4D 階層リストの保存と読み込み

By Charles “ Charlie” Vass, Technical Services Team Member, 4D Inc.

Technical Note 09-26

概要

このテクニカルノートでは、4D の階層リストを書き出す方法を 2 つ示します。一つは XML ファイル、もう一つは Blob ドキュメントへの書き出しです。またこれらの XML あるいは Blob ファイルから階層リストを読み込む方法も示します。

XML ファイルの処理では、4D の SAX XML コマンドを再帰呼び出しのメソッドで使用して、階層リストを効率的に書き出し/読み込みする方法をお見せします。

はじめに

4D リストを使用してデータ入力制御を効果的に行うことができます。リストはデータファイルへの入力を許可するデータを制限したり制御したりする等の目的で使用できます。しかしリストにはストラクチャ(.4DB)ファイルに保存されるという欠点もあります。4D リストはユーザによる編集を許可することができ、ゆえにリストはユーザサイドで変更されているかもしれません。ここで既存のストラクチャをあなたが開発した新しいストラクチャで置き換えると、ユーザがリストに対して行った変更は失われてしまいます。

ユーザが更新したリストをデータベースストラクチャ更新時にも保持するためには、開発者が計画を行わなければなりません。一つのテクニックとして On Backup Startup データベースメソッドや On Backup Shutdown データベースメソッドを使用できます。このテクニカルノートではこれらのデータベースメソッドを使用してピクチャライブラリと階層リストを標準のバックアップ処理の一部としてアーカイブを試みます。

4D 階層リスト

4D は階層リストをメモリ中に存在するランゲージオブジェクトおよびフォームオブジェクトとして定義します。

ランゲージオブジェクトは、4D ランゲージリファレンスで"ListRef"と表記される、倍長整数型の ID でユニークに参照されます。この ID はリストを作成するコマンド **New list**、**Copy list**、**Load list**、**BLOB to list** コマンドから返されます。メモリ中にはただ 1 つのランゲージオブジ

ェクトインスタンスがあり、このオブジェクトに対して行われた変更は即座にそれを使用する個所に反映されます。

ランゲージオブジェクトとの大きな違いとして、フォームオブジェクトはユニークである必要はありません。同じフォーム上あるいは異なるフォーム上で、同じ階層リストを複数表示できます。他のフォームオブジェクトと同様、ランゲージ中で (*;"ListName"; ...) のシンタックスを使用してオブジェクトを指定できます。

フォームオブジェクトとしてのリストはそれぞれ固有の属性と、他の同じリスト表現との共有属性を持っています。以下の属性はそれぞれのリスト表現に固有のものです

- 選択された項目
- 項目の展開/折りたたみ状態
- カーソルのスクロール位置

他の属性 (フォント、フォントサイズ、スタイル、入力制御、カラー、リストの内容、アイコン等) はすべての表示で共通で、個別に変更はできません。

したがって、展開/折りたたみ状態や選択された項目に基づくコマンドを使用するとき、例えば Count list items を最後の * 引数なしで呼び出す場合は、明示的に使用するフォーム上の階層リストオブジェクトを指定できなければなりません。

メモリ中の階層リストを指定するには “ListRef”をランゲージコマンドに渡します。

フォームレベルで階層リストオブジェクトを指定するには (*;"ListName"; ...) シンタックスの “オブジェクト名” (文字型) を使用しなければなりません。

このテクニカルノートではメモリ中の階層リストのみを扱うので、すべてのコードで “ListRef” ID が使用されます。

4D 階層リストの書き出し - XML

階層リストを書き出す 2 つの方法のうち、XML を使用する方は複雑で教育的です。*HL_Export* メソッドを通し、このドキュメントでは 4D SAX コマンドを使用したリストのロード、解析、書き出しの方法を示します。

すべてのファイルは外部ファイルとして “Active 4D Folder”に格納されます。このフォルダの場所は以下の通りです(POSIX パス):
Mac OS: “/Users/<user>/Library/Preferences/4D/”
Windows XP: “/Documents and Settings/<user>/Application Data/4D/”
Windows Vista: “/Users/<user>/AppData/Roaming/4D/”

HL_Export

4D 階層リストを XML ファイルに書き出すおもな利点は、それを再び読み込む前に、開発者あるいはデータベース管理者がその中身を確認し、必要であれば XML を編集できることです。

このテクニックを実装するには、メソッドが自分自身を呼び出す再帰呼び出しを使用しなければなりません。注意深くコードを書かないと無限ループに陥ってしまい、スタックオーバーフローが発生して、最後にはアプリケーションがクラッシュするかもしれません。

だからと言って再帰呼び出しを使ってはいけないというわけではありません。これは未知のレベルの階層構造を持つデータの処理にとても効率的なテクニックです。これがよく使用されるのは未知の数のレイヤーのフォルダに格納されたファイルやフォルダを読み込んだり処理したりする場合です。再帰呼び出しはよく使用されるテクニックであり、すべての 4D デベロッパはこれに慣れておくべきです。

再帰呼び出しの中でどのような処理を行うかにかかわらず、スタックサイズは余裕を持って割り当てる必要があります。タスクを実行するためにどれほどのレベルが必要になるか分からないためです。

```
$InstalledErrHandler_T:=Method called on error
ON ERR CALL("HL_ErrorHandler")

OK:=1
    `// Active 4D folder に保存先フォルダが存在するか確認する
    `{
$Path_T:=HL_FolderPath ("ExportedLists")
    `}

If (OK=1)
    `// インストール済みのリスト一覧を取得
    `{
  ARRAY LONGINT($ListNum_aL;0)
  ARRAY TEXT($ListName_aT;0)
  LIST OF CHOICE LISTS($ListNum_aL;$ListName_aT)
  $SOA:=Size of array($ListName_aT)
    `}

  For ($Ndx;1;$SOA)
    $FileName_T:="List_"+$ListName_aT{$Ndx}+".XML"
    $DocRef_H:=Create document($Path_T+$FileName_T)
    If (OK=1)

       `// XML エンコーディングとタイムスタンプコメントを追加
       `// <!-- File created 2009-05-25T07:29:34-->
       `{
     SAX SET XML OPTIONS($DocRef_H;"UTF-8";True)
     SAX ADD XML COMMENT($DocRef_H;"File created "+Replace
string(String(Current date;ISO Date );"00:00:00";String(Current time)))
       `}
```

最初のポイントは、すべての"親"項目の数を取得することです。そのため次のコードブロックではリスト項目を下から上に走査し、すべての展開属性を False に設定、そして 2 回目の Count list items を呼び出します。

```
       `// リストを開き、すべての項目を折りたたむ
       `{
     $HL_Ref_L:=Load list($ListName_aT{$Ndx})
     $Cnt:=Count list items($HL_Ref_L)
     For ($Idx;$Cnt;1;-1)
GET LIST ITEM($HL_Ref_L;$Idx;$ItemRef_L;$ItemText_T;$SubList_L;$Expanded_B)

       If ($SubList_L#0)) `// “expanded” 属性を false に設定
SET LIST ITEM($HL_Ref_L;$ItemRef_L;$ItemText_T;$ItemRef_L;$SubList_L; False)
       End if
     End for
     $Cnt:=Count list items($HL_Ref_L)
       `}
```

XML 形式でリスト項目を保存する際には、2 つの選択肢があります。要素として書き出すか、属性として書き出すかです。このテクニカルノートでは属性値として書き出す方法を選択しました。

```
      `// リストプロパティを取得して XML 要素の属性として保存する
      `{
GET LIST PROPERTIES
($HL_Ref_L;$Appearance_L;$Icon_L;$LineHeight_L;$DoubleClick_L;$MultiSelections_
L;$Editable_L)

SAX OPEN XML ELEMENT
($DocRef_H;"List";"Name";$ListName_aT{$Ndx};"ItemCount";String($Cnt);"Appearanc
e";String($Appearance_L);"Icon";String($Icon_L);"LineHeight";String($LineHeight
_L);"DoubleClick";String($DoubleClick_L);"MultiSelections";String($MultiSelecti
ons_L);"Editable";String($Editable_L))

      `====================== Method Actions ================================

     For ($Idx;1;$Cnt)
      `// リスト項目を取得し、書き出す
      `{
GET LIST ITEM ($HL_Ref_L;$Idx;$ItemRef_L;$ItemText_T;$SubList_L;$Expanded_B)
GET LIST ITEM PROPERTIES
($HL_Ref_L;$ItemRef_L;$Enterable_B;$Styles_L;$Icon_L;$Color_L)
```

このテクニカルノート執筆の時点では、コマンド Count list items の動作に不審な点がありました。SubList をテストしているとき、そのリストのいずれかの要素にアイコンが割り当てられていると、リストを走査してすべての項目の展開属性を False に設定したにもかかわらず、 返される値は折りたたみ状態を反映しないものでした。メソッド *HL_Count_In_SubList* はこの問題を避けるように書かれていて、この問題が修正された後も動作します。

```
      If ($SubList_L#0)
        $Rdx:=HL_Count_In_SubList ($SubList_L;$ItemRef_L)
          `// XML 属性としてすべての情報を保存する
          `{
SAX OPEN XML ELEMENT
($DocRef_H;"SubList";"Name";$ItemText_T;"ItemCount";String($Rdx);"ItemRef";Stri
ng($ItemRef_L);"Enterable";Choose($Enterable_B;"True";"False");"Styles";String(
$Styles_L);"Icon";String($Icon_L);"Color";String($Color_L))
          `// 含まれるサブリストの処理
          ` {
        HL_ExportSublist ($SubList_L;$ItemRef_L;$ItemText_T;$DocRef_H)
          ` }
        SAX CLOSE XML ELEMENT($DocRef_H)
          ` }
      Else
          `// XML 属性としてリスト項目のすべての情報を保存する
```

```
          `{
SAX OPEN XML ELEMENT
($DocRef_H;"Item";"Name";$ItemText_T;"ItemRef";String($ItemRef_L);"Enterable";C
hoose($Enterable_B;"True";"False");"Styles";String($Styles_L);"Icon";String($Ic
on_L);"Color";String($Color_L))
         SAX CLOSE XML ELEMENT($DocRef_H)
           `}

       End if
         `}
     End for
     SAX CLOSE XML ELEMENT($DocRef_H)
       `}

     CLOSE DOCUMENT($DocRef_H)
     CLEAR LIST($HL_Ref_L)
   End if
  End for
End if

ON ERR CALL($InstalledErrHandler_T)
```

HL_Count_In_Sublist

メソッド *HL_Count_In_SubList* はサブリスト中にアイコンを割り当てられた項目が存在する場合の Count list items の不具合に対応するものです。Count list items はすべてが折りたたまれた項目の数か、サブリストを含めたすべての項目数に対して正しい数を返します。"親"項目をテストすることにより、このメソッドは第一レベルの子要素の正しい数を、展開/折りたたみ状態にかかわらず返します。

```
$SUBLIST_L:=$1
$SUBLIST_ItemRef_L:=$2
$SUBLIST_Cnt_L:=Count list items($SubList_L)

For ($Ndx;1;$SUBLIST_Cnt_L)
  GET LIST ITEM($SUBLIST_L;$Ndx;$ItemRef_L;$ItemText_T;$Child_L;$Expanded_B)
  $SubItemRef_L:=List item parent($SUBLIST_L;$ItemRef_L)
  If ($SubItemRef_L=$SUBLIST_ItemRef_L)
    $RIS:=$RIS+1
  End if
End for

$0:=$RIS
```

HL_ExportSublist

HL_ExportSublist は再帰的に呼び出されるメソッドで、添付されたサブリストの要素数がいくつであれ、正しく書き出しを処理します。

```
  `====================== Initialize and Setup ================================

$List_L:=$1
$ItemRef_L:=$2
$ItemText_T:=$3
$DocRef_H:=$4

  `// 折りたたまれたリスト項目数をカウントする
  `{
$Cnt_L:=Count list items($List_L)
For ($Ndx;$Cnt_L;1;-1)
  GET LIST ITEM($List_L;$Ndx;$ItemRef_L;$ItemText_T;$SubList_L;$Expanded_B)
  If ($SubList_L#0) `// Set the “expanded” attribute to false.
    SET LIST ITEM($List_L;$ItemRef_L;$ItemText_T;$ItemRef_L;$SubList_L;False)
  End if
End for
$Cnt_L:=Count list items($List_L)
  `}

  `======================== Method Actions ==================================

  `// サブリスト項目を処理
  `{
For ($Ndx;1;$Cnt_L)
  GET LIST ITEM($List_L;$Ndx;$ItemRef_L;$ItemText_T;$SubList_L)
  GET LIST ITEM PROPERTIES
($List_L;$ItemRef_L;$Enterable_B;$Styles_L;$Icon_L;$Color_L)
  If ($SubList_L#0)
     `// XML 属性にリスト項目のすべての情報を保存
     `{
    SAX OPEN XML ELEMENT
($DocRef_H;"SubList";"Name";$ItemText_T;"ItemCount";String(Count list
items($SubList_L));"ItemRef";String($ItemRef_L);"Enterable";Choose($Enterable_B
;"True";"False");"Styles";String($Styles_L);"Icon";String($Icon_L);"Color";Stri
ng($Color_L))

     `// 再帰呼び出しでサブリストの中身を処理
     ` {
    HL_ExportSublist ($SubList_L;$ItemRef_L;$ItemText_T;$DocRef_H)
     ` }
    SAX CLOSE XML ELEMENT($DocRef_H)
     `}

  Else
    `// XML 属性にリスト項目のすべての情報を保存
    `{
  SAX OPEN XML ELEMENT
($DocRef_H;"Item";"Name";$ItemText_T;"ItemRef";String($ItemRef_L);"Enterable";C
hoose($Enterable_B;"True";"False");"Styles";String($Styles_L);"Icon";String($Ic
on_L);"Color";String($Color_L))
  SAX CLOSE XML ELEMENT($DocRef_H)
    `}
  End if
End for
  `}
```

4D 階層リストの書き出し - BLOB

HL_Export_BLOB

HL_Export_BLOB メソッドはすべてのリストを Blob に書き出します。4D はメモリ中の階層リストをそのまま Blob に格納します。このメソッドを実行した後、ファイルを開いても階層リストの状態を見ることはできない点に留意してください。リストは 4D のバイナリフォーマットで書き出されます。

```
$InstalledErrHandler_T:=Method called on error
ON ERR CALL("HL_ErrorHandler")

OK:=1
  `// Active 4D folder に保存先フォルダが存在するか確認
  `{
$Path_T:=HL_FolderPath ("ExportedBLOBs")
  `}

If (OK=1) `// インストール済みのリスト一覧を取得
    `{
  ARRAY LONGINT($ListNum_aL;0)
  ARRAY TEXT($ListName_aT;0)
  LIST OF CHOICE LISTS($ListNum_aL;$ListName_aT)
  $SOA:=Size of array($ListName_aT)
    `}
  For ($Ndx;1;$SOA)
    $FileName_T:="List_"+$ListName_aT{$Ndx}+".BLOB"

      `======================== Method Actions ================================

    $HL_Ref_L:=Load list($ListName_aT{$Ndx})
    LIST TO BLOB($HL_Ref_L;$BLOB)
    BLOB TO DOCUMENT($Path_T+$FileName_T;$BLOB)
    CLEAR LIST($HL_Ref_L)

      `======================== Clean up and Exit ============================

  End for
End if

ON ERR CALL($InstalledErrHandler_T)
```

割り当てられたピクチャの保存

リスト項目にはアイコンを割り当てることができるため、ピクチャライブラリの書き出しと読み込みも必要です。実際 ‘cicn’ や ‘PICT’ リソース、および 4D ピクチャライブラリのピクチャ

をアイコンとして使用できます。ここで示すメソッドはピクチャライブラリのみを考慮に入れています。GET LIST ITEM PROPERTIES コマンドから返される icon 値をテストすれば、値が 65536 より小さければアイコンは ‘cicn’、 65536 より大きくて 131072 より小さければ‘PICT’ リソース、131072 より大きければピクチャライブラリだと分かります。

PL_Export

メソッド *PL_Export* はデータベースのピクチャライブラリを走査し、4D の SEND VARIABLE コマンドを使用して画像を一つのファイルに書き出します。

```
ARRAY LONGINT($PicRef_aL;0)
ARRAY TEXT($PicName_aT;0)

PICTURE LIBRARY LIST($PicRef_aL;$PicName_aT)
$SOA:=Size of array($PicRef_aL)

$InstalledErrHandler_T:=Method called on error
ON ERR CALL("HL_ErrorHandler")

OK:=1
  `// Active 4D folder に保存先フォルダがあるか確認
  `{
$Path_T:=HL_FolderPath ("ExportedPictures")
  `}

If ((OK=1)& ($SOA>0))

    `======================== Method Actions ==================================

  SET CHANNEL(12;$Path_T+"PictLibExport")
  If (OK=1)
    $Tag_T:="4D_PictureLibraryExport"
    SEND VARIABLE($Tag_T)
    SEND VARIABLE($SOA)
    $Error_L:=0
    For ($Ndx;1;$SOA)
      $PicRef_L:=$PicRef_aL{$Ndx}
      $PicName_T:=$PicName_aT{$Ndx}
      GET PICTURE FROM LIBRARY($PicRef_aL{$Ndx};$Picture_G)
      If (OK=1)
        SEND VARIABLE($PicRef_L)
        SEND VARIABLE($PicName_T)
        SEND VARIABLE($Picture_G)
      Else
        $Ndx:=$SOA+1
        $Error_L:=-108
      End if
    End for
    SET CHANNEL(11)
    If ($Error_L#0)
      ALERT("ピクチャライブラリを書き出せません。メモリを増やして再試行してください。")
      DELETE DOCUMENT(Document)
```

```
    End if
  End if
Else
  ALERT("ピクチャライブラリは空です。")
End if
```

PL_Import

PL_Import メソッドは先のメソッドに対応するもので、ファイルから画像を読み込み、ピクチャライブラリに格納します。

```
$Path_T:=HL_FolderPath ("ExportedPictures")+"ExportedPictures"
SET CHANNEL(10;$Path_T)
If (OK=1)

   `======================== Method Actions ==================================

  RECEIVE VARIABLE($Tag_T)
  If ($Tag_T="4D_PictureLibraryExport")
    RECEIVE VARIABLE($SOA)
    If ($SOA>0)
      For ($Ndx;1;$SOA)
        RECEIVE VARIABLE($PicRef_L)
        If (OK=1)
          RECEIVE VARIABLE($PicName_T)
        End if
        If (OK=1)
          RECEIVE VARIABLE($Picture_G)
        End if
        If (OK=1)
          SET PICTURE TO LIBRARY($Picture_G;$PicRef_L;$PicName_T)
        Else
          $Ndx:=$SOA+1
          ALERT("ファイルが壊れているようです。")
        End if
      End for
    Else
      ALERT("ファイルが壊れているようです。")
    End if
  Else
    ALERT("ファイル \""+Document+"\" はピクチャライブラリが書き出されたファイルではありません。")
  End if

   `======================== Clean up and Exit ================================

  SET CHANNEL(11)

End if
```

4D 階層リストの読み込み - XML

ここで紹介するメソッドは“ExportedLists” フォルダに保存された XML ファイルを読み込み、メモリ中に階層リストを作成して、カレントの 4D データベースストラクチャファイルに保存します。書き出しと同様、サブリストを処理するために再帰呼び出しが使用されます。

HL_Import

XML ファイルを DOM を使用して読み込むときは、メモリを気にする必要があります。どの程度の量があるか分からない状況では 4D の SAX コマンドを使用できます。4D の SAX コマンドを使用すれば XML を上から下に一行ごと読み込み、イベントを受け取って、情報を処理することができます。

SAX コマンドを使用する際のキーポイントは SAX イベントを理解することです。SAX イベントは **SAX Get XML node** コマンドから返されます。このコマンドを **Repeat/Until** ループで、SAX XML End Document イベントが返されるまで繰り返し呼び出します。

```
$Path_T:=HL_FolderPath ("ExportedLists")
ARRAY TEXT($Documents_aT;0)
DOCUMENT LIST($Path_T;$Documents_aT)
`// Process the list of files
`{
$SOA:=Size of array($Documents_aT)
For ($Ndx;1;$SOA)
```

Note: SAX コマンドで読み込むドキュメントは Open document コマンドの読み込みのみモードで開かなくてはなりません。これにより 4D と Xerces ライブラリ間の衝突が避けられます。読み書きモードで開かれたドキュメントを SAX コマンドで解析しようとすると、警告メッセージが表示され解析は行われません。

```
$Ref_In_H:=Open document($Path_T+$Documents_aT{$Ndx};"XML";Read Mode )
If (OK=1)
  ARRAY TEXT($ATT_Names_aT;0)
  ARRAY TEXT($ATT_Values_aT;0)
  $Exit_B:=False

    `======================== Method Actions ==================================

    `// XML ファイルを解析
```

```
   `{
  Repeat
    $SAX_Event_L:=SAX Get XML node($Ref_In_H)

    Case of
      : ($SAX_Event_L=XML Start Document )
        $Start_B:=True

      : ($SAX_Event_L=XML Start Element )
        SAX GET XML ELEMENT
($Ref_In_H;$ELE_Name_T;$ELE_Prefix_T;$ATT_Names_aT;$ATT_Values_aT)

        If ($Start_B)
          $ELE_Root_T:=$ELE_Name_T
          $Start_B:=False
        End if

        Case of
          : ($ELE_Name_T="List")
            $LIST_Ref_L:=New list
            $LIST_Name_T:=$ATT_Values_aT{1}
            $LIST_ItemCnt_L:=Num($ATT_Values_aT{2})

`// appearance; icon; lineHeight; doubleClick; multiSelections; editable )
            `{

            SET LIST PROPERTIES
($LIST_Ref_L;Num($ATT_Values_aT{3});Num($ATT_Values_aT{4});Num($ATT_Values_aT{5
});Num($ATT_Values_aT{6});Num($ATT_Values_aT{7});Num($ATT_Values_aT{8}))
            `}

          : ($ELE_Name_T="SubList")
            $SUB_List_L:=New list
            $SUB_Name_T:=$ATT_Values_aT{1}
            $SUB_ItemCnt_L:=Num($ATT_Values_aT{2})
            $SUB_ItemRef_L:=Num($ATT_Values_aT{3})
            APPEND TO LIST
($LIST_Ref_L;$SUB_Name_T;$SUB_ItemRef_L;$SUB_List_L;False)

            `// enterable; styles; icon; color)
            `{
            SET LIST ITEM PROPERTIES
($LIST_Ref_L;$SUB_ItemRef_L;($ATT_Values_aT{4}="True");Num($ATT_Values_aT{5});N
um($ATT_Values_aT{6});Num($ATT_Values_aT{7}))
            `}
            HL_ImportSublist ($Ref_In_H;$SUB_List_L;$SUB_ItemCnt_L;$SUB_Name_T)

          : ($ELE_Name_T="Item")
            $ITEM_Name_T:=$ATT_Values_aT{1}
            $ITEM_Ref_L:=Num($ATT_Values_aT{2})
            APPEND TO LIST($LIST_Ref_L;$ITEM_Name_T;$ITEM_Ref_L)

            `// enterable; styles; icon; color)
            `{
            SET LIST ITEM PROPERTIES
($LIST_Ref_L;$SUB_ItemRef_L;($ATT_Values_aT{3}="True");
Num($ATT_Values_aT{4});Num($ATT_Values_aT{5});Num($ATT_Values_aT{6}))
            `}
          End case

        : ($SAX_Event_L=XML Comment )

        : ($SAX_Event_L=XML DATA )
```

```
      : ($SAX_Event_L=XML CDATA )

      : ($SAX_Event_L=XML End Element )
        SAX GET XML ELEMENT
($Ref_In_H;$ELE_Name_T;$ELE_Prefix_T;$ATT_Names_aT;$ATT_Values_aT)
        $Exit_B:=($ELE_Root_T=$ELE_Name_T)
     End case

   Until (($Exit_B) | ($SAX_Event_L=XML End Document ))
     `}

     `======================== Clean up and Exit ============================

   CLOSE DOCUMENT($Ref_In_H)
   SAVE LIST($LIST_Ref_L;$LIST_Name_T)
   CLEAR LIST($LIST_Ref_L;*)
  End if
End for
  `}
```

HL_ImportSublist

HL_ImportSublist メソッドは要素値の代わりに要素属性を使用し、またサブリストの項目を読み込むために再帰呼び出しを使用する XML の解析例です。

```
$Ref_In_H:=$1
$LIST_Ref_L:=$2
$LIST_ItemCnt_L:=$3
$LIST_Name_T:=$4

If ($LIST_ItemCnt_L>0)
  ARRAY TEXT($ATT_Names_aT;0)
  ARRAY TEXT($ATT_Values_aT;0)

`======================== Method Actions ==================================

  $Ndx:=0
  Repeat
    $SAX_Event_L:=SAX Get XML node($Ref_In_H)

    Case of
      : ($SAX_Event_L=XML Start Element )
        SAX GET XML ELEMENT
($Ref_In_H;$ELE_Name_T;$ELE_Prefix_T;$ATT_Names_aT;$ATT_Values_aT)

        Case of
          : ($ELE_Name_T="SubList")
            $SUB_List_L:=New list
            $SUB_Name_T:=$ATT_Values_aT{1}
            $SUB_ItemCnt_L:=Num($ATT_Values_aT{2})
            $SUB_ItemRef_L:=Num($ATT_Values_aT{3})
            APPEND TO LIST
($LIST_Ref_L;$SUB_Name_T;$SUB_ItemRef_L;$SUB_List_L;False)
```

```
            `// appearance; icon; lineHeight; doubleClick; multiSelections;
editable )
            `{
           SET LIST ITEM PROPERTIES
($SUB_List_L;$SUB_ItemRef_L;($ATT_Values_aT{4}="True");Num($ATT_Values_aT{5});N
um($ATT_Values_aT{6});Num($ATT_Values_aT{7}))
            `}

           HL_ImportSublist ($Ref_In_H;$SUB_List_L;$SUB_ItemCnt_L;$SUB_Name_T)

         : ($ELE_Name_T="Item")
           $ITEM_Name_T:=$ATT_Values_aT{1}
           $ITEM_Ref_L:=Num($ATT_Values_aT{2})
           APPEND TO LIST($LIST_Ref_L;$ITEM_Name_T;$ITEM_Ref_L)

            `// enterable; styles; icon; color)
            `{
           SET LIST ITEM PROPERTIES
($LIST_Ref_L;$SUB_ItemRef_L;($ATT_Values_aT{3}="True");Num($ATT_Values_aT{4});
Num($ATT_Values_aT{5});Num($ATT_Values_aT{6}))
            `}

       End case

     : ($SAX_Event_L=XML DATA )

     : ($SAX_Event_L=XML End Element )
       $Ndx:=$Ndx+1
       SAX GET XML ELEMENT
($Ref_In_H;$ELE_Name_T;$ELE_Prefix_T;$ATT_Names_aT;$ATT_Values_aT)
   End case
  Until ($Ndx=$LIST_ItemCnt_L)

   `======================== Clean up and Exit ================================

End if
```

4D 階層リストの読み込み - BLOB

HL_Import_BLOB

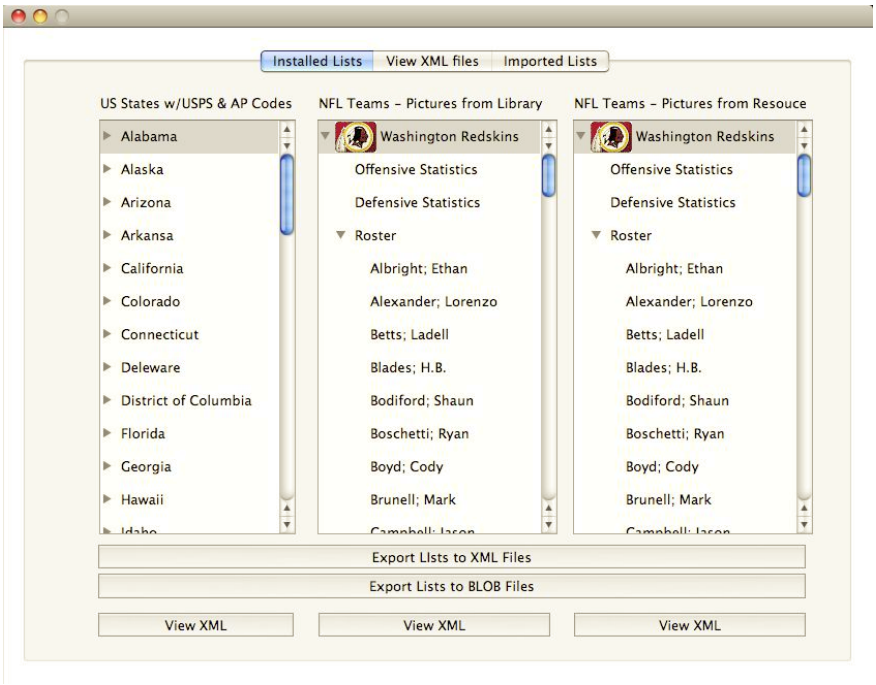
ここで示すメソッドは ExportedBLOBs フォルダに保存されたドキュメントを処理し、4D ストラクチャファイルに階層リストを保存します。

```
$InstalledErrHandler_T:=Method called on error
ON ERR CALL("HL_ErrorHandler")

$Path_T:=HL_FolderPath ("ExportedBLOBs")
ARRAY TEXT($Documents_aT;0)
DOCUMENT LIST($Path_T;$Documents_aT)
$SOA:=Size of array($Documents_aT)
```

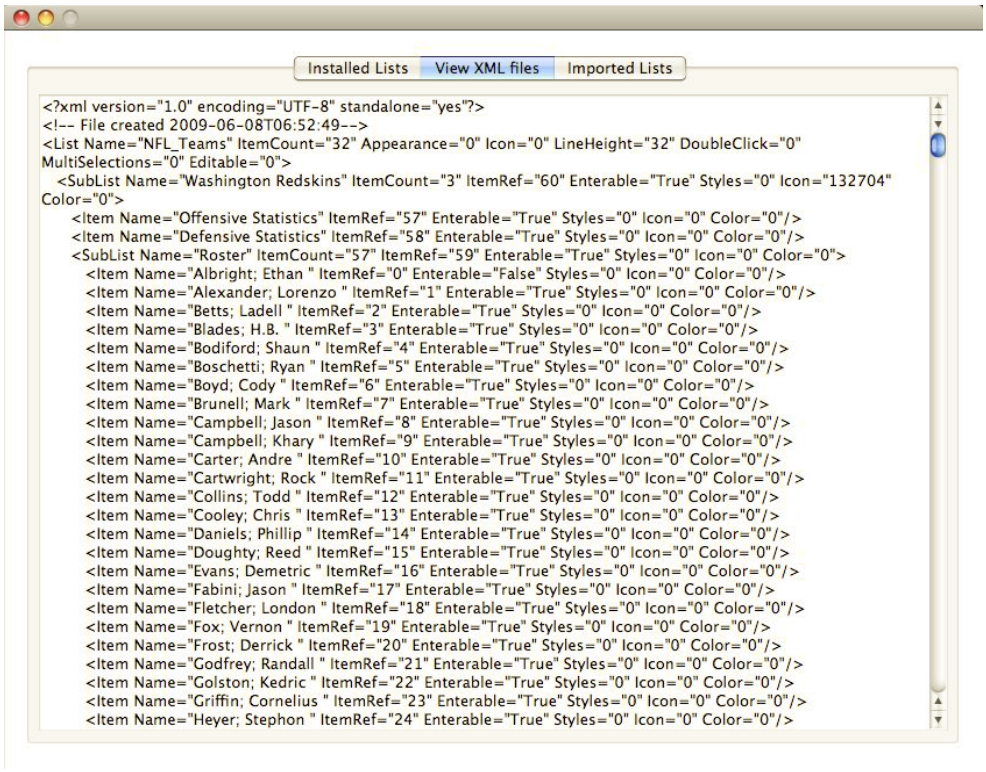
```
`======================== Method Actions ==================================

For ($Ndx;1;$SOA)
  DOCUMENT TO BLOB($Path_T+$Documents_aT{$Ndx};$BLOB)

  $HL_Ref_L:=BLOB to list($BLOB)
  SAVE LIST($HL_Ref_L;"Test2")
  CLEAR LIST($HL_Ref_L)
End for

`======================== Clean up and Exit ================================

ON ERR CALL($InstalledErrHandler_T)
```

デモ

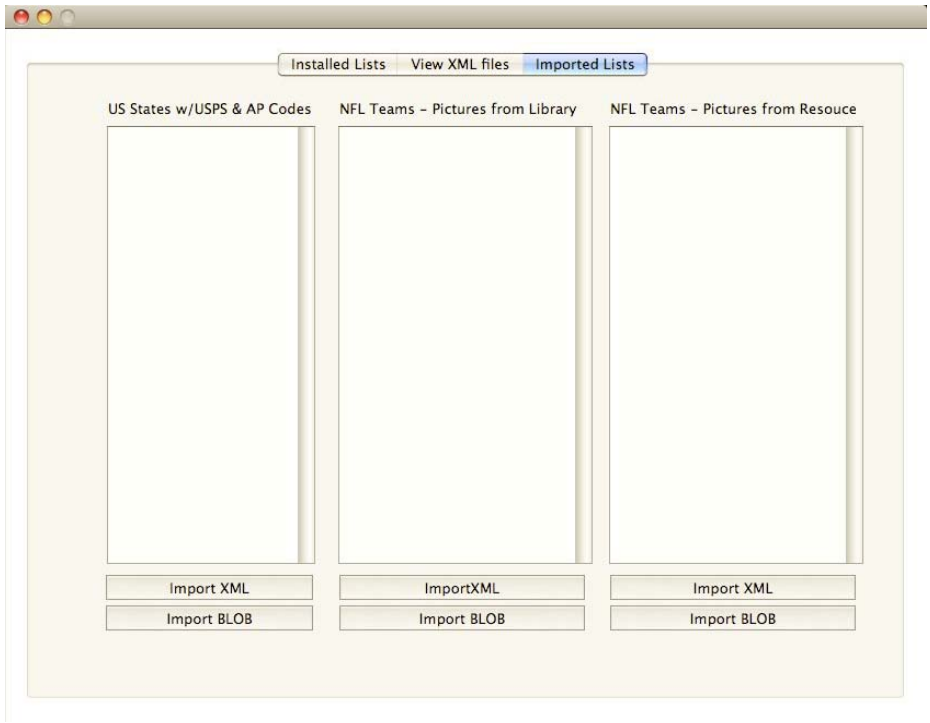
このテクニカルノートに含まれるデモは、ダイアログの 1 ページめに 3 つのリストを表示します。これらのリストを XML ファイルや BLOB ファイルに書き出し、XML ファイルについては読み込んで表示させることができます。



書き出した XML ファイルの表示を選択すると、ファイルがロードされ、自動で 2 ページ目に切り替わります。このページにはディスクに保存された XML が表示されます。要素値ではなく要素属性を使用してデータを保存する方法を見ていただくことができます。



Imported Lists ページでは、書き出したリストを読み込んで、このページのリストオブジェクトに表示できます。デモデータベース中、読み込んだリストはフォーム上にのみ表示され、データベースには保存されません。メソッド *DEMO_HL_Import* と *DEMO_HL_Import_BLOB* は先に説明したメソッドを変更したもので、書き出したリストをデモフォームオブジェクトにロードするために作成されています。



まとめ

このテクニカルノートでは 4D v11 SQL において階層リストを書き出し読み込む 2 つのテクニックを紹介しました。また 4D ピクチャライブラリを書き出し、読み込むメソッドも含まれています。既存あるいは新規のデータベースにこれらの機能を実装するために必要なコードがサンプルデータベースに含まれています。