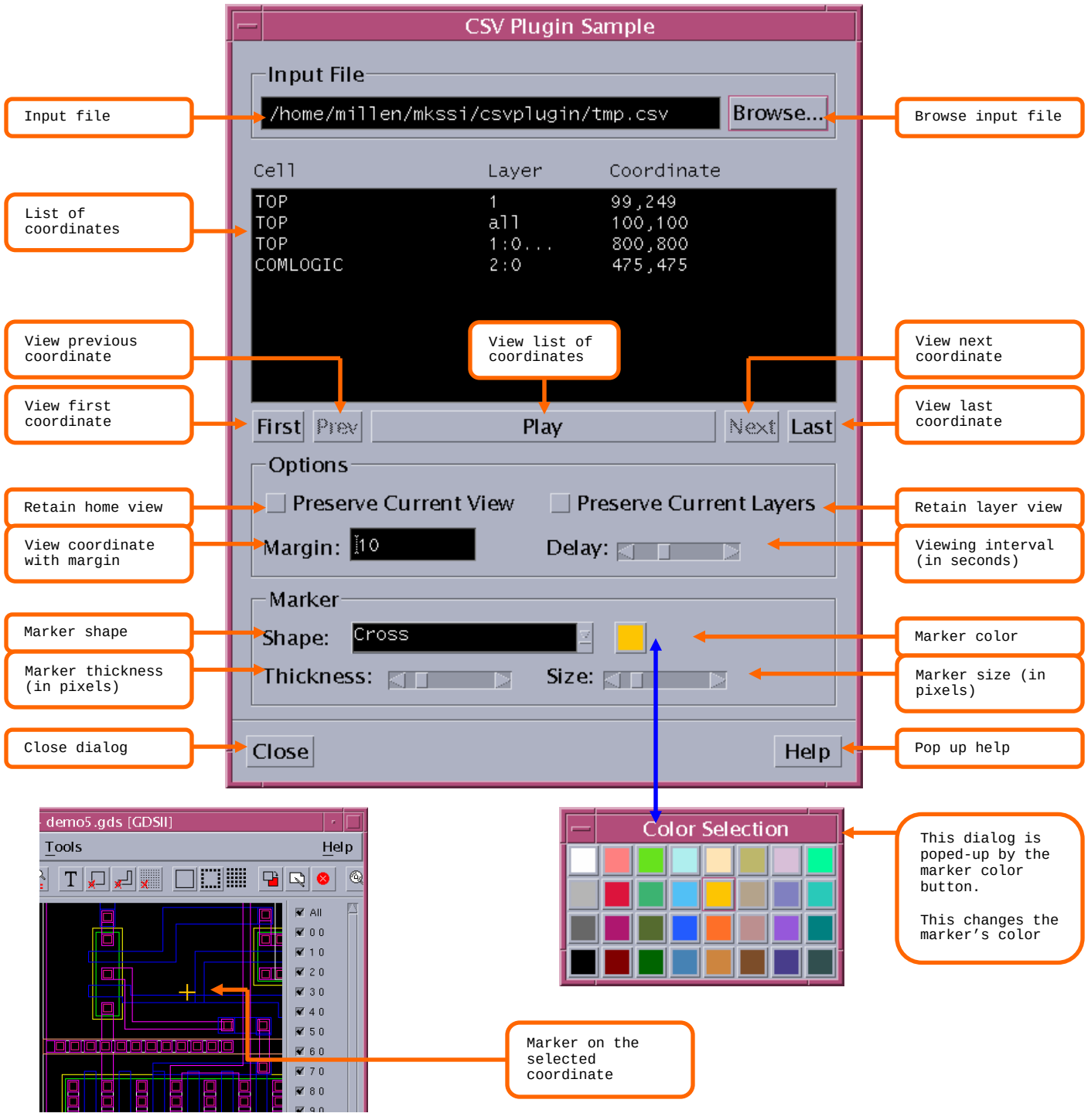
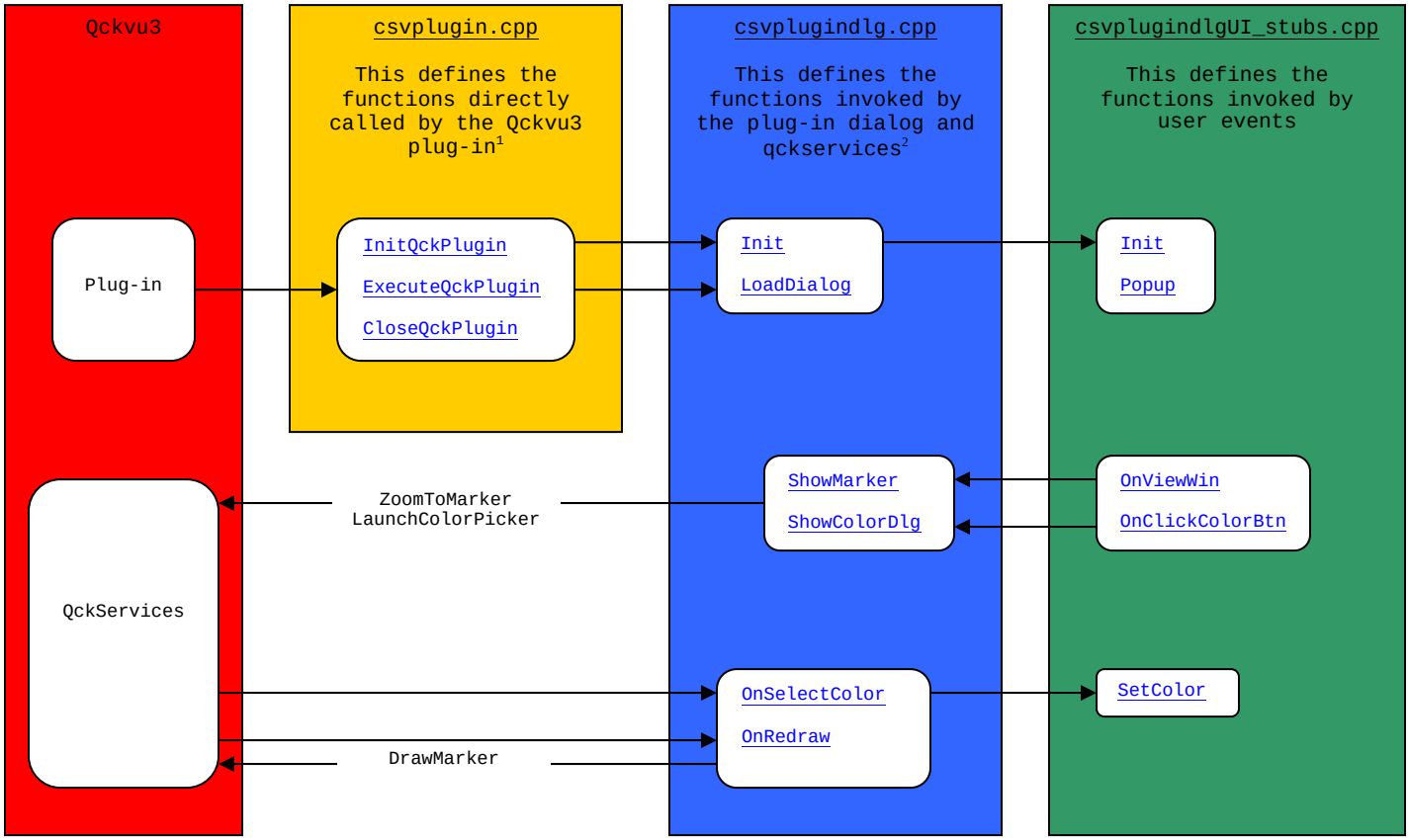
CSV PLUG-IN OVERVIEW:

This package illustrates how a plug-in can allow a user to navigate Qckvu3 via a comma separated value (CSV) file. This plug-in allows the selection of cell, layer and window views. It also illustrates the use of marker and color palettes. The plug-in could easily be modified so that markers (of color and type) could be associated with entries in the CSV file.

Data from the CSV file populates the plug-in's dialog. An entry can be selected from the list and the associated cell/layer/coordinate view is displayed in Qckvu3. The whole list can also be viewed with some interval by pressing the play button.



CSV PLUG-IN FLOW:



[1] Reference “How to write a QckPlugin” document (pg.04) updated on 08/28/08
[2] Reference “How to write a QckPlugin” document (pg.10) updated on 08/28/08

csvplugin.cpp

InitQckPlugin: ***This function is called at startup when Qckvu3 finds a QckPlugin and tries to load it into memory. It creates the plug-in's menu button and initializes the plug-in.***

```
int InitQckPlugin(const char *ProgFileName, void* PlgHandle, void* SrvcMgr)
{
  IQckMenuSrvc * MenuServ = ((IQckSrvcMgr *)SrvcMgr)->RegisterForMenuItemService();
  MenuServ->CreateMenuItem(PlgHandle, "CSV Plugin Sample");

  ICsvPluginDlg *CsvPluginDialog = ICsvPluginDlg::GetInstance();
  CsvPluginDialog->Init(ProgFileName, MenuServ->GetAppShell(), PlgHandle, (IQckSrvcMgr *)SrvcMgr);

  return 0;
}
```

ExecuteQckPlugin: ***This function will be called every time the user clicks on the menu item corresponding to the QckPlugin. It loads the plug-in dialog.***

```
void ExecuteQckPlugin(int argc,void *argv)
{
  ICsvPluginDlg *CsvPluginDialog = ICsvPluginDlg::GetInstance();
  CsvPluginDialog->LoadDialog();
}
```

CloseQckPlugin: ***This function is called after the user closes Qckvu3 when where the cleanup routines are executed before exiting.***

```
void CloseQckPlugin()
{
}
```

csvplugindlg.cpp

Constructors and Destructors: ***Only one instance of the plug-in exists. GetInstance is called to return a pointer of the plug-in instead of creating one via the constructor.***

```
//Get an instance of ICsvPluginDlg
ICsvPluginDlg* ICsvPluginDlg::GetInstance()
{
  return(CsvPluginDlg::GetInstance());
}
//Get the one and only instance of CsvPluginDlg
CsvPluginDlg* CsvPluginDlg::GetInstance()
{
  static CsvPluginDlg lTheOnlyInstance;
  return(&lTheOnlyInstance);
}

//Constructor
CsvPluginDlg::CsvPluginDlg()
{
  wCsvPluginDlgForm = NULL;

  mServMgr = NULL;
  mMServ = NULL;
  mWServ = NULL;
  mGServ = NULL;
  mDServ = NULL;
  mLServ = NULL;
  mCServ = NULL;

  mMarkerOn = false;
  mMarker.mXY[0] = 0.0;
  mMarker.mXY[1] = 0.0;
  mMarker.mShape = DEFSHAPE;
  mMarker.mSize = DEFSIZE;
  mMarker.mThickness = DEFWIDTH;
  mMarker.mRGB[0] = DEFCOLOR;
  mMarker.mRGB[1] = DEFCOLOR;
  mMarker.mRGB[2] = DEFCOLOR;
  mWinMargin = DEFMARGIN;
  mCell = NULL;
  mLayer = NULL;
}
//Destructor
CsvPluginDlg::~CsvPluginDlg()
{
  mMarkerOn = false;
  mMarker.mXY[0] = 0.0;
  mMarker.mXY[1] = 0.0;
  mMarker.mShape = DEFSHAPE;
  mMarker.mSize = DEFSIZE;
  mMarker.mThickness = DEFWIDTH;
  mMarker.mRGB[0] = DEFCOLOR;
  mMarker.mRGB[1] = DEFCOLOR;
  mMarker.mRGB[2] = DEFCOLOR;
  mWinMargin = DEFMARGIN;
  mCell = NULL;
  mLayer = NULL;

  mServMgr = NULL;
  mMServ = NULL;
  mWServ = NULL;
  mGServ = NULL;
  mDServ = NULL;
  mLServ = NULL;
  mCServ = NULL;

  if(wCsvPluginDlgForm)
    delete(wCsvPluginDlgForm);
  wCsvPluginDlgForm = NULL;
}
```

Init: ***This function registers other QckServices and enable notification services. It also creates and initializes the plug-in's dialog.***

```
//Initialize CsvPluginDlg (must be called once at the beginning)
void CsvPluginDlg::Init(const char * ProgPath, void* Parent, void *PlgHandle, IQckSrvcMgr *SrvcMgr)
{
  mServMgr = SrvcMgr;

  //Setup the plugin services
  if(mServMgr)
  {
    mMServ = mServMgr->RegisterForMarkerService();
    mWServ = mServMgr->RegisterForWindowService();
    mGServ = mServMgr->RegisterForGraphicsService();
    mDServ = mServMgr->RegisterForDatabaseService();
    mLServ = mServMgr->RegisterForLayerService();
    if(PlgHandle)
    {
      if(mWServ)
        mWServ->SetWindowServiceNotify(PlgHandle, this);
      if(mGServ)
        mGServ->SetDrawServiceNotify(PlgHandle, this);
      if(mDServ)
        mDServ->SetDatabaseServiceNotify(PlgHandle, this);
      if(mLServ)
        mLServ->SetLayerServiceNotify(PlgHandle, this);
    }
  }

  //Setup the dialog
  if(!wCsvPluginDlgForm && Parent)
  {
    create_wCsvPluginDlgShell((Widget) Parent);
    if(wCsvPluginDlgForm)
      wCsvPluginDlgForm->Init(Parent, ProgPath, this);
  }
}
```

LoadDialog: ***This functions loads the plug-in's dialog. Any extended services are also obtained here.***

```
//Load the dialog
void CsvPluginDlg::LoadDialog()
{
  //Get the color dialog plugin (extended). This cannot be called in INIT
  if(!mCServ && mServMgr)
    mCServ = (IQckColorSrvc*) mServMgr->GetExtendedService(_ACS_QCK_COLOR_SERVICE_NAME_);

  if(wCsvPluginDlgForm)
  {
    //Check if there is a cell and popup the dialog
    if(mDServ && mDServ->GetViewCell() != NULL)
      wCsvPluginDlgForm->Popup(true);
    else
      wCsvPluginDlgForm->Popup(false);
  }
}
```

Plug-in dialog's callback functions: ***These functions are invoked by the plug-in's dialog.***

```
//Check if specified cell exists
bool CsvPluginDlg::CellExist(const char* str){
  if(mDServ && mDServ->GetCellID(str) >= 0)
    return true;
  else
    return false;
}
```

```
//Hide the marker
void CsvPluginDlg::HideMarker()
{
  //Reset the marker/cell/layer
  mMarkerOn = false;
  mMarker.mXY[0] = 0.0;
  mMarker.mXY[1] = 0.0;
  mMarker.mShape = DEFSHAPE;
  mMarker.mSize = DEFSIZE;
  mMarker.mThickness = DEFWIDTH;
  mMarker.mRGB[0] = DEFCOLOR;
  mMarker.mRGB[1] = DEFCOLOR;
  mMarker.mRGB[2] = DEFCOLOR;
  mWinMargin = DEFMARGIN;

  mCell = NULL;
  mLayer = NULL;

  //Redraw
  if(mGServ)
    mGServ->Redraw(false);
}

//Show the marker
void CsvPluginDlg::ShowMarker(const char * cell, const char * layer, const double margin,
                              const bool nomargin, const bool layerson, const sQCKMARKER marker)
{
  //Set the marker/cell/layer
  mMarkerOn = true;
  mMarker = marker;
  mWinMargin = margin;

  mCell = (char*) cell;
  mLayer = (char*) layer;

  bool forcedraw = false;
  if(mDServ && mLServ && mWServ && mMServ){
    //If the current cell is different, set the correct cell

    sQCKWINDOW currwin;
    if(strcmp(mDServ->GetViewCell(), mCell) != 0){
      mDServ->SetViewCell(mCell, false);
      mDServ->GetCellExtents(mCell, currwin);
      forcedraw = true;
    }
    else
      mWServ->GetCurrentWindow(currwin);

    //If using specified layers, set the correct layers
    if(!layerson){
      if(strcasecmp(layer, "all") == 0)
        mLServ->SetAllLayersOnOff(true, false);
      else{
        mLServ->SetAllLayersOnOff(false, false);

        char tmp[1024];
        strcpy(tmp, mLayer);
        char * ptr = strtok(tmp, ";");
        while(ptr){
          unsigned short lyr = 0;
          unsigned short dtp = 0;

          int x = sscanf(ptr, "%hd:%hd", &lyr, &dtp);
          if(x == 2)
            mLServ->SetLayerOnOff(lyr, dtp, true, false);
          else if(x == 1){
            //Get all the possible datatypes for the given layer
            unsigned short * datatypeList;
            int datatypeNum = mDServ->GetDatatypesForLayer(lyr, &datatypeList);

            for(int i = 0; i < datatypeNum; i++)
              mLServ->SetLayerOnOff(lyr, datatypeList[i], true, false);

            mDServ->FreeLayerList(datatypeList);
          }
          ptr = strtok(NULL, ";");
        }
```

```
      }
      forcedraw = true;
    }

    //If using specified margin, zoom to the marker else show the home view
    if(!nomargin)
      mMServ->ZoomToMarker(mMarker, mWinMargin, mWinMargin, false);
    else
      mWServ->SetWindow(currwin, false);

    //Redraw
    mGServ->Redraw(forcedraw);
  }
}

//Change the current marker
void CsvPluginDlg::ChangeMarker(const sQCKMARKER marker)
{
  mMarker = marker;
  if(mGServ && mMarkerOn)
    mGServ->Redraw(false);
}

//Show the color dialog
void CsvPluginDlg::ShowColorDlg(){
  if(mCServ)
    mCServ->LaunchColorPicker(this);
}
```

Extended Services' callback functions: ***These functions are invoked by the extended service (color dialog).***

```
//IColorPickerCb's callback
void CsvPluginDlg::OnSelectColor(const sQCKCOLOR& aColor)
{
  //Change the dialog's marker color
  if(wCsvPluginDlgForm)
    wCsvPluginDlgForm->SetColor(aColor.mR, aColor.mG, aColor.mB);
}
```

QckServices' callback functions: ***These functions are invoked by QckServices.***

```
//IQckEventNotify's callback
void CsvPluginDlg::OnRedraw()
{
  //On redraw, draw the marker if marker is on
  if(mMServ && mMarkerOn)
    mMServ->DrawMarker(mMarker);
}
void CsvPluginDlg::PostFileOpen(const char* openedFileName, int fileType)
{
  //If a file is opened, close the dialog
  if(wCsvPluginDlgForm)
    wCsvPluginDlgForm->Close();
}
```

csvplugindlgUI_stubs.cpp

Constructors and Destructors:

```
wCsvPluginDlgForm_c::wCsvPluginDlgForm_c()
{
  wFileSelDlgForm = NULL;

  mClient = NULL;
  mHasTop = false;
  mReadCSV = false;

  mMarker.mXY[0] = 0.0;
  mMarker.mXY[1] = 0.0;
  mMarker.mShape = DEFSHAPE;
  mMarker.mSize = DEFSIZE;
  mMarker.mThickness = DEFWIDTH;
  mMarker.mRGB[0] = 253;
  mMarker.mRGB[1] = 197;
  mMarker.mRGB[2] = 2;
  mWinMargin = DEFMARGIN;
  mNoMargin = false;
  mAllLayersOn = false;
  mWinSpeed = DEFSPEED;
  mPBtn = false;

  mColor.pixel = 0;
  mPmap = XmUNSPECIFIED_PIXMAP;
  mAllocColor = false;

  strcpy(mProgPath, "");
  mContext = (XtAppContext) 0;
  .
  .
  .
}

wCsvPluginDlgForm_c::~wCsvPluginDlgForm_c()
{
  Close();
  .
  .
  .
  strcpy(mProgPath, "");
  mContext = (XtAppContext) 0;

  mMarker.mXY[0] = 0.0;
  mMarker.mXY[1] = 0.0;
  mMarker.mShape = DEFSHAPE;
  mMarker.mSize = DEFSIZE;
  mMarker.mThickness = DEFWIDTH;
  mMarker.mRGB[0] = 253;
  mMarker.mRGB[1] = 197;
  mMarker.mRGB[2] = 2;
  mWinMargin = DEFMARGIN;
  mNoMargin = false;
  mAllLayersOn = false;
  mWinSpeed = DEFSPEED;
  mPBtn = false;

  mColor.pixel = 0;
  mPmap = XmUNSPECIFIED_PIXMAP;
  mAllocColor = false;

  mClient = NULL;
  mHasTop = false;

  if(wFileSelDlgForm)
    delete(wFileSelDlgForm);
  wFileSelDlgForm = NULL;
}
```

Init: ***This function initializes the dialog***

```
void
wCsvPluginDlgForm_c::Init(void *Parent, const char *ProgPath, ICsvPluginDlg *Client)
{
  //Initialize dialog
  strcpy(mProgPath, ProgPath);
  mContext = XtWidgetToApplicationContext((Widget) Parent);
  mClient = Client;

  //Set the area list's label
  char tmp[1024];
  sprintf(tmp, "%-20s %-10s %s", "Cell", "Layer", "Coordinate");
  XmString lStr = XmStringCreateLocalized((char*)tmp);
  XtVaSetValues(wAreaLbl,XmNlabelString,lStr,NULL);
  XmStringFree(lStr);

  //Add the shapes to the drop down list
  for(int i = 0; i < shapenum; i++){
    XmString shapelist = XmStringCreateLocalized((char *) shapename[i]);
    XmListAddItemUnselected(wMarkList,shapelist,i+1);
    XmStringFree(shapelist);
  }

  //Disable the view buttons
  XtSetSensitive(wPrevBtn, False);
  XtSetSensitive(wNextBtn, False);
  XtSetSensitive(wFirstBtn, False);
  XtSetSensitive(wPlayBtn, False);
  XtSetSensitive(wLastBtn, False);

  //Initialize the area list to default
  InitList();

  //Create the file selection dialog
  if(!wFileSelDlgForm)
    create_wFileSelDlgShell((Widget) Parent);
}
```

Show/Hide: ***These functions show/hide the dialog.***

```
void
wCsvPluginDlgForm_c::PopupMsgBox(const char *Msg)
{
  //This is local to artwork conversion software
#if(!defined(LOCALOFF))
  MyMessageBox(wCsvPluginDlgShell, (char*) Msg, (char*) "CSV Plugin Sample",  MB_OK);
#endif
}

bool
wCsvPluginDlgForm_c::IsManaged()
{
  //Check if the dialog is up
  return XtIsManaged(wCsvPluginDlgForm);
}
void
wCsvPluginDlgForm_c::HideDlg()
{
  //Lower the dialog
  if(IsManaged())
    XLowerWindow(XtDisplay(wCsvPluginDlgShell),XtWindow(wCsvPluginDlgShell));

  //Lower the file selection dialog
  if(wFileSelDlgForm)
    wFileSelDlgForm->HideDlg();
}
void
wCsvPluginDlgForm_c::UnHideDlg()
{
  //Raise the dialog
  if(IsManaged())
```

```
    XRaiseWindow(XtDisplay(wCsvPluginDlgShell),XtWindow(wCsvPluginDlgShell));

  //Raise the file selection dialog
  if(wFileSelDlgForm)
    wFileSelDlgForm->UnHideDlg();
}

void
wCsvPluginDlgForm_c::Popup(bool HasTop)
{
  //Popup the dialog
  mHasTop = HasTop;
  XtSetSensitive(wCsvPluginForm, mHasTop);
  XtSetSensitive(wHelpBtn, mHasTop);

  if(IsManaged())
    UnHideDlg();
  else{
    XtManageChild(wCsvPluginDlgForm);

    //This is local to artwork conversion software
#if(!defined(LOCALOFF))
    if(mFirstLoad){
      GetXdefaultsWidgetDimension((char*)"csvplugin", (char*)"CsvPluginDlg", wCsvPluginDlgForm, 0);
      mFirstLoad = false;
    }
#endif
  }
}
void
wCsvPluginDlgForm_c::Close()
{
  //Close the dialog
  if(IsManaged())
    XtUnmanageChild(wCsvPluginDlgForm);

  //Close the file selection dialog
  if(wFileSelDlgForm)
    wFileSelDlgForm->Close();
}
```

ReadCSV: ***This function reads the selected input file.***

```
void
wCsvPluginDlgForm_c::ReadCSV()
{
  mReadCSV = true;
  //Reset the dialog
  OnClickClose(NULL, NULL);
  //Initialize the area list
  InitList();
  //Set the dialog
  OnClickOpen(NULL, NULL);

  mReadCSV = false;
}
```

OnClickOpen: ***This function sets up the dialog***

```
void
wCsvPluginDlgForm_c::OnClickOpen (Widget w, XtPointer xt_call_data )
{
  char str[256];

  //Set the dialog's margin/layer/speed
  sprintf(str, "%g", mWinMargin);
  XmTextFieldSetString(wMarginTxt, (char *) str);
  XmToggleButtonSetState(wMarginTgl, mNoMargin, True);
  XmToggleButtonSetState(wLayerTgl, mAllLayersOn, True);
  XtVaSetValues(wSpeedScale, XmNvalue, mWinSpeed, NULL);

  //Set the dialog's marker shape/thickness/size/color
```

```
  XmListSelectPos(wMarkList, mMarker.mShape, True);
  XtVaSetValues(wMarkWScale, XmNvalue, mMarker.mThickness, NULL);
  XtVaSetValues(wMarkSScale, XmNvalue, mMarker.mSize, NULL);
  SetColor(mMarker.mRGB[0], mMarker.mRGB[1], mMarker.mRGB[2]);

  //Enable the dialog's play/first/last view buttons
  if(mWinNum){
    XtSetSensitive(wFirstBtn, True);
    XtSetSensitive(wPlayBtn, True);
    XtSetSensitive(wLastBtn, True);
  }
}
```

OnClickClose: ***This function resets the dialog***

```
void
wCsvPluginDlgForm_c::OnClickClose (Widget w, XtPointer xt_call_data )
{
  //Free the xpm color
  FreeColor();
  //Pause the view
  OnPauseView();
  //Hide the marker
  if(mClient)
    mClient->HideMarker();

  //Clear the area list
  ClearList();
}
```

OnClickInBtn: ***This function pops up the file selection dialog***

```
void
wCsvPluginDlgForm_c::OnClickInBtn (Widget w, XtPointer xt_call_data )
{
  //Popup the file selection dialog
  if(wFileSelDlgForm){
    if(wFileSelDlgForm->IsManaged())
      wFileSelDlgForm->UnHideDlg();
    else
      wFileSelDlgForm->Popup();
  }
}
```

InitList: ***This function reads the selected input file and populates the listbox***

```
void
wCsvPluginDlgForm_c::InitList()
{
  //Default area list values
  mWinIdx = 1;
  mWinNum = 0;
  mCelTxt = NULL;
  mLyrTxt = NULL;
  mWinTxt = NULL;

  if(wFileSelDlgForm && mReadCSV){
    struct stat statbuf;
    //Validate the selected area file
    if(!stat(wFileSelDlgForm->mFileName,&statbuf) && (statbuf.st_mode & S_IFREG)) {
      XmTextFieldSetString(wInTxt, (char *) wFileSelDlgForm->mFileName);

      char tmp[1024];
      //Open the selected area file
      FILE * in = fopen(wFileSelDlgForm->mFileName, "r");
      if(!in){
        sprintf(tmp, "Failed to open input file: %s", wFileSelDlgForm->mFileName);
        PopupMsgBox(tmp);
      }
```

```
      else{
        int x;
        int num;
        char line[1024*4];
        char * readin;
        //Read the number of areas
        while(readin = fgets(line, sizeof(line), in)){
          x = sscanf(line, "%d", &num);
          if(x == 1)
            break;
        }
        if(readin == NULL){
          sprintf(tmp, "Failed reading input file: %s", wFileSelDlgForm->mFileName);
          PopupMsgBox(tmp);
        }
        else{
          double xcoor,ycoor;
          mCelTxt = (char**) malloc(sizeof(char *) * num);
          memset(mCelTxt, 0, sizeof(char *) * num);
          mLyrTxt = (char**) malloc(sizeof(char *) * num);
          memset(mLyrTxt, 0, sizeof(char *) * num);
          mWinTxt = (char**) malloc(sizeof(char *) * num);
          memset(mWinTxt, 0, sizeof(char *) * num);
          mWinNum = 0;
          //Read/save the areas
          for(int i = 0; i < num; i++){
            mCelTxt[mWinNum] = (char*) realloc(mCelTxt[mWinNum], sizeof(char) * 256);
            mLyrTxt[mWinNum] = (char*) realloc(mLyrTxt[mWinNum], sizeof(char) * 256);
            mWinTxt[mWinNum] = (char*) realloc(mWinTxt[mWinNum], sizeof(char) * 256);
            while(readin = fgets(line, sizeof(line), in)){
              x = sscanf(line, "%[^,],%[^,],%s", mCelTxt[mWinNum], mLyrTxt[mWinNum], mWinTxt[mWinNum]);
              if(x == 3)
                break;
            }
            if(readin == NULL){
              ClearList();
              sprintf(tmp, "Failed reading input file: %s", wFileSelDlgForm->mFileName);
              PopupMsgBox(tmp);
              break;
            }
            if(mClient && mClient->CellExist(mCelTxt[mWinNum]))
              mWinNum++;
          }
        }
        //Close the selected area file
        fclose(in);

        char cellstr[16];
        char layerstr[8];
        //Add the saved areas to the dialog's area list
        for(int i = 0; i < mWinNum; i++){
          if(strlen(mCelTxt[i]) > 12){
            for(int j = 0; j < 12; j++)
              cellstr[j] = mCelTxt[i][j];
            cellstr[12] = '\0';
            strcat(cellstr, "...");
          }
          else
            strcpy(cellstr, mCelTxt[i]);
          if(strlen(mLyrTxt[i]) > 3){
            for(int j = 0; j < 3; j++)
              layerstr[j] = mLyrTxt[i][j];
            layerstr[3] = '\0';
            strcat(layerstr, "...");
          }
          else
            strcpy(layerstr, mLyrTxt[i]);
          sprintf(tmp, "%-20s %-10s %s", cellstr, layerstr, mWinTxt[i]);
          XmString attrlist = XmStringCreateLocalized((char *) tmp);
          XmListAddItemUnselected(wAreaList,attrlist,i+1);
          XmStringFree(attrlist);
        }

      } // If open area file successful
    } // If valid area file
  } // If read area file
}
```

ClearList: ***This function clears the listbox***

```
void
wCsvPluginDlgForm_c::ClearList()
{
  //Reset the dialog
  XmTextFieldSetString(wInTxt, (char *) "");
  XmListDeleteAllItems(wAreaList);
  XtSetSensitive(wPrevBtn, False);
  XtSetSensitive(wNextBtn, False);
  XtSetSensitive(wFirstBtn, False);
  XtSetSensitive(wPlayBtn, False);
  XtSetSensitive(wLastBtn, False);

  //Clear the saved area list
  for(int i = 0; i < mWinNum; i++){
    if(mCelTxt && mCelTxt[i])
      free(mCelTxt[i]);
    if(mLyrTxt && mLyrTxt[i])
      free(mLyrTxt[i]);
    if(mWinTxt && mWinTxt[i])
      free(mWinTxt[i]);
  }
  if(mCelTxt)
    free(mCelTxt);
  mCelTxt = NULL;
  if(mLyrTxt)
    free(mLyrTxt);
  mLyrTxt = NULL;
  if(mWinTxt)
    free(mWinTxt);
  mWinTxt = NULL;

  mWinNum = 0;
  mWinIdx = 1;
}
```

View functions: ***These functions view the selected item in the list.***

```
void
wCsvPluginDlgForm_c::OnSelectAreaList(Widget w, XtPointer xt_call_data)
{
  //Pause the view
  OnPauseView();

  //Get the index of the selected area
  int selcount;
  int * selitems;
  XmListGetSelectedPos(wAreaList, &selitems, &selcount);
  if(selcount > 0){
    mWinIdx = selitems[0];
    SetVisible(wAreaList, mWinIdx);
    //View the selected area
    OnViewWin();
  }
  XtFree((char*)selitems);
}

void
wCsvPluginDlgForm_c::OnClickFirstBtn(Widget w, XtPointer xt_call_data)
{
  //Select the first area
  XmListSelectPos(wAreaList, 1, true);
}

void
wCsvPluginDlgForm_c::OnClickLastBtn(Widget w, XtPointer xt_call_data)
{
  //Select the last area
  XmListSelectPos(wAreaList, mWinNum, true);
}
```

```
void
wCsvPluginDlgForm_c::OnClickPrevBtn(Widget w, XtPointer xt_call_data)
{
  //Select the previous area
  if(mWinIdx > 1)
    XmListSelectPos(wAreaList, mWinIdx-1, true);
}

void
wCsvPluginDlgForm_c::OnClickNextBtn(Widget w, XtPointer xt_call_data)
{
  //Select the next area
  if(mWinIdx < mWinNum)
    XmListSelectPos(wAreaList, mWinIdx+1, true);
}

void
wCsvPluginDlgForm_c::OnClickPlayBtn(Widget w, XtPointer xt_call_data)
{
  //Play/Pause the view
  if(!mPBtn)
    OnPlayView();
  else
    OnPauseView();
}

void
wCsvPluginDlgForm_c::OnPauseView()
{
  //Pause view
  if(mPBtn){
    XmString lStr = XmStringCreateLocalized((char*)"Play");
    XtVaSetValues(wPlayBtn,XmNlabelString,lStr,XmNalignment, XmALIGNMENT_CENTER,NULL);
    XmStringFree(lStr);
    mPBtn = false;
  }
}

void
wCsvPluginDlgForm_c::OnPlayView()
{
  //Play view
  XmString lStr = XmStringCreateLocalized((char*)"Pause");
  XtVaSetValues(wPlayBtn,XmNlabelString,lStr,XmNalignment, XmALIGNMENT_CENTER,NULL);
  XmStringFree(lStr);
  mPBtn = true;

  mWinIdx--;
  XtAppAddTimeOut(mContext,0,OnStartShow,this);
}
void
wCsvPluginDlgForm_c::OnStartShow(XtPointer data, XtIntervalId *id)
{
  wCsvPluginDlgForm_c *athis=(wCsvPluginDlgForm_c*)data;

  if(athis){
    //If pause, stop view
    if(!athis->mPBtn)
      return;

    //Select the next area and view
    athis->mWinIdx++;
    XmListSelectPos(athis->wAreaList, athis->mWinIdx, false);
    athis->SetVisible(athis->wAreaList, athis->mWinIdx);
    athis->OnViewWin();

    //If pause, stop view
    if(!athis->mPBtn)
      return;

    //Continue onto the next area or pause if end is reached
    if(athis->mWinIdx >= athis->mWinNum)
      athis->OnPauseView();
    else
      XtAppAddTimeOut(athis->mContext,athis->mWinSpeed*1000,&OnStartShow,athis);
  }
}
```

```
void
wCsvPluginDlgForm_c::OnViewWin()
{
  //Enable/Disable previous button
  if(mWinIdx <= 1)
    XtSetSensitive(wPrevBtn, False);
  else
    XtSetSensitive(wPrevBtn, True);

  //Enable/Disable next button
  if(mWinIdx >= mWinNum)
    XtSetSensitive(wNextBtn, False);
  else
    XtSetSensitive(wNextBtn, True);

  if(mWinIdx >= 1 && mWinIdx <= mWinNum){
    //Lower the dialog
    HideDlg();

    double x, y;
    sscanf(mWinTxt[mWinIdx-1], "%lf,%lf", &x, &y);

    mMarker.mXY[0] = x;
    mMarker.mXY[1] = y;
    //Show the marker
    if(mClient)
      mClient->ShowMarker(mCelTxt[mWinIdx-1], mLyrTxt[mWinIdx-1], mWinMargin, mNoMargin, mAllLayersOn, mMarker);

    //Raise the dialog
    UnHideDlg();
  }
}
void
wCsvPluginDlgForm_c::SetVisible(Widget w, int pos)
{
  //Set an area visible in the dialog
  int top, visible;
  XtVaGetValues(w, XmNtopItemPosition, &top, XmNvisibleItemCount, &visible, NULL);
  if(pos < top)
    XmListSetPos(w, pos);
  else if(pos >= top+visible)
    XmListSetBottomPos(w, pos);
}
```

Events functions: ***These functions are invoked by user events***

```
void
wCsvPluginDlgForm_c::OnClickSpeedScale (Widget w, XtPointer xt_call_data )
{
  XmScaleCallbackStruct *call_data = (XmScaleCallbackStruct *) xt_call_data;
  //Change the viewing speed
  mWinSpeed = call_data->value;
}

void
wCsvPluginDlgForm_c::OnChangeMargin (Widget w, XtPointer xt_call_data )
{
  //Change the viewing margin
  mWinMargin = DEFMARGIN;

  char * str = XmTextFieldGetString(wMarginTxt);
  if(str){
    sscanf(str, "%lf", &mWinMargin);
    XtFree(str);
  }
  if(mWinMargin <=0)
    mWinMargin = DEFMARGIN;
}

void
wCsvPluginDlgForm_c::OnMarginTgl (Widget w, XtPointer xt_call_data )
{
  //Set the viewing margin on/off
  mNoMargin = XmToggleButtonGetState(wMarginTgl);
  if(mNoMargin)
```

```
    XtSetSensitive(wMarginTxt, False);
  else
    XtSetSensitive(wMarginTxt, True);
}

void
wCsvPluginDlgForm_c::OnLayerTgl (Widget w, XtPointer xt_call_data )
{
  //Set the viewing layer on/off
  mAllLayersOn = XmToggleButtonGetState(wLayerTgl);
}

void
wCsvPluginDlgForm_c::ChangeMarker()
{
  //If an area is selected change the marker
  if(mWinIdx >= 1 && mWinIdx <= mWinNum){
    //Pause view
    OnPauseView();
    //Lower the dialog
    HideDlg();
    //Change the marker
    if(mClient)
      mClient->ChangeMarker(mMarker);
    //Raise the dialog
    UnHideDlg();
  }
}

void
wCsvPluginDlgForm_c::OnClickWidthScale (Widget w, XtPointer xt_call_data )
{
  XmScaleCallbackStruct *call_data = (XmScaleCallbackStruct *) xt_call_data;
  //Change the marker's thickness
  mMarker.mThickness = call_data->value;
  ChangeMarker();
}

void
wCsvPluginDlgForm_c::OnClickSizeScale (Widget w, XtPointer xt_call_data )
{
  XmScaleCallbackStruct *call_data = (XmScaleCallbackStruct *) xt_call_data;
  //Change the marker's size
  if(call_data->value % 10 == 0){
    mMarker.mSize = call_data->value;
  }
  else if(call_data->value % 10 < 5){
    mMarker.mSize = (call_data->value / 10)*10 + 10;
    XmScaleSetValue(wMarkSScale, mMarker.mSize);
  }
  else{
    mMarker.mSize = (call_data->value / 10)*10;
    XmScaleSetValue(wMarkSScale, mMarker.mSize);
  }
  ChangeMarker();
}

void
wCsvPluginDlgForm_c::OnDragSizeScale (Widget w, XtPointer xt_call_data )
{
  XmScaleCallbackStruct *call_data = (XmScaleCallbackStruct *) xt_call_data;
  //Change the marker's size
  if(call_data->value % 10 < 5)
    XmScaleSetValue(wMarkSScale, (call_data->value / 10)*10);
  else
    XmScaleSetValue(wMarkSScale, (call_data->value / 10)*10 + 10);
}

void
wCsvPluginDlgForm_c::OnClickShapeList (Widget w, XtPointer xt_call_data )
{
  //Change the marker's shape
  mMarker.mShape = DEFSHAPE;

  int selcount;
  int * selitems;
  XmListGetSelectedPos(wMarkList, &selitems, &selcount);
```

```
  if(selcount > 0)
    mMarker.mShape = selitems[0];
  XtFree((char*)selitems);

  ChangeMarker();
}

void
wCsvPluginDlgForm_c::OnClickColorBtn (Widget w, XtPointer xt_call_data )
{
  //Popup the color dialog
  if(mClient)
    mClient->ShowColorDlg();
}

void
wCsvPluginDlgForm_c::SetColor(const unsigned short R, const unsigned short G, const unsigned short B)
{
  //Set the marker's color
  char lR[8], lG[8], lB[8];
  sprintf(lR,"000%X",R * 257);
  sprintf(lG,"000%X",G * 257);
  sprintf(lB,"000%X",B * 257);

  char **pixData = (char**)malloc(sizeof(char*)*PIXSIZE);
  memset(pixData, 0, sizeof(char*)*PIXSIZE);
  for(int i = 0; i < PIXSIZE; i++)
    pixData[i] = (char*)malloc(sizeof(char*)*PIXSIZE);

  strcpy(pixData[0], "20 20 1 1");
  sprintf(pixData[1],". c #%s%s%s",&lR[strlen(lR)-4],&lG[strlen(lG)-4],&lB[strlen(lB)-4]);
  for(int i = 2; i < PIXSIZE; i++)
    strcpy(pixData[i], "....................");

  FreeColor();
  //Create a pixmap
  XpmCreatePixmapFromData(XtDisplay(wCsvPluginDlgForm),  DefaultRootWindow(XtDisplay(wCsvPluginDlgForm)),
pixData, &mPmap, NULL, NULL);
  //Set the color button's label
  XtVaSetValues(wMarkBtn, XmNlabelType, XmPIXMAP, XmNlabelPixmap, mPmap, XmNarmPixmap, mPmap, NULL);

  mMarker.mRGB[0] = R;
  mMarker.mRGB[1] = G;
  mMarker.mRGB[2] = B;

  for(int i = 0; i < PIXSIZE; i++)
    free(pixData[i]);
  free(pixData);
  pixData = NULL;

  //Change the marker's color
  ChangeMarker();
}

void
wCsvPluginDlgForm_c::FreeColor()
{
  //free the pixmap
  if(mPmap != XmUNSPECIFIED_PIXMAP)
    XFreePixmap(XtDisplay(wCsvPluginDlgForm), mPmap);
  mPmap = XmUNSPECIFIED_PIXMAP;
}
```

OnClickCancelBtn: ***This function closes the dialog***

```
void
wCsvPluginDlgForm_c::OnClickCancelBtn (Widget w, XtPointer xt_call_data )
{
  //Close the dialog
  Close();
}
```

OnClickHelpBtn: ***This function pops up the help information***

```
void
wCsvPluginDlgForm_c::OnClickHelpBtn (Widget w, XtPointer xt_call_data )
{
  //This is local to artwork conversion software
#if(!defined(LOCALOFF))
  IAcsHelpSystem *acshelp = IAcsHelpSystem::Instance();
  if(acshelp){
    char progPath[1024];
    char helpPath[1024];
    char pagePath[1024];

    wSplitFilename((char*)mProgPath,progPath,NULL);
    wJoinFilename(helpPath,progPath,(char*)"help");
    wJoinFilename(pagePath,helpPath,(char*)"csvplugin_dialog.htm");
    if(!wFileExist(pagePath))
      wJoinFilename(pagePath,helpPath,(char*)"contents.htm");

    if(acshelp->LoadHelpPage(progPath,pagePath))
      PopupMsgBox((char*)acshelp->GetLastErrorMsg());
  }
#endif
}
```

File Selection dialog: ***These functions selects a file to populate the list box***

```
wFileSelDlgForm_c::wFileSelDlgForm_c()
{
  strcpy(mFileName, "");
  //This is local to artwork conversion software
#if(!defined(LOCALOFF))
  mFirstLoad = true;
#endif
}
wFileSelDlgForm_c::~wFileSelDlgForm_c()
{
  //This is local to artwork conversion software
#if(!defined(LOCALOFF))
  if(!mFirstLoad)
    SetXdefaultsWidgetDimension((char*)"csvplugin",(char*)"FileSelDlg",wFileSelDlgForm);
  mFirstLoad = true;
#endif
  strcpy(mFileName, "");
}

void
wFileSelDlgForm_c::Popup()
{
  //Popup the file selection dialog
  if(IsManaged())
    UnHideDlg();
  else{
    XmTextFieldSetString(wFFileTxt, (char *) mFileName);
    XtManageChild(wFileSelDlgForm);

    //This is local to artwork conversion software
#if(!defined(LOCALOFF))
    if(mFirstLoad){
      GetXdefaultsWidgetDimension((char*)"csvplugin", (char*)"FileSelDlg", wFileSelDlgForm, 0);
      mFirstLoad = false;
    }
#endif
  }
}
void
wFileSelDlgForm_c::Close()
{
  //Close the file selection dialog
  if(IsManaged())
    XtUnmanageChild(wFileSelDlgForm);
}
```

```
bool
wFileSelDlgForm_c::IsManaged()
{
  //Check if the file selection dialog is up
  return XtIsManaged(wFileSelDlgForm);
}
void
wFileSelDlgForm_c::HideDlg()
{
  //Lower the file selection dialog
  if(IsManaged())
    XLowerWindow(XtDisplay(wFileSelDlgShell),XtWindow(wFileSelDlgShell));
}
void
wFileSelDlgForm_c::UnHideDlg()
{
  //Raise the file selection dialog
  if(IsManaged())
    XRaiseWindow(XtDisplay(wFileSelDlgShell),XtWindow(wFileSelDlgShell));
}

void
wFileSelDlgForm_c::OnClickOkBtn (Widget w, XtPointer xt_call_data )
{
  char * str = XmTextFieldGetString(wFFileTxt);
  strcpy(mFileName, str);
  XtFree(str);

  //Close the file selection dialog
  Close();

  //Read the selected area file
  wCsvPluginDlgForm->ReadCSV();
}
void
wFileSelDlgForm_c::OnClickCancelBtn (Widget w, XtPointer xt_call_data )
{
  //Close the file selection dialog
  Close();
}
```