PIPEOCX

WILL

株式会社ウィル

目　次

はじめに

動作環境について

■対応 OS

PIPEOCX は、以下に示す OS で動作確認を行っております。

Microsoft Windows 2000、Microsoft Windows XP、
Microsoft Windows 2003、Microsoft Windows VISTA

■開発に必要なソフトウエア

PIPEOCX をご使用いただくには、以下のいずれかのソフトウエアが必要です。

Microsoft Visual Basic Ver 6.0
Microsoft Office 2003 (Access、Excel)
Microsoft Visual Studio 2005

PIPEOCX は、Microsoft Visual C++ Ver 6.0 で作成しています。サンプルは、Microsoft Visual Basic Ver 6.0 または VS.NET2003 または 2005 で作成しています。
※ 本製品は日本語環境のみの対応となります。

インストール

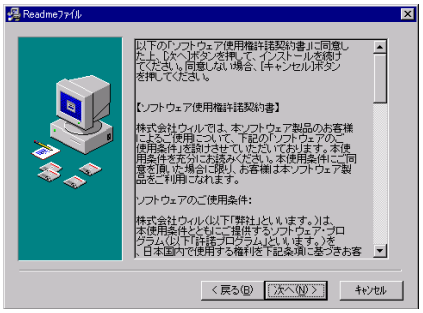
ダブルクリックします。

製品のCD-ROMに含まれているセットアップキット（PIPEOCX.exe）を

画面にしたがって、インストールを進めて下さい。

1. インストールを始めます。「次へ」をクリックして下さい。

2. 使用許諾契約書です。内容に同意される場合は「次へ」をクリックして下さい。



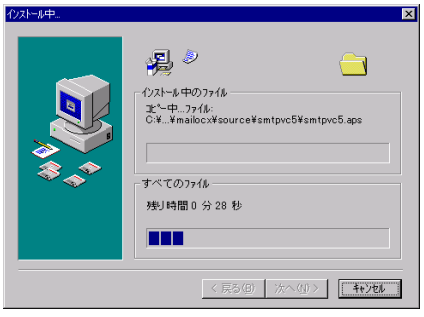
3. インストール先のフォルダを指定します。初期設定でよろしければ「次へ」をクリックして下さい。別のフォルダを指定したい場合は「参照」をクリックし、フォルダを指定して下さい。

4. インストール中に置換されるファイルのバックアップを作成できます。そのバックアップファイルの保存先フォルダを指定します。初期設定でよろしければ「次へ」をクリックして下さい。

5. WILL を登録するスタートメニュー又はプログラムマネージャのグループフォルダを指定します。初期設定では、新規に「WILL」の名前でフォルダを作成します。特に指定する必要がなければ、初期設定をお勧めします。

6. プログラムのコピーを開始します。「次へ」をクリックして下さい。

7. プログラムのコピーをしています。中断する場合は、「キャンセル」をクリックして下さい。



8. インストールが完了しました。「完了」をクリックし、インストールを終了して下さい

ライセンスの登録

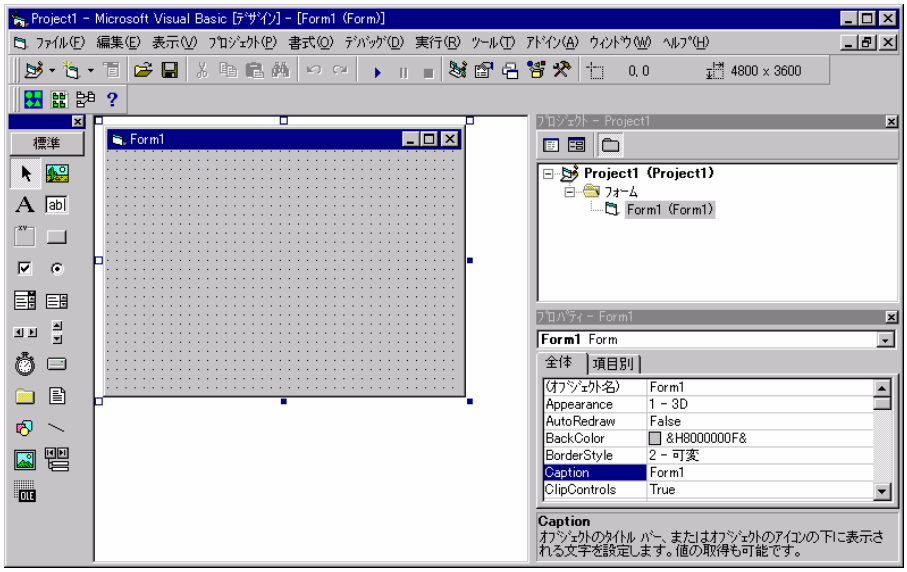
■レジストリファイルから登録する

ライセンスを登録します。製品に含まれている CD のレジストリファイル(EFXXXXXXXXX.reg)をダブルクリックして下さい。(「XXXXXXXXX」は、任意の数字がファイル名として付けられています。)

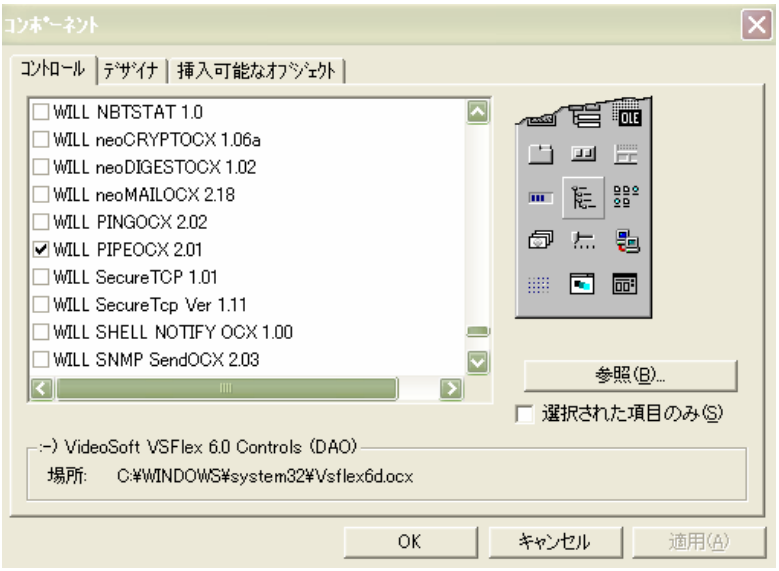
以下のメッセージボックスが表示され、ライセンスがレジストリに登録されます。

■手動で登録する

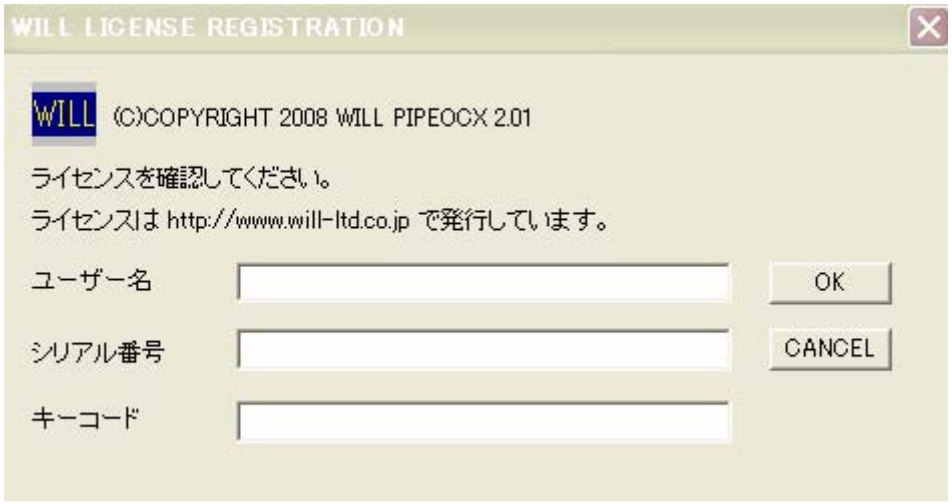
あらかじめ電子メールで通知しているライセンス情報を利用してライセンスを登録する等、レジストリファイルを利用しない場合は、VisualBasic 起動後に新規プロジェクトを選択し以下のデザイン画面を開きます。



ツールバーの「プロジェクト」から、「コンポーネント」を選択し、「コンポーネント」画面を開きます。次にコントロールタブの一覧から PIPEOCX を選択して「OK」をクリックすると、PIPEOCX がツールボックスに追加され、アイコンが表示されます。



ツールボックスに追加された PIPEOCX を選択し、フォームにアイコンを貼り付けると、以下の「WILL LICENSE REGISTRATION」画面が表示されます。ここで、ユーザー名、シリアル番号、キーコードをそれぞれ入力してライセンス登録を行います。



■トライアルライセンスから正規ライセンスへの移行

既にトライアルライセンスが登録されている場合には、デザイン画面にあるPIPEOCXのプロパティで「バージョン情報」をクリックして下さい。

「WILL LICENSE REGISTRATION」画面が表示されますので、ここで正規ライセンスを入力して下さい。

■ライセンス入力時のご注意

※ライセンスが入力できない！？

入力したライセンスにスペースが含まれていないか確認して下さい。（ライセンスに、スペースは使用していません。）

※登録したライセンスを認識しない！？

ライセンスを登録しても、オブジェクトが新規ライセンスを認識していない場合は、PIPEOCX のアイコンを少し動かして下さい。この作業により、オブジェクトにライセンスが記憶されます。

※トライアルライセンスで作成したアプリケーションはどうする！？

既にトライアルライセンスで作成したアプリケーションは、正規ライセンスを登録した後、再コンパイルする必要があります。

サポートについて（無償）

サポートは基本的に電子メールで受け付けております。サポートは無償でご利用いただけます。

■お問い合わせの前に

サポート作業を円滑に行うために、お問い合わせの際には以下の情報をご用意下さい。

1. 製品名及びバージョン
2. 開発環境（OS の種類及びバージョン、サービスパッケージの種類）
3. 開発ツール及びバージョン
4. サーバーの種類
5. 問題点
   (1) エラー内容又は、エラー状況のハードコピー
   (2) 問題点となる部分のサンプルソースコード
   (3) Trace イベントの msg の内容

■お問合せ先

http://www.will-ltd.co.jp/で御確認ください。

バージョンアップについて（無償）

製品のバージョンアップは、無償です。

■バージョンアップ情報の入手方法

バーションアップの情報は、弊社ホームページの新着情報で通知し、各商品のページの更新履歴で更新内容を掲示致します。

■最新バージョンの入手方法

最新バージョンのプログラムは、弊社ホームページ（http://www.will-ltd.co.jp/）のダウンロードのページよりダウンロードすることが出来ます。ダウンロードするファイルは、以下のバージョンアップの目的により異なりますのでご注意下さい。

再配布について

■作成したアプリケーションの配布時

PIPEOCX を利用して作成したアプリケーションの配布時のランタイムライセンスはフリーです。但し、開発ライセンスの配布はできません。

■再配布時に必要な配布可能ファイル

PIPEOCX を利用して作成したアプリケーションを配布する場合には、以下のファイルを添付する必要があります。

- PIPEOCX201.OCX

■著作権

プログラミング概要

子プロセスを開始する

子プロセスを開始するには、Run メソッドを使用します。子プロセスが起動されると、子プロセスからの出力が、Recv イベントで通知されます。また、Send メソッドを使うと、子プロセスにデータを送信することができます。

子プロセスが起動中かどうかは、IsRunning メソッドで確認することができます。

子プロセスを終了する

子プロセスに終了を伝えるには、Close を実行してください。これにより、子プロセスの標準入力が閉じられ、子プロセスはデータがこれ以上ないことを知り、子プロセスの処理が終わった時点で自主的に子プロセスを終了します。

この方法で、子プロセスが終了しないときは、KILL メソッドにより、子プロセスを強制的に終了されることができます。ただ、子プロセスの子プロセス（孫プロセス）が実行中であれば、子プロセスは終了できません。このような場合は、Terminate メソッドにより、子プロセスを切り離すことができます。

子プロセスが終了すると、Done イベントが発生します。

メソッドの発行

■Send メソッドの発行

Send メソッドは、子プロセスにデータを送信します。正常に送信すると Sent イベントが発生します。

Send メソッドは、データをすべて送信し終わるまで、次のデータを送信することができません。

データを送信可能かどうかは、Sendable メッソドで確認することができます。

■Close メソッドの発行

Close メソッドは、子プロセスへの送信を終わることを子プロセスに通知します。入力待ちをしているコンソールアプリケーションはこれを合図に終了します。

しかし、notepad のような入力待ちをしていないアプリケーションの場合は、Close メソッドを呼び出しても終了しませんの、この場合は、KILL メソッドで終了させます。ただし、PIPEOCX は、notepad のような標準入出力を扱わないアプリケーションは想定していません。

■Kill メソッドの発行

Kill メソッドは、子プロセスを強制終了させます。

たとえば、Ping.exe を終了条件なしに起動した場合(-t)、Close メソッドを行っても Ping.exe は終了しません。このときは、Kill メソッドでプロセスを終了してください。

■Terminate メソッドの発行

Terminate メソッドは、子プロセスを起動したまま、子プロセスとの通信を終わらせたいときに使用します。CLose メソッドで終了しないアプリケーションであれば、引き続き子プロセスは動作し続けることができます(例：notepad.exe)。

プログラミング例

■ 例 1：CMD.EXE の TIME コマンドを呼び出し、その後 CMD.EXE を終了する

```
PIPEOCX1.Run “CMD”
PIPEOCX1.Send “TIME /T” & vbCrLf & “EXIT” & vbCrLf
```

【解説】
Run メソッドで CMD.EXE を起動し、CMD.EXE に TIME /T コマンドと EXIT コマンドを入力しています。CMD.EXE は TIME/T コマンドで時刻を表示し、EXIT コマンドでプロセスを終了します。CMD.EXE では、コマンドの区切りは vbCrLf ですので、コマンドの後ろに vbCrLf を付加しています。

2 つのコマンドを送信する場合、2 つのメソッドに分けて送信することもできます。この場合は、

```
PIPEOCX1.Send “TIME /T” & vbCrLf
PIPEOCX1.Send “EXIT” & vbCrLf
```

とは書く事はできません。これは、最初の Send による送信が完了するまで、次の Send を呼び出すことが出来ないためです。最初の Send が終わったことを知るには、Sendable イベントが True であるか、Sent イベントが発生したことを確認してください。

```
PIPEOCX1.Send “TIME /T” & vbCrLf
Do While PIPEOCX1.Sendable = False
    DoEvents
Loop
PIPEOCX1.Send “EXIT” & vbCrLf
```

この中の DoEvents は必須です。DoEvents を呼び出さないと、Sendable プロパティは変化しません。

■ 例 2：TIME コマンドで/T オプションを指定しない場合

```
PIPEOCX1.Run “CMD”
PIPEOCX1.Send “TIME /T” & vbCrLf & vbCrLf & “EXIT” & vbCrLf
```

【解説】
入力待ちに対し、2 つ目の vbCrLf により改行コードを入力しています。このように、入力待ちに対しデータを入力することができます。
ただし、たとえば、FTP.EXE でユーザーID の入力待ちのようにコンソールからの入力を待つ場合には、PIPEOCX の Send メソッドではデータの入力ができません。FTP.EXE の場合には、ユーザーID とパスワードを書いたファイルを用意し(userinof.txt とする)、

```
FTP –s::userinfo.txt ftp-server-address
```

のように起動すれば、ログインまでは自動的に実行された後、FTP コマンドを PIPEOCX で入力することができます。

■ 例 3:終了コードを得る

```
PIPEOCX1.Run “CMD”
PIPEOCX1.Send “EXIT 12” & vbCrLf

Private Sub PIPEOCX1_Done(ByVal Code As Long, ByVal ErrCode As Long)
    Debug.Print ″PIPEOCX1_Done Code=″ & Code & ″,ErrCode″ & ErrCode
End Sub
```

【解説】
コマンドの終了コードは、Done イベントの ErrCode 変数に格納されます。(Code が 0 の場合、ErrCode は子プロセスの終了コードを示している)
CMD.EXE の場合、CMD.EXE から起動したプログラムの終了コードを ERRORLEVEL 変数に格納しています。CMD.EXE から起動したプログラムの終了コードを PIPEOCX で受け取るには、プログラム起動直後に”EXIT %ERRORLEVEL%”を CMD.EXE に渡します。

```
PIPEOCX1.Run “CMD”
PIPEOCX1.Send “COPY A B” & vbCrLf & “EXIT %ERRORLEVEL%” & vbCrLf
```

上記プログラムでは、COPY コマンドが成功したかどうかを PIPEOCX の Done イベントの ErrCode 変数で判定することができます。

■ 例 4：CMD.EXE 経由でバッチファイルを起動する

```
PIPEOCX1.Run “CMD”
PIPEOCX1.Send “A.bat” & vbCrLf & “EXIT” & vbCrLf
```

【解説】
CMD.EXE にバッチファイルを起動させています。
コマンドラインから入力するのと同じですので、複数のバッチファイルを続けて起動することも可能です。

```
PIPEOCX1.Send “A.bat” & vbCrLf & “B.bat” & vbCrLf & “EXIT” & vbCrLf
```

■ 例 5：直接バッチファイルを起動する

```
PIPEOCX1.Run “A.BAT”
```

【解説】
直接バッチファイルを起動する場合、”EXIT”コマンドを付加する必要はありません。

■ 例 6:出力を処理する

```
Private LINEEND As String
Private Sub PIPEOCX1_Received(Msg As String)
    Static Remain$   ‘ 改行コードで終了していない最終行
    Dim p As Long    ‘ 改行コードの先頭文字の位置。なければ 0
    Dim q As Long    ‘ 改行コードの次の文字
    LINEEND = vbCrLf ‘ 改行コード(Windows の場合)
    Msg = Remain & Msg ‘ 前回のデータの残りを受け取る
    q = 1 ‘ 最初の文字は 1 番目
    p = InStr(q, Msg, LINEEND) ‘ 改行コードを検索する
    Do While p > 0 ‘ 改行が見つかる限り続ける
        Debug.Print Mid$(Msg, q, p - q)  ‘ ---途中行---
        q = p + Len(LINEEND) ‘ 次のデータは改行コードの直後の文字
        p = InStr(q, Msg, LINEEND)  ‘ そこから改行コードを検索する
    Loop
    Remain = Mid$(Msg, q)  ‘ 最後の改行コード以降のデータ
    If Remain <> “” Then
        Debug.Print Rmain  ‘ ---最終行(たぶん続くデータがある)---
    End If
End Sub
```

【解説】

子プロセスから来るデータは、複数行まとめて受け取ることになります。

また、最後の行が改行コードで終わっているとは限りません。

改行コードは Windows のコマンドのように vbCrLf を使うものもあれば、cygwin のコマンドのように vbLf を使うものもあります。上記のコードでは、LINEEND という変数で表現していますが、Windows の場合は vbCrLf を cygwin の場合は vbLf を代入すればどちらも対応できます。

受け取ったデータを改行で区切っていくと、最後の行は改行で終わる場合と、改行で終わらない場合があります。改行で終わらない場合は、変数 Remain に記録されます。残りの行を受け取ったときは、その行の前に、Remain を付けて処理をします。

■ 例 7:出力を処理する

```
Private LINEEND As String
Private Sub PIPEOCX1_Received(Msg As String)
    Static Remain$
    Dim arr, n&, i&
    LINEEND = vbCrLf ‘ 改行コード(Windows の場合)
    Msg = Remain & Msg
    arr = Split(Msg, LINEEND)    ‘ 改行コードで分割する
    If arr(n) = “” Then ‘ 最終行が改行コードで終わっているか？
        Remain = ””    ‘全部処理をしたので残りはない
        ReDim Preserve arr(n - 1)    ‘ 空行を arr から削除する
    Else
        n = UBound(arr)    ‘ 最終行の位置
        Remain = arr(n)    ‘ 最終行を記録する
        ReDim Preserve arr(n - 1) ‘ 最終行を arr から削除して Remain に移す
    End If
    For i = LBound(arr) To UBound(arr)
        Debug.Print arr(i) ‘ ---改行コードで終わっている行---
    Next
    If Remain <> “” Then
        Debug.Print Rmain    ‘ ---Remain に残っているデータ---
    End If
End Sub
```

【解説】
例 6 を split を使って書き直したものです。
arr(0)から arr(n)までに受信したデータが入っています。Remain には最終行が改行コードで終わっていない場合に、最終行のデータが入っています。これは次回の受信時に合わせて処理されます。

出力情報を取り扱うクラスの詳しい例はサンプルをご覧ください。

■ 例 8：CMD.EXE で複数コマンドを連続して起動したときの処理

```
PIPEOCX1.Run “CMD”
PIPEOCX1.Send “PROMPT ---UniqLine--$_” & vbCrLf & “CD” & vbCrLf _
      & “DIR” & vbCrLf & “EXIT” & vbCrLf
```

【解説】
複数のコマンドを起動すると、出力データがどのコマンドのものか見分けが付かなくなります。このようなときは、PROMPT を変更して、出力の区切りにします。PROMPT を使うには ECHO は ON の状態にしておいてください。
上記の PROMPT は、「---UniqLine--」の後に改行($_)されます。ですから、最初の「---UniqLine--」までが CME.EXE の起動メッセージ、次が、CD の結果、その次が DIR の結果となります。

■ 例 9：PowerShell を起動する

```
PIPEOCX1.Run Environ("WINDIR") & _
"¥system32¥WindowsPowerShell¥v1.0¥powershell.exe -command -"
PIPEOCX1.Send "echo 'Hello'" & vbCrLf & "exit" & vbCrLf
```

【解説】

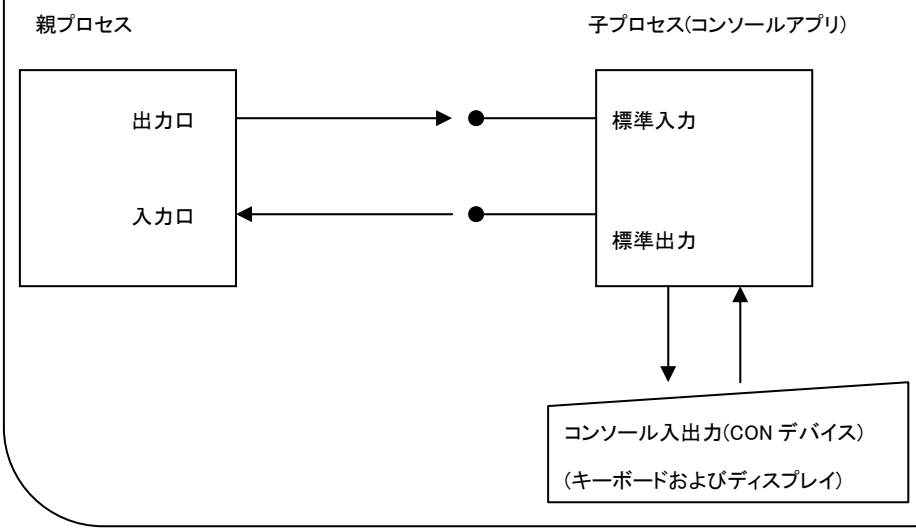
PowerShell を起動するときは、引数に「-command -」を付けて起動します。

■ プロパティの活用

Priority プロパティは、子プロセスの優先度を指定できます。緊急性のない処理の場合は、IDLE_PRIORITY_CLASS か BELOW_NORMAL_PRIORITY_CLASS を指定し、緊急度の高い処理は、ABOVE_NORMAL_PRIORITY_CLASS か HIGH_PRIORITY_CLASS を指定してください。
なお、HIGH_PRIORITY_CLASS を指定すると CPU を占有しますので、十分テストしてから指定するようにしてください。

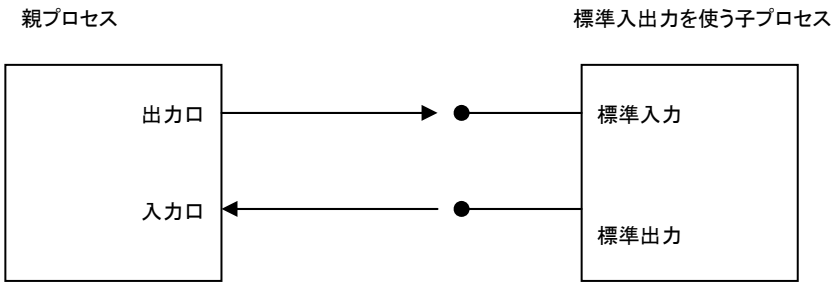
Style プロパティは、子プロセスのウィンドウを表示するかどうか指定できます。デフォルトでは子プロセスのウィンドウは開きませんが、SHOWNORMAL を指定するとウィンドウが表示されます。
CMD.EXE の場合は、ウィンドウを表示しても、なにも表示されませんが、コンソールウィンドウの役割を果たしていて、PIPEOCX から「echo OK > con」を send するとコンソール画面に OK の文字が表示されます。

「Type con」を行うとコンソールからの入力待ちになります。コンソールウィンドウから文字列を入れた後、^Z(コントロールキーと Z を同時に)を押してください。(例: 12345^Z)



■ プロセスの終了

PIPEOCX で起動した子プロセスは、



のように、親プロセスの出力口が子プロセスの標準入力につながり、子プロセスの標準出力が親プロセスの入力口につながります。データの流れとしては、親プロセスの出力口から送信されたデータが子プロセスの標準入力に送られ、そのデータを子プロセスが処理して、子プロセスの標準出力から親プロセスの入力口に向けて出力されます。
子プロセスは、親プロセスからの入力データが来なくなったときに、データの終了と判定します。一般的にこの時点で子プロセスは終了します。
PIPEOCX は、入力データの終了を Close メソッドで通知することができます。上記の図でいうと、親プロセスの出力口を閉じることになります。その結果、子プロセスの標準入力のデータが来なくなり、子プロセスはデータの終わりと判断して終了します。
これが、PIPEOCX が想定している一般的な終了方法です。すなわち、親プロセスは送信するデータがなくなると、Close メソッドを呼び出し、それに連動して子プロセスが終了します。標準入力からデータを読み込むプログラムは、この方法で終了させることができます。たとえば、CMD.EXE や PowerShell.EXE などがこれにあたります。

もちろん、Close メソッドを呼び出すとどんな子プロセスも終了するかといえば、そうではありません。Close メソッドでは、親プロセスのデータが来なくなったという意味しか伝えることができませんので、子プロセスはこれを無視することができます。たとえば、バッチプログラムは、親プロセスが Close メソッドを呼び出したとしても、自分がすべき処理をすべて行うまでは、終了しません。バッチのような終了の仕方を、自主的な終了と読んでいます(弊社の造語です)。

子プロセスが終了すると、子プロセスの標準出力が閉じられ、その結果、親プロセスは入力口からデータが来なくなることを知ります。このとき、PIPEOCX は Doen イベントを発生します。

上記の 2 つの終了の仕方をしない場合は、強制的に終了させるしかありません。強制的に終了させるには、Kill メソッドを使います。Kill メソッドを呼び出すと、子プロセスを強制終了させます。これは、TerminateProcessAPI を使って終了させていますので、子プロセスはリソースの開放などが出来ない可能性がありますので、この点は良く理解して使ってください。サンプルでは、ping.exe を子プロセスとして起動したとき、自主的に終了しない場合に、Kill メソッドで終了させるようにしています。
この場合でも、子プロセスの標準出力は閉じられますので、Done イベントが発生します。

Kill メソッドを使っても子プロセスが終了しないケースがあります。子プロセスが起動したプロセスが生きているケースなどです。その場合には、子プロセスとの通信を強制的に切断する方法があります。Terminate メッソドは、親プロセスの入力口を強制的に閉じます。

■ コンポーネントの終了

子プロセスを起動した状態でコンポーネントを終了するとき、PIPEOCX は下記の処理を行います。

1. 子プロセスからのデータを読み取り処理を停止します。
   ただし、Tarminatable プロパティが True(デフォルト)の場合のみ。

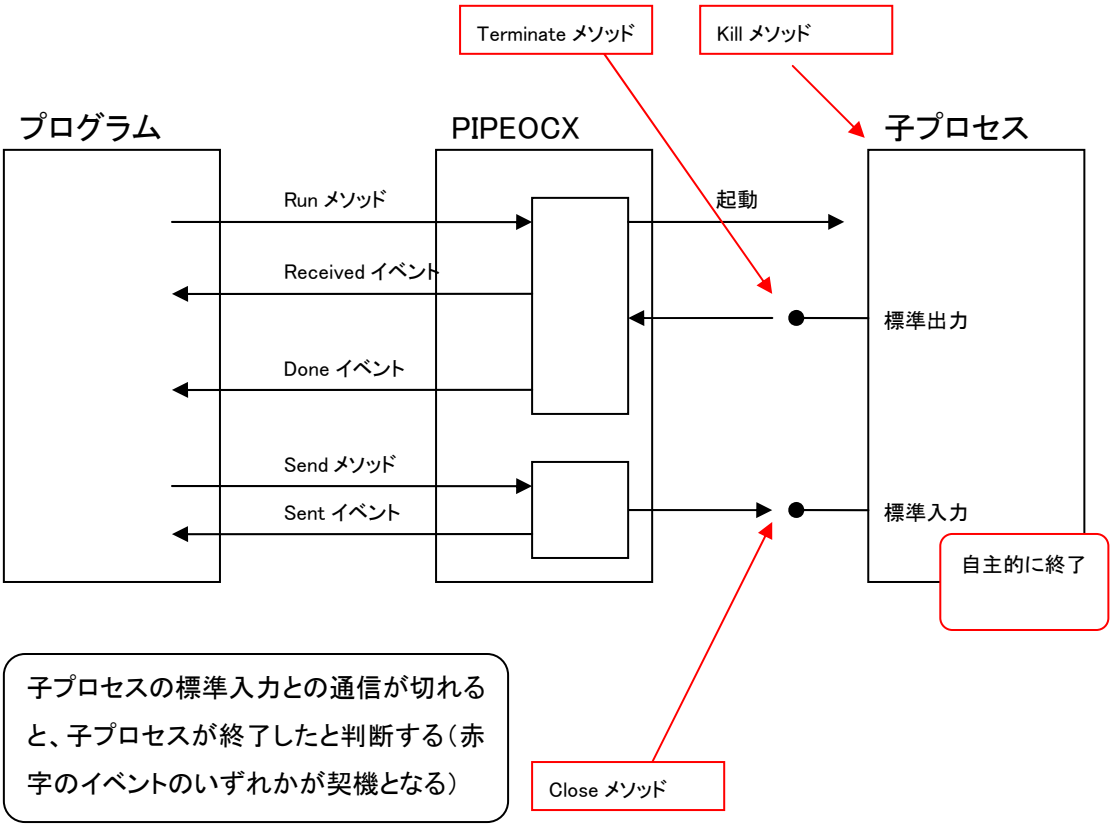
2. 親プロセスの出力口(子プロセスの標準入力)を閉じます。

3. 子プロセスを TerminateProcess します。
   ただし、DontKill プロセスが False(デフォルト)の場合のみ。

4. 親プロセスの入力口(子プロセスの標準出力)を閉じます。
   この時、子プロセスが終了していない場合、この処理は子プロセスの終了を待ちますので、親プロセスはハングしたような状態になり、子プロセスが終了するまで、親プロセスは終了できません。なお、Terminatable プロパティが True の場合は、このような状況になる(ハングする)ことはありません。

> Terminatable プロパティ
> Close メソッドを呼び出すと確実にプロセスが終了するタイプのアプリケーションを子プロセスとして呼び出す場合に限り、Terminatabe プロパティを False にすることができます。
> Terminateble プロパティを False にすると、親プロセスの読みこみスレッドにおけるデータ取得のレスポンスが若干向上します。

状態遷移図


Terminate メソッド
Kill メソッド
プログラム
PIPEOCX
子プロセス
Run メソッド
起動
Received イベント
標準出力
Done イベント
Send メソッド
Sent イベント
標準入力
自主的に終了
子プロセスの標準入力との通信が切れると、子プロセスが終了したと判断する（赤字のイベントのいずれかが契機となる）
Close メソッド

プロパティ

Copyright プロパティ

■機　能

　PIPEOCX　のコピーライト文字列。プログラム、ヘルプファイル、マニュアルなどのいずれかにこのプロパティの文字列を表示してください。この値は参照のみ可能です。

■構　文

Object. Copyright

Copyright プロパティの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■データ型

文字列(String)

Directory プロパティ

■機　能

子プロセスのワーキングディレクトリを指示します。

■構　文

Object. Directory[=Value]
Directory プロパティの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
Value	ワーキングディレクトリを指定する文字列式です。

■データ型

文字列(String)

DontKill プロパティ

■機　能

PIPEOCX の Unload 時に起動中の子プロセスを終了させるかどうか指示する。

■構　文

Object. DontKill[=Value]
DontKill プロパティの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
Value	子プロセスを終了させるかどうか指示する論理値です。

■設定値

Value の設定値は次のとおりです。

(値)	(説　明)
True	Unload 時に起動中の子プロセスを終了させる。(初期値)
False	Unload 時に起動中の子プロセスを終了させません。

■データ型

ブール型(Boolean)

Priority プロパティ

■機　能

子プロセスの優先度を指示します。デフォルトは親プロセスと同じ優先度。

■構　文

Object.Priority[=Value]
Priority プロパティの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
Value	子プロセスの優先度を指定する長整数です

■設定値

Value の設定値は次のとおりです。

(値)	(説　明)	
NORMAL_PRIORITY_CLASS	&H20	標準の優先度
IDLE_PRIORITY_CLASS	&H40	システムのアイドル時に実行
HIGH_PRIORITY_CLASS	&H80	CPU サイクル独占
BELOW_NORMAL_PRIORITY_CLASS	&H4000	標準より下の優先度
ABOVE_NORMAL_PRIORITY_CLASS	&H8000	標準より上の優先度

■データ型

長整数(Long)

Style プロパティ

■機　能

子プロセスのウィンドウスタイルを指示します。

■構　文

Object. Style[=Value]

Style プロパティの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
Value	子プロセスのウィンドウスタイルを示す長整数です。

■設定値

Value の設定値は次のとおりです。

(値)	(説　明)
HIDE	0　非表示(初期値)
SHOWNORMAL	1　通常はこれを指定する
SHOWMINIMIZED	2　最小化
SHOWMAXIMIZED	3　最大化
SHOW	5　同じ位置と大きさで表示

■データ型

長整数(Long)

Terminatable プロパティ

■機　能

Close メソッドを呼び出すと確実にプロセスが終了するタイプのアプリケーションを子プロセスとして呼び出す場合に限り、Terminatabe プロパティを False にすることができます。
Terminateble プロパティを False にすると、親プロセスの読みこみスレッドにおけるデータ取得のレスポンスが若干向上します。

子プロセスを実行中は、値の変更をすることが出来ません。実行中に設定を行うと、29931 エラーが発生します。

■構　文

Object. Terminatable[=Boolean]
Terminatable プロパティの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
Boolean	Terminate メソッドを利用するかどうかを決めます。

■設定値

Value の設定値は次のとおりです。

(値)	(説　明)
True	Terminate メソッドを利用します。(初期値)
False	Terminate メソッドを利用しません。

■データ型

ブール型(Boolean)

メソッド

Cancel メソッド

■機　能

送信中のデータを中断します。
送信中のデータがある場合は、TRUE が返ります。
送信中のデータがない場合は、FALSE が返ります。
子プロセスを起動していない場合は、29927 エラーが発生します。

■構　文

Object. Cancel()
Cancel メソッドの構文の指定項目は次の通りです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■戻り値

戻り値(Boolean)が示す値は次のとおりです。

(値)	(説　明)
True	中断を指示しました。
False	中断の必要はありません。

Close メソッド

■機　能

子プロセスの標準入力を閉じます。これによる子プロセスは親プロセスからのデータがこれ以上来ないことを知ることができます（End Of Data）。

たとえば、cmd.exe を引数なしで起動すると標準入力からの入力待ちになります。Cmd.exe への入力がすべて終わったとき、Close メソッドを呼び出すと、cmd.exe は入力の終わりと判断して、終了します。
また、PowerShell を引数「-command -」をつけて起動すると標準入力からの入力待ちになります。PowerShell への入力がすべて終わったとき、Close メソッドを呼び出すと、PowerShell は入力の終わりと判断して、終了します。

なお、データ送信中の場合は、送信が完了してから、子プロセスの標準入力を閉じます。
Close メソッドが成功すると、そのプロセスに対してはデータを送信することはできません。
子プロセスを起動していない場合は、29927 エラーは発生します。

■構　文

Object.Close()
Close メソッドの構文の指定項目は次のとおりです。

（指定項目）	（内　容）
Object	PIPEOCX オブジェクトです。

■戻り値

戻り値(Boolean)が示す値は次のとおりです。

（値）	（説　明）
True	Close メソッドは成功しました。
False	Close メソッドは成功しませんでした。

ErrorMessage メソッド

■機　能

Err.Number が 29900 のエラーが発生した場合、さらに詳しいエラーメッセージを得ることができます。

■構　文

Object.ErrorMessage()

ErrorMessage メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■戻り値

文字列(String)

ErrorNumber メソッド

■機　能

　Err.Number が 29900 のエラーが発生した場合、さらに詳しいエラーコードを得ることができます。

■構　文

Object.ErrorNumber()

ErrorNumber メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■戻り値

長整数(Long)

IsRunnung メソッド

■機　能

　Run メソッドで子プロセスを起動してから、Done イベントが発生するまで、このメソッドは True を返します。
　子プロセスを起動中は、Run メソッドを呼び出す事はできませんので、Run メソッドを呼び出す前に、このメソッドで子プロセス起動中かどうか、確認してください。

■構　文

Object.IsRunnig()
IsRunnig メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■戻り値

戻り値(Boolean)が示す値は次のとおりです。

(値)	(説　明)
True	子プロセスを起動中のため、Run メソッドは利用できません。
False	Run メソッドを利用可能です。

Kill メソッド

■機　能

起動している子プロセスを強制的に終了させます。

強制終了ですので、アプリケーションによってはデータをセーブできないなどの問題がでるかもしれません。子プロセスとして呼び出すアプリケーションをよく確認してから、このメソッドを呼び出してください。

なお、子プロセスを起動していないときに呼び出すと、29927 エラーを発生します。

また、強制終了できないプロセスの場合、29900 エラーが発生する可能性がありますので、この場合は、ErrorMessage を参照してください。

■構　文

Object.Kill()

Kill メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■戻り値

なし。

Run メソッド

■機　能

引数で指定したプログラムを起動します。

起動時に、Priority プロパティ、Style プロパティ、Directory プロパティ、Terminatable プロパティを参照します。

プロセスの起動の過程でエラーを検出すると、29900 エラーが発生しますので、この場合は、ErrorMessage を参照してください。

また、すでに Run メソッドで子プロセスを起動中は、29901 エラーが発生します。

子プロセスが終了するか、Terminate メソッドで子プロセスの切り離しを行うと、Done イベントが発生します。Done イベントが発生すると、再び Run メソッドを呼び出すことができます。

複数の子プロセスを起動する場合は、PIPEOCX のインスタンスを複数用意して、それぞれの Run メソッドを呼び出してください。

■構　文

Object.Run(Command As String)

Run メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内 容)
Object	PIPEOCX オブジェクトです。
Command	ディレクトリのパスを指定する文字列式です。

■戻り値

なし。

Send メソッド

■機　能

子プロセスに対し、データを送信します。

子プロセスを起動していない場合や、前回の Send メソッドによるデータ送信が完了していない場合は、29925 エラーが発生します。

なお、Sendable メソッドを使うと、送信可能かどうかわかります。

送信データは、Unicode 文字列です。内部で ShiftJis に変換して送信しています。送信データサイズの制限は、文字変数の大きさの制限に依存しますが、内部では 1024 バイトに分割して送信します。

StopSend メソッドにより送信の中断を行うことができるのは、1024 バイトごとの送信後のタイミングになります。

■構　文

Object.Send (Msg As String)

Send メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内 容)
Object	PIPEOCX オブジェクトです。
Msg	送信するデータを指定する文字列式です。

■戻り値

なし。

Sendable メソッド

■機　能

Send メッソドによる送信可能か知らせます。

■構　文

Object.Sendable()
Sendable メソッドの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。

■戻り値

戻り値(Boolean)が示す値は次のとおりです。

(値)	(説　明)
True	Send メッソドは可能です。
False	Send メソッドは利用できません。

Terminate メソッド

■機　能

子プロセスとの通信を切断します。
Done イベントが発生します。
Terminatable が True でなければなりません。そうでない場合は、29930 エラーが発生します。
また、子プロセスを起動していない場合は、29927 エラーが発生します。

■構　文

Object.GetFile(File As String, [LocalFile As String])
GetFile メソッドの構文の指定項目は次のとおりです

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
File	取り出したいリモートファイルを指定する文字列式です。
LocalFile	GetOpen イベントの SaveFileName パラメータとして使われる文字列式です。

■戻り値

なし。

イベント

Done イベント

■機　能

Run メソッドにより起動した子プロセスとの通信を切断したときに発生します。
発生原因は、
　子プロセスが終了した。
　子プロセスとの通信でエラーが発生した
　Terminate メソッドを呼び出した
があります。
それぞれに応じて Code の値が変わるので、イベントハンドラで必要に応じて確認をしてください。

■構　文

Private Sub Object_Done(Code As Long, ErrCode As Long)
Done イベントの構文の指定項目は次のとおりです。

(指定項目)	(内 容)		
Object	PIPEOCX オブジェクトです。		
	正常終了	通信エラー	Terminate メソッド
Code	0	1	2
ErrCode	プロセスの終了コード	Win32API エラーコード	0

Received イベント

■機　能

子プロセスから受信したデータを通知します。
Msg は Unicode 文字列です。
複数行を一度に受信することがあります。
改行コードは、子プロセスの改行コードに依存します。
たとえば、CMD.EXE や PowerShell.exe は vbCRLF が改行コードですが、cygwin のコマンドの場合は、vbLF です。
子プロセスの文字コードは ShiftJIS であると仮定しているので、子プロセスの出力を ShiftJis に変換してください。変換しない場合は、文字化けが生じます。

■構　文

Private Sub Object_Received(Msg As String)
Received イベントの構文の指定項目は次のとおりです。

(指定項目)	(内　容)
Object	PIPEOCX オブジェクトです。
Msg	子プロセスの出力した文字列。

Sent イベント

■機　能

Send メソッドで指示したデータの送信が完了したことを通知します。
送信の完了の原因は、
　送信は正常に完了した
　送信途中でエラーが発生した
　StopSend メソッドで送信を中断した
があります。
また、Close メソッドが実施されている場合、このイベントの発生時点で Sendable メソッドは FALSE を返し、この子プロセスに対する送信は出来ません。

■構　文

Private Sub Object_Sent (ByVal Code As Long, ByVal ErrCode As Long)
Sent イベントの構文の指定項目は次のとおりです。

(指定項目)	(内　容)		
Object	PIPEOCX オブジェクトです。		
	正常終了	Cancel メソッド※	通信エラー
Code	0	0	1
ErrCode	0	1	Win32API エラーコード

※ Cancel メソッドが呼ばれても、正常に送信できた場合は、正常終了を返します。
Cancel メソッドが呼ばれて、未送信のデータがある場合に、Cancel メソッドの状態になります。

エラーコード

29900	Win32API エラーです。ErrorNumber メソッドを参照してください。
29901	すでに起動中です。
29925	起動前および送信中は、送信できません。
29926	送信中は停止できません。
29927	子プロセスがありません。
29929	引数が正しくありません。
29930	このメソッドは、Terminatable が True でなければ、使えません。
29931	子プロセスを実行中は変更できません。

PIPEOCX マニュアル

2008 年 2 月 20 日 初版第 1 版
発行　株式会社ウィル
http://www.will-ltd.co.jp/