前田 和寛

株式会社ミガロ.

システム事業部 システム2課

Delphi/400のモジュールバージョン管理手法

バージョンチェック用アプリケーションで
バージョンの比較と最新モジュールの取得方法を紹介する。

略歴
1983 年 11 月 21 日生まれ
2006 年大阪工業大学情報科学部卒
2006 年 04 月株式会社ミガロ . 入社
2006 年 04 月システム事業部配属

現在の仕事内容
Delphi/400 を利用したシステムの受託開発・保守対応などを担当している。

1. なぜExeの置き換えが必要?

通常、システムを運用 / 稼働していると、アップデート（バージョンアップ）が必要不可欠である。機能拡張、あるいは不具合の修正など理由は多岐に渡るが、運用当初のままの状態でシステムを稼働し続けることは難しい。

例えば、Windows の OS などに代表されるパッケージソフトは、Service Pack のように常に新しいパッチプログラムでアップデートが適用される。このパッチプログラムはインターネット経由でユーザーに配布されたり、個別のインストーラーの形で提供されたりすることが多い。

他方、各企業で独自に開発している業務アプリケーションにおいてはどうなるだろうか。

同様のアップデートの仕組みを実現する場合には、もちろん別途に配布を行わなければならず、その仕組みそのものを構築する必要がある。

Exe（モジュール）

アップデートを実施する場合に気を付けなければならないのが、最新の Exe（モジュール）をいかに各クライアント PC に適用するかである。

アプリケーションを機能拡張した場合、新しい Exe を、各クライアント PC の古い Exe と置き換える必要がある。

その時、プログラムの変更部分が Exe だけであれば、クライアント PC の Exe を順次置き換えるという対応をがとれ、問題が生じる可能性も小さい。

しかし、IBM i（AS/400）など、データベース・サーバー側のプログラム（例えば RPG）と連携したパラメータ部分などが変更になった場合には、問題がある。サーバー側の変更と同時に、クライアント側の Exe の置き換えを行わなければ、アップデートが逆にエラーの原因となってしまう。そういった場合には、サーバー側の変更と同期をとって、クライアント PC 側の Exe も置き換えることが必須になってくる。

もちろん Delphi/400 では、SQL などで、データベースの処理もすべて Exe 側で行うことができる。そういった Exe に処理を集約したプログラムであれば、クライアント / サーバー間におけるインターフェースの懸念は少なくて済むだろう。

2. Exe置き換えの手法

Exe の置き換えの必要性は前述の通りだが、ここでは Exe を置き換える手法について整理してみる。大きくは 2 つの運用手法が考えられるだろう。また、この 2 つの手法の特徴について列挙する。

(1) 配布ツールを利用して、Exe置き換えを運用

【実現コスト】
・ツールの導入費用が必要である。
・ツールで置き換えの仕組みが提供されるため、開発が不要である。

【汎用性】
・複数の Exe を管理する場合も、ツールですべて管理できる。
・Exe の置き換えだけでなく、新規 Exe の配布も行える。
・ツールとして提供されるため、機能のカスタマイズは行えない。
【Exe 置き換えのタイミング】
・Exe の置き換えは PC 起動時やスケジュールされたタイミングで行われるため、それにあわせて新しい Exe を配布する必要がある。

(2)独自に仕組み、プログラムを構築して Exe置き換えを運用

【実現コスト】
・ツールの導入費用が不要である。
・置き換えの仕組み、プログラムの作成（開発工数とスキル）が必要である。
【汎用性】
・個々の Exe ごとに、置き換えのプログラムを用意する必要がある。
・新規に Exe を作成した場合、初回配布は手動で行う必要がある。
・自動置き換えのプログラムに機能拡張や修正が必要となった場合、自動置き換えのプログラム自体の置き換えが必要となる。
・必要に応じてカスタマイズした仕組みの開発が可能である。
【Exe 置き換えのタイミング】
・Exeの実行時に置き換えができるため、確実に新しい Exe を利用することができる。

このように2つの手法は、それぞれメリット / デメリットとなる点が異なる。

開発の必要性や汎用性を考慮すると、(1) 配布ツールを使って運用する場合も多いが、本稿では、Delphi/400 の開発テクニックとして (2) Exe の自動置き換えの仕組みをプログラムで実現する例を紹介したい。

3. 自動置き換えの実現方法

ここからは、Exe のバージョンチェック～置き換えを行うにあたり、技術的な実現方法について説明していきたい。

まず、自動置き換えを行うには、Exe を 3 つ準備する必要がある。

● AExe：チェック / 置き換えを行うExe
● BExe：置き換えが行われる実行 Exe
● CExe：最新版の実行 Exe

AExe はバージョン情報をチェックして、BExe を最新の CExe に置き換える。その後、BExe を起動する。つまり、AExe を起動することで、自動的に最新に置き換えられた BExe が起動する仕組みである。【図 1】

AExe と BExe はクライアント PC の同一フォルダに保持し、CExe はファイルサーバーなど、システム利用者がアクセス可能な領域で一元管理を行うものとする。なお、システム利用者の権限によりアクセス制限が設けられている場合、自動置き換えが実施できなくなるため注意が必要である。

次に、ここで 3 つの Exe 構成をとる理由について説明しておく。

最新版としての CExe はいうまでもないが、AExe と BExe をわざわざ分けるのには理由がある。

本来、バージョン情報のチェックだけであれば、BExe 単体でも可能である。しかし、同時にファイルの置き換えを行う必要があるため、BExe 単体では実行中の自身を最新版の CExe と置き換えることができない。そのため、バージョン情報のチェックと置き換え処理を兼ねた AExe が必要となるのである。

バージョン情報

BExe と CExe の置き換え有無の判断については、各 Exe ファイルが保持しているバージョン情報を利用する。

そこで、Exe ファイルバージョン情報の設定方法、バージョン情報を取得する GetProductVersion 関数についてそれぞれ紹介する。

バージョン情報の設定方法は以下の通り。【図 2】

①Delphi の［メニュー］→［プロジェクト］→［オプション］
②バージョン情報タブの「バージョン番号を含める」にチェックを付ける。
③「モジュールのバージョン番号」に任意の番号を設定する。

また、バージョン情報の取得は、次のようになる。

ソース例のように、関数を利用することにより、BExe と CExe のバージョン情報を取得 / 比較し、ファイル置き換えの有無をチェックする。【図 3】

引数 PGNM に指定したファイルのバージョン情報を、文字列として返却する。なお、例外が発生した場合は、N/A を返却する。【図 4】

自動置き換え

準備についての説明を終えたところで、本題である実現方法の説明に話を進めたい。BExe と CExe はバージョンチェック、置き換えに利用するだけであり、ここからは AExe の動作について詳細に述べる。

まず、AExe の動作は大きく以下の 5 ステップに分かれる。【図 5】

ステップ①：BExe の絶対パスを取得
ステップ②：CExe の絶対パスを取得
ステップ③：BExe と CExe のバージョン情報を比較
ステップ④：バージョン情報が異なる場合、最新版への置き換え
ステップ⑤：BExe を実行

AExe の動作（5 つのステップ）について、少し掘り下げて説明を続けたい。

以下、Delphi ソースを交えて説明を行うにあたり、BExe、CExe の絶対パスとして変数宣言が行われているものとする。【ステップ 0】

① BExe の絶対パスを取得

AExe と BExe が同一フォルダに保持されていることを前提として、ExtractFilePath 関数を利用して、BExe の絶対パスを取得する。

また、ソース上の cExeName には、実際に実行する BExe のファイル名（拡張子含む）を定数として定義しておく。【ステップ 1】

② CExe の絶対パスを取得

CExe の絶対パスは、別途準備した ini ファイル（設定ファイル）より取得する。

ini ファイルの利用については、次章「4. 実行環境の変化への対策」で補足説明を行う。【ステップ 2】

図1　Exeファイルの構成

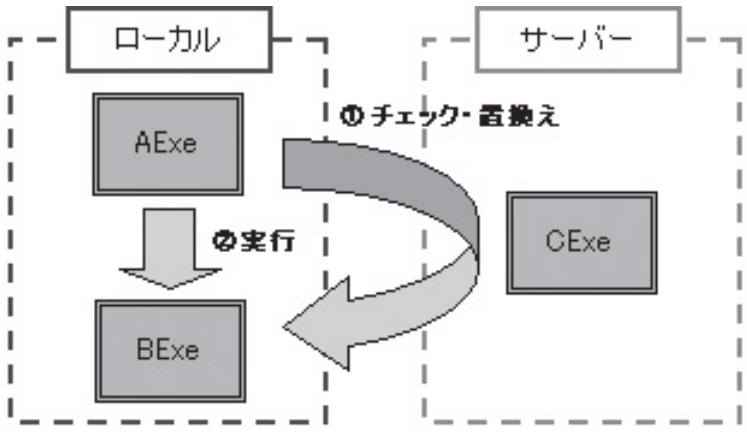


図2　バージョン情報の設定方法

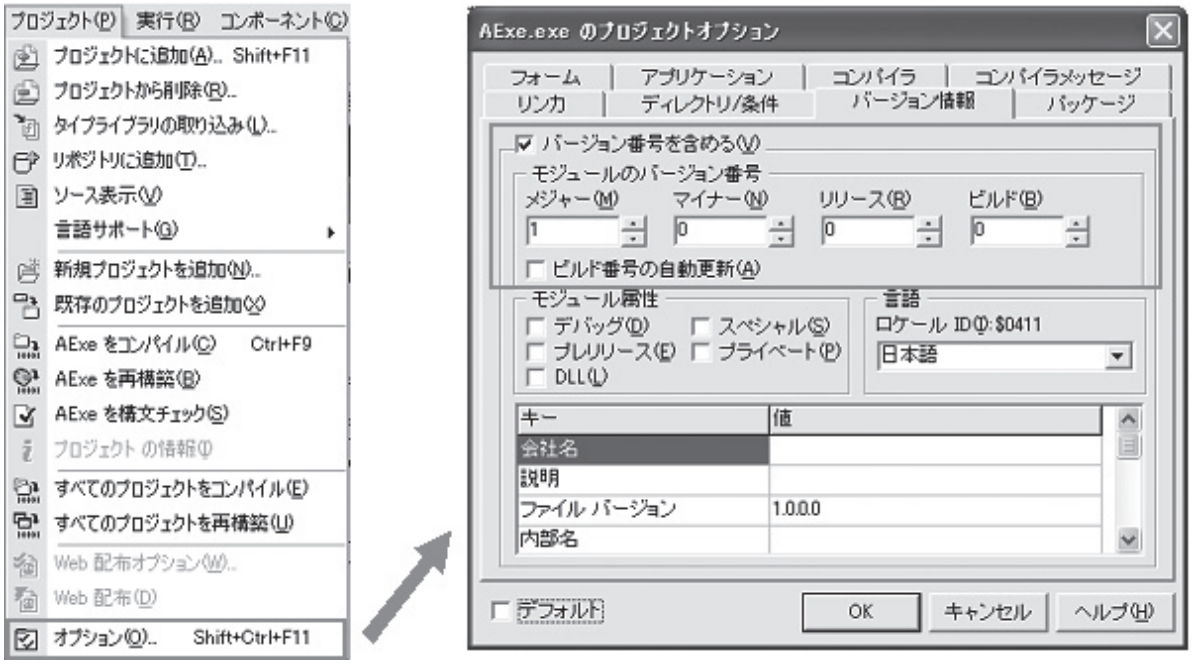


図3　バージョン情報の取得処理

```
{*****************************************************************************
 目的：バージョン情報の取得処理
 引数：
 戻値：取得したバージョン情報
*****************************************************************************}
function GetProductVersion(PGNM: String): String;
var
  Size, SizeFileInfo, Ret: DWORD; pData, pInfo: Pointer;
begin
  try
    Size := GetFileVersionInfoSize(PCHAR(PGNM), Ret);
    GetMem(pData, Size);
    try
      Assert(GetFileVersionInfo(PCHAR(PGNM), 0, Size, pData));
      Assert(VerQueryValue(pData,
        PCHAR('¥StringFileInfo¥041103A4¥FileVersion'), pInfo,
        SizeFileInfo));
      Result := PCHAR(pInfo);
    finally
      FreeMem(pData);
    end;
  except
    Result := 'N/A';  //----- データ取得できない場合
  end;
end;
```

③ BExe と CExe のバージョン情報を比較

①②で取得した BExe、CExe の絶対パスをもとに、それぞれのファイルのバージョン情報を取得する。CExe が常に最新バージョンであるため、バージョン情報が異なる場合、BExe のバージョンを古いものとして扱う。【ステップ 3】

④バージョン情報が異なる場合、最新版への置き換え

DeleteFile 関数で、クライアント PC の BExe の削除を行う。CopyFile 関数で、サーバーの最新版 CExe を、クライアント PC に BExe としてコピーする。【ステップ 4】

⑤ BExe を実行

最後に、CreateProcess 関数で BExe を起動する。ステップ①～⑤の手順を踏むことで、最新版への置き換え～ Exe 実行が実現可能となる。ユーザーからは特に意識されることもない。【ステップ 5】

実際に、図 5 のような構成で AExe を実行すると、BExe が「Ver.1.0.0.0」から、最新版の CExe へ置き換えられ、それにより「Ver.1.0.0.1」にアップデートされたことを確認できる。【図 6】【図 7】【図 8】

4. 実行環境の変化への対策

システムのアップデート（バージョンアップ）が行われるのと同様に、システムを運用する実行環境についても変化が訪れる可能性はきわめて高い。

例えば、接続先サーバーの変更、帳票レイアウトの変更、接続ユーザーの変更などである。当然、ファイルサーバーの変更（フォルダの変更）も例外ではない。

仮に、チェック / 置き換えを行う Exe（AExe）に、ファイルサーバーの接続先情報を保持しているとしよう。ファイルサーバーの変更があった場合、AExe 自体の置き換えが必要となってしまう。それでは運用にたえない。

そこで、ファイルサーバーの接続先情報を、ini ファイルにより外部で管理する手法を紹介したい。ここでは、前章のステップ②に登場した、ini ファイルを例に説明を行う。

はじめに、ini ファイルについて説明する。ini ファイル（設定ファイル）とは一般に、拡張子「.ini」を持つテキストファイルを指し、ini ファイルの記述は以下の構成をとる。【図 9】

セクション：キーのグループ、[] で囲んで記述する
キー　　：名前＝値のセット

次に、ini ファイル情報を取得する GetIniFile 関数について紹介する。なお、ソースの記述は、ini ファイルが AExe と同じフォルダに保持されていることを前提としている。【ソース 1】【ソース 2】

ソースの設定の GetIniFile 関数を実行すると、図 9 の AExe.ini に示したように、セクション：Common、キー：CExePath として、返却値には以下が設定される。

C:\SampleSource\CExe\CExe.exe

つまり、ini ファイルを利用すると、ファイルサーバーの変更があった場合、ini ファイルの設定値を変更するだけでよく、容易な対応が可能となる。

5. まとめ

モジュールのバージョンチェック / 置き換え手法というと一見難しい処理を想像するが、参考ソースを見ていただくと分かる通り、比較的少ないソース記述と ini ファイルの設定で実装することができる。

初期導入として、チェック用 Exe と ini ファイルの配布は必須となるが、それ以降のアップデートに関してはサーバー上の最新版 Exe を置き換えることで運用できる。

たしかに配布ツールを利用した場合は、専用のツールとなるので汎用性や導入してすぐ使えるなどの優れた点も多々あるだろう。

しかし、オーダーメイドのシステムであれば、システム運用にあわせて Exe 置き換えの仕組みをカスタマイズしていくなど、管理しやすくなる面がある。

また、本稿で説明したバージョンの取得や ini ファイルの利用、ファイルの置き換えといった開発テクニックは、さまざまな応用ができる。Exe の置き換え以外にも、おおいに利用していただきたい。これらの技術が、Delphi/400 のシステム開発 / 運用の助けとなれば幸いである。

M

図4　Ver.1.0.0.1の返却

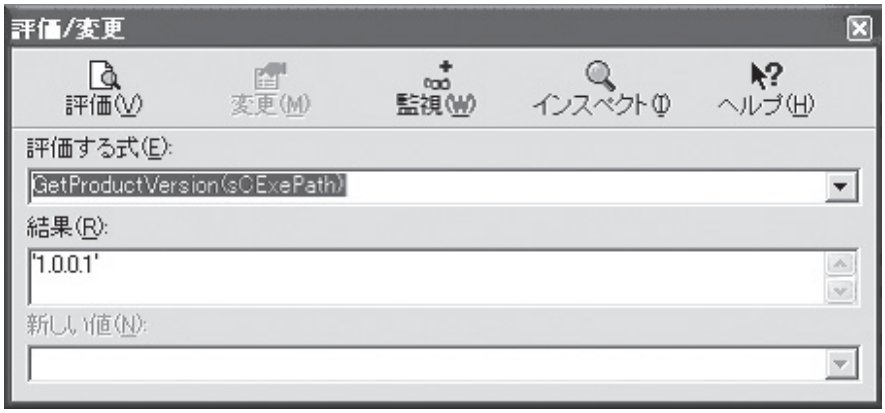


図5　AExeの動作

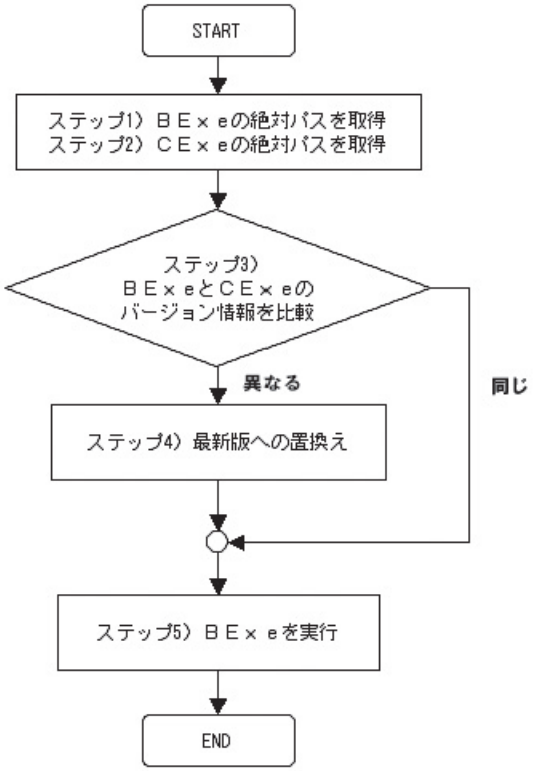
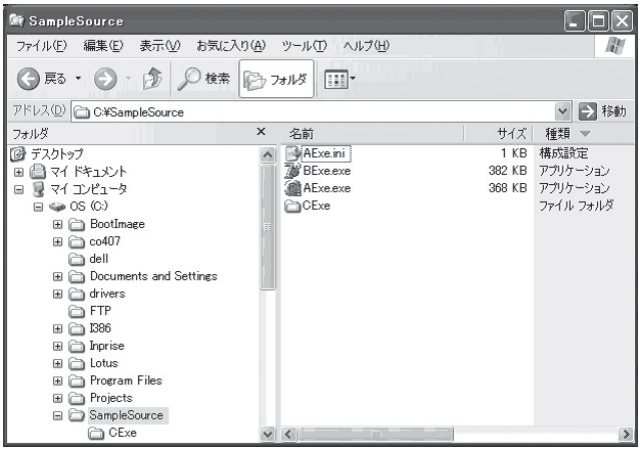


図6　ファイル構成



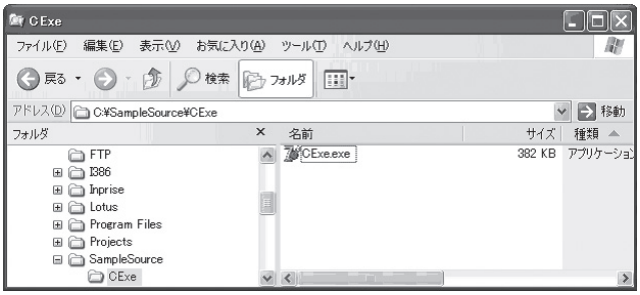


図7-1　AExe実行前のBExeのバージョン情報

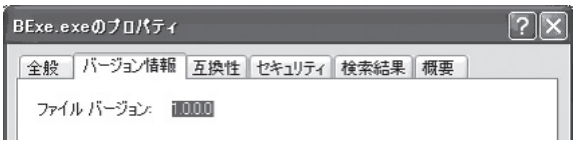


図7-2　AExe実行前のCExeのバージョン情報

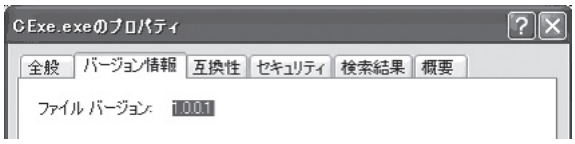


図8　AExe実行後のBExeのバージョン情報

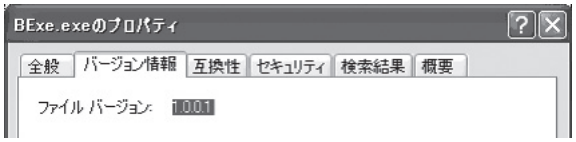
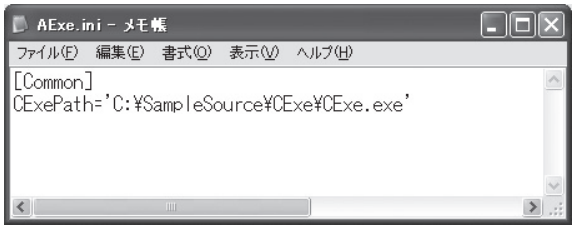


図9　AExeのIniファイル



ソース1　iniファイル取

```
{*****************************************************************************
 目的: Iniファイル取得処理
 引数: sSection - セクション
       sKey     - キー
       sDefault - デフォルト値
 戻値: 取得文字列
*****************************************************************************}
function GetIniFile(sSection, sKey, sDefault :string): string;
var
  FileIni: TIniFile;
  sPath: String;
begin
  // フォルダの指定
  sPath := ExtractFilePath(Application.ExeName);

  // INIファイルの読み込み
  FileIni := TIniFile.Create(sPath + cINIFile);
  try
    Result := FileIni.ReadString(sSection, sKey, sDefault);
  finally
    FileIni.Free;
  end;
end;
```

ソース2　iniファイル関連の設定

```
// 全般
cINIFile = 'AExe.ini';              // INIファイル名
cExeName = 'BExe.exe';              // 置換えExe (BExe)

// INIファイル
cSection_Common = 'Common';         // セクション 共通情報
cKey_CExePath   = 'CExePath';       // キー 最新版Exe (CExe) のパス
```

ステップ0

```
var
  sBExePath : String;     // BExeの絶対パス
  sCExePath : String;     // CExeの絶対パス
```

ステップ1

```
// BExeの絶対パスを取得（ローカル）
sBExePath := ExtractFilePath(Application.ExeName) + cExeName;
```

ステップ2

```
// CExeの絶対パスを取得（サーバー）
sCExePath := GetIniFile(cSection_Common, cKey_CExePath, '');
```

ステップ3

```
// バージョン情報チェック
if  GetProductVersion(sBExePath) <> GetProductVersion(sCExePath) then
```

ステップ4

```
// 最新版への置換え
DeleteFile(PChar(sBExePath));
CopyFile(PChar(sCExePath), PChar(sBExePath), True);
```

ステップ5

```
// BExeの起動
if not CreateProcess(nil, PChar(sBExePath), nil, nil, False,
                     CREATE_DEFAULT_ERROR_MODE, nil, nil, SI, PI) then
```

参考ソース1　AExe.dpr

```
program AExe;

uses
  Forms,
  Dialogs,
  SysUtils,
  Windows,
  Controls,
  pasConst in 'pasConst.pas',
  pasCommon in 'pasCommon.pas';

{$R *.res}
var
  sBExePath : String;     // BExeの絶対パス
  sCExePath : String;     // CExeの絶対パス
  SI          : TstartupInfo;
  PI          : TProcessInformation;
  i           : Integer;
  sPrmStr     : String;
begin
  Application.Initialize;
  Application.Run;

  // BExeの絶対パスを取得（ローカル）
  sBExePath := ExtractFilePath(Application.ExeName) + cExeName;

  // CExeの絶対パスを取得（サーバー）
  sCExePath := GetIniFile(cSection_Common, cKey_CExePath, '');

  //サーバーに最新版Exeが存在する場合、チェック実行
  if FileExists(sCExePath) then
  begin
    // バージョン情報チェック
    if  GetProductVersion(sBExePath) <> GetProductVersion(sCExePath) then
    begin
      try
        // 最新版への置換え
        DeleteFile(PChar(sBExePath));
        CopyFile(PChar(sCExePath), PChar(sBExePath), True);
      except
        //
      end;
    end;
  end;

  // 実行時引数の取得
  sPrmStr := '';
  for i := 1 to ParamCount do
  begin
    sPrmStr := sPrmStr + ' ' + ParamStr(i);
  end;
  sBExePath := sBExePath + Trim(sPrmStr);

  // BExeの起動
  if not CreateProcess(nil, PChar(sBExePath), nil, nil, False,
                       CREATE_DEFAULT_ERROR_MODE, nil, nil, SI, PI) then
  begin
    raise Exception.Create('Exec Error' + IntToStr(GetLastError));
  end;
end.
```

参考ソース2　pasConst.pas

```
unit pasConst;

interface

resourcestring
  // 全般
  cINIFile = 'AExe.ini';              // INIファイル名
  cExeName = 'BExe.exe';              // 置換えExe（BExe）

  // INIファイル
  cSection_Common = 'Common';         // セクション 共通情報
  cKey_CExePath   = 'CExePath';       // キー 最新版Exe（CExe）のパス

implementation

end.
```

参考ソース3　pasCommon.pas

```
unit pasCommon;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  IniFiles, Registry;

function GetIniFile(sSection, sKey, sDefault :string): string;
function GetProductVersion(PGNM: String): String;

implementation

uses pasConst;

{*******************************************************************************
 目的：Iniファイル取得処理
 引数：sSection - セクション
       sKey     - キー
       sDefault - デフォルト値
 戻値：取得文字列
*******************************************************************************}
function GetIniFile(sSection, sKey, sDefault :string): string;
var
  FileIni: TIniFile;
  sPath: String;
begin
  // フォルダの指定
  sPath := ExtractFilePath(Application.ExeName);

  // INIファイルの読み込み
  FileIni := TIniFile.Create(sPath + cINIFile);
  try
    Result := FileIni.ReadString(sSection, sKey, sDefault);
  finally
    FileIni.Free;
  end;
end;

{*******************************************************************************
 目的：バージョン情報の取得処理
 引数：
 戻値：取得したバージョン情報
*******************************************************************************}
function GetProductVersion(PGNM: String): String;
var
  Size, SizeFileInfo, Ret: DWORD; pData, pInfo: Pointer;
begin
  try
    Size := GetFileVersionInfoSize(PCHAR(PGNM), Ret);
    GetMem(pData, Size);
    try
      Assert(GetFileVersionInfo(PCHAR(PGNM), 0, Size, pData));
      Assert(VerQueryValue(pData,
        PCHAR('¥StringFileInfo¥041103A4¥FileVersion'), pInfo,
        SizeFileInfo));
      Result := PCHAR(pInfo);
    finally
      FreeMem(pData);
    end;
  except
    Result := 'N/A';  //----- データ取得できない場合
  end;
end;

end.
```