Column Name: Advanced PM Programming
Column Title: Introducing Containers

by: Guy Scharf
(c) Copyright 1992 Software Architects, Inc.

## Introducing Containers

OS/2 2.0 introduces four new controls—slider, value set, notebook, and container.  This month we will take a first look at containers.

First, what *is* a container?  You have seen containers—lots of them. The container control is one of the key components of the Workplace Shell.  A container displays to the user the items, or objects, in contains in a visual way.  Virtually everything you see on the OS/2 desktop before starting a program or opening a settings view is a container.

If you open the drives folder, select a drive, and open it, you will see a container showing directories on the drive, arranged hierarchially.  This representation is called a tree view.  Select Open from the system menu again, press the right arrow button, and you can open Settings, Icon view, Tree view, and Details view.  These three views are all different ways a container can display the directories and files on a drive.

The container control supports five views of its contents:

Icon View  Icons or bitmaps with strings beneath.  The Workplace Shell desktop normally displays objects this way.

Name View  Icons or bitmaps with text to the right.

Text View  Text strings without any pictorial representation.

Tree View  Icons or bitmaps with text to the right and with a way to display a hierarchy (often indicated by + and - signs).

Details View  Icons or bitmaps, text strings, numbers, times, and/or dates.

Container windows support *direct manipulation*—the user can select items with the mouse and drag them around.  The user can drag items and drop them onto other containers, applications, or windows.

It is the container control that gives OS/2 2.0 and the Workplace Shell much of its flavor.

A container may change the way in which its contents are displayed by changing the view.  In the Workplace Shell, different views are typically seen by opening windows with that alternate view.  It is possible to change the view within a single window.  Only one view can be displayed in one container window at a time.

## A Step at a Time

As you might imagine, a control that performs all of these functions and many more and shows data in all of these views is complex.  The documentation is daunting.  The CUA Library/2 manual has 34 pages describing the container control and how it works and 74 pages of programming reference material.  The documentation for the container control is larger than that for any of the other new

controls.

Here, we will cut the problem down to a simpler case—we will use the details view of a container to build a multi-column listbox with scrolling, container and column titles, and other bells and whistles. Programmers frequently need multi-column listboxes for applications. While you can use the owner-draw feature of the listbox control to do that, the container with its details view is easier and more powerful.

Figure 1 shows a dialog with a container details view that is displayed by the sample program. Look at some of its features that you could not easily achieve in a list box:

o The container has a two-line, centered heading that does not scroll. This heading is part of the container itself; it is not static text outside of the container. A horizontal line is placed beneath the heading.

o The columns have headings too. Some of the headings are left justified, some are right justified. These headings also have a horizontal separator line beneath them.

o The container has a vertical scroll bar. The scroll bar does not extend to the headings, indicating that the headings stay fixed as you scroll the container vertically.

o The container has a vertical double line between the first and second columns. Grab the line with the mouse and you can drag it left and right. This is called a *split bar*. The split bar separates the container into two pieces, each of which is individually scrolled horizontally. You can adjust the split of the window into two pieces by dragging the split bar.

If you scroll the container horizontally, the column headings scroll too.

o The data in the columns is formatted. Text strings are left-justified; numeric values and dates have correct punctuation as specified by the system settings and are right-justified.

This looks complex, but is all pretty easy to do. Let's first look at the basics of programming a container.

## Container Basics

The basic steps for programming a container are the same as for other controls you are familiar with. You must create the container, load it with data, possibly monitor or modify its operation, possibly retrieve data from the container, and shut the container down when you are done with it. However, the container control requires more complex setup than older controls.

## Creating a Container

As with other control, you create a container by placing a CONTROL statement in a resource script. WC_CONTAINER is the window class name (CCL_CONTAINER if you are using CUA Library/2). You can also create a container programmatically using WinCreateWindow.

In the sample program, the container is in a dialog box. In the Workplace Shell drive folders, the containers occupy the entire window. That difference is a design choice. For the sample program, I wanted the appearance of a list box in a dialog.

The container control supports a small set of style flags. We look here only at those that apply to the details view.

CCS_READONLY makes the entire container read-only.  If you don't make the container read-only, the user can modify the container title and container records.  If you want to modify some data, but not all, you would omit this style and later mark container titles, records, column titles, or column contents that are not to be changed as read-only.

CCS_SINGLESEL specifies that only one container item can be selected at a time.  CCS_MULTIPLESEL allows the user to select zero or more items.  Both of these are similar to listbox styles.  The container control also has the CCS_EXTENDSEL style, which allows the user to select *one* or more container items.

CCS_MINIRECORDCORE specifies that a smaller record structure is to be used for this container. *Records* are a key concept of containers.  We examine them in the next section.

Notice that the type of view was not specified as a style, and other information is not yet known either.  You must initialize the container programmatically using the CM_SETCNRINFO message. We'll look at that later.

**Understanding Container Items and Records**

A container item is anything the user or programmer might store in a container.  The container is really a very general-purpose object, and can hold many different types of items.  Car, truck, bicycle, and unicycle objects could all be stored in the same container.

Programmatically, the data for each item in the container is stored in a *record*.  The container control manages the records.  The programmer asks the control to allocate memory for the records, fills in the fields of the record, and tells the container to insert the records in the container.  When the container window is closed, the container frees the memory for the records.

All records of a container must have the same structure.  There are two issues in determining the record format:

o  Whether the full-sized record format (RECORDCORE) or a small record format (MINIRECORDCORE) should be used.

o  How much additional space should be included in each record for programmer-maintained data.

The basic record structure (RECORDCORE or MINIRECORDCORE) has control information used by the container to maintain the records.

The MINIRECORDCORE structure contains only the minimum information necessary for a container item: a pointer to a text string, a handle to an icon, a chain pointer to the next record, the position of the icon in an icon view, and some attribute flags.

The larger RECORDCORE structure contains the above information plus more.  RECORDCORE supports different text strings for text, name, and tree views.  RECORDCORE is the default data structure.  If you wish to use MINIRECORDCORE, you must specify the CCS_MINIRECORDCORE style when creating the container.

The information in these data structures is sufficient to control the basic operation of a container, but is usually not sufficient for an application.  Also, many of the items in the structure are pointers to strings.  Where are the strings stored?  If the strings are in memory the programmer has allocated, then that memory must remain allocated for the life of the record.  That may not be

convenient.

The container control allows you to append any information to the basic record structure. When you ask the container to allocate memory, you tell it how many *extra bytes* you want allocated in each record. The usual way to program this is to declare a C struct. The first element of the struct will be RECORDCORE or MINIRECORDCORE. The following elements are your data fields. For example:

```
typedef struct
{
        MINIRECORDCORE    RecordCore;
        ULONG         ulMyNumber;
        CHAR                  szMyString[10];
        CDATE         cdate;
} USERRECORD, *PUSERRECORD;
```

You ask the container to allocate memory for one or more records by sending a CM_ALLOCRECORD message. To be more efficient, allocate many records at the same time.

Now, complete the record contents. Assign values to elements in both the base record structure and the extra bytes that follow the base structure.

After completing the records, you insert them into the container by completing a RECORDINSERT structure and sending the address of it and the first record allocated to the container with a CM_INSERTRECORD message.

When the container terminates, it will free the memory it allocated for you. However, if you used malloc() to obtain any additional space while filling in the records, the container cannot release that space, since it does not know about it. Instead, you will need to examine the records, perhaps while processing the WM_DESTROY message to the dialog, and free any additional space you allocated.

You may remove or insert additional records in the container at any time.

**Container Columns**

The details view shows several columns of information. As you might expect, these columns require additional control information so that the container can manage the columns.

In a way similar to managing records, you ask the container to allocate column control structures with the CM_ALLOCDETAILFIELDINFO message. You then complete the FIELDINFO data structures with information about each column. Finally, you construct a FIELDINFOINSERT structure and insert the column information with a CM_INSERTDETAILFIELDINFO message.

As with records, you can remove or insert additional columns at will.

Unlike container records, the container does not allow you to add extra bytes to the column control data. Instead, the FIELDINFO structure contains a pUserData field that you can use to hold a pointer to your information.

**Example Program**

The example program constructs a details view container that lists the states of the Union, their

capital cities, their population according to the 1980 census, and the dates they were admitted. The container can be scrolled vertically and horizontally. A split bar allows the user to adjust the amount of space in which the state name is displayed.

**The Resource Script**

Let's look first at the CNRDTL.RC file. As usual, all strings that the user might see are placed in a STRINGTABLE. No user-visible text is allowed in the program code. All of the strings are container or column headings. The *\012* in the strings is a new-line character, forcing that heading to be displayed in two lines.

The dialog template contains a CONTROL statement for a window of WC_CONTAINER class. The user may select only one item at a time (CCS_SINGLESEL) and cannot modify any data (CCS_READONLY). The program uses the minimal record structure (CCS_MINIRECORDCORE).

A PRESPARAMS statement follows to set the font within the container to 8 point Helvetica. A *\0* is included at the end of the font name string. This is required to make PRESPARAMS work with OS/2.0. This extra null is not required with OS/2 1.3.

The container control itself has no margin surrounding it. This makes it especially easy to use when the entire client area of a window is occupied by a container. We used the GROUPBOX statement to place a margin around the container for appearance's sake. This GROUPBOX statement should follow the CONTROL statement for the container itself.

**Main Program**

First, we define sample state data. Each record has the state and capital names, population, and date admitted.

Next we define the container record structure, RECORD. The first element of this structure must be MINIRECORDCORE (or RECORDCORE if desired). We then add three fields that represent the columns to be displayed: a pointer to the capital name, population of the state, and the date admitted. We'll examine the data types and the following COLDESC data in the next module.

The main program simply runs the example dialog and terminates; it is of little interest itself.

Finally, we come to the dialog procedure, ExampleDlgProc(). The dialog procedure for this program is extremely simple. In WM_INITDLG, CnCreateDetailsView() defines the columns of the details view and InitContainer() loads the state data into the container. In WM_COMMAND, we dismiss the dialog when a button is pushed. In WM_DESTROY, CnDestroyDetailsView() frees memory allocated for the container.

**Creating the Details View**

CnCreateDetailsView() is a utility function in CNFUNC.C to initialize the details view of a container, based on control structures passed to it. This function requires the number of columns, a pointer to the column control structure, the column the split bar is to follow, the position of the split bar as a percentage of the container width, the STRINGTABLE id of the container heading, and a handle to the resource module containing the STRINGTABLE.

The column information is the COLDESC data in CNRDTL.C. The COLDESC data contains an offset to the data within the record, column attribute flags, a STRINGTABLE id for the column heading, an

optional width for the column, and an NULL pointer used within CnCreateDetailsView().

The first thing CnCreateDetailsView does is go through the table of column descriptions, load the column titles from the resource file, and place pointers to the title text in the pszTitle field of the structure.

Next we initialize the container itself. We clear the CNRINFO structure on the stack and set the structure size in cb. We set the container attributes field in cnrinfo.flWindowAttr to CV_DETAIL (details view), CA_DETAILSVIEWTITLES (we want column titles), CA_CONTAINERTITLE (we want a title area for the container itself), and CA_TITLESEPARATOR (we want a horizontal line under the container title).

Then we load the container title from the resource file and place a pointer to it in cnrinfo.pszCnrTitle. The title contains a newline character (*012*), which causes the title to appear as two lines.

The CNRINFO structure contains a great many fields. When we send the CM_SETCNRINFO structure to the container, we must tell it what parts of the CNRINFO structure should be examined. We tell it this with CMA_* flags OR'ed together in mp2. In this case, we say CMA_FLWINDOWATTR (set window attributes) and CMA_CNRTITLE (set the title text). Those are the only two fields in CNRINFO that we filled in. If we had completed other fields, we would have needed to specify other CMA_* values to ask the container to pay attention to those fields.

Now that we have established the container, we can define the columns. First, we use the CM_ALLOCDETAILFIELDINFO message to ask the container to allocate the column control structures for the number of columns we need. The container returns to us a pointer to the first FIELDINFO structure. The other structures have been allocated and chained to this structure. The structures have been initialized as zero except for the structure length and chain fields.

We now loop through the COLDESC array, filling in one FIELDINFO structure for each COLDESC entry.

We copy the field offset and the attributes from COLDESC. These attributes are key to the operation of the column. You must specify the type of field this is. Details view supports CFA_ULONG (offset is to a ULONG), CFA_DATE (offset is to a CDATE structure), CFA_TIME (offset is to a CTIME structure), CFA_BITMAPORICON (offset is to an HPOINTER), or CFA_STRING (offset is to a PSZ, not to a CHAR array as the name might suggest). We can specify the justification of the data (CFA_LEFT, CFA_RIGHT, CFA_CENTER), vertical positioning of data in the row (DT_VCENTER, DT_TOP, DT_BOTTOM), and visual attributes (CFA_HORZSEPARATOR for a separator below column headings, CFA_SEPARATOR for a vertical separator to right of column, CFA_INVISIBLE, CFA_FIREADONLY for a read-only field, CFA_OWNERDRAW).

We don't use any bitmaps or time values in this sample. The CDATE and CTIME structures are sets of three ULONG values. The container will format them according to the current system preference settings.

We copy the title attribute and mark the title read-only. We use CFA_LEFT and CFA_RIGHT on columns to left or right justify the column headings. Since the container has the CCS_READONLY style, CFA_FITITLEREADONLY does not add anything. We have no user data and set the column width to 0. With a 0 width, the container will determine the width of the columns based on the data to be displayed. We could have set a width of our own choice if we wished to.

Setting the split bar requires modifying the container setup information. It is not part of the FIELDINFO information as such. We need to set a pointer to the last FIELDINFO that is left of the splitbar, and specify the position of the splitbar. To determine the position, we get the dimensions

of the container window and calculate the percent of that width that was passed as a parameter. We set the calculated x position and the pointer to the last FIELDINFO in the CNRINFO structure. Finally, we send a CM_SETCNRINFO message to the container, specifying which fields we have set (CMA_XVERTSPLITBAR for the location, and CMA_PFIELDINFOLAST for the last column to the left of the split bar).

Having completed all FIELDINFO structures, we construct a FIELDINFOINSERT structure, tell it how many structures we want to insert, and insert the fields by sending CM_INSERTDETAILFIELDINFO.

**Loading the Container**

Now we are ready to load the container using InitContainer() in CNRDTL.C.  First, we ask the container to create the records.  The number of extra bytes is the difference between our RECORD size and the system's MINIRECORDCORESIZE.  The container allocates the requested records, initializing them to zero except for the size and chain fields.

All we need to do is to run the chain and our array of state data, copying information into the RECORD structure.  We used the pszIcon field, which is the name of the item, as a pointer to the state name.  We used the pszCapital name we created for the capital city.  We could have used our own pointers for both.  We could also have added CHAR fields for these names in the RECORD structure and copied the actual strings in.  However, if we had done that, we would still have had to have PSZ values, as the CFA_STRING must specify the offset of a PSZ, not the beginning of a string.

Next we create a RECORDINSERT structure, give it the number of records, tell it where to insert these records, and insert the records with a CM_INSERTRECORD message.

We're done.  The container is now up and running.  Since this is a read-only display container, we don't have to do anything until the user terminates the dialog.

**Shutting Down**

While initializing the container and details view, we allocated memory with malloc.  When the container closes, we need to free that memory to prevent a memory leak.  Upon receipt of WM_DESTROY, the dialog calls CnDestroyDetailsView().  Here we free the strings loaded from the resource file.  We read the CNRINFO data and free the container title text that we allocated.  Next we free the column data and all records.

In this example, freeing the column data and records is not required, as the container will free that memory.  However, if we had been doing something else, such as switching from a details view to a tree view, we might have wanted to free the column data before the container terminated.

**Summary**

There you have it—a multi-column listbox with titles, column headings, vertical splits, and nice formatting.  Later, we can add many more features—direct manipulation (drag/drop), direct editing of column data, and more.

The programs in this article are available on the OS2DEV forum on CompuServe.  GO OS2DEV and download file CNRDTL.ZIP from library 4, Ver 2.0.

---

Guy Scharf is president of Software Architects, Inc., 2163 Jardin Drive, Mountain View, CA 94040. Software Architects, Inc. specializes in OS/2 Presentation Manager software development and consulting.  Guy can be reached on the CompuServe OS2DEV forum or on CompuServe Mail at 76702,557 or through Internet at 76702.557@compuserve.com.

**LISTING 1.  CNRDTL.C**

```
// cnrdtl.c -- Sample program to demonstrate container detail

//--------------------------------------------------------------
//  cnrdtl.c
//
//      Sample program to demonstrate container detail view
//
//  By: Guy Scharf                    (415) 948-9186
//      Software Architects, Inc.       FAX: (415) 948-1620
//      2163 Jardin Drive
//      Mountain View, CA   94040
//      CompuServe: 76702,557
//      Internet: 76702.557@compuserve.com
//  (c) Copyright 1992 Software Architects, Inc.
//
//      All Rights Reserved.
//
//  Software Architects, Inc. develops software products for
//  independent software vendors using OS/2 and Presentation
//  Manager.
//--------------------------------------------------------------

#define INCL_PM                       // Basic OS/2 PM
#define INCL_BASE                       // components
#include <OS2.H>

#include <stdlib.h>                   // C runtime library
#include <stddef.h>
#include <string.h>

#include "cnrdtl.h"                  // Container demo defs
#include "cnfunc.h"                   // Container utilities


//  Sample data

typedef struct
{
  SHORT   mm, dd, yy;                 // General date storage
} OURDATE, *POURDATE;

typedef struct
{
  PSZ     pszState;                 // Name of state
  PSZ     pszCapital;                // Name of city
  ULONG   ulPopulation;                // '80 census population
  OURDATE ourdateAdm;                   // Date admitted
}  STATEREC, *PSTATEREC;

STATEREC statedata[] =
{
  {"Alabama",     "Montgomery",   3893888, {12, 14, 1819}},
```

```
  {"Alaska",      "Juneau",       401851, { 1,  3, 1959}},
  {"Arizona",     "Phoenix",      2718425, { 2,  4, 1912}},
  {"Arkansas",    "Little Rock",  2286435, { 6, 15, 1836}},
  {"California",  "Sacramento",  23667565, { 9,  9, 1850}},
  {"Colorado",    "Denver",       2889735, { 8,  1, 1876}},
  {"Connecticut", "Hartford",     3107576, { 1,  9, 1788}},
  {"Delaware",    "Dover",         594317, {12,  7, 1787}},
  {"Florida",     "Tallahassee",  9746342, { 3,  3, 1845}},
  {"Georgia",     "Atlanta",      5463105, { 1,  2, 1788}},
  {"Hawaii",      "Honolulu",      964691, { 8, 21, 1959}}
};

typedef struct                      // Container data record
{
  MINIRECORDCORE  RecordCore;       // MINIRECORDCORE structure
  PSZ         pszCapital;           // Capital city
  ULONG       ulPopulation;         // Population
  CDATE       cdateAdmitted;        // Date admitted
} RECORD, *PRECORD;

COLDESC cdState[] =
{
 {offsetof(RECORD, RecordCore.pszIcon), CFA_STRING | CFA_FIREADONLY |
    CFA_HORZSEPARATOR | CFA_LEFT, IDS_HEAD_STATE,
    CFA_LEFT, 0, NULL},
 {offsetof(RECORD, pszCapital), CFA_STRING | CFA_FIREADONLY |
    CFA_HORZSEPARATOR | CFA_LEFT, IDS_HEAD_CAP,
    CFA_LEFT, 0, NULL},
 {offsetof(RECORD, ulPopulation), CFA_ULONG | CFA_FIREADONLY |
    CFA_HORZSEPARATOR | CFA_RIGHT, IDS_HEAD_POP,
    CFA_RIGHT, 0, NULL},
 {offsetof(RECORD, cdateAdmitted), CFA_DATE | CFA_FIREADONLY |
    CFA_HORZSEPARATOR | CFA_RIGHT, IDS_HEAD_ADM,
    CFA_RIGHT, 0, NULL}
};

//  Prototypes of procedures

static MRESULT EXPENTRY ExampleDlgProc (HWND, MSGID, MPARAM,
                                MPARAM);

static MRESULT InitContainer (          // Initialize slider
HWND    hwndDlg,                    // I - Dialog window
USHORT  idContainer,                // I - Container id
USHORT  cStates,                    // I - Number of states
PSTATEREC pastate);                 // I - State data array

//---------------------------------------------------------------
//
//  Main program to drive container example
//
//---------------------------------------------------------------

int main (void)
{
```

```
  HAB     hab;                    // Handle to anchor blk
  HMQ     hmqMsgQueue;            // Handle to msg queue
#ifndef OS220
  HMODULE hmodContainer;          // Handle to cntr module
#endif

  hab = WinInitialize (0);        // Initialize PM

  hmqMsgQueue = WinCreateMsgQueue (hab, 0);// Create msg queue

#ifndef OS220
  if (DosLoadModule (NULL, 0, CCL_CONTAINER_DLL,
                  &hmodContainer))
     return FALSE;
#endif

  WinDlgBox (HWND_DESKTOP, HWND_DESKTOP, ExampleDlgProc, 0,
           IDLG_EXAMPLE, NULL);

#ifndef OS220
  DosFreeModule (hmodContainer);
#endif

  WinDestroyMsgQueue (hmqMsgQueue);   // Shutdown
  WinTerminate       (hab);
  return 0;
}

//--------------------------------------------------------------
//
//  ExampleDlgProc() -- Show state info
//
//--------------------------------------------------------------

static MRESULT EXPENTRY ExampleDlgProc (
HWND    hwndDlg,
MSGID   msg,
MPARAM  mp1,
MPARAM  mp2)
{
  switch(msg)
  {
     //-------------------------------------------------------
     //  Initialize dialog by defining the details view
     //  in the container, loading records, etc.
     //-------------------------------------------------------
     case WM_INITDLG:

        CnCreateDetailsView (WinWindowFromID (hwndDlg,
                                    IDCN_STATEINFO),
                     sizeof cdState / sizeof (COLDESC),
                     cdState,
                     0,
                     33,
                     IDS_TITLE,
```

```
                    0);

        //--------------------------------------------------
        //  Load the container
        //--------------------------------------------------
        InitContainer (hwndDlg, IDCN_STATEINFO,
                    sizeof statedata / sizeof (STATEREC),
                    statedata);
        return 0;

      //--------------------------------------------------------
      //  Process pushbuttons.  They both just quit dialog.
      //--------------------------------------------------------
      case WM_COMMAND:
         switch (SHORT1FROMMP(mp1))
         {
                              // Cancel pressed
                              // Dismiss dialog
            case DID_CANCEL:
               WinDismissDlg (hwndDlg, FALSE);
               return 0;

                              // OK button pressed
            case DID_OK:            // We're done
               WinDismissDlg (hwndDlg, TRUE);
               return 0;
         }
         return 0;

      //--------------------------------------------------------
      //  Recover memory allocated for the container
      //--------------------------------------------------------
      case WM_DESTROY:
         CnDestroyDetailsView (WinWindowFromID (hwndDlg,
                                    IDCN_STATEINFO),
                     sizeof cdState / sizeof (COLDESC),
                     cdState);
         return 0;

      //--------------------------------------------------------
      //  All other messages go to default window procedure
      //--------------------------------------------------------
      default:
         return (WinDefDlgProc (hwndDlg, msg, mp1, mp2));
   }
   return FALSE;
}



//------------------------------------------------------------
// Function: InitContainer
// Outputs:  none
//
// This function loads the container with all of the state data.
```

```
// See the article for a complete discussion of this function.
//------------------------------------------------------------

static MRESULT InitContainer (          // Initialize slider
HWND    hwndDlg,                        // I - Dialog window
USHORT  idContainer,                    // I - Container id
USHORT  cStates,                        // I - Number of states
PSTATEREC pastate)                      // I - State data array
{
  USHORT      i;                    // Loop counter
  ULONG       cbExtraBytes;         // Extra bytes in record structure */

  PRECORD     precord;              // -> container records
  PRECORD     precordFirst;         // -> first container record

  RECORDINSERT recordInsert;        // Record insertion control

  //------------------------------------------------------------
  //  Allocate memory for all user records
  //------------------------------------------------------------

  cbExtraBytes = (ULONG)(sizeof(RECORD) -
                                sizeof(MINIRECORDCORE));
  precord = (PRECORD) WinSendDlgItemMsg (hwndDlg, idContainer,
                           CM_ALLOCRECORD,
                           MPFROMLONG (cbExtraBytes),
                           MPFROMSHORT (cStates));
  precordFirst = precord;

  //------------------------------------------------------------
  //  Initialize all records
  //------------------------------------------------------------

  for (i = 0; i < cStates; i++, pastate++)
  {
     //---------------------------------------------------
     //   Initialize the container record control structure
     //---------------------------------------------------
     precord->RecordCore.cb        = sizeof (MINIRECORDCORE);

     //---------------------------------------------------
     //  Copy our data to the container record control struct
     //---------------------------------------------------
     precord->RecordCore.pszIcon  = pastate->pszState;
     precord->pszCapital          = pastate->pszCapital;
     precord->ulPopulation        = pastate->ulPopulation;
     precord->cdateAdmitted.month = pastate->ourdateAdm.mm;
     precord->cdateAdmitted.day   = pastate->ourdateAdm.dd;
     precord->cdateAdmitted.year  = pastate->ourdateAdm.yy;

                                 // Move to next record
     precord = (PRECORD) precord->RecordCore.preccNextRecord;
  }

  //------------------------------------------------------------
```

```
  //  Construct record insertion control structure
  //-------------------------------------------------------------

  memset (&recordInsert, 0, sizeof (RECORDINSERT));
  recordInsert.cb                 = sizeof (RECORDINSERT);
  recordInsert.pRecordParent      = NULL;
  recordInsert.pRecordOrder       = (PRECORDCORE)((ULONG)
                                  MPFROMSHORT(CMA_END));
  recordInsert.zOrder             = (USHORT) CMA_TOP;
  recordInsert.cRecordsInsert     = cStates;
  recordInsert.fInvalidateRecord  = TRUE;

  //-------------------------------------------------------------
  //  Insert the records
  //-------------------------------------------------------------

  WinSendDlgItemMsg (hwndDlg, idContainer,
                  CM_INSERTRECORD,
                  MPFROMP (precordFirst),
                  MPFROMP (&recordInsert));

  //-------------------------------------------------------------
  //  Return to caller
  //-------------------------------------------------------------

  return 0;
}
```

**LISTING 2.  CNFUNC.C**
==========

```
// cnfunc.c -- Container utility functions

//------------------------------------------------------------------
//
//  cnfunc.c
//
//      Container utility functions
//
//------------------------------------------------------------------

#define INCL_WIN
#include <os2.h>

#include <stdlib.h>
#include <string.h>

#include "cnfunc.h"


#define MAXMSGSIZE  255                 // Max size of string


//------------------------------------------------------------------
//  UtilLoadString - load a string from the resource file
//------------------------------------------------------------------

PSZ UtilLoadString (            // Load string from res
USHORT  usStringID,               // I - String identifier
HAB     hab,                    // I - Current HAB
HMODULE hmod)                     // I - Resource module or NULL
{
  PSZ     pszBuf;             // String buffer

  pszBuf = malloc (MAXMSGSIZE);
  if (WinLoadString (hab, hmod, usStringID, MAXMSGSIZE,
                  pszBuf) == 0)
  {
     free (pszBuf);
     return (NULL);
  }

  return (pszBuf);            // Return -> string
}


//------------------------------------------------------------------
//
//  CnCreateDetailsView
//
//      Create the details view of the container.  This requires
//      setting up the column information.
```

```
//
//------------------------------------------------------------------

USHORT CnCreateDetailsView (            // Create details view
HWND    hwndContainer,                  // I - container window
USHORT  cColumns,                       // I - Number of colums
COLDESC acd[],                          // IO->column descriptor
SHORT   sLastLeftColumn,                // I - Last column in
                                        // left split window
                                        //     -1 means no split
LONG    lPctSplitBarPos,                // Percent of cntr width
                                        // to set split bar
USHORT  idTitle,                        // I - container hdg id
HMODULE hmod)                           // I -hmod for resources
{
  USHORT      i;                    // Loop counter

  CNRINFO     cnrinfo;              // Cntr info structure
  PFIELDINFO  pFieldInfoHead;       // --> First field
  PFIELDINFO  pFieldInfo;           // --> Current field
  FIELDINFOINSERT FieldInsertInfo;

  PCOLDESC    pcd;                  // Column descriptor

  //-------------------------------------------------------------
  //  Get container text information
  //-------------------------------------------------------------

  for (pcd = acd, i = 0; i < cColumns; i++, pcd++)
  {
     if (pcd->pszTitle == NULL)
        pcd->pszTitle = UtilLoadString (pcd->idTitle,
                    WinQueryAnchorBlock (hwndContainer),
                                hmod);
  }


  //-------------------------------------------------------------
  //  Initialize the container for a simple details view
  //-------------------------------------------------------------

  memset (&cnrinfo, 0, sizeof(CNRINFO));
  cnrinfo.cb = sizeof (CNRINFO);

  // Set for Details View, with column titles, with icons and
  // not bitmaps, include a container title, with a separator
  // bar underneath it, and don't let the user change title
  cnrinfo.flWindowAttr = CV_DETAIL            |
                     CA_DETAILSVIEWTITLES |
                     CA_CONTAINERTITLE    |
                     CA_TITLESEPARATOR;

  cnrinfo.pszCnrTitle = UtilLoadString (idTitle,
                      WinQueryAnchorBlock (hwndContainer),
                      hmod);
```

```
WinSendMsg (hwndContainer, CM_SETCNRINFO,
          MPFROMP (&cnrinfo),
          MPFROMLONG (CMA_FLWINDOWATTR | CMA_CNRTITLE));


//------------------------------------------------------------
//  Get memory for the column information
//------------------------------------------------------------

pFieldInfoHead = (PFIELDINFO) PVOIDFROMMR (
                    WinSendMsg (hwndContainer,
                              CM_ALLOCDETAILFIELDINFO,
                              MPFROMSHORT (cColumns),
                              0));

pFieldInfo = pFieldInfoHead;        // Start at top of list


//------------------------------------------------------------
//  Load the field info structures
//------------------------------------------------------------

for (pcd = acd, i = 0; i < cColumns; i++, pcd++)
{
   pFieldInfo->cb          = sizeof (FIELDINFO);
   pFieldInfo->flData      = pcd->flAttributes;
   pFieldInfo->flTitle     = pcd->flTitle;
   pFieldInfo->pTitleData  = pcd->pszTitle;
   pFieldInfo->offStruct   = pcd->offField;
   pFieldInfo->pUserData   = NULL;
   pFieldInfo->cxWidth     = pcd->cxWidth;

   if (sLastLeftColumn >= 0)
   {
      if (sLastLeftColumn == i)
      {
         SWP     swp;
         WinQueryWindowPos (hwndContainer, &swp);
         cnrinfo.pFieldInfoLast = pFieldInfo;
         cnrinfo.xVertSplitbar  =
                     (swp.cx * lPctSplitBarPos) / 100;
         WinSendMsg (hwndContainer, CM_SETCNRINFO,
                   MPFROMP (&cnrinfo),
                   MPFROMLONG (CMA_PFIELDINFOLAST |
                             CMA_XVERTSPLITBAR));
      }
   }
   pFieldInfo = pFieldInfo->pNextFieldInfo;
};


//------------------------------------------------------------
//  Construct the FIELDINFOINSERT structure that describes
//  the number of fields to be inserted, where they are to
```

```
  //  be inserted, etc.
  //-------------------------------------------------------------

  memset (&FieldInsertInfo, 0, sizeof(FIELDINFOINSERT));
  FieldInsertInfo.cb                 = sizeof (FIELDINFOINSERT);
  FieldInsertInfo.pFieldInfoOrder  = (PFIELDINFO) CMA_END;
  FieldInsertInfo.cFieldInfoInsert = cColumns;
  FieldInsertInfo.fInvalidateFieldInfo = TRUE;


  //-------------------------------------------------------------
  //  Insert the fields.
  //-------------------------------------------------------------

  WinSendMsg (hwndContainer, CM_INSERTDETAILFIELDINFO,
            MPFROMP (pFieldInfoHead),
            MPFROMP (&FieldInsertInfo));
  return 0;

}



//-------------------------------------------------------------------
//
//  CnDestroyDetailsView
//
//      Destroy the details view of the container.  This
//      requires freeing column information.
//
//-------------------------------------------------------------------

USHORT CnDestroyDetailsView (   // Destroy details view
HWND    hwndContainer,          // I - Handle to cntr window
USHORT  cColumns,               // I - Number of colums
COLDESC acd[])                  // IO--> column descriptor
{
  USHORT      i;              // Loop counter
  PCOLDESC    pcd;            // Column descriptor entry
  CNRINFO     cnrinfo;        // Container info structure


  //-------------------------------------------------------------
  //  Remove any column titles loaded into memory
  //-------------------------------------------------------------

  for (pcd = acd, i = 0; i < cColumns; i++, pcd++)
  {
     if (    (pcd->pszTitle != NULL)
          && (pcd->idTitle != 0) )
     {
        free (pcd->pszTitle);
        pcd->pszTitle = NULL;
     }
  }
```

```
  //-----------------------------------------------------------
  //  Remove the column heading from memory
  //-----------------------------------------------------------

  WinSendMsg (hwndContainer,
              CM_QUERYCNRINFO,
              MPFROMP(&cnrinfo),
              MPFROMSHORT (sizeof(CNRINFO)));

  free (cnrinfo.pszCnrTitle);


  //-----------------------------------------------------------
  //  Remove the column information of the container
  //-----------------------------------------------------------

  WinSendMsg (hwndContainer,
              CM_REMOVEDETAILFIELDINFO,
              0,
              MPFROM2SHORT (0, CMA_FREE));


  //-----------------------------------------------------------
  //  Remove any records in the container
  //-----------------------------------------------------------

  WinSendMsg (hwndContainer,
              CM_REMOVERECORD,
              0,
              MPFROM2SHORT (0, CMA_FREE));

  return 0;
}
```

**LISTING 3.  CNRDTL.H**
==========

```
// cnrdtl.h -- Definitions for cnrdtl demo

//---------------------------------------------------------------
//  Change the following as required for target system
//---------------------------------------------------------------
#define OS220                       // Define target system
//#define OS213                     // OS/2 1.3 + CUA Lib/2

#ifdef OS220
  #define MSGID   ULONG             // OS/2 2.0
#else
  #define MSGID   USHORT            // OS/2 1.3
  #include <fclcnrp.h>              // CUA Library/2
  #define WC_CONTAINER CCL_CONTAINER  // Window class
#endif

//  Defines for dialogs, controls
#define IDLG_EXAMPLE        100
#define IDCN_STATEINFO      200
#define IDS_HEAD_STATE      500
#define IDS_HEAD_CAP        501
#define IDS_HEAD_POP        502
#define IDS_HEAD_ADM        503
#define IDS_TITLE           504
```

**LISTING 4.  CNFUNC.H**
==========

```
// cnfunc.h -- Container utility functions

//---------------------------------------------------------------
//
//  cnfunc.h
//
//      Container Functions
//
//---------------------------------------------------------------

//---------------------------------------------------------------
//  Data structure used to describe field information
//---------------------------------------------------------------

typedef struct                  // Field (column) descriptors
{
  ULONG   offField;             // Offset of field in record
  ULONG   flAttributes;         // Field attributes
  USHORT  idTitle;              // Identifier of column title
  ULONG   flTitle;              // Title Attributes
```

```
  USHORT  cxWidth;           // Column width (0 = auto calc)
  PSZ     pszTitle;          // Pointer to column title text
} COLDESC, *PCOLDESC;


//------------------------------------------------------------
//  Function prototypes
//------------------------------------------------------------

USHORT CnCreateDetailsView (    // Create details view
HWND    hwndContainer,          // I - Handle to container window
USHORT  cColumns,               // I - Number of colums
COLDESC acd[],                  // IO--> column descriptor
SHORT   sLastLeftColumn,        // I - Last column in left split
LONG    lPctSplitBarPos,        // Percent of container width
USHORT  idTitle,                // I - container heading id
HMODULE hmod);                  // I - handle to resource file

USHORT CnDestroyDetailsView (   // Destroy details view
HWND    hwndContainer,          // I - Handle to container window
USHORT  cColumns,               // I - Number of colums
COLDESC acd[]);                 // IO--> column descriptor
```

**LISTING 5.  CNRDTL.RC**
==========

```
#include <os2.h>                    // OS/2 definitions
#include "cnrdtl.h"                 // Application defs

STRINGTABLE
BEGIN
 IDS_HEAD_STATE, "\012State"
 IDS_HEAD_CAP,   "\012Capital"
 IDS_HEAD_POP,   "Population\012(1980 Census)"
 IDS_HEAD_ADM,   "Entered\012Union"
 IDS_TITLE,      "State Information Table\012(1987 Almanac)"
END

DLGTEMPLATE IDLG_EXAMPLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
 DIALOG  "State Information", IDLG_EXAMPLE, 40, 27, 200, 150,
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
 BEGIN
   CONTROL    "",   IDCN_STATEINFO,
                20,  31, 160, 110, WC_CONTAINER,
                CCS_SINGLESEL | CCS_READONLY |
                CCS_MINIRECORDCORE |
                WS_GROUP | WS_TABSTOP | WS_VISIBLE
            PRESPARAMS PP_FONTNAMESIZE,"8.Helv\0"

   GROUPBOX   "", 300, 19, 30, 162, 116

DEFPUSHBUTTON "OK",     DID_OK,     15, 9, 48, 13, WS_GROUP
```

```
   PUSHBUTTON "Cancel", DID_CANCEL, 78, 9, 48, 13,
            NOT WS_TABSTOP
 END
END
```

**LISTING 6.  CNRDTL.MAK**
==========

```
# Make File Creation run in directory:
#   D:\P\MAGAZINE\CNRDTL;

.SUFFIXES:

.SUFFIXES: .c .rc

ALL: CNRDTL.EXE \
    CNRDTL.RES

cnrdtl.exe:  \
 CNFUNC.OBJ \
 CNRDTL.OBJ \
 CNRDTL.RES \
 CNRDTL.MAK
  @REM @<<CNRDTL.@0
    /CO /M /NOL /PM:PM +
    CNFUNC.OBJ +
    CNRDTL.OBJ
    cnrdtl.exe


    ;
<<
  LINK386.EXE @CNRDTL.@0
  RC CNRDTL.RES cnrdtl.exe

{.}.rc.res:
  RC -r .\$*.RC

{.}.c.obj:
  ICC.EXE /Ss /Kbger /Ti /W2 /C .\$*.c

!include CNRDTL.DEP
```

**LISTING 7.  CNRDTL.DEP**
==========

```
# Make File Creation run in directory:
#   D:\P\MAGAZINE\CNRDTL;

# Assumed INCLUDE environment variable path:
```

```
#   C:\TOOLKT20\C\OS2H;
#   C:\TOOLKT20\ASM\OS2INC;
#   C:\IBMC\INCLUDE;
#   E:\GPF\INCLUDE;


CNRDTL.RES:  CNRDTL.RC  \
#  {.;$(INCLUDE)}OS2.H  \
  {.;$(INCLUDE)}CNRDTL.H  \
#    {.;$(INCLUDE)}FCLCNRP.H  \
  CNRDTL.MAK

CNFUNC.OBJ:  CNFUNC.C  \
#  {$(INCLUDE);}os2.h  \
#  {$(INCLUDE);}stdlib.h  \
#  {$(INCLUDE);}string.h  \
  {.;$(INCLUDE);}cnfunc.h  \
  CNRDTL.MAK

CNRDTL.OBJ:  CNRDTL.C  \
#  {$(INCLUDE);}os2.h  \
#  {$(INCLUDE);}stdlib.h  \
#  {$(INCLUDE);}stddef.h  \
#  {$(INCLUDE);}string.h  \
  {.;$(INCLUDE);}cnrdtl.h  \
#    {$(INCLUDE);}fclcnrp.h  \
  {.;$(INCLUDE);}cnfunc.h  \
  CNRDTL.MAK
```