%Library.File の使用

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

%Library.File の使用

InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

%Library.File の使用

%Library.File クラス (%File が省略形) では、ファイルとディレクトリを操作するための広範な API が提供されています。このページでは、この API の主な機能について説明します。プロパティ、メソッド、およびクエリのキャノニック形式のリストは、クラス・リファレンスを参照してください。

注釈　ファイル名またはディレクトリ名の一部を指定すると、ほとんどのメソッドでは、作業中のネームスペースの既定のグローバル・データベースが含まれるディレクトリを基準とした項目を参照しているものと想定されます。このディレクトリをここでは "既定のディレクトリ" と呼びます。このルールが当てはまらない場合は注記しています。

また、これらのメソッドでファイル名とディレクトリ名の大文字と小文字が区別されるのは、基盤となるオペレーティング・システムでファイル名とディレクトリ名の大文字と小文字が区別される場合だけです。つまり、Unix ではファイル名またはディレクトリ名の大文字と小文字が区別されますが、Windows では区別されません。

1 ディレクトリとドライブに対するクエリ

%Library.File クラスには、ドライブとディレクトリにクエリを実行できるクラス・クエリが用意されています。

- ディレクトリの内容のリスト
- ドライブまたはマウントされたファイル・システムのリスト

1.1 ディレクトリの内容のリスト

FileSet クラス・クエリで、ディレクトリの内容をリストします。このクエリは、以下のパラメータをこの順序で受け入れます。

1. directory ー 確認するディレクトリの名前を指定します。
2. wildcards ー 一致させるファイル名パターンを指定します (使用する場合)。詳細は、$ZSEARCH のリファレンスの“ワイルドカード”のセクションを参照してください。
3. sortby ー 結果の並べ替え方法を指定します。以下の値のいずれかを使用します。
   - `Name` ー ファイルの名前 (既定値)
   - `Type` ー 項目タイプ
   - `DateCreated` ー ファイルが作成された日時
   - `DateModified` ー ファイルが前回変更された日時
   - `Size` ー ファイル・サイズ
4. includedirs ー 指定したディレクトリ内のディレクトリを処理する方法を指定します。この引数が true (1) の場合、クエリはファイルより先にすべてのディレクトリを返します。また、ディレクトリ名では wildcards 引数は無視されます。この引数が false (0) の場合、wildcards 引数はファイルとディレクトリの両方に適用されます。既定値は 0 です。
5. delimiter ー wildcards 引数内のワイルドカードの区切り文字を指定します。既定値は ; です。

このクエリによって返される結果セットには、以下のフィールドがあります。

- Name ー 項目の完全パス名。

- Type ー 項目のタイプ。F はファイル、D はディレクトリ、S はシンボリック・リンクを示します。
- Size ーファイル・サイズ (バイト単位)。このフィールドは、ディレクトリおよびシンボリック・リンクでは NULL になります。
- DateCreated ー 項目が作成された日時。形式は yyyy-mm-dd hh:mm:ss。
- DateModified ー 項目が前回変更された日時。形式は yyyy-mm-dd hh:mm:ss。
- ItemName ー項目の短い名前。ファイルの場合、ディレクトリの付かないファイル名のみです。ディレクトリの場合は、ディレクトリ・パスの最後の部分のみです。

注釈　現在、実際に作成された日付を追跡しているプラットフォームは Windows のみです。その他のプラットフォームは、ファイル・ステータスの最後の変更の日付を格納しています。

このクラス・クエリを使用した簡単な例を以下に示します。

Class Member

```
ClassMethod ShowDir(dir As %String = "", wildcard As %String = "", sort As %String = "Name")
{
  set stmt = ##class(%SQL.Statement).%New()
  set status = stmt.%PrepareClassQuery("%File", "FileSet")
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = stmt.%Execute(dir, wildcard, sort)
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write !, rset.%Get("Name")
    write " ", rset.%Get("Type")
    write " ", rset.%Get("Size")
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
}
```

メソッドが User.FileTest クラスにあると仮定した場合、ターミナルから指定のディレクトリに対してこのメソッドを実行し、ログ・ファイルをフィルタリングで抽出してファイル・サイズで並べ替えると、以下のようになります。

```
USER>do ##class(FileTest).ShowDir("C:\InterSystems\IRIS\mgr", "*.log", "Size")

C:\InterSystems\IRIS\mgr\alerts.log F 380
C:\InterSystems\IRIS\mgr\FeatureTracker.log F 730
C:\InterSystems\IRIS\mgr\journal.log F 743
C:\InterSystems\IRIS\mgr\ensinstall.log F 12577
C:\InterSystems\IRIS\mgr\iboot.log F 40124
C:\InterSystems\IRIS\mgr\SystemMonitor.log F 483865
C:\InterSystems\IRIS\mgr\messages.log F 4554535
```

別の例として、以下のメソッドは、ディレクトリとそのすべてのサブディレクトリを再帰的に調べて、見つかった各ファイルの名前を書き出します。

Class Member

```
ClassMethod ShowFilesInDir(directory As %String = "")
{
  set stmt = ##class(%SQL.Statement).%New()
  set status = stmt.%PrepareClassQuery("%File", "FileSet")
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = stmt.%Execute(directory)
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    set name = rset.%Get("Name")
    set type = rset.%Get("Type")

    if (type = "F") {
```

```
      write !, name
     } elseif (type = "D"){
      do ..ShowFilesInDir(name)
    }
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
}
```

ターミナルで既定のディレクトリに対してこのメソッドを実行すると、以下のようになります。

```
USER>do ##class(FileTest).ShowFilesInDir()

C:\InterSystems\IRIS\mgr\user\IRIS.DAT
C:\InterSystems\IRIS\mgr\user\iris.lck
C:\InterSystems\IRIS\mgr\user\userenstemp\IRIS.DAT
C:\InterSystems\IRIS\mgr\user\userenstemp\iris.lck
C:\InterSystems\IRIS\mgr\user\usersecondary\IRIS.DAT
C:\InterSystems\IRIS\mgr\user\usersecondary\iris.lck
```

1.2 ドライブまたはマウントされたファイル・システムのリスト

DriveList クラス・クエリは、使用可能なドライブ (Windows の場合) またはマウントされているファイル・システム (Unix の場合) をリストします。このクエリは、以下の 1 つのパラメータを受け入れます。

1. fullyqualified ー この引数が 1 の場合、各 Windows ドライブ名の最後に円記号が追加されます。この引数は他のプラットフォームには影響しません。既定値は 0 です。

このクエリによって返される結果セットには、フィールドが 1 つあります。

- Drive ードライブの名前 (Windows の場合) またはマウントされているファイル・システムの名前 (Unix の場合)。

以下の例は、このクエリを使用する方法を示しています。

```
ClassMethod ShowDrives()
{
  set stmt = ##class(%SQL.Statement).%New()
  set status = stmt.%PrepareClassQuery("%File","DriveList")
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = stmt.%Execute(1)
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write !, rset.%Get("Drive")
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
}
```

ここでも、このメソッドが User.FileTest クラス内にあると仮定した場合、ターミナルでこのメソッドを実行すると、以下のようになります。

```
USER>do ##class(FileTest).ShowDrives()

c:\
x:\
```

2 ファイルおよびディレクトリのプロパティと属性の操作

%Library.File クラスには、ファイルとディレクトリに関する情報の取得や、そのプロパティと属性の表示または設定に使用できるクラス・メソッドも多数用意されています。

2.1 ファイルとディレクトリの有無のチェック

指定したファイルが存在するかどうかを確認するには、Exists() メソッドを使用し、ファイル名を引数として指定します。例えば、以下のように指定します。

```
USER>write ##class(%File).Exists("C:\temp\test.html")
1
```

同様に、指定したディレクトリが存在するかどうかを確認するには、DirectoryExists() メソッドを使用し、ディレクトリを引数として指定します。例えば、以下のように指定します。

```
USER>write ##class(%File).DirectoryExists("C:\temp")
1
```

前述のとおり、これらのメソッドでも、Unix ではファイル名またはディレクトリ名の大文字と小文字が区別されますが、Windows では区別されません。また、ファイル名またはディレクトリ名の一部を指定すると、作業中のネームスペースの既定のグローバル・データベースが含まれるディレクトリを基準としたファイルまたはディレクトリを指しているものと想定されます。例えば、以下のようになります。

```
USER>write ##class(%File).Exists("iris.dat")
1
```

2.2 ファイルとディレクトリの許可の表示と設定

%Library.File クラスには、ファイルまたはディレクトリの許可の表示または設定に使用できるクラス・メソッドが多数用意されています。

2.2.1 ファイルまたはディレクトリが読み取り専用か書き込み可能かの確認

ファイル名またはディレクトリ名を指定すると、ReadOnly() メソッドは、そのファイルまたはディレクトリが読み取り専用の場合は 1 を返し、そうでない場合は 0 を返します。

```
USER>write ##class(%File).ReadOnly("export.xml")
1
USER>write ##class(%File).ReadOnly("C:\temp")
0
```

同様に、ファイル名またはディレクトリ名を指定すると、Writeable() メソッドは、そのファイルまたはディレクトリが書き込み可能な場合は 1 を返し、そうでない場合は 0 を返します。

```
USER>write ##class(%File).Writeable("export.xml")
0
USER>write ##class(%File).Writeable("C:\temp")
1
```

2.2.2 ファイルまたはディレクトリを読み取り専用または書き込み可能にする方法 (Windows)

Windows 上のファイルまたはディレクトリを読み取り専用にするには、SetReadOnly() メソッドを使用します。このメソッドは、成功または失敗を示すブーリアン値を返します。このメソッドは 3 つの引数を取りますが、2 つ目は Windows では省略されます。1 つ目の引数は、ファイルまたはディレクトリの名前です。3 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下の例では、SetReadOnly() の呼び出しによって、ファイル C:¥temp¥testplan.pdf が正常に読み取り専用に変更されます。

```
USER>write ##class(%File).ReadOnly("C:\temp\testplan.pdf")
0
USER>write ##class(%File).SetReadOnly("C:\temp\testplan.pdf",,.return)
1
USER>write ##class(%File).ReadOnly("C:\temp\testplan.pdf")
1
```

以下の例では、SetReadOnly() の呼び出しが失敗し、"アクセスが拒否された" ことを示す Windows システム・エラー・コード 5 が返されます。

```
USER>write ##class(%File).SetReadOnly("C:\",,.return)
0
USER>write return
-5
```

Windows 上のファイルまたはディレクトリを書き込み可能にするには、SetWriteable() メソッドを使用します。このメソッドは同じ 3 つの引数を取りますが、同じようにその 2 つ目は Windows では省略されます。

```
USER>write ##class(%File).Writeable("export.xml")
0
USER>write ##class(%File).SetWriteable("export.xml",,.return)
1
USER>write ##class(%File).Writeable("export.xml")
1
```

2.2.3 ファイルまたはディレクトリを読み取り専用または書き込み可能にする方法 (Unix)

Unix でもメソッド SetReadOnly() と SetWriteable() を使用できますが、2 つ目のパラメータがあることで動作が多少異なります。詳細は、"クラス・リファレンス" の "%Library.File.SetReadOnly()" または "%Library.File.SetWriteable()" を参照してください。

ただし、Unix では、所有者、グループ、およびユーザに異なる許可を指定できます。ファイルとディレクトリの許可のきめ細かい制御については、"ファイルとディレクトリの属性の表示と設定" を参照してください。

2.3 ファイルとディレクトリの属性の表示と設定

より詳細なレベルでファイルまたはディレクトリの属性を表示または設定するには、%Library.File の Attributes() メソッドおよび SetAttributes() メソッドを使用します。ファイルの属性は、1 つの整数として集合的に表現されたビット・シーケンスで表されます。個々のビットの意味は、基盤となるオペレーティング・システムによって異なります。

属性のビットの完全なリストは、"クラス・リファレンス" の "%Library.File.Attributes()" を参照してください。

属性のビットの文字列を操作する際のヒントは、"整数として実装されるビット文字列の操作" を参照してください。

2.3.1 ファイルとディレクトリの属性の表示

%Library.File の Attributes() メソッドは、引数としてファイル名またはディレクトリ名を取り、整数として表現された属性のビット・シーケンスを返します。

以下は、Windows システムで実行した場合の例です。

```
USER>write ##class(%File).Attributes("iris.dat")
32
USER>write ##class(%File).Attributes("C:\temp")
16
USER>write ##class(%File).Attributes("secret.zip")
35
```

1 つ目の例の 32 は、iris.dat がアーカイブ・ファイルであることを示します。2 つ目の例の 16 は、C:¥temp がディレクトリであることを示します。3 つ目の例では、複数のビットが設定されており、35 は、secret.zip が非表示で (2) 読み取り専用 (1) のアーカイブ (32) であることを示します。32 + 2 + 1 = 35 の計算になります。

以下は、Unix システムで実行した場合の例です。

```
write ##class(%File).Attributes("/home")
16877
```

この例の 16877 は、/home が、所有者は読み取り (256)、書き込み (128)、および実行 (64) の許可を持ち、グループは読み取り (32) および実行 (8) の許可を持ち、それ以外は読み取り (4) および実行 (1) の許可を持つディレクトリ (16384) であることを示します。16384 + 256 + 128 + 64 + 32 + 8 + 4 + 1 = 16877 の計算になります。

2.3.2 ファイルとディレクトリの属性の設定

一方、SetAttributes() メソッドは、ファイルまたはディレクトリの属性を設定して (可能な場合)、成功または失敗を示すブーリアン値を返します。このメソッドは、3 つの引数を取ります。1 つ目の引数は、ファイルまたはディレクトリの名前です。2 つ目の引数は、ファイルまたはディレクトリに設定する属性を表す整数です。3 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下は Windows で実行した場合の例で、1 ビットを設定して C:¥temp¥protectme.txt を読み取り専用にします。

```
USER>write ##class(%File).Attributes("C:\temp\protectme.txt")
32
USER>write ##class(%File).SetAttributes("C:\temp\protectme.txt",33,.return)
1
USER>write ##class(%File).Attributes("C:\temp\protectme.txt")
33
```

以下は Unix で実行した場合の例で、既定のディレクトリ内の myfile に対する許可を 644 から完全な許可 (777) に変更します。

```
USER>write ##class(%File).Attributes("myfile")
33188
USER>write ##class(%File).SetAttributes("myfile",33279,.return)
1
USER>write ##class(%File).Attributes("myfile")
33279
```

目的の属性値は、通常ファイルの値 (32768) に所有者 (448)、グループ (56)、およびその他 (7) のマスクを加算して計算されます。

2.4 ファイルとディレクトリのその他のプロパティの表示

%Library.File のその他のクラス・メソッドでは、ファイルとディレクトリのその他のさまざまなプロパティを確認できます。

GetFileDateCreated() メソッドは、ファイルまたはディレクトリが作成された日付を $H 形式で返します。

```
USER>write $zdate(##class(%File).GetFileDateCreated("stream"))
12/09/2019
```

注釈　現在、実際に作成された日付を追跡しているプラットフォームは Windows のみです。その他のプラットフォームは、ファイル・ステータスの最後の変更の日付を格納しています。

GetFileDateModified() メソッドは、ファイルまたはディレクトリが変更された日付を $H 形式で返します。

```
USER>write $zdate(##class(%File).GetFileDateModified("iris.dat"))
08/20/2020
```

GetFileSize() メソッドは、ファイルのサイズをバイト単位で返します。

```
USER>write ##class(%File).GetFileSize("export.xml")
2512
```

GetDirectorySpace() メソッドは、ドライブまたはディレクトリの空き容量と合計容量を返します。容量は、4 つ目の引数の値が 0、1、2 のいずれであるかによって、バイト、MB (既定値)、GB の単位で返されます。以下の例の 2 は、容量が GB 単位で返されることを示します。

```
USER>set status = ##class(%File).GetDirectorySpace("C:", .FreeSpace, .TotalSpace, 2)

USER>write FreeSpace
182.87
USER>write TotalSpace
952.89
```

Windows では、ディレクトリ名をこのメソッドに渡した場合に返される容量は、ドライブ全体の値です。

GetDirectorySpace() メソッドでは、返されるエラー状態はオペレーティング・システム・レベルのエラーです。以下の例の Windows システム・エラー・コード 3 は、"指定されたパスが見つからない" ことを示しています。

```
USER>set status = ##class(%File).GetDirectorySpace("Q:", .FreeSpace, .TotalSpace, 2)

USER>do $system.Status.DisplayError(status)

ERROR #83: Error code = 3
```

3 ファイル名とディレクトリ名の処理

%Library.File クラスには、ファイル名とディレクトリ名の操作に使用できるクラス・メソッドがいくつか用意されています。ほとんどの場合、これらのメソッドを使用するためにファイルとディレクトリが存在する必要はありません。

- ファイル名とディレクトリ名の取得
- ファイル名とディレクトリ名の正規化
- スペースが含まれるファイル名とディレクトリ名の処理
- ファイル名とディレクトリ名の構成と分解
- システム・マネージャ・ディレクトリの取得

3.1 ファイル名とディレクトリ名の取得

%Library.File クラスには、ファイル名とディレクトリ名の一部分を取得するのに使用できるクラス・メソッドが用意されています。

完全パス名を指定して GetDirectory() と GetFilename() を使用すると、ディレクトリと短いファイル名をそれぞれ取得できます。このメソッドでは、部分的なディレクトリ名は許可されていません。

```
USER>set filename = "C:\temp\samples\sample.html"

USER>write ##class(%File).GetDirectory(filename)
C:\temp\samples\
USER>write ##class(%File).GetFilename(filename)
sample.html
```

ファイル名を指定して CanonicalFilename() を使用すると、ルートからの完全なパスを取得できます。

```
USER>set filename = "iris.dat"

USER>write ##class(%File).CanonicalFilename(filename)
c:\intersystems\IRIS\mgr\user\iris.dat
USER>write ##class(%File).CanonicalFilename("foo.dat")
```

指定したファイルを開くことができない場合、CanonicalFilename() メソッドは空の文字列を返します。

ディレクトリ名を指定して ComputeFullDBDir() を使用すると、キャノニック形式のディレクトリ名を構成できます。

```
USER>write ##class(%File).ComputeFullDBDir("foodirectory")
c:\intersystems\IRIS\mgr\user\foodirectory\
```

ディレクトリ名を指定して GetDirectoryLength() と GetDirectoryPiece() を使用すると、ディレクトリ内の構成要素の数と特定の構成要素をそれぞれ取得できます。構成要素は、オペレーティング・システムに応じて、スラッシュ (/) または円記号 (¥) で区切ることができます。

```
USER>set dir = "C:\temp\samples"

USER>write ##class(%File).GetDirectoryLength(dir)
3
USER>write ##class(%File).GetDirectoryPiece(dir,1)
C:
```

ファイル名またはディレクトリ名を指定して ParentDirectoryName() を使用すると、その親ディレクトリを取得できます。

```
USER>set dir = "stream"

USER>write ##class(%File).ParentDirectoryName(dir)
C:\InterSystems\IRIS\mgr\user\
```

3.2 ファイル名とディレクトリ名の正規化

%Library.File クラスでは、正規化されたファイル名およびディレクトリ名を返すクラス・メソッドが用意されています (サーバが実行されているオペレーティング・システムの命名規則に従います)。これらのメソッドは、既存の名前に名前要素を追加してファイル名やディレクトリ名を新規作成する場合に有用です。

ファイル名を指定すると、NormalizeFilename() は、正規化されたファイル名を返します。

ディレクトリ名を指定すると、NormalizeDirectory() は、正規化されたディレクトリ名を返します。

これらのメソッドは、基盤となるオペレーティング・システムでの使用に適した正規化された名前を返し、スラッシュ (/) または円記号 (¥) のパス区切り文字の正規化を試みます。

Windows の例 :

```
USER>set filename = "C:\temp//samples\myfile.txt"

USER>write ##class(%File).NormalizeFilename(filename)
C:\temp\samples\myfile.txt
USER>write ##class(%File).NormalizeDirectory("stream")
C:\InterSystems\IRIS\mgr\user\stream\
```

Unix の例 :

```
USER>set filename = "/tmp//samples/myfile.txt"

USER>write ##class(%File).NormalizeFilename(filename)
/tmp/samples/myfile.txt
USER>write ##class(%File).NormalizeDirectory("stream")
/InterSystems/IRIS/mgr/user/stream/
```

これらのメソッドのいずれかを呼び出して、指定したディレクトリを基準としたディレクトリ名またはファイル名を正規化する場合、2 つ目の引数を追加します。このディレクトリは存在している必要があります。

Windows の例 :

```
USER>write ##class(%File).NormalizeFilename("myfile.txt", "C:\temp\samples")
C:\temp\samples\myfile.txt
USER>write ##class(%File).NormalizeDirectory("stream", "")
C:\InterSystems\IRIS\mgr\user\stream\
```

Unix の例：

```
USER>write ##class(%File).NormalizeFilename("myfile.txt", "/tmp/samples")
/tmp/samples/myfile.txt
USER>write ##class(%File).NormalizeDirectory("stream", "")
/InterSystems/IRIS/mgr/user/stream/
```

SubDirectoryName() メソッドは、引数が 2 つの形式の NormalizeDirectory() と似ていますが、引数の順序が逆になっている点が異なります。また、このディレクトリが存在する必要はありません。末尾の区切り文字を追加する場合は 3 つ目の引数で `1` を渡し、省略する場合は `0` (既定値) を渡します。

Windows の例：

```
USER>write ##class(%File).SubDirectoryName("C:\foobar", "samples")
C:\foobar\samples
USER>write ##class(%File).SubDirectoryName("", "stream", 1)
C:\InterSystems\IRIS\mgr\user\stream\
```

Unix の例：

```
USER>write ##class(%File).SubDirectoryName("/foobar", "samples")
/foobar/samples
USER>write ##class(%File).SubDirectoryName("", "stream", 1)
/InterSystems/IRIS/mgr/user/stream/
```

3.3 スペースが含まれるファイル名とディレクトリ名の処理

スペースが含まれるファイル名またはディレクトリ名の場合、NormalizeFilenameWithSpaces() を使用します。これにより、ホスト・プラットフォームに適した方法でパス名に含まれるスペースが処理されます。NormalizeFilename() や NormalizeDirectory() とは異なり、このメソッドは引数を 1 つしか取りません。また、別のディレクトリを基準としたファイル名またはディレクトリ名を正規化することはできず、既定のディレクトリを基準としたファイル名またはディレクトリ名の一部分を正規化することもありません。

Windows システムでは、パス名にスペースが含まれていて、ファイルまたはディレクトリが存在しない場合、このメソッドはパス名を二重引用符で囲んで返します。パス名にスペースが含まれていて、ファイルまたはディレクトリが存在する場合、このメソッドはパス名の短い形式を返します。パス名にスペースが含まれない場合、このメソッドはパス名を変更なしで返します。

```
USER>write ##class(%File).NormalizeFilenameWithSpaces("C:\temp\nonexistant folder")
"C:\temp\nonexistant folder"
USER>write ##class(%File).NormalizeFilenameWithSpaces("C:\temp\existant folder")
C:\temp\EXISTA~1
USER>write ##class(%File).NormalizeFilenameWithSpaces("iris.dat")
iris.dat
```

詳細は、"クラス・リファレンス" の "%Library.File.NormalizeFilenameWithSpaces()" を参照してください。

Unix システムでは、パス名にスペースが含まれている場合、このメソッドはパス名を二重引用符で囲んで返します。パス名にスペースが含まれない場合、このメソッドはパス名を変更なしで返します。

```
USER>write ##class(%File).NormalizeFilenameWithSpaces("/InterSystems/my directory")
"/InterSystems/my directory"
USER>write ##class(%File).NormalizeFilenameWithSpaces("iris.dat")
iris.dat
```

3.4 ファイル名とディレクトリ名の構成と分解

%Library.File クラスには、パスの配列からファイル名を構成する際、およびファイル名をパスの配列に分解する際に使用できるクラス・メソッドが用意されています。

パスの配列を指定すると、Construct() はパスを構成して、ファイル名を返します。構成されるファイル名は、サーバ・プラットフォームに適したものです。引数なしでこのメソッドを呼び出すと、既定のディレクトリが返されます。

ファイル名を指定すると、Deconstruct() は、ファイル名を分解してパスの配列を返します。配列の内容は、サーバ・プラットフォームに適したものです。

以下の Windows の例では、配列 dirs を Construct() に渡しています。配列の最後の位置にある空の文字列は、返されるファイル名が \ で終わる必要があることを示します。

```
USER>zwrite dirs
dirs=4
dirs(1)="C:"
dirs(2)="Temp"
dirs(3)="samples"
dirs(4)=""
USER>write ##class(%File).Construct(dirs...)
C:\Temp\samples\
```

以下の Unix の例では、引数なしで Construct() を呼び出しています。メソッドは、既定のディレクトリを返します。

```
USER>set default = ##class(%File).Construct()

USER>write default
/InterSystems/IRIS/mgr/user
```

以下の Unix の例では、Deconstruct() を呼び出しています。これは、変数 default でパスを取得し、それらのパスを配列 defaultdir に格納します。

```
USER>do ##class(%File).Deconstruct(default, .defaultdir)

USER>zwrite defaultdir
defaultdir=4
defaultdir(1)="InterSystems"
defaultdir(2)="IRIS"
defaultdir(3)="mgr"
defaultdir(4)="user"
```

3.5 システム・マネージャ・ディレクトリの取得

ManagerDirectory() メソッドを使用すると、installdir/mgr ディレクトリの完全修飾名を取得できます。例えば、以下のように指定します。

```
USER>write ##class(%File).ManagerDirectory()
C:\InterSystems\IRIS\mgr\
```

4 ディレクトリの操作

%Library.File クラスには、ディレクトリに対して以下のようなさまざまな操作を実行できるクラス・メソッドがいくつか用意されています。

- ディレクトリの作成
- ディレクトリのコピー
- ディレクトリの削除
- ディレクトリの名前変更

4.1 ディレクトリの作成

ディレクトリを作成するには、CreateDirectory() メソッドを使用します。このメソッドは、成功または失敗を示すブーリアン値を返します。このメソッドは、2 つの引数を取ります。1 つ目の引数は、作成するディレクトリの名前です。2 つ目の引数

は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

C:¥temp が既に存在する場合、以下のコマンドは失敗し、"既に存在するファイルを作成することはできない" という意味の Windows システム・エラー・コード 183 が返されます。

```
USER>write ##class(%File).CreateDirectory("C:\temp", .return)
0
USER>write return
-183
```

C:¥temp が既に存在していても、C:¥temp¥test は存在しない場合、以下のコマンドは失敗します。CreateDirectory() は、最大でも、指定したディレクトリ・パスの最後のディレクトリしか作成しないためです。したがって、返される Windows システム・エラー・コードは 3 ("指定されたパスを見つけられませんでした") です。

```
USER>write ##class(%File).CreateDirectory("C:\temp\test\this", .return)
0
USER>write return
-3
```

以下の例は成功し、Windows システム・コード 0 ("操作は正常に終了しました") が返されます。

```
USER>write ##class(%File).CreateDirectory("C:\temp\test", .return)
1
USER>write return
0
```

類似する CreateNewDir() メソッドは、指定した親ディレクトリ内に新しいディレクトリを作成します。このメソッドは、3 つの引数を取ります。1 つ目の引数は、親ディレクトリの名前です。2 つ目の引数は、作成するディレクトリの名前です。3 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下の 1 つ目の例では、親ディレクトリ C:¥temp 内に newdir という名前のディレクトリを作成します。2 つ目の例では、既定のディレクトリ内に newdir という名前の新しいディレクトリを作成します。

```
USER>write ##class(%File).CreateNewDir("C:\temp", "newdir", .return)
1
USER>write ##class(%File).CreateNewDir("", "newdir", .return)
1
```

関連するもう 1 つのメソッド CreateDirectoryChain() は、指定したディレクトリ・パスにすべてのディレクトリを作成します (可能な場合)。

以下の 1 つ目の例では、親ディレクトリ C:¥temp 内に、入れ子になった 3 つのディレクトリを作成します。以下の 2 つ目の例では、既定のディレクトリ内に、入れ子になった 3 つのディレクトリを作成します。

```
USER>write ##class(%File).CreateDirectoryChain("C:\temp\one\two\three", .return)
1
USER>write ##class(%File).CreateDirectoryChain("one\two\three", .return)
1
```

4.2 ディレクトリのコピー

ディレクトリをコピーするには、CopyDir() メソッドを使用します。このメソッドは、成功または失敗を示すブーリアン値を返します。

このメソッドは、以下の 5 つの引数を取ります。

1. pSource ー ソース・ディレクトリの名前を指定します。
2. pTarget ー ターゲット・ディレクトリの名前を指定します。
3. pOverlay ー ターゲット・ディレクトリが存在する場合にそれを上書きするかどうかを指定します。既定値は 0 です。

4. pCreated ー コピー・プロセスで作成されたファイルまたはディレクトリの数が含まれる出力引数です。
5. pDeleteBeforeCopy ー コピーを実行する前に、ターゲット・ディレクトリに存在するファイルを削除するかどうかを指定します。既定値は 0 です。

pSource または pTarget にディレクトリ名の一部を指定した場合、ディレクトリ名は、作業中のネームスペースの既定のグローバル・データベースが含まれるディレクトリを基準として計算されます。

ディレクトリ作成メソッドとは異なり、CopyDir() には、システム・エラー・コードを返すための出力引数はありません。

以下の 1 つ目の例では、コピー操作が成功し、46 個のファイルとディレクトリが C:¥temp から C:¥temp2 にコピーされます。2 つ目の例では、コピー操作が成功し、46 個のファイルとディレクトリが、C:¥temp から既定のディレクトリ内の temp2 にコピーされます。

```
USER>write ##class(%File).CopyDir("C:\temp", "C:\temp2", 0, .pCreated, 0)
1
USER>write pCreated
46
USER>write ##class(%File).CopyDir("C:\temp", "temp2", 0, .pCreated, 0)
1
USER>write pCreated
46
```

最後の例では、pOverlay が 0 に設定されていて、ターゲット・ディレクトリが既に存在するため、コピーは失敗します。

```
USER>write ##class(%File).CopyDir("C:\temp", "C:\temp2", 0, .pCreated, 0)
0
USER>write pCreated
0
```

4.3 ディレクトリの削除

空でないディレクトリを削除するには、RemoveDirectory() メソッドを使用します。このメソッドは、成功した場合に 1 を返し、失敗した場合に 0 を返します。このメソッドは、2 つの引数を取ります。1 つ目の引数は、削除するディレクトリの名前です。2 つ目の引数は出力引数で、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下の 1 つ目の例では、メソッドは成功します。2 つ目の例は失敗し、Windows エラー・コード 145 (″ディレクトリが空ではありません″) が返されます。

```
USER>write ##class(%File).RemoveDirectory("C:\temp2\newdir", .return)
1
USER>write ##class(%File).RemoveDirectory("C:\temp2", .return)
0
USER>write return
-145
```

サブディレクトリが含まれるディレクトリを削除するには、RemoveDirectoryTree() メソッドを使用します。このメソッドは、成功した場合に 1 を返し、失敗した場合に 0 を返します。RemoveDirectory() メソッドとは異なり、RemoveDirectoryTree() には、システム・エラー・コードを返すための出力引数はありません。

RemoveDirectoryTree() は、ディレクトリおよびサブディレクトリが空でない場合も成功します。

```
USER>write ##class(%File).RemoveDirectoryTree("C:\temp2")
1
```

4.4 ディレクトリの名前変更

ディレクトリの名前を変更するには、Rename() メソッドを使用します。このメソッドは、成功した場合に 1 を返し、失敗した場合に 0 を返します。このメソッドは、3 つの引数を取ります。1 つ目の引数は名前を変更するディレクトリの名前で、2 つ目の引数は新しい名前です。3 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

Rename() を使用したディレクトリの名前変更は、そのディレクトリのファイル・システムが、作業中のファイル・システムと同じ場合にのみ機能します。

以下の 1 つ目の例では、メソッドは成功します。2 つ目の例では、C:¥nodir は存在しないため、メソッドは失敗し、Windows エラー・コード 3 (″指定されたパスを見つけられませんでした″) が返されます。

```
USER>write ##class(%File).Rename("C:\temp\oldname", "C:\temp\newname", .return)
1
USER>write ##class(%File).Rename("C:\nodir\oldname", "C:\nodir\newname", .return)
0
USER>write return
-3
```

このメソッドを使用する際には、パスの指定を慎重に行う必要があります。例えば、以下の例は、C:¥temp¥oldname を既定のディレクトリに移動してから、newname に名前変更する処理となります。

```
USER>write ##class(%File).Rename("C:\temp\oldname", "newname", .return)
1
```

5 ファイルの操作

%Library.File クラスには、ファイルに対して以下のようなさまざまな操作を実行できるクラス・メソッドがいくつか用意されています。ファイル自体の操作の詳細は、″%File オブジェクトの操作″ を参照してください。

- ファイルのコピー
- ファイルの削除
- ファイルの切り捨て
- ファイルの名前変更
- ファイルの比較
- 一時ファイルの生成

5.1 ファイルのコピー

ファイルをコピーするには、CopyFile() メソッドを使用します。このメソッドは、成功または失敗を示すブーリアン値を返します。

このメソッドは、以下の 4 つの引数を取ります。

1. from ー ソース・ファイルの名前を指定します。
2. to ー ターゲット・ファイルの名前を指定します。
3. pDeleteBeforeCopy ー ターゲット・ファイルが存在する場合、コピーの実行前にそのターゲット・ファイルを削除するかどうかを指定します。既定値は 0 です。
4. return ー 出力引数。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

例を以下に示します。

以下の 1 つ目の例では、ファイル old.txt をディレクトリ C:¥temp 内の new.txt にコピーします。2 つ目の例では、同じファイルを既定のディレクトリ内の new.txt にコピーします。

```
USER>write ##class(%File).CopyFile("C:\temp\old.txt", "C:\temp\new.txt", 0, .return)
1
USER>write ##class(%File).CopyFile("C:\temp\old.txt", "new.txt", 0, .return)
1
```

以下の最後の例は失敗し、Windows エラー・コード 2 (″ファイルが見つかりません″) が返されます。

```
USER>write ##class(%File).CopyFile("foo.txt", "new.txt", 0, .return)
0
USER>write return
-2
```

5.2 ファイルの削除

ファイルを削除するには、Delete() メソッドを使用します。このメソッドは、成功した場合に 1 を返し、失敗した場合に 0 を返します。このメソッドは、2 つの引数を取ります。1 つ目の引数は、削除するファイルの名前です。2 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下の 1 つ目の例では、メソッドは成功します。2 つ目の例は失敗し、Windows エラー・コード 2 (″ファイルが見つかりません″) が返されます。

```
USER>write ##class(%File).Delete("C:\temp\myfile.txt", .return)
1
USER>write ##class(%File).Delete("C:\temp\myfile.txt", .return)
0
USER>write return
-2
```

ファイルを削除する際にワイルドカードに一致させるには、ComplexDelete() メソッドを使用します。1 つ目の引数には、削除するファイルの名前を指定します。2 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下の例では、C:¥temp ディレクトリ内にある拡張子が .out のファイルをすべて削除します。

```
USER>write ##class(%File).ComplexDelete("C:\temp\*.out", .return)
1
```

5.3 ファイルの切り捨て

ファイルを切り捨てるには、Truncate() メソッドを使用します。このメソッドは、成功した場合に 1 を返し、失敗した場合に 0 を返します。このメソッドは、2 つの引数を取ります。1 つ目の引数は、切り捨てるファイルの名前です。2 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

既存のファイルを切り捨てる場合、このメソッドでは、ファイルから内容が削除されますが、ファイル・システムからファイルが削除されることはありません。存在しないファイルを切り捨てた場合、空のファイルが新規作成されます。

以下の 1 つ目の例では、メソッドは成功します。2 つ目の例は失敗し、Windows エラー・コード 5 (″アクセスが拒否されました″) が返されます。

```
USER>write ##class(%File).Truncate("C:\temp\myfile.txt", .return)
1
USER>write ##class(%File).Truncate("C:\no access.txt", .return)
0
USER>write return
-5
```

5.4 ファイルの名前変更

ファイルを名前変更するには、Rename() メソッドを使用します。このメソッドは、成功した場合に 1 を返し、失敗した場合に 0 を返します。このメソッドは、3 つの引数を取ります。1 つ目の引数は名前を変更するファイルの名前で、2 つ目の引数は新しい名前です。3 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

以下の 1 つ目の例では、メソッドは成功します。2 つ目の例は失敗し、Windows エラー・コード 183 (″既に存在するファイルを作成することはできません″) が返されます。

```
USER>write ##class(%File).Rename("C:\temp\oldname.txt", "C:\temp\newname.txt", .return)
1
USER>write ##class(%File).Rename("C:\temp\another.txt", "C:\temp\newname.txt", .return)
0
USER>write return
-183
```

このメソッドを使用する際には、パスの指定を慎重に行う必要があります。例えば、以下の例は、C:¥temp¥oldname.txt を既定のディレクトリに移動してから、newname.txt に名前変更する処理となります。

```
USER>write ##class(%File).Rename("C:\temp\oldname.txt", "newname.txt", .return)
1
```

5.5 ファイルの比較

2 つのファイルを比較するには、Compare() メソッドを使用します。このメソッドは、2 つのファイルがまったく同じ場合にブーリアン値 1 を返し、そうでない場合に 0 を返します。このメソッドには、システム・エラー・コードを返すための出力引数はありません。

以下の 1 つ目の例では、2 つのファイルはまったく同じで、このメソッドは 1 を返します。2 つ目の例では、2 つのファイルは異なるため、このメソッドは 0 を返します。

```
USER>write ##class(%File).Compare("C:\temp\old.txt", "C:\temp\new.txt")
1
USER>write ##class(%File).Compare("C:\temp\old.txt", "C:\temp\another.txt")
0
```

以下の例に示すように、一方または両方のファイルが存在しない場合も、このメソッドは 0 を返します。

```
USER>write ##class(%File).Compare("foo.txt", "bar.txt")
0
USER>write ##class(%File).Exists("foo.txt")
0
```

5.6 一時ファイルの生成

一時ファイルを生成するには、TempFilename() メソッドを使用します。このメソッドは、一時ファイルの名前を返します。このメソッドは、3 つの引数を取ります。1 つ目の引数は、一時ファイルの目的のファイル拡張子です。2 つ目は、一時ファイルの生成先のディレクトリです。これを指定しない場合、メソッドは OS 提供の一時ディレクトリにファイルを生成します。3 つ目の引数は、出力引数です。これが負の値の場合、メソッドが失敗したときにオペレーティング・システムから返されるエラー・コードが含まれます。

Windows の例 :

```
USER>write ##class(%File).TempFilename("txt")
C:\WINDOWS\TEMP\GATqk8a6.txt
USER>write ##class(%File).TempFilename("txt","C:\temp")
C:\temp\WpSwuLlA.txt
```

Unix の例：

```
USER>write ##class(%File).TempFilename("", "", .return)
/tmp/filsfHGzc
USER>write ##class(%File).TempFilename("tmp", "/InterSystems/temp", .return)
/InterSystems/temp/file0tnuh.tmp
USER>write ##class(%File).TempFilename("", "/tmp1", .return)

USER>write return
-2
```

上記の 3 つ目の例では、ディレクトリが存在しないためメソッドは失敗し、システム・エラー・コード 2 (″このようなファイルまたはディレクトリはありません″) が返されます。

6 %File オブジェクトの操作

ファイル自体を操作する場合、%Library.File クラスの %New() メソッドを使用して、%File オブジェクトをインスタンス化する必要があります。このクラスには、ファイルの操作に使用できるインスタンス・メソッドも用意されています。

注釈　ここでは、%File オブジェクトを使用した例を説明のためにいくつか紹介しています。

単にファイルの読み取りおよび書き込みを行う場合は、%Stream.FileCharacter クラスおよび %Stream.FileBinary クラスを使用してください。これらのクラスでは、ファイルを自動的に適切なモードで開くといった追加機能を利用できるためです。

6.1 %File オブジェクトのインスタンスの作成

ファイルを操作するには、%New() メソッドを使用して、そのファイルを表す %File オブジェクトをインスタンス化する必要があります。対象のファイルは、ディスクに既に存在する場合も存在しない場合もあります。

以下の例では、既定のディレクトリ内のファイル export.xml の %File オブジェクトをインスタンス化します。

```
set fileObj = ##class(%File).%New("export.xml")
```

6.2 ファイルの開閉

%File オブジェクトをインスタンス化した後、ファイルに対して読み取りまたは書き込みを行うには、Open() メソッドを使用してファイルを開く必要があります。

```
USER>set status = fileObj.Open()

USER>write status
1
```

ファイルを閉じるには、Close() メソッドを使用します。

```
USER>do fileObj.Close()
```

6.3 %File オブジェクトのプロパティの確認

ファイルをインスタンス化したら、ファイルのプロパティを直接確認できます。

```
USER>write fileObj.Name
export.xml
USER>write fileObj.Size
2512
USER>write $zdate(fileObj.DateCreated)
11/18/2020
USER>write $zdate(fileObj.DateModified)
11/18/2020
USER>write fileObj.LastModified
2020-11-18 14:24:38
USER>write fileObj.IsOpen
0
```

LastModified は人が読める形式のタイムスタンプであり、$H 形式の日付ではないことに注意してください。

Size、DateCreated、DateModified、および LastModified の各プロパティは、アクセスされたときに計算されます。存在しないファイルのプロパティにアクセスすると、ファイルが見つからなかったことを示す -2 が返されます。

注釈　現在、実際に作成された日付を追跡しているプラットフォームは Windows のみです。その他のプラットフォームは、ファイル・ステータスの最後の変更の日付を格納しています。

```
USER>write ##class(%File).Exists("foo.xml")
0
USER>set fooObj = ##class(%File).%New("foo.xml")

USER>write fooObj.Size
-2
```

ファイルが開いているときに CanonicalName プロパティにアクセスすると、そのファイルのキャノニック形式の名前を表示できます。これは、ルート・ディレクトリからのフル・パスです。

```
USER>write fileObj.CanonicalName

USER>set status = fileObj.Open()

USER>write fileObj.IsOpen
1
USER>write fileObj.CanonicalName
c:\intersystems\IRIS\mgr\user\export.xml
```

6.4 ファイルからの読み取り

ファイルから読み取るには、ファイルを開いてから Read() メソッドを使用します。

以下の例では、messages.log の最初の 200 文字を読み取ります。

```
USER>set messages = ##class(%File).%New(##class(%File).ManagerDirectory() _ "messages.log")

USER>set status =  messages.Open("RU")

USER>write status
1
USER>set text = messages.Read(200, .sc)

USER>write text


*** Recovery started at Mon Dec 09 16:42:01 2019
     Current default directory: c:\intersystems\IRIS\mgr
     Log file directory: .\
     WIJ file spec: c:\intersystems\IRIS\mgr\IR
USER>write sc
1
USER>do messages.Close()
```

ファイルから 1 行全体を読み取るには、ReadLine() メソッドを使用します。これは、%Library.File の親クラス %Library.AbstractStream から継承されるメソッドです。

以下の例では、C:¥temp¥shakespeare.txt の最初の行を読み取ります。

```
USER>set fileObj  = ##class(%File).%New("C:\temp\shakespeare.txt")

USER>set status =  fileObj.Open("RU")

USER>write status
1
USER>set text = fileObj.ReadLine(,.sc)

USER>write text
Shall I compare thee to a summer's day?
USER>write sc
1
USER>do fileObj.Close()
```

6.5 ファイルへの書き込み

ファイルに書き込むには、ファイルを開いてから Write() メソッドまたは WriteLine() メソッドを使用します。

以下の例では、新規ファイルに 1 行のテキストを書き込みます。

```
USER>set fileObj = ##class(%File).%New("C:\temp\newfile.txt")

USER>set status = fileObj.Open("WSN")

USER>write status
1
USER>set status = fileObj.WriteLine("Writing to a new file.")

USER>write status
1
USER>write fileObj.Size
24
```

6.6 ファイルの巻き戻し

ファイルに対して読み取りまたは書き込みを行った後に、Rewind() メソッドを使用してファイルを巻き戻すと、ファイルの先頭から操作を実行できます。

前述の例で停止した場所から処理を始める場合、fileObj は現在、ファイルの末尾にあります。ファイルを巻き戻してもう一度 WriteLine() を使用すると、ファイルを上書きする処理が行われます。

```
USER>set status = fileObj.Rewind()

USER>write status
1
USER>set status = fileObj.WriteLine("Rewriting the file from the beginning.")

USER>write status
1
USER>write fileObj.Size
40
```

ファイルをいったん閉じてから再度開くことでも、ファイルを巻き戻すことができます。

```
USER>do fileObj.Close()

USER>set status = fileObj.Open("RU")

USER>write status
1
USER>set text = fileObj.ReadLine(,.sc)

USER>write sc
1
USER>write text
Rewriting the file from the beginning.
```

6.7 ファイルのクリア

ファイルをクリアするには、ファイルを開いてから Clear() メソッドを使用します。これにより、ファイル・システムからファイルが削除されます。

以下の例では、既定のディレクトリ内のファイル junk.xml がクリアされます。

```
USER>write ##class(%File).Exists("junk.xml")
1
USER>set fileObj = ##class(%File).%New("junk.xml")

USER>set status = fileObj.Open()

USER>write status
1
USER>set status = fileObj.Clear()

USER>write status
1
USER>write ##class(%File).Exists("junk.xml")
0
```

7 例

以下の例は、ここで紹介した %Library.File メソッドのいくつかを使用したサンプル・クラスを示しています。

サンプル・クラス User.FileTest では、ProcessFile() メソッドが入力ファイルと出力ファイルを受け入れ、SetUpInputFile() と SetUpOutputFile() を呼び出してファイルを開き、一方を読み取り、他方を書き込みに使用します。次に、入力ファイルを 1 行ずつ読み取り、ProcessLine() を呼び出して、各行の内容に対して 1 つ以上の置換を実行して、各行の新しい内容を出力ファイルに書き込みます。

```
Include %occInclude

Class User.FileTest Extends %Persistent
{

/// Set up the input file
/// 1. Create a file object
```

```
/// 2. Open the file for reading
/// 3. Return a handle to the file object
ClassMethod SetUpInputFile(filename As %String) As %File
{
    Set fileObj = ##class(%File).%New(filename)
    Set status = fileObj.Open("RU")
    if $$$ISERR(status) {
        do $system.Status.DisplayError(status)
        quit $$$NULLOREF
    }
    quit fileObj
}

/// Set up the output file
/// 1. Create the directory structure for the file
/// 2. Create a file object
/// 3. Open the file for writing
/// 4. Return a handle to the file object
ClassMethod SetUpOutputFile(filename As %String) As %File
{
    set dir=##class(%File).GetDirectory(filename)
    do ##class(%File).CreateDirectoryChain(dir)
    Set fileObj = ##class(%File).%New(filename)
    Set status = fileObj.Open("WSN")
    If ($SYSTEM.Status.IsError(status)) {
        do $system.Status.DisplayError(status)
        quit $$$NULLOREF
    }
    quit fileObj
}

/// Process one line, using $REPLACE to perform a series of substitutions on the line
ClassMethod ProcessLine(line As %String = "") As %String
{
    set newline = line

    set newline = $REPLACE(newline, "Original", "Jamaican-Style")
    set newline = $REPLACE(newline, "traditional", "innovative")
    set newline = $REPLACE(newline, "orange juice", "lime juice")
    set newline = $REPLACE(newline, "orange zest", "ginger")
    set newline = $REPLACE(newline, "white sugar", "light brown sugar")

    quit newline
}

/// Process an input file, performing a series of substitutions on the content and
/// writing the new content to an output file
ClassMethod ProcessFile(inputfilename As %String = "", outputfilename As %String = "")
{
    // Make sure filenames were passed in
    if (inputfilename="") || (outputfilename="") {
        write !, "ERROR: missing file name"
        quit
    }

    // Open input file for reading
    set inputfile = ..SetUpInputFile(inputfilename)
    if (inputfile = $$$NULLOREF) quit

    // Open output file for writing
    set outputfile = ..SetUpOutputFile(outputfilename)
    if (outputfile = $$$NULLOREF) quit

    // Loop over each line in the input file
    // While not at the end of the file:
    // 1. Read a line from the file
    // 2. Call ProcessLine() to process the line
    // 3. Write the new contents of the line to the output file
    while (inputfile.AtEnd = 0) {
        set line = inputfile.ReadLine(,.status)
         if $$$ISERR(status) {
             do $system.Status.DisplayError(status)
         }
         else {
             set newline = ..ProcessLine(line)
             do outputfile.WriteLine(newline)
         }
    }

    // Close the input and output files
     do inputfile.Close()
     do outputfile.Close()
}

}
```

ProcessFile() メソッドは、以下のように呼び出します。

```
USER>do ##class(FileTest).ProcessFile("C:\temp\original cranberry sauce.txt",
"C:\temp\jamaican-style cranberry sauce.txt")
```

入力ファイル C:¥temp¥original cranberry sauce.txt の内容が以下であるとします。

```
Original Whole Berry Cranberry Sauce

This traditional whole berry cranberry sauce gets its distinctive flavor
from the freshly squeezed orange juice and the freshly grated orange zest.

2 tsp freshly grated orange zest
1 1/4 cups white sugar
1/4 cup freshly squeezed orange juice
3 cups cranberries (12 oz. package)

1. Grate orange zest into a bowl and set aside.
2. Combine the sugar and orange juice in a saucepan. Bring to a boil over medium-low
heat and stir until sugar is dissolved.
3. Add cranberries and cook over medium-high heat, stirring occasionally, until the
cranberries have popped.
4. Add the cranberry mixture into the bowl with the orange zest, and stir. Let cool.
5. Cover bowl and chill.
```

この場合、出力ファイル C:¥temp¥jamaican-style cranberry sauce.txt の内容は以下のようになります。

```
Jamaican-Style Whole Berry Cranberry Sauce

This innovative whole berry cranberry sauce gets its distinctive flavor
from the freshly squeezed lime juice and the freshly grated ginger.

2 tsp freshly grated ginger
1 1/4 cups light brown sugar
1/4 cup freshly squeezed lime juice
3 cups cranberries (12 oz. package)

1. Grate ginger into a bowl and set aside.
2. Combine the sugar and lime juice in a saucepan. Bring to a boil over medium-low
heat and stir until sugar is dissolved.
3. Add cranberries and cook over medium-high heat, stirring occasionally, until the
cranberries have popped.
4. Add the cranberry mixture into the bowl with the ginger, and stir. Let cool.
5. Cover bowl and chill.
```