How to Save and Restore 4D Hierarchical Lists

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

Technical Note 09-26

Table of Contents

Abstract

This Tech Note demonstrates two techniques of exporting 4D Hierarchical Lists. First to an XML file and then to a BLOB document. It demonstrates two complimentary techniques for importing the archived Hierarchical Lists from these XML or BLOB files.

In the process of exporting the XML files, this Tech Note demonstrates effective use of 4D SAX XML commands and 4D method recursion to efficiently export and import Hierarchical Lists.

Introduction

4D Lists are very useful in data entry control. They can be used to limit and control what data is committed to the data file and much more. But there is a drawback in that they are stored in the structure file (.4DB). Because 4D Hierarchical Lists can be made modifiable by the user, therein lies the rub. If you are deploying a new structure file for a client any changes they've made to these list will be lost without prior intervention on the part of the developer.

To be able to preserve any user modified lists between database structure updates, the developer has to plan ahead. One potential technique is to incorporate either the `On Backup Startup Database Method` or the `On Backup Shutdown Database Method`. `Both of` theese methods are incorporated in this TN to archive the Picture Library and Hierarchical Lists as a standard part of the backup process.

4D Hierarchical Lists

4D defines a hierarchical list as both a language object existing in memory and a form object.

The language object is referenced by an unique internal ID of the Longint type, designated by "ListRef" in the 4D language documentation. This ID is returned by the commands that can be used to create lists: **New list**, **Copy list**, **Load list**, **BLOB to list.** There is only one instance of the language object in memory and any modification carried out on this object is immediately carried over to all the places where it is used.

A significant difference from the language object, the form object is not necessarily unique: There may be several representations of the same hierarchical list in the same form or in different ones. As with other form objects, you specify the object in the language using the syntax (*;"ListName", ...).

Each form representation of the list has its own specific characteristics as well as sharing common characteristics with all the other representations. The following characteristics are specific to each representation of the list:

- The selection,
- The expanded/collapsed state of its items,
- The position of the scrolling cursor,

The other characteristics (font, font size, style, entry control, color, list contents, icons, etc.) are common to all the representations and can't be modified separately.

Consequently, when using commands based on the expanded/collapsed configuration or the current item, for example **Count list items** (when the final * parameter is not passed), it is important to be able to specify the representation to be used without any ambiguity.

You must use the "ListRef" ID with language commands when you want to specify the hierarchical list found in memory.

If you want to specify the representation of a hierarchical list object at the form level, you must use the "object name" (string type) in the command, via the syntax (*;"ListName", ...).

Since this tech note only deals with hierarchical lists in memory, only the "ListRef" ID is used in all code.

Exporting 4D Hierarchical Lists - XML

Of the two ways to save Hierarchical Lists, doing it in XML is the most involved and instructional of the two. With the method ***HL_Export***, this document walks through loading, parsing, and exporting the list using the 4D SAX commands.

All files are saved to the "Active 4D Folder", which is external to the Structure folder. On Mac that is (POSIX path) "/Users/<user>/Library/Preferences/4D/" and on Windows it is (POSIX path) "/Documents and Settings/<user>/Application Data/4D/".

HL_Export

The chief advantage of exporting 4D hierarchical lists to XML files is that it gives the developer or DBMS manager a way to inspect what is being stored in the lists and, if necessary, edit the XML to add, delete, or change what is in the list the next time it is imported into a 4D database.

A challenge to this technique is the advanced level of coding, namely the use of recursion, which is a method calling itself. If you are not careful in code, you may create an infinite loop, a stack overflow can occur and ultimately the application may fail..

These are not reasons to avoid using recursion, for it is a very efficient technique for handling unknown levels of recursively embedded data. Most commonly it is used for reading and manipulating files and folders embedded

with unknown layers of folders. Recursion happens naturally in nature and computer systems. Becoming comfortable with using it within 4D is important to every developer.

In whatever process you are going to use recursive techniques, ensure that you apply ample stack space to the process for, by its nature, you do not know how many levels of recursion will be needed to complete the task.

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

OK:=1
  `//  Make sure the archiving folder exists in the Active 4D folder
  `{
$Path_T:=HL_FolderPath ("ExportedLists")
  `}

If (OK=1)
          `//  Get the list of installed lists
          `{
        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)

         `//  Set the XML encoding and add a comment containing the a timestamp
         `//  <!-- 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)))
            `}
```

The first important point of detail is to make sure that we get a count of all root "Parent" items. To do that, the next block of code walks the list, from bottom to top, sets all of the "expanded" attributes to false and then calls **Count list items** a second time.

```
            `//  Open the list and force all items to be collapsed
            `{
          $HL_Ref_L:=Load list($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 ($SubList_L#0)  `//  Set the "expanded" attribute to false.
   SET LIST ITEM($HL_Ref_L;$ItemRef_L;$ItemText_T;$ItemRef_L;$SubList_L; False)

           End if
          End for
          $Cnt:=Count list items($HL_Ref_L)
            `}
```

There are two choices of style when it comes to saving list items as XML. You can use elements or attributes. This Tech Note demonstrates using attributes exclusively.

```
  `//  Get the list properties and save them as XML Element attributes
  `{
            GET LIST
PROPERTIES($HL_Ref_L;$Appearance_L;$Icon_L;$LineHeight_L;$DoubleClick_L;$MultiS
elections_L;$Editable_L)
            SAX OPEN XML ELEMENT($DocRef_H;"List";"Name";$ListName_aT{$Ndx};
"ItemCount";String($Cnt);"Appearance";String($Appearance_L);"Icon";
String($Icon_L);"LineHeight";String($LineHeight_L);"DoubleClick";
String($DoubleClick_L);"MultiSelections";String($MultiSelections_L);
"Editable";String($Editable_L))

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

            For ($Idx;1;$Cnt)
                `//  Get a list item and export it
                `{
          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)
```

At the time of the writing of this Tech Note there is an anomaly in the command **Count in list**. If you are testing a SubList and there are icons attached to any element in the list, the count may not reflect the collapsed count even after you have gone through and set all of their expanded properties to false. The method ***HL_Count_In_SubList*** is a workaround for this anomaly and will work even when this anomaly no longer exists.

```
              If ($SubList_L#0)
                $Rdx:=HL_Count_In_SubList ($SubList_L;$ItemRef_L)

`//  Save a list item using XML attributes to save all information, not element
values.
                  `{
                SAX OPEN XML
ELEMENT($DocRef_H;"SubList";"Name";$ItemText_T;"ItemCount";String($Rdx);"ItemRe
f";String($ItemRef_L);"Enterable";
Choose($Enterable_B;"True";"False");"Styles";String($Styles_L);"Icon";
String($Icon_L);"Color";String($Color_L))
                  `//  Process an included sublist
                  `  {
                HL_ExportSublist ($SubList_L;$ItemRef_L;$ItemText_T;$DocRef_H)
                  `  }
                SAX CLOSE XML ELEMENT($DocRef_H)
                  `}

              Else
`// Save a list item using XML attributes to save all information, not element
values.
                  `{
                SAX OPEN XML ELEMENT($DocRef_H;"Item";"Name";$ItemText_T;
"ItemRef";String($ItemRef_L);"Enterable";Choose($Enterable_B;"True";"False");
"Styles";String($Styles_L);"Icon";String($Icon_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

The method ***HL_Count_In_SubList*** is a workaround for the anomaly in **Count in list** when testing a SubList and icons have been attached to any element in the list. **`Count list items`** either returns the correct number of items with all collapsed or the number of total items assigned to the list and all associated sublists. By testing for the "Parent" item, this method returns the correct number of first generation child elements, expanded or collapsed.

```
$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

The method below, ***HL_ExportSublist***, is the "recursive" method that handles the exporting of sublists regardless of how many child elements have sublists attached.

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

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

  `//  Get the "collapsed" list item count
  `{
$Cnt_L:=Count list items($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  ==================================

  `//  Process the Sublist items
  `{
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)
`//  Save a list item using XML attributes for all information, not element values
           `{
          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($Icon_L);"Color";String($Color_L))

           `//  Process an included sublist using recursion
           `  {
          HL_ExportSublist ($SubList_L;$ItemRef_L;$ItemText_T;$DocRef_H)
           `  }
          SAX CLOSE XML ELEMENT($DocRef_H)
           `}

        Else
`//  Save a list item using XML attributes for all information, not element values
           `{
          SAX OPEN XML ELEMENT($DocRef_H;"Item";"Name";$ItemText_T;
"ItemRef";String($ItemRef_L);"Enterable";Choose($Enterable_B;"True";"False");
"Styles";String($Styles_L);"Icon";String($Icon_L);"Color";String($Color_L))
          SAX CLOSE XML ELEMENT($DocRef_H)
           `}

        End if
End for
  `}
```

Exporting 4D Hierarchical Lists - BLOB

HL_Export_BLOB

The next method ***HL_Export_BLOB*** exports all list in BLOBS. 4D saves the hierarchical list as it exists in memory and saves it into a BLOB. Be aware that after running this method, The list cannot be reviewed once exported for it is in 4D binary format for saving and importing BLOBs.

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

OK:=1
    `//  Make sure the archiving folder exists in the Active 4D folder
```

```
    `{
$Path_T:=HL_FolderPath ("ExportedBLOBs")
    `}

If (OK=1)     `//  Get the list of installed lists
          `{
         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)
```

Preserving associated pictures

Because icons can be associated to list items, it is also necessary to have the ability to export and import the picture library. True, icons can be assigned to a list item from a 'cicn' or 'PICT' resources or from 4D's Picture Library. The method shown below only address the Picture Library. Using 4Ds Resources commands and testing the icon values returned in **GET LIST ITEM PROPERTIES**, you can determine if the icon is a 'cicn' if the value is less than 65536, a 'PICT' resource if greater than 65536 but less than 131072, and from the Picture Library if greater than 131072.

PL_Export

The method below, ***PL_Export***, cycles through a database Picture Library and saves all of the images into a single file using 4D's **SEND VARIABLE** command.

```
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
    `//  Make sure the archiving folder exists in the Active 4D folder
    `{
```

```
$Path_T:=HL_FolderPath ("ExportedPictures")
    `}

If ((OK=1)& ($SOA>0))    `//  Get the list of installed lists

`========================  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("The Picture Library could not be exported, retry with more
memory.")
            DELETE DOCUMENT(Document)
          End if
        End if
Else
        ALERT("The Picture Library is empty.")
End if
```

PL_Import

The method below, ***PL_Import***, is the compliment to the one above. It restores images to the Picture Library from the file that was previously saved.

```
$Path_T:=HL_FolderPath ("ExportedPictures")+"ExportedPictures"
SET CHANNEL(10;$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("This file looks like being damaged.")
            End if
          End for

        Else
          ALERT("This file looks like being damaged.")

        End if
      Else
        ALERT("The file \""+Document+"\" is not a Picture Library export
file.")

      End if

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

      SET CHANNEL(11)
End if
```

Importing 4D Hierarchical Lists - XML

The methods below read the XML files saved to the "ExportedLists" folder, build a Hierarchical List in memory, and then save that list in the current 4D database structure file. As with the export, recursion is used to handle sublists.

HL_Import

When you don't want to tie up an unknown amount of memory by importing an entire XML file into a DOM, an excellent alternative is the 4D SAX commands. 4D SAX commands do not get a lot of attention for they do a very straight top-to-bottom task: Read an XML document one line at a time, tell you what is has just read and give you the opportunity to act upon that information.

The key point in using SAX commands is understanding SAX events. SAX events are returned by the command **SAX Get XML node**. This command is called repeatedly in a Repeat/Until loop, the Until is when the SAX XML End Document event is reported.

```
$Path_T:=HL_FolderPath ("ExportedLists")
ARRAY TEXT($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: Documents read by SAX commands must be opened in read-only mode by the Open document command. This avoids any conflict between 4D and the Xerces library when you open "standard" and XML documents simultaneously. If you execute a SAX parsing command with a document open in read-write mode, an alert message is displayed and parsing is impossible.

```
        $Ref_In_H:=Open document($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  ==================================

            `//  Parse the XML file
            `{
             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});
Num($ATT_Values_aT{6});Num($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

The method below, ***HL_ImportSublist***, demonstrates the heavy use of element attributes instead of element values and uses recursion to handle the import of items belonging to a sublist.

```
$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});
Num($ATT_Values_aT{6});Num($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
```

Importing 4D Hierarchical Lists - BLOB

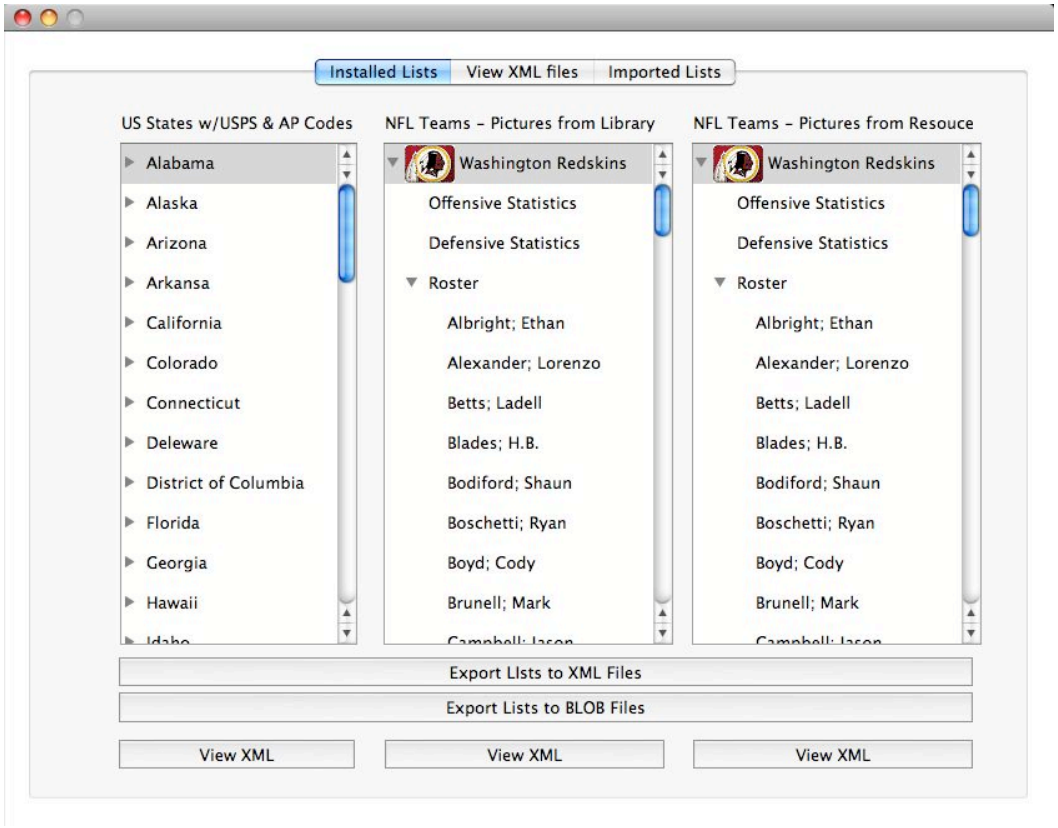
HL_Import_BLOB

The method below processes the documents saved in ExportedBLOBs folder and saves the hierarchical lists into the current 4D structure file.

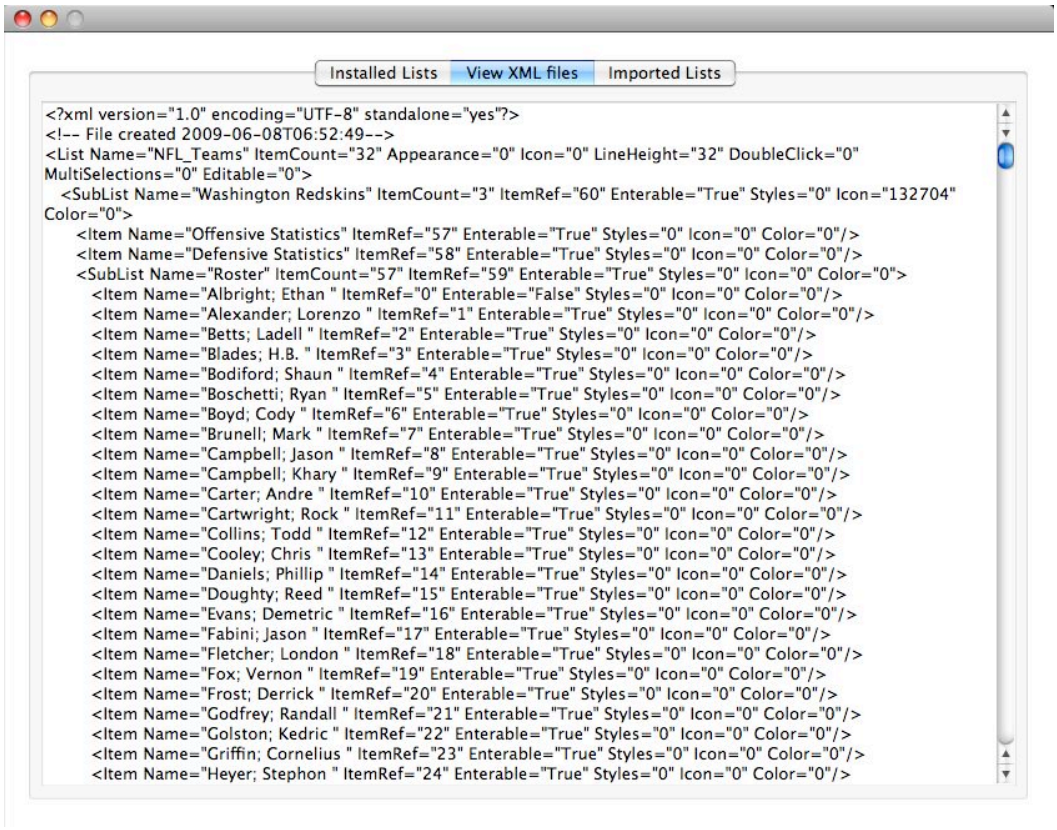
```
$InstalledErrHandler_T:=Method called on error
ON ERR CALL("HL_ErrorHandler")

$Path_T:=HL_FolderPath ("ExportedBLOBs")
ARRAY TEXT($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)
```

The Demo

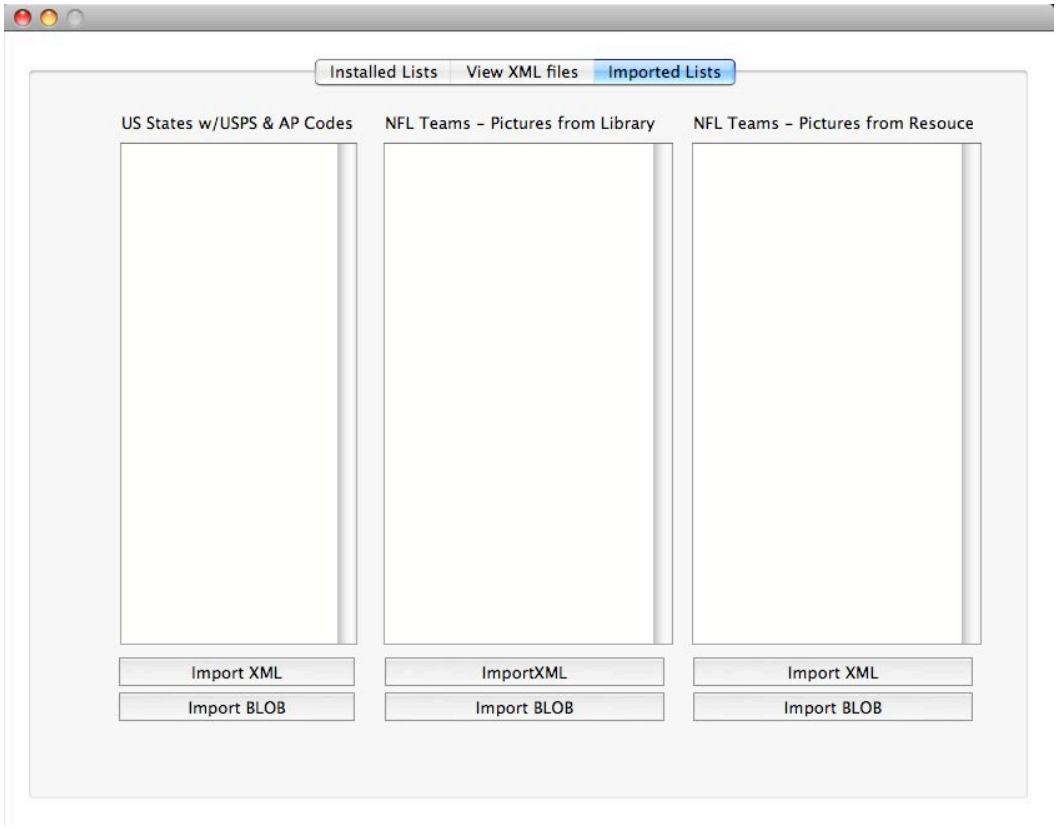
The demo that's included with this database displays the three lists installed in the database on page one. You have the ability to export the lists to XML files, BLOB files, and to load a view of the exported XML files.



When selecting to view an exported XML file, the file is loaded and the page is automatically switched to page two as shown below. On this page the XML that was save to disk is displayed. Here you can inspect how the list data is saved element attributes instead of element values.



On the Imported Lists page, the exported lists can be imported and viewed in the lists objects on this page. In the demo database, the imported lists are only presented in the form and are not saved back to the database. The methods **DEMO_HL_Import** and **DEMO_HL_Import_BLOB** are modified versions of the earlier documented methods meant specifically to load the exported lists into the demo form objects.



Conclusion

This Technical Note described two techniques for exporting and importing 4D hierarchical lists in 4D v11 SQL. It also includes methods for importing and exporting the 4D Picture Library. Included is a sample database containing the code needed to implement these new commands and functions in an existing or new databases.