 とすると、0 から 100 までの範囲でランダムな値。4 とだけ指定すると、0 から 3 までの数値となります。

例 1)
ループ開始位置　10
$random$=ランダム　0-100
メッセージ　　$random$
ループ終了位置

<クリップボード取得> / <クリップボード書き込み> コマンド

書き方：

変数名=[クリップボード]　　(値 1:動作[GET / SET])

概要　：

クリップボードから取得するには　クリップボード　　GET

クリップボードに書き込むには　　クリップボード　　SET

クリップボードにテキストを格納したり、取り出したりします。

例 1) $クリップボードの中身$=クリップボード　GET

例 2)クリップボード　　SET　　クリップボードの中に入れたい文字列

＜「ループ開始」と「ループ終了」＞ コマンド

書き方：

ループ開始位置　(値 1:[繰り返し回数　or 繰り返すリスト]) ((値 2:繰り返し要素を格納する変数)　　(値 3:動作のオプション))

・・・(繰り返したい処理を書く)

ループ終了位置

概要　：ショートカットメニューにある「ループ開始」と「ループ終了」で選択範囲を囲む コマンドでは、繰り返したい区間を「ループ開始位置」と「ループ終了位置」という文字で囲みます。
値 1 にループ数またはリストの指定をしないと無限ループとなり、永遠に終わらなくなります。
値 2 を指定しない場合、@loopitem@ という変数名で要素を取り出すことができます。
なお、要素をリストにするには、値 1 のデータを半角の | で区切ってください。 値 3 に区切り文字を入力して | 以外の文字を区切りにすることができます。

※値 1 にフォルダパスを指定した場合のみ。値はタブ、括弧([])を除く 1 文字以上の文字であれば OK です。

また、値 1 は 文字列でもファイルパスでも両方を同じように処理しますが、場合によっては文字としてうまく動かない場合があります。

そのような場合に[あいうえお]と[]で囲むことで必ず文字として処理されるようになります。

※値 3:は動作のオプションを指定できます。
[値 1 を分割する区切り文字,サブフォルダ以下を対象とする]

例 1：メッセージが 3 回表示され、$数値$に現在の回数が表示されます

```
ループ開始位置　3　　$数値$
　　メッセージ　　3 回表示されます$数値$回目
ループ終了位置
```

例 2：異なる文字を順番に表示(要素を 半角の | で区切ってください)

```
ループ開始位置　あいうえお|かきくけこ　　$文字$
　　メッセージ　　$文字$
ループ終了位置
```

例 3：変数名が指定されていない場合、@loopitem@ という変数が予約されています

```
ループ開始位置　あいうえお|かきくけこ
　　メッセージ　　@loopitem@
ループ終了位置
```

例 4：フォルダ指定でワイルドカードを付与すると、そのフォルダ以下のファイルを対象に処理できます。

```
ループ開始位置　c:¥test¥*.txt
　　メッセージ　　@loopitem@
ループ終了位置

ループ開始位置　c:¥test¥*.txt　$ファイル$　　サブフォルダも対象とする
　　メッセージ　　$ファイル$
ループ終了位置
```

例 5：ループ数を変数で指定することもできます

```
変数　　$数字$　　3
ループ開始位置　$数字$　　$現在$
　　$現在$=計算　$現在$*3
　　メッセージ　　$現在$　　// 3 , 6 , 9 と表示される
```

```
ループ終了位置
```

例 6：データを分割する区切り文字を指定することもできます

```
変数　　$文字列$　　あいうえお--かきくけこ--さしすせそ
ループ開始位置　$文字列$　　$現在$　　[--]
　　メッセージ　　$現在$　　// あいうえお → かきくけこ　→ さしすせそ と表示される
ループ終了位置

ループ開始位置　c:¥test¥*.txt_c:¥work¥*.txt　$ファイル$　　[_,SUBFOLDER]
　　メッセージ　　$ファイル$
　　//c:¥test 、c:¥work 以下の txt ファイルとそのサブフォルダをすべて表示します。
ループ終了位置
```

<ループの中断> コマンド

書き方：

```
ループ中断
```

概要　：ループ処理中にループを中断します。

```
例)
ループ開始位置　10　　$現在$
　　メッセージ　　$現在$<5
　　計算.error　　$現在$<5
　　メッセージ.!error　　$現在$<5(ここでエラー)　　エラー
　　ループ中断.!error
ループ終了位置
```

<ループを１つスキップさせる> コマンド

書き方：

```
ループ継続
```

概要　：ループ処理中、そのループの処理を 1 つスキップします。エラー時に処理させない場合で次処理が必要な場合などに使います。

例)

```
ループ開始位置　10　　$現在$
　　計算.error　　$現在$<5
　　ループ継続.!error
　　メッセージ　　$現在$<5
ループ終了位置
```

＜「関数開始」と「関数終了」＞ コマンド

書き方：

関数開始　　(値 1:関数名) ((値 2:～値 4: 引数))

・・・(関数としてまとめたい処理を書く)

関数終了

概要　：ショートカットメニューにある「関数開始」と「関数終了」で選択範囲を囲む　コマンドでは関数として定義したいまとまった処理を定義できます。
登録した関数を呼び出す場合は「関数実行」コマンドを利用します。
値 2 以降は指定しなくても OK。処理の中に 「Return　$変数名$」を入れておくと、$変数名$ の中身が、関数実行 の返り値となります。
関数実行コマンドで返り値を受け取るには $変数名$=関数実行 のような書き方をするか、引数の最後を変数として定義します。
なお、関数内で初めて定義された変数は関数の外からは参照できません。

例)

```
関数開始 FuncName　　$valueOne　　$valueTwo　　$valueThree
$rep$=文字を置換　　$valueOne　　$valueTwo　　$valueThree
return $rep$
関数終了

$result$=関数実行　　FuncName　　これはテストです。　　テスト  TEST
メッセージ　　$result$
```

<関数の実行> コマンド

書き方：

変数名=[関数実行]　　(値 1：呼び出す関数名)　((値 2：引数 1　値 3：引数 2　　値 4：引数 3　値 5：引数 4))

概要　：定義された関数名のものを呼び出して実行します。

(※上記　関数開始　～　関数終了 の例を参照)

<関数の返り値> コマンド

書き方：

Return　　(値 1：関数呼び出し元に返すデータまたは変数名)

概要　：実行時に呼び出し元にデータを返します。関数の定義内でのみ有効です。

(※上記　関数開始　～　関数終了 の例を参照)

<スクリプト終了時実行する関数> コマンド

書き方：

終了時実行　　(値 1：スクリプト終了時に呼び出す関数名)

概要　：スクリプトの実行が終わったタイミングで実行する関数名を設定します。これにより、中断時に設定を元に戻したり、どんなことがあっても必ず最後に表示するメッセージを指定できます。
スクリプト実行時に初期化されるので、毎回指定が必要です。

例)終了時実行　appQuit

関数開始 appQuit

　メッセージ　　終了します

関数終了

<コマンドライン引数を取得> コマンド

書き方：

変数名=[コマンドライン取得]　　(値 1：引数の何番目か？)　((値 2：区切り文字))

概要　：KitSprocket.exe 実行時に渡された引数を取得するコマンドです。
専用ショートカットを作成すると、ショートカットの中身として、KitSprocket.exe の引数にスクリプトファイルのパスが渡される形のコマンドラインになりますが、さらにこのショートカットにファイルをドロップして処理したいような場合、2 番目の引数を処理しなければ

ならなくなります。
値 1 は 1,2,3.. のように指定が可能。値 2 には複数指定された場合の区切り文字を指定できます（指定しない場合 | が区切り文字）

例 1)
$test$=コマンドライン取得　　2
メッセージ　　$test$
実行中止
//※このスクリプトを「専用ショートカット作成」コマンドでデスクトップに保存してから、そのショートカットになにかファイルをドロップすると、ドロップしたファイルのフルパスが表示されます。(2 を 1 にすると保存された上記スクリプト本体のパスとなる。実行中止を最後に配置しているのは、そのまま実行するとドロップしたファイルも実行されるため)

<スクリプト実行を中止> コマンド

書き方：

実行中止

概要　：スクリプトの実行を中断します。エラー発生時のグループ機能を使って、条件に応じて処理を中断するときなどに使います

例 1)※以下の処理では 「終了」というメッセージが表示されません
メッセージ　　開始
実行中止
メッセージ　　終了

例 2) メッセージ.do　　実行しますか？
　　実行中止.!do

■■ 拡張 ■■

<レジストリ操作> コマンド

書き方：

レジストリ　　(値 1:コマンド)　　(値 2:レジストリパス) (値 3:レジストリ名) (値 4:レジストリの詳細設定

概要　：レジストリの読み書きを行います。　値 1 にはコマンドをそれぞれ　読み込み/書き込み/削除/種類取得/存在確認　　を指定し、<レジストリパス>　<レジストリ名>　　(書きこむ場合は)レジストリ種類とデータを[]で囲んで指定します。例：[レジストリ種類|データ]
レジストリの種類は、文字(REG_SZ)、複数行(REG_MULTI_SZ)、展開可能な文字列(REG_EXPAND_SZ)、DWORD、ＱWORD を指定します。
レジストリの(既定)の値をとりたいときは、値３を　"" にしてください。

例 1)ビルド番号=レジストリ　　読み出し
HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion
　CurrentBuild
　メッセージ　　ビルド番号

例２）レジストリパス=HKEY_CURRENT_USER¥Software¥Classes¥CLSID¥{86ca1aa0-34aa-4e8b-a509-50c905bae2a2}¥InprocServer32
レジストリ　　書込み　レジストリパス　　""　　""
メッセージ　　再起動すると、Windows 11 の右クリックメニューが旧仕様になります。

例３）レジストリパス=HKEY_CURRENT_USER¥Software¥Classes¥CLSID¥{86ca1aa0-34aa-4e8b-a509-50c905bae2a2}
レジストリ　　削除　　レジストリパス
メッセージ　　再起動すると、Windows 11 の右クリックメニューが元に戻ります。

<INI ファイル操作> コマンド

書き方：

INI ファイル　　(値 1:コマンド)　　(値 2:INI ファイルパス) (値 3:グループ名) (値 4:変数名)　　(値 5:初期値または書きこむ値)

概要　：INI ファイルを操作できます。値 1 には、読みこみ　または　書込み　を指定。INI ファイルのパスを指定し、グループ名、変数名、値を指定します。読み込む場合は、読み込み失敗した場合の初期値を 値 5 に指定します。

例 1)てすと=INI ファイル　　読み込み　　c:¥Windows¥system.ini　　drivers wave メッセージ　　てすと

■■ その他 ■■

<スクリーンショット> コマンド

書き方：

スクリーンショット　　(値 1：ウインドウ名 または 領域)　(値 2:画像を保存するファイルパス)

概要　：スクリーンショットを取得します。値 1 がウインドウタイトルの場合は、そのウインドウタイトルを持つ一番最初のウインドウの領域が保存されます。値 1 を[]とすると、領域を指定する+カーソルが表示されて、マウスドラッグで範囲を指定できます。(esc キーで範囲指定を中断できます)
値 2 を指定しない場合、保存先はログフォルダで種類は PNG となります。値 2 に指定するファイル名の拡張子で画像種類を指定してください(jpg または png)

例 1)スクリーンショット メモ帳

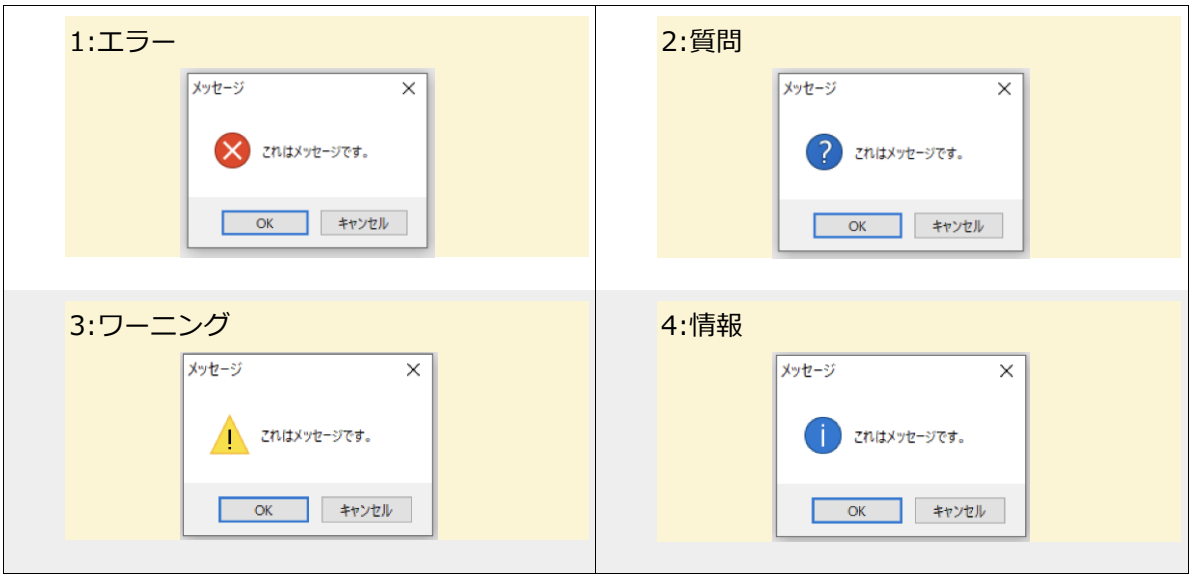
例 2)スクリーンショット [10,10,200,200]　　c:¥temp¥saveimage.png

例 3)スクリーンショット []　　//マウスカーソルが+に変わり、ドラッグで領域を指定できます。esc キーで指定を中断できます。

<メッセージ> コマンド

書き方：

変数名=メッセージ　　(値 1:テキスト)　　((値 2：アイコンタイプ：1～4)　　(値 3：



ダイアログタイプ：0～5))

概要　：メッセージを表示します。

注意)グループ指定がない場合にキャンセルすると、スクリプトが中断されます。

値 2 にアイコンタイプ(1～4)を指定することで、以下のように表示されます。ダイアログによって異なる音が鳴ります。アイコンを指定したくない場合は 0 を入れます。

アイコンタイプは文字でも指定可能です(エラー/質問/ワーニング/情報)

また、値 3 に 0～5 の数値(:の左側)または、文字(:の右側)を入れることで、ボタンの種類を変更できます。

0：OK

1：OK キャンセル

2：中止 再試行 無視

3：はい いいえ キャンセル

4：はい いいえ

5：再試行 キャンセル

メッセージで押したボタンの結果によって、変数名には数値または文字が入ります。上記ボタンの種類を数値で指定すると数値となり、文字で指定すると文字が返ります。

1:OK
2:キャンセル
3:中止
4:再試行
5:無視
6:はい
7:いいえ

例 1)メッセージ　　ハロー！

例 2)メッセージ.do　　コマンドを実行しますか？

シェルコマンド.do　　dir c:¥

例 3)変数名=メッセージ 処理を継続しますか？　2　　4

比較.chk 変数名　6 //はいであるかどうか？

実行.chk domessage

関数開始 domessage

メッセージ　　関数は実行されました

関数終了

例 4)変数名=メッセージ 処理を続けますか　　はい いいえ キャンセル

メッセージ　　変数名

<バルーンメッセージ> コマンド

書き方：

バルーン　　(値 1:テキスト)　　((値 2:アイコン種類) (値 3:音を出す))

概要　：KitSprocket 本体が背面にある場合に実行すると Windows の通知センターでポップアップメッセージを表示します。値 2 に 0～3(0:アイコンなし、1:情報、2:警告、3:停止) を指定すると、アイコンが変化します。
(なし/情報/警告/停止 いずれかの文字を入れるのでも OK)
値 3 になにか文字をいれると音が出るようになります。

※以下のコマンドは KitSprocket が背面にある場合に動作します

例 1)バルーン　ハロー！

例 2)バルーン　ハロー！！　　2　　音あり

例 3)バルーン　ハロー！！！！　　警告

<ツールチップ表示> コマンド

書き方：

ツールチップ　(値 1:テキスト)　((値 2:表示する秒数)(値 3:表示する座標))

概要　：ツールチップを指定秒数表示します。秒数を指定しない場合はツールチップコマンドを再度実行するか、全体スクリプトが終わるか、ほかのコマンドによりツールチップが表示されるまで消えません。座標を指定しない場合は画面右下に表示されます。座標の指定で「マウス～」と入れるとマウスポインタの右横に表示されます。
また、3.0.61 からツールチップ実行後すぐに次の処理が始まるようになっているので注意してください

例 1)ツールチップ　ハロー！

例 2)ツールチップ　ハロー！！！　2

例 3)ツールチップ　ハロー？　2　[100,100]

例 4)ツールチップ　ハロー？　2　マウス位置　//マウス位置にツールチップが表示

例 5)ツールチップ
//表示されているツールチップを消したい場合はツールチップコマンドを引数なしで実行してください

<リストから選択> コマンド

書き方：

変数名=[リスト選択ダイアログ](値 1:リスト項目(| で区切った文字列))　((値 2:キャプション)　(値 3:複数選択))

概要　：値 1 で指定したリスト項目を選択ダイアログで表示します。値 2 と値 3 は省略可能で、最後の引数が変数名として扱われます。値 3 の複数選択は、なにか文字が入っていれば OK。
グループ指定がないときにキャンセルすると、スクリプトが中断されます。

例 1) $結果$=リスト選択ダイアログ　aaa|bbb

例 2) $結果$=リスト選択ダイアログ　aaa|bbb　選んでね

例 3) $結果$=リスト選択ダイアログ　aaa|bbb|ddd　選んでね　複数選択

例 4) $結果$=リスト選択ダイアログ.do　aaa|bbb|ccc　選択してください

メッセージ.!do　キャンセルされました

<キー入力を待機する> コマンド

書き方：

キー入力待機　(値 1:待機するキー) ((値 2:待機をキャンセルするキー))

概要　：指定したキーを待機し、押されたら続行します。なにかキーが押されるまで処理を中断したい時などに使います。キーの指定方法は「キー入力」コマンドの一覧表を参考にしてください。

例 1)キー入力待機　[tab]　[esc]

<ホットキーを登録する> コマンド

書き方：

ホットキー設定　(値 1:設定するキー) ((値 2:関数名またはスクリプトのパス))

概要　：スクリプト実行中に指定したキーを押したときに別のスクリプトまたは関数を実行するように登録します。
何も実行したくない場合（キーを無効化したい場合）は 値 2 に [] を指定してください。解除する場合は、値 1 のみ指定して実行します。

例 1)ホットキー設定　[tab]　"C:¥Users¥Public¥Documents¥KitSprocket¥SampleScripts¥スタートアップ.txt"

<キーボード、マウス操作を記録> コマンド

書き方：

変数名= [操作記録]　　(値 1:キー/マウス/ショートカット/すべて)(値 2:待機時間を記録する/しない)　(値 3:ウインドウ切り替えを記録する/しない)　　(値 4:ウインドウドキュメント名を記録する/しない)

概要　：キーボードとマウス操作を記録します。記録の終了は Ctrl+.キーを押します。キー操作だけにしたい場合は値 1 を　キー　とします。値 2 は操作と操作の空白時間を記録するかどうか、値 3 はウインドウ切り替えを記録するかどうか、値 4 はウインドウ切り替えを記録する場合に、ドキュメント名を含めるかどうかを指定します。
記録された内容は、そのまま KitSporcket で実行できます。
なお、値 1 を「ショートカット」とすることで、 1 つのキーコンビネーションを取得するモードになります。ホットキー設定などで、Ctrl+alt+z など 1 操作のみ取得したい場合はこちらを使います。

例 1)変数名=操作記録　すべて　記録する　　記録しない
メッセージ　　変数名
例 2)変数名=操作記録　ショートカット
メッセージ　　変数名

<ゲーム操作をキー操作として記録> コマンド

書き方：

変数名=[ゲーム操作キー記録]　(値 1:デバイス番号(-1 で実行時に選択))　(値 2:使用するテーブル番号 1~5)　(値 3:最適に動作するための設定ヘッダを追加する)

概要　：ゲームパッドやアーケードスティックの操作をあらかじめ用意したテーブルのキー操作として記録します。記録を終了するには ESC キーを押します。
[テーブル番号]
1 が鉄拳 7 の 1P
2 が鉄拳 7 の 2P
3 がストリートファイター
4 がデッドオアアライブ
5 がログ記録用ボタン表記になります（そのまま実行しても動かない）
となっています。
%appdata%¥Kitworks¥KitSprocket¥ksXinput.ini
を編集することで、割り当てを変更できます。

例 1) 変数名=[ゲーム操作キー記録]　　-1　　3　　ヘッダを追加する

メッセージ　　変数名

<ゲームデバイス番号を取得> コマンド

書き方：

変数名=[コントローラーの検出](値 1:検出を待つ秒数)

概要　：ゲームパッドまたはアーケードスティックを検出するための待機モードに入ります。使用したいゲームパッドまたはアーケードスティックのボタンを押すと終了し、そのデバイスの番号が返ります。ゲーム操作キー記録　のデバイス番号に指定する番号を取得したい場合に使用します。

例 1) 変数名=[コントローラーの検出]

メッセージ　　変数名

<ログ> コマンド

書き方：

ログ　　(値 1:ログに残したい文字列)　　((値 2:日付追加指定))

概要　：指定したテキストをログファイルに記憶します。値 2 に任意の文字('日付追加する'とか'date'とかなんでも OK)を入れると、最後尾に日付と時間が記載されます。
(※日付フォーマットは変更できません)

例 1)ログここで入力テストが終了　　日付付き

<入力モード設定> コマンド

書き方：

入力モード　　(値 1:ウインドウ名) ((値 2:アプリ名 -省略可-)　(値 3:設定したいモード))

概要　：日本語入力モードを変更します。Windows ではウインドウ単位で入力モードが別管理されているので、ウインドウ名　とアプリ名を指定します。ウインドウ名が指定されている場合、アプリ名は省略可能。 モード指定は on/off/ひらがな/全角カナ/半角カナ/全角英数/半角英数と指定します。Off の状態から ひらがな にしたい場合は　ひらがな　とだけ指定します。引数を指定しないと、現在の最前面にあるウインドウの入力モードが取得されます。

例 1)入力モード　無題 – メモ帳　半角カナ

例 2)入力モード　*　　notepad.exe　ひらがな

例 3)現在設定=[入力モード]

<シェルコマンド> コマンド

書き方：

変数名=[シェルコマンド]　　(値 1:コマンド)

概要：コマントプロンプト(Windows) で値 1 のコマンドを実行します。値 2 に変数名を入れておくと、結果を変数に覚えさせておくことができます。値 2 を指定しない場合は、@result@ という変数に格納されます。

※変数名には[]括弧、タブ文字、<>、@は使えません

例)

dir 出力結果=シェルコマンド　dir c:¥

　メッセージ　dir の出力結果

<スクリプト実行> コマンド

書き方：

スクリプト実行　(値 1:[スクリプトファイルのパス] または スクリプトの文字列)

概要：値 1 に書かれていた文字列をスクリプトとして実行します。 []でファイルパスが囲まれていた場合、そのファイルを読み込んで実行します。

処理結果を取得するには、スクリプトの中で宣言されている変数名を使います。

例 1)

//「$結果$=[計算]　　1+1」と書かれた keisan.txt をデスクトップに保存してから実行します。

スクリプト実行　[@デスクトップフォルダ@/keisan.txt]

メッセージ　　$結果$

例 2）

スクリプト="

$結果$=[計算]　1+1

メッセージ　　$結果$

"

スクリプト実行　スクリプト

<タスクスケジューラを登録…> コマンド

書き方：

タスクスケジューラ　　(値 1:コマンド名) (値 2 :タスク名) (値 3:スケジュール指定) (値 4:実行する項目)

概要：タスクスケジューラを制御します。　登録、登録解除、実行、停止、無効化、有効化 と 現在登録されている項目のリストを表示します。 ¥記号を入れずにタスク名を指定した場合、 タスクスケジューラ上の KitSprocket というフォルダに登録されます。値 1 に使えるコマンドは 登録、削除、実行、停止、有効化、無効化 ができます。リストというコマンドで、タスクのリストを取得できます。（初期値は KitSprocket フォルダのリストを取得します）
なおすでにあるタスク名を指定した場合、(1) のような連番が付与されます。上書きはできないためいったん削除してください。

値１：登録、削除、実行、停止、有効か、無効化、リスト
値２：タスク名は ¥test と書くとルートに作成され、test と書くと ¥KitSprocet¥test と作成されます
値３：[スケジュールタイプ,開始時刻,実行間隔,繰り返し間隔]　の形で , で区切って書きます。スケジュールタイプには、日時指定,毎日,毎週,アイドル,ログオン,スタートアップ が使えます。開始時刻 は 7/30 11:05 でも 7 月 30 日 午後 1:00 など比較的自由に書くことができます。
実行間隔 は 毎日の場合 2 と書くと 2 日間隔、毎週では 日月火水木金土 で実行したい曜日を指定します。例えば　火曜日木曜日:2 と :で区切って書くと 2 週間ごとの 火曜日と木曜日となります。曜日指定も Tuesday Friday と書いたり自由に書けます。

繰り返し間隔も 1Hour:1Day のように書くと、 1 時間ごとで継続時間が 1 日 となります。

値４の実行する項目は [プログラムパス,引数,作業フォルダ] の形で指定してください。引数や作業フォルダは省略可能です。
値３、値４の各項目は , で区切られたブロックをさらに + や：で 2 の設定の意味を持たせてあります。例えば 開始日時は指定したくないけど、遅延は設定したい場合
[毎日,+10 分,2,1 時間]
のように書くこともできます。
（繰り返し間隔では継続時間を指定しない場合、継続時間は 1 日となります）

例 1)タスクスケジューラ 登録　　new-task　　[毎日,7/30 11:00+10min,2,1h:1day] [@kitsprocket@,c:¥script.txt]

例 2) タスクスケジューラ　　登録　new-task　　[毎週,7/30 11:00+10min,火曜日 木曜日:2,1h:1day]　[@kitsprocket@,c:¥script.txt]

例３）タスクスケジューラ　　削除　new-task　　//¥KitSprocket¥new-task が削除されます

例４）$タスクリスト$=タスクスケジューラ　　リスト　¥KitSprocket　//¥KitSprocket 以下のタスクリストが　$タスクリスト$に格納されます。

<OS 再起動> コマンド

書き方：

OS 再起動　　((値 1：強制的に行うかどうか？))

概要　：OS を再起動します。値 1 になにか文字が入っていると、保存されていないデータがあっても強制的に再起動します。

例)OS 再起動　　強制再起動

<OS シャットダウン> コマンド

書き方：

OS シャットダウン　　((値 1：強制的に行うかどうか？))

概要　：OS をシャットダウンします。値 1 になにか文字が入っていると、保存されていないデータがあっても強制的にシャットダウンします。コピーが終わるまで帰れない・・・なんて場合は KitSprocket に任せて帰りましょう！

例)OS シャットダウン　　強制シャットダウン

<OS バージョンの確認> コマンド

書き方：

変数名=[OS バージョン]

概要　：現在利用している OS が何かを取得、判定します。Win7 、Win8.1、Win10、Win11、Win2012、Win2012R2、Win2016 まで対応。

※Windows Server 2019 では Win2016 と判定されます

例 1) $OSVER$=[OS バージョン]

メッセージ $OSVER$

<OS ビルド番号の確認> コマンド

書き方：

変数名=[OS ビルド]

概要　：現在利用している OS のビルド番号を数値で返します。

例) $build$=[OS ビルド]

メッセージ　　$build$

<OS アーキテクチャの確認> コマンド

書き方：

変数名=[OS アーキテクチャ]

概要　：現在利用している OS が 32bit 版か 64bit 版かを判定します。値 1 を変数として扱い、現在のアーキテクチャが格納されます。

例 1) $OS アーキテクチャ$=[OS アーキテクチャ]

メッセージ　　$OS アーキテクチャ$

■■ 動作設定 ■■

KitSprocket の動作設定をします。これらのコマンドは引数なしで　変数名=<コマンド名>で実行すると、現在設定値を取得できます。

<実行結果ダイアログの表示> コマンド

書き方：

実行結果を表示　(値 1:する　または　しない)

概要　：全体処理が終了したら表示されるダイアログを表示しないようにすることができます

例)実行結果を表示　　しない

<キー入力の速度調整> コマンド

書き方：

キー入力速度　　(値 1:キー入力間隔　または　キー入力の長さ)　(値 2：5～100 の数字)

概要　：「キー入力」コマンドで文字を入力する速度をミリ秒単位で変更するコマンド。文字がうまく入力されない場合に変更します。

例)　キー入力速度　　キー入力間隔　20

<処理をタイムアウトとする待ち時間の設定(秒)> コマンド

書き方：

タイムアウト　　(値 1:秒数)

概要　：ウインドウ待機やアプリ存在確認など、ある程度処理を待機するコマンドの待ち時間の初期値を設定します。

例)　タイムアウト　　10

<マウスを移動するときの速度> コマンド

書き方：

マウス移動スピード　　(値 1:数値)

概要　：マウス移動コマンドでマウスが移動するときに、どのくらいの速度で移動させるかを設定します。10 でおよそ画面の端から端まで 1 秒程度で移動します。（初期値は 1、最大は 100）

例)　マウス移動ディレイ　　10　　//10 でおよそ画面の端から端まで 1 秒程度で移動します

<実行を中断するキーの設定(秒)> コマンド

書き方：

実行中断キー設定 (値 1:ショートカットキー)

概要　：KitSprocet の実行を中断するショートカットキーを設定します。
値 1 には [ctrl+/]のようにキー入力と同じ表記を行います。実際の入力操作で設定したい場合 は値 1 に[]を指定します。

例 1)　実行中断キー設定 [ctrl+.]

例２）実行中断キー設定　　[]

<変数の変換に符号が必要かどうかを切り替え> コマンド

書き方：

変数変換符号　　(値 1:必要　または　不要)

概要　：環境設定「変数の取得に%が必要 (例: %変数名%)」を変更するコマンド。この設定を変更すると、サンプルスクリプトが動作しなくなりますので注意してください

例)　変数変換符号　　必要

<ループ変数の区切り文字の変更> コマンド

書き方：

ループ引数セパレータ　(値 1:@改行@ など)

概要　：ループ開始 コマンドの引数をリストとして区切る文字を指定できます。

例)ループ引数セパレータ @改行@

<ファイルパスを自動的に修復する> コマンド

書き方：

ファイルパス自動修正　(値 1: する|しない)

概要　：ファイルパスをコマンドに渡すとき KitSprocket はパスが正しいかどうかのチェックを行いますが、この設定を「しない」することで、パスのチェックを行わなくなります。この設定を使うことで大量のファイルを扱いたい場合に、スクリプトの実行速度が高速化します。
実行中のみ有効。記憶されないのでスクリプト毎にコマンドを実行してやる必要があります。

例) ファイルパス自動修正　　しない

<ツールチップの表示の切り替え> コマンド

書き方：

ツールチップ表示(値 1:する　または　しない)

概要　：実行中に右下に表示されるツールチップを表示するかしないかを設定できます。

例)ツールチップ表示　　しない

<ログ保存の切り替え> コマンド

書き方：

ログを保存　　(値 1:する　または　しない　あるいは、0 か 1 の設定)

概要　：実行結果などのログを保存するかしないかを設定します。3.2.2 より仕様を変更して、エラー以外のログを保存しないようにしました。この設定で 1 を指定することで従来のとおり細かいログを保存するようになります。

例 1)ログを保存　しない

例 2)ログを保存　1

<ログ保存フォルダの変更> コマンド

書き方：

ログフォルダ　　(値 1:保存先パス)

概要　：ログやスクリーンショットを保存するフォルダ名のフルパスを指定します。このコマンドの指定がない場合、または指定したフォルダが見つからない場合は、デスクトップに KitSprocket_logs というフォルダが作成されます。

例 1)ログフォルダ　　/Users/takahark/Desktop/test/

<エラー発生時スクリプトを継続しない> コマンド

書き方：

エラー時スクリプト継続 (値 1:する　または　しない)

概要　：グルーピングをしていない場合。ウインドウが見つからなかったり、ボタンクリックを失敗したり、メッセージをキャンセルした場合にスクリプトの処理を続行するかどうかを設定します。（初期値：しない）
過去バージョンの互換性を保持するために存在しますが、基本的には初期値で使っていただくことを推奨します。

例)エラー時スクリプト継続　　する

<画像比較時の一致率の変更> コマンド

書き方：

イメージ比較レート　　(値 1:　0.1 ～ 1　)

概要　：イメージクリックなどで画像から画面上の領域を検出する場合に、どのくらいの一致率で成立させるかを指定します。0.1～1 の範囲で設定します。1 にすると、完全一致しないと検出されなくなります。初期値は 0.7 です。

例)イメージ比較レート　0.8

<ファンクションキーの使用> コマンド

書き方：

ファンクションキー使用 (値 1：する/しない)

概要　：KitSprocket が起動中、ファンクションキーの押下によって指定したスクリプトを実行する機能を ON/OFF することができます。

例)ファンクションキー使用　　しない

<ファンクションキーへのスクリプト登録> コマンド

書き方：

ファンクションキー設定 (値 1:F1～F10)　　(値 2:スクリプトファイルパス)　　(値 3:有効化しない)

概要　：KitSprocket が起動中、ファンクションキーの押下によって指定したスクリプトを実行します。KitSprocket が前面にいなくても機能します。設定を削除するには、値 1 のみ指定します。この設定は引数なしで利用しても現在設定を返しません。

例 1) ファンクションキー設定　F4　　c:¥temp¥メッセージ.txt　　//有効化して、スクリプトを割り当てます

例 2) ファンクションキー設定　F4　　c:¥temp¥メッセージ.txt　　OFF　　//スクリプトを割り当てますが、有効化しません

例 3)ファンクションキー設定　F4　　//設定を削除(無効化)します

<スタートアップスクリプトパスの設定> コマンド

書き方：

スタートアップスクリプト　　(値 1:スクリプトファイルのパス)

概要　：スタートアップスクリプトのパスを設定できます。

例)スタートアップスクリプト　　@ホームフォルダ@¥スタートアップ.txt

<実行を中断するキーの設定> コマンド

書き方：

実行中断キー設定　　(値 1:キー操作)

概要　：KitSprocket　の実行を中断するキーの組み合わせを指定できます。キーの書き方については、キー入力コマンドの表を参照してください
値 1 を指定しないで実行すると、キー操作を自分で入力して指定できます

例)実行中断キー設定　　[esc]

コマンドのグルーピングについて

■コマンドのグルーピング

概要　：コマンド名に「.abc」のように拡張子をつけることで、コマンドのグルーピングができます。同じグループ名を持つコマンドを、エラーが発生した場合に実行したくない場合は、同じ拡張子をつけます。

＜判定用の符号について＞
拡張子の先頭に「!」をつけると否定となり、エラーが発生した場合にのみ実行されるコマンドになります。(例：メッセージ.!abc)
また「?」を先頭にすることで、そのコマンドの実行結果がグルーピングに影響しないように設定できます。

＊＊＊？の使い方＊＊＊

data=ファイル選択.test ファイルを選択してください

メッセージ.?test　　ファイルが選択されました

メッセージ.!test　　ファイルが選択されませんでした

// 上記 1 つめの メッセージをキャンセルすると、本来ならば test がエラー発生のグループと再判定されてしまい、2 つめのメッセージが表示されてしまいます。
//このため、1 つめのメッセージのグループに?を付けることで、ファイル選択 のときの test の判定を保持できるようになり、正しくファイルを選択した場合は、2 つめのメッセージが実行されなくなります。

なお、拡張子の先頭に「@」をつけることで、そのグループの初期化が行われます。たとえば、ループで繰り返し実行する場合、最初のコマンドに@付きのグループ名をつけます。
※「?」と「!」と「@」は半角文字で入力してください

＜書式＞

(コマンド名).グループ識別子

概要　：コマンド実行結果がどうなったかを判定する。同じ記号を別のコマンドに付与することで同じグループとなり、コマンドが失敗した時点で、そのグループのコマンドは実行されなくなります。

例 1：

ボタン.kana　実行

メッセージ.kana　　実行しました!

メッセージ.!kana　　実行できませんでした　//ボタン操作が失敗するとこちらのメッセージが表示

例 2：

メッセージ.test　キャンセルしてください(結果がエラーになります)

メッセージ.@test　　キャンセルされても表示されます

```
メッセージ.test  キャンセルされると表示されません
```

その他　：v3 より、以下のように書くことで、グルーピングを省略できるようになりました。

```
<コマンド 0>                引数
(
        <コマンド 1 >
        <コマンド 2 >
)(
        <失敗したときのコマンド>
)
```

※<コマンド 0> の位置のコマンドが実行結果で成功したときに最初のブロック（コマンド 1、コマンド 2）を実行します。　失敗した場合は、<失敗したときのコマンド>の位置に書かれたコマンドが実行されます。

■定義済みの変数

このアプリでは、あらかじめ制御用の文字や環境の定数などが定義されています。

定数	実際の値	説明または書き方例
@result@	実行結果を変数で取得するタイプのコマンドで変数が指定されていなかった場合に結果が格納される値	シェルコマンド　dir c: メッセージ　@result@
@loopitem@	ループで変数が指定されてない場合の現在の値	ループ開始位置 3 メッセージ　@loopitem@ ループ終了位置
@改行@	改行コード	メッセージ　今日@改行@明日
@LF@	改行コード LF	主に Unix 系 OS で使われている改行コード
@CR@	改行コード CR	特殊な OS で使われている改行コード
@タブ@	タブ文字	メッセージ　今日@タブ@明日
@ユーザ名@	ログインしているユーザ名	@ユーザ名@
@コンピューター名@	コンピュータ名	@コンピューター名@
@作業フォルダ@	スクリプトファイルを読み込んでいる場合、その親となるフォルダ	@作業フォルダ@
@デスクトップフォルダ@	デスクトップフォルダのパス	@デスクトップフォルダ@
@ホームフォルダ@	ホームフォルダのパス	@ホームフォルダ@
@display_dpi@	高解像度ディスプレイなどでの文字拡大率	メッセージ　@display_dpi@
@AppDataCommonFolder@	C:¥ProgramData	ショートカットメニューの 定義文字 以下に CommonAppDataFolder から呼び出せます
@DesktopCommonFolder@	C:¥Users¥Public¥Desktop	ショートカットメニューの 定義文字 以下に CommonDeskTopFolder から呼び出せます。すべてのユーザーのデスクトップに表示されるファイルを置く場所です
@DocumentsCommonFolder@	C:¥Users¥Public¥Documents	ショートカットメニューの 定義文字 以下に CommonDocumentsFolder から呼び出せます。すべてのユーザーのドキュメントに表示されるフ

定数	実際の値	説明または書き方例
		ァイルを置く場所です
@StartMenuProgramsCommonFolder@	C:¥ProgramData¥Microsoft¥Windows¥Start Menu¥Programs	ショートカットメニューの 定義文字 以下に CommonStartMenuProgramsFolder から呼び出せます。すべてのユーザーのスタートメニューのアプリ一覧に表示されるファイルを置く場所です
@StartMenuCommonFolder@	C:¥ProgramData¥Microsoft¥Windows¥Start Menu	ショートカットメニューの 定義文字 以下に CommonStartMenuFolder から呼び出せます。すべてのユーザーのスタートメニューに表示されるファイルを置く場所です
@StartupCommonFolder@	C:¥ProgramData¥Microsoft¥Windows¥Start Menu¥Programs¥Startup	ショートカットメニューの 定義文字 以下に CommonStartupFolder から呼び出せます。すべてのユーザーのログイン時に実行されれるプログラムファイル(またはショートカット)を置く場所です
@AppDataFolder@	C:¥Users¥<UserName>¥AppData¥Roaming	ログインしている<UserName>のアプリケーションの管理フォルダです
@LocalAppDataFolder@	C:¥Users¥<UserName>¥AppData¥Local	ログインしている<UserName>のアプリケーションの管理フォルダです
@DesktopFolder@	C:¥Users¥<UserName>¥Desktop	ログインしている<UserName>のデスクトップフォルダです

※上記以外にも対応している変数があります。詳しくはショートカットメニュー「定義文字」を開いて確認してください。

特殊な書き方

■変数名=文字列

- コマンド名より前のタブ文字、スペース(全角スペース含む)を無視します

- コマンド直後の引数はスペース区切りでも区切りとなります。ただし 値 2 の引数はタブ区切りでなければならない
- 変数名=計算式　と書くことですばやく 変数に値を格納できるように対応。""や[]で囲むことで動作を変更できます

例 1) $変数$=1+1　・・・ $変数$ に計算結果　2 が格納されます

例 2) $変数$="1+1"　・・・ $変数$ には　1+1 という文字列が格納されます

例 3) $変数$=現在日時　・・・ $変数$ には　現在日時 という文字列が格納されます

　$変数$=[現在日時]　・・・ $変数$ には　現在日時 コマンドの結果(2021/5/26 ...) が入ります

例 4) $変数$="2021/1/1 10:11"　・・・ スペースを含みたい場合は、前後を " で囲みます

例 5) $変数$="テストと""問題"""　・・・ 前後を " で囲った場合に " を途中で入れるには、"" と書くことが必要です →　上記は「テストと"問題"」というデータになります

コマンド別名

KitSprocket では 1 つのコマンドにいろいろな別名を用意しています。これによりコマンド名をうろ覚えで思い出すことができ、よりストレスのないスクリプト作成を実現します。

例：アプリ実行　notepad.exe　を　run notepad.exe と書くことができます。

マニュアル記載のコマンド名	別名
「アプリ起動」	"アプリ実行" or "exec" or "run" or "launch"
「アプリ待機」	"アプリが起動するのを待つ" or "アプリを待つ" or "起動待機" or "waitprocess" or "waitingstart"
「アプリ存在確認」	"アプリの存在を確認" or "アプリ存在チェック" or "checkprocess" or "processcheck" or "existsapp"
「アプリ強制終了」	"アプリを強制終了" or "forcequitapp"
「アプリ終了待機」	"アプリが終了するのを待つ" or "終了待機" or "waitdisappear" or "waitingquit"
「起動アプリ情報」	"実行アプリ情報" or "processinfo" or "runappinfo"
「起動アプリリスト」	"実行アプリリスト" or "プロセスリスト" or "processlist" or "runapplist"
「ウインドウ待機」	"ウインドウの出現を待つ" or "waittitle" or

	"waitwindowtitle"
「ウインドウ存在確認」	"ウインドウ確認" or "ウインドウの存在を確認" or "existswindow" or "windowexists"
「ウインドウ前面」	"ウインドウ最前面" or "windowactive"
「ウインドウ」	"ウインドウ切り替え" or "ウインドウを最前面に移動" or "アクティベート" or "activate" or "switchwindow"
「ウインドウ最小化」	"windowminimize" or "minimizewindow"
「ウインドウ最大化」	"windowmaximize" or "maximizewindow"
「ウインドウ復元」	"ウインドウ復帰" or "restorewindow" or "windowrestore"
「ウインドウ非表示」	"ウインドウ隠す" or "hidewindow"
「ウインドウ表示」	"showwindow"
「ウインドウ閉じる」	"ウインドウを閉じる" or "closewindow" or "windowclose"
「ウインドウ移動」	"ウインドウサイズ変更" or "movewindow" or "sizewindow"
「ウインドウ閉じるのを待つ」	"ウインドウ閉じるまで待つ" or "処理を待つ" or "waitprocessclose" or "waitwindowclose"
「ウインドウサイズ」	"ウインドウの現在位置" or "ウインドウ位置" or "windowize" or "windowpos"
「ウインドウ情報」	"ウインドウの情報" or "windowinfo"
「ウインドウリスト」	"ウインドウ名リスト" or "listwindow" or "windowlist"
「ウインドウ数変化待機」	"ウインドウが増えるのを待つ" or "waitwindowcount" or "waitwincount"
「メニュー」	"メニュー操作" or "menu"
「貼り付け」	"ペースト" or "paste"
「ボタン」	"ラジオ" or "ラジオボタン" or "button" or "radio" or "radiobutton"
「チェックボックス」	"チェック" or "checkbox" or "check"
「エディット」	"エディットフィールド" or "入力枠" or "edit" or "editfield"
「エディット追加」	"エディットフィールド追加" or "入力枠追加" or

	"editadd" or "editfieldadd"
「リストボックス」	"コンボボックス" or "リストビュー" or "ドロップダウンメニュー" or "listbox" or "combobox" or "listview" or "list" or "dropdownmenu"
「タブ」	"タブ切り替え" or "タブ切替" or "tab" or "selecttab"
「ラベル」	"テキストチェック" or "文字列チェック" or "テキスト" or "スタティックテキスト" or "label" or "static" or "text" or "findtext"
「マウスクリック」	"クリック" or "click" or "mouseclick"
「マウスダブルクリック」	"ダブルクリック" or "dclick" or "doubleclick"
「マウストリプルクリック」	"トリプルクリック" or "tclick" or "tripleclick"
「マウス右クリック」	"右クリック" or "rightclick" or "rclick"
「マウス移動」	"ポインタ移動" or "mousemove" or "pointermove" or "pmove"
「マウスドラッグ」	"ドラッグ" or "mousedrag" or "drag"
「右マウスドラッグ」	"マウス右ドラッグ" or "rightmousedrag" or "rdrag"
「マウスホイール」	"ホイール" or "wheel"
「マウスダウン」	"マウス下げる" or "mousedown"
「マウスアップ」	"マウス上げる" or "mouseup"
「マウスをもどす」	"マウスをもとの位置に戻す" or "restoremousepos" or "restoremouse"
「マウスの現在位置」	"マウス位置" or "マウス座標" or "mousepos"
「キー入力」	"文字入力" or "キー送信" or "textinput" or "keyinput" or "keywait"
「文字入力ダイアログ」	"文字入力パネル" or "入力パネル" or "inputdialog"
「文字数」	"文字数を数える" or "文字を数える" or "文字数カウント" or "countstring" or "stringlen"
「文字の位置」	"文字を見つける" or "文字を比較" or "stringcompare" or "stringinstr"
「文字を置換」	"文字置き換え" or "文字置換" or "replace" or "replacestring"
「文字を切り取り」	"文字切り取り" or "文字をカット" or "cutstring"
「文字を切り取りした残り」	"文字をカットした残り" or "残りの文字" or

	"cutstringremain" or "stringremain"
「文字並べ替え」	"文字を並べ替え" or "並べ替え" or "文字ソート" or "sort" or "sortstring"
「文字コード」	"文字コードを取得" or "ユニコード" or "stringid" or "stringtounicode"
「コードから文字」	"ユニコード文字" or "codetostring" or "tostring"
「16 進数に変換」	"HEX に変換" or "hex" or "converttohex"
「10 進数に変換」	"DEC に変換" or "dec" or "converttodec"
「ファイル開く」	"フォルダ開く" or "open" or "openfile" or "openfolder"
「ファイル移動」	"ファイルを移動" or "フォルダ移動" or "フォルダを移動" or "movefile" or "filemove" or "moveFolder" or "foldermove"
「ファイルコピー」	"ファイルをコピー" or "フォルダコピー" or "フォルダをコピー" or "copyfile" or "filecopy" or "copyfolder" or "foldercopy"
「ファイル削除」	"ファイルを削除" or "フォルダ削除" or "フォルダを削除" or "deletefile" or "filedelete" or "deletefolder" or "folderdelete"
「ファイル選択」	"ファイルを選択" or "selectfile" or "fileselect"
「ファイル名取得」	"ファイル名を取得" or "ファイル名" or "フォルダ名取得" or "フォルダ名を取得" or "フォルダ名" or "getfilename" or "filename" or "getfoldername" or "foldername"
「親フォルダ取得」	"親フォルダを取得" or "親フォルダ" or "getparentfolder" or "getpfolder"
「ファイル名変更」	"フィル名を変更" or "フォルダ名変更" or "フォルダ名を変更" or "changefilename" or "changefoldername"
「ファイル情報」	"ファイルの情報を取得" or "フォルダ情報" or "フォルダの情報を取得" or "fileinfo" or "folderinfo"
「ファイル確認」	"ファイル存在確認" or "existsfile" or "fileexists"
「フォルダ確認」	"フォルダ存在確認" or "existsfolder" or

	"folderexists"
「ファイル読み込み」	"ファイル読込" or "読み込み" or "read" or "readfile" or "fileread"
「ファイル書き込み」	"ファイル書込" or "ファイル保存" or "filewrite" or "writefile" or "savefile"
「ファイル消去」	"ファイル初期化" or "消去" or "cleanup"
「ファイルリスト」	"ファイルリストを取得" or "filelist" or "listfile"
「フォルダリスト」	"フォルダリストを取得" or "folderlist" or "listfolder"
「フォルダ作成」	"フォルダを作成" or "createfolder" or "makefolder" or "foldercreate"
「フォルダ選択」	"フォルダを選択" or "selectfolder"
「URL を開く」	"サイトを開く" or "openurl"
「サーバーに接続」	"ファイルサーバーに接続" or "connectserver"
「インターネット読み込み」	"サイトを読み込み" or "inetread" or "readinternet"
「インターネットダウンロード」	"サイトをダウンロード" or "ダウンロード" or "inetdownload" or "downloadinternet"
「専用ショートカットの作成」	"専用ショートカット作成" or "createoriginalshortcut"
「イメージチェック」	"イメージ確認" or "imagecheck" or "checkimage" or "isinimage"
「イメージへ移動」	"イメージの場所へ移動" or "imagemoveto" or "movetoimage"
「イメージクリック」	"イメージをクリック" or "imageclick" or "clickimage"
「イメージダブルクリック」	"イメージをダブルクリック" or "imagedclick" or "dclickimage"
「イメージ右クリック」	"イメージを右クリック" or "imagerightclick" or "rightclickimage"
「イメージ待機」	"イメージを待機" or "イメージの出現を待機" or "waitimage" or "imagewait"
「イメージファイル比較」	"画像ファイル比較" or "compareimagefile" or "twoimagefiles"
「変数」	"変数追加" or "変数を追加" or "val" or "dim" or "local" or "variable" or "newvariable"

「変数取得」	"変数を取得" or "getval" or "getvariable"
「変数削除」	"変数を削除" or "delval" or "removeval" or "removevariable"
「比較」	"変数比較" or "変数を比較" or "compare" or "compval" or "compareval"
「配列数カウント」	"配列数" or "arrayMax" or "arraysize"
「配列」	"配列作成" or "配列を作成" or "array" or "newarray"
「配列追加」	"配列編集" or "arrayadd" or "arrayedit"
「配列取得」	"配列検索" or "arrayget" or "arraysearch"
「配列削除」	"arraydelete" or "arraydel"
「結果取得」	"結果" or "getresult" or "getresultvariable" or "result"
「変数展開」	"usevariable" or "useval"
「グループ有効化」	"enablegroup"
「グループ履歴リセット」	"グループ履歴初期化" or "resetallgroup"
「グループ継続」	"keepgroup" or "currentgroup"
「ウエイト」	"スリープ" or "ディレイ" or "待機" or "遅延" or "一時停止" or "wait" or "sleep" or "delay"
「計算」	"判定" or "ダイレクトコマンド" or "AutoIt スクリプト" or "executedirect" or "autoitscript"
「現在日時」	"現在時刻" or "今日" or "currentdate" or "currenttime" or "today" or "currentdatetime" or "now"
「日付比較」	"日付を比較" or "日時比較" or "時刻比較" or "時間比較" or "comparedate" or "datecompare"
「ランダム」	"乱数" or "random"
「クリップボード」	"clipboard"
「ループ開始」	"ループ開始位置" or "ループ" or "loop" or "startloop" or "loopstart" or "for"
「ループ終了」	"ループ終了位置" or "endloop" or "nextloop" or "loopend" or "next"
「ループ中断」	"ループ中止" or "ループ脱出" or "ループから出る" or "exitloop"

「ループ継続」	"スキップ" or "コンティニュー" or "skip" or "continue"
「関数開始」	"繰り返し利用開始" or "Func" or "Sub"
「関数終了」	"繰り返し利用終了" or "EndFunc" or "EndSub"
「関数実行」	"関数呼び出し" or "実行" or "ExecFunc" or "Call"
「return」	"返り値" or "関数の返り値"
「終了時実行」	"終了マクロ" or "onquit" or "onexit"
「コマンドライン取得」	"起動引数取得" or "getarg" or "getcmdline"
「実行中止」	"実行中断" or "abort" or "stop"
「INI ファイル」	"ini" or "inifile"
「レジストリ」	"reg" or "registory"
「スクリーンショット」	"スクリーンキャプチャ" or "screenshot" or "screencapture"
「メッセージ」	"メッセージボックス" or "文字を表示" or "msg" or "message" or "msgbox" or "display"
「バルーン」	"ポップアップメッセージ" or "balloon" or "balloonmessage" or "popupmessage"
「ツールチップ」	"tooltip"
「リストから選択」	"リスト選択ダイアログ" or "choosebox"
「キー入力待機」	"入力待機" or "waitinput"
「ホットキー設定」	"ホットキー" or "hotkey" or "hotkeyset"
「ホットキー設定取得」	"gethotkey"
「ホットキー削除」	"ホットキー解除" or "hotkeydelete" or "deletehotkey" or "removehotkey"
「操作記録」	"操作を記録" or "recordall" or "recordoperation" or "guirecording"
「ゲーム操作キー記録」	"ゲーム操作をキー操作で記録" or "recxinput" or "recordXinput"
「コントローラーの検出」	"checkxinput" or "checkgamecontroller"
「ログ」	"ログとして保存" or "memo" or "log" or "savelog"
「入力モード」	"入力モード設定" or "日本語モード" or "inputmode" or "setinputmode"
「入力モード取得」	"日本語モード取得" or "getinputmode"

「シェルコマンド」	"コマンドプロンプト" or "shellcommand" or "commandprompt"
「スクリプト実行」	"runscript"
「タスクスケジューラ」	"タスク管理" or "taskscheduler"
「OS 再起動」	"OS を再起動" or "再起動" or "restart" or "reboot" or "osrestart"
「OS シャットダウン」	"OS をシャットダウン" or "シャットダウン" or "shutdown" or "osshutdown"
「OS バージョン」	"システムバージョン" or "osver" or "systemversion" or "sysver"
「OS ビルド」	"OS ビルド番号" or "ビルド番号" or "osbuild" or "buildno"
「OS アーキテクチャ」	"アーキテクチャ" or "checkarch" or "architecture"

キットワークス株式会社では、製品に関するご意見・ご要望を広くお待ちしております。
今後の製品への参考とさせていただきますので、ぜひ皆様のご意見をお聞かせください。
ご意見・ご要望は、下記のキットワークスホームページで承っております。
https://kitworksinc.com/
https://kitworksinc.com/オリジナルアプリ/sprocket/

KitSprocket

発行者　木下幹司
発行所　キットワークス株式会社
〒 770-8024 徳島県徳島市西須賀町鶴島 47-13
第 3.2.6 版