

```
//-----  
// Project: POS Business Application  
// Module: POS_System.cpp  
// Author: Stephen W. McClure  
// Date: Oct 2007  
//-----  
// Description:  
//  
// This module contains the POS system functions.  
//-----  
// This program is the exclusive property of Quantum Blue Technology LLC.  
// Copyright(c) 2005, 2006, 2007, 2008.  
//  
// Reproduction, disclosure, or use, in whole or in part, are not to be  
// undertaken except with prior written authorization from the owner  
// Quantum Blue Technology LLC.  
//  
// Contact Information: Quantum Blue Technology LLC  
//                        1424 Welsh Way  
//                        Ramona  
//                        CA 92065  
//  
//                        Phone: (858) 837-2160  
//                        Email: info@quantumbluetechology.com  
//-----  
  
#include <windows.h>                // This is required  
#undef GetEnvironmentVariable        // for proper operation of the GetEnvironmentVariable  
#include <vcl.h>  
#pragma hdrstop  
  
#include "Definitions.h"  
#include "Typedefs.h"  
#include "Externals.h"  
  
#include "POS_System.h"  
#include "POS_Main.h"  
#include "POS_Log.h"  
#include "POS_Files.h"  
#include "POS_Accounts_Payable.h"  
#include "POS_Accounts_Receivable.h"  
#include "POS_Bank_Reconciliation.h"  
#include "POS_General_Ledger.h"  
#include "POS_Inventory.h"  
#include "POS_Payroll.h"  
#include "POS_Purchase_Order.h"  
#include "POS_Register.h"  
#include "POS_Registry.h"  
#include "POS_Import.h"  
#include "POS_Invoice_Number.h"  
#include "POS_Register_Reports.h"  
  
#include <process.h>  
#include <sysutils.hpp>  
#include <dir.h>  
#include <fcntl.h>  
#include <io.h>  
#include <sys\stat.h>  
#include <stdio.h>  
#include <mem.h>  
#include <errno.h>  
#include <math.hpp>  
#include <Inifiles.hpp>
```

```
//-----  
// POS MessageBox  
//  
// This function displays an error message box.  
//  
// Returns: status of MessageBox key press.  
//-----  
  
int POS_MessageBox (void * Null_ptr, const char * Text, const char * Caption, int Flags)  
{  
    int status;  
  
    Application->NormalizeTopMosts();  
    status = MessageBox(Null_ptr, Text, Caption, Flags | MB_TASKMODAL);  
    Application->RestoreTopMosts();  
  
    return (status);  
}  
  
//-----  
// Status Display Clear  
//  
// This function removes any displayed message from the status display.  
//  
// Returns: SUCCESS  
//-----  
  
int Status_display_clear (void)  
{  
  
    /* Erase the Status display message */  
    Status_display ("", INDEFINITELY);  
  
    /* Note that the display is inactive */  
    status_display = INACTIVE;  
  
    return (SUCCESS);  
}  
  
//-----  
// Status Display Beep  
//  
// This function beeps and places the passed data into the status bar for the  
// specified timeout period.  
//  
// Returns: SUCCESS  
//-----  
  
int Status_display_beep (char * status_string, int timeout)  
{  
  
    /* The status display is now active */  
    status_display = ACTIVE;  
  
    /* Sound the warning */  
    MessageBeep(MB_ICONEXCLAMATION);  
  
    /* Display the status message */  
    Status_display (status_string, timeout);  
  
    /* SUCCESS */  
    return (SUCCESS);  
}
```

```
//-----  
// Status Display Blank  
//  
// This function displays the new status message only if the currently  
// displayed status message is blank.  
//  
// Returns: SUCCESS  
//-----  
  
int Status_display_blank (char * status_string, int timeout)  
{  
  
/* Is the Status Message currently blank? */  
if (status_display == INACTIVE)  
{  
    /* Status display is now active */  
    status_display = ACTIVE;  
  
    /* Yes - Display the status message */  
    Status_display (status_string, timeout);  
  
    /* SUCCESS */  
    return (SUCCESS);  
}  
  
/* FAILURE */  
return (FAILURE);  
}  
  
  
//-----  
// Status Display Beep Blank  
//  
// This function beeps and places the passed data into the status bar for the  
// specified timeout period only if the currently displayed status message is  
// blank.  
//  
// Returns: SUCCESS  
//-----  
  
int Status_display_beep_blank (char * status_string, int timeout)  
{  
  
/* Is the Status Message currently blank? */  
if (status_display == INACTIVE)  
{  
    /* Status display is now active */  
    status_display = ACTIVE;  
  
    /* Yes - Display the status message */  
    Status_display_beep (status_string, timeout);  
  
    /* SUCCESS */  
    return (SUCCESS);  
}  
  
/* FAILURE */  
return (FAILURE);  
}
```

```
//-----  
// Clean Up String  
//  
// This function cleans up the parameter string by removing leading and  
// trailing spaces, multiple embedded spaces are replaced by a single space,  
// and all control characters are removed.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int clean_up_string (AnsiString * parameter)  
{  
    int    parameter_length;  
    char   buffer [MAX_STRING_SIZE];  
    char * read_ptr;  
    char * write_ptr;  
  
    AnsiString local_parameter;  
    AnsiString trimmed_parameter;  
  
    /* Set Text to Uppercase */  
    local_parameter = *parameter;  
  
    /* Remove leading and trailing spaces and any control characters */  
    trimmed_parameter = local_parameter.Trim();  
  
    /* Make a copy of the parameter string */  
    strcpy (buffer, trimmed_parameter.c_str());  
  
    /* Initialize pointers */  
    read_ptr = buffer;  
    write_ptr = buffer;  
  
    /* Scan string and remove excess spaces */  
    while (*read_ptr != 0x00)  
    {  
        /* Copy non-blank characters */  
        if (*read_ptr != ' ')  
            *write_ptr++ = *read_ptr++;  
  
        /* Copy single space character */  
        if (*read_ptr == ' ')  
        {  
            /* Copy single space character */  
            *write_ptr++ = *read_ptr++;  
  
            /* Remove any excess spaces */  
            while (*read_ptr == ' ')  
            {  
                /* Step to next character */  
                read_ptr++;  
            }  
        }  
    }  
  
    /* Null terminate the updated string */  
    *write_ptr = 0x00;  
  
    /* Return updates string */  
    *parameter = buffer;  
  
    /* All is well... */  
    return (SUCCESS);  
}
```

```
//-----  
// Read System Registry File  
//  
// This function reads the system configuration registry for specific  
// system variables.  
//  
// Note: The POS variable is set to UPPER CASE.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int read_system_registry_parameters (void)  
{  
  
/* Note: Only the Computer Name is currently obtained from the System Registry */  
POS_computer_name = AnsiUpperCase(get_registry_string  
    ("SYSTEM\\CurrentControlSet\\Control\\ComputerName\\ComputerName",  
    "ComputerName"));  
  
/* Is the Computer Name defined? */  
if (POS_computer_name == "")  
{  
    /* No - Set the computer name to "Undefined" */  
    POS_computer_name = "UNDEFINED";  
}  
  
/* Success */  
return (SUCCESS);  
}
```

```
//-----  
// Read Configuration  
//  
// This function reads the specified configuration parameter from the  
// configuration file. If the configuration parameter does not exist then  
// the default is returned.  
//  
// Returns: Configuration Parameter  
//-----  
  
AnsiString Read_configuration (AnsiString section_name,  
                              AnsiString configuration_name,  
                              AnsiString default_setting)  
{  
    int index;  
    int start_of_section;  
    int end_of_section;  
    int number_of_configuration_lines;  
    AnsiString full_section_name;  
    AnsiString configuration_string;  
    AnsiString first_token;  
    AnsiString second_token;  
    AnsiString third_token;  
  
    /* Load local configuration file into Memo Box */  
    POS_Main_Form->System_Configuration_Memo->Lines->Clear();  
    POS_Main_Form->System_Configuration_Memo->Lines->LoadFromFile(system_configuration_file_name);  
  
    /* Initialization */  
    index = 0;  
    start_of_section = 0;  
    end_of_section = 0;  
    number_of_configuration_lines = POS_Main_Form->System_Configuration_Memo->Lines->Count;  
    full_section_name = "[" + section_name + "];"  
  
    /* Scan for section name */  
    while (index < number_of_configuration_lines)  
    {  
        /* Section Name found? */  
        if (AnsiUpperCase(POS_Main_Form->System_Configuration_Memo->Lines->Strings[index]) ==  
            AnsiUpperCase(full_section_name))  
        {  
            /* Start of section found */  
            start_of_section = index + 1;  
            break;  
        }  
  
        /* Step to next configuration entry */  
        index++;  
    }  
  
    /* Section name not found? */  
    if (index >= number_of_configuration_lines)  
    {  
        /* Return default value */  
        return (AnsiUpperCase(default_setting));  
    }  
  
    /* Step to next line */  
    index++;  
}
```

```
/* Scan for subsequent section name or end of file */
while (index < number_of_configuration_lines)
{
    /* Next section name detected? */
    if (POS_Main_Form->System_Configuration_Memo->Lines->Strings[index].c_str()[0] == '[')
    {
        /* End of section found */
        break;
    }

    /* Step to next configuration entry */
    index++;
}

/* End of section found */
end_of_section = index;

/* Search through section for required configuration variable */
for (index = start_of_section; index < end_of_section; index++)
{
    /* Get the next configuration string */
    configuration_string = POS_Main_Form->System_Configuration_Memo->Lines->Strings[index];
    configuration_string = configuration_string.Trim();

    /* Check that configuration string is not null and is not a comment */
    if ((configuration_string != "") &&
        (configuration_string.c_str()[0] != ';'))
    {
        /* Extract tokens from string */
        if (extract_token (configuration_string, &first_token, 0) == FAILURE)
            return (default_setting);
        if (extract_token (configuration_string, &second_token, 1) == FAILURE)
            return (default_setting);
        if (extract_token (configuration_string, &third_token, 2) == FAILURE)
            return (default_setting);

        /* Check for configuration variable name */
        if (AnsiUpperCase(first_token) == AnsiUpperCase(configuration_name))
        {
            /* Check that second token is the "=" sign */
            if (second_token == "=")
            {
                /* Return configuration setting */
                return (AnsiUpperCase(third_token));
            }

            /* Error detected */
            return (AnsiUpperCase(default_setting));
        }
    }
}

/* Configuration variable name not found - return default value */
return (AnsiUpperCase(default_setting));
}
```

```
//-----  
// Write Configuration  
//  
// This function writes the specified configuration parameter to the  
// configuration file. If the configuration parameter exists then its value  
// is changed. If the configuration parameter does not exist then a new  
// parameter is written.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int Write_configuration (AnsiString section_name,  
                        AnsiString configuration_name,  
                        AnsiString configuration_value)  
{  
    int index;  
    int start_of_section;  
    int end_of_section;  
    int number_of_configuration_lines;  
    AnsiString full_section_name;  
    AnsiString configuration_string;  
    AnsiString first_token;  
    AnsiString new_configuration_section;  
    AnsiString new_configuration_variable;  
  
    /* Initialization */  
    new_configuration_section = "[" + section_name + "];  
  
    /* Does the configuration value contain a space? */  
    if (StrPos (configuration_value.c_str(), " ") == NULL)  
    {  
        /* No - Just use the configuration value */  
        new_configuration_variable = configuration_name + " = " + configuration_value;  
    }  
    else  
    {  
        /* Yes - Place the configuration value in double quotes */  
        new_configuration_variable = configuration_name + " = \"" + configuration_value + "\"";  
    }  
  
    /*-----  
    /* First determine if the configuration name exists in the section */  
    /*-----  
  
    /* Load local configuration file into Memo Box */  
    POS_Main_Form->System_Configuration_Memo->Lines->Clear();  
    POS_Main_Form->System_Configuration_Memo->Lines->LoadFromFile(system_configuration_file_name);  
  
    /* Initialization */  
    index = 0;  
    start_of_section = 0;  
    end_of_section = 0;  
    number_of_configuration_lines = POS_Main_Form->System_Configuration_Memo->Lines->Count;  
    full_section_name = "[" + section_name + "];
```

```

/* Scan for section name */
while (index < number_of_configuration_lines)
{
    /* Section Name found? */
    if (AnsiUpperCase(POS_Main_Form->System_Configuration_Memo->Lines->Strings[index]) ==
        AnsiUpperCase(full_section_name))
    {
        /* Start of section found */
        start_of_section = index + 1;
        break;
    }

    /* Step to next configuration entry */
    index++;
}

/* Section name not found? */
if (index >= number_of_configuration_lines)
{
    /* Create New Section Name */
    POS_Main_Form->System_Configuration_Memo->Lines->Add(" ");
    POS_Main_Form->System_Configuration_Memo->Lines->Add(new_configuration_section);

    /* Write New Configuration Variable */
    POS_Main_Form->System_Configuration_Memo->Lines->Add(new_configuration_variable);

    /* Save Configuration File */
    POS_Main_Form->System_Configuration_Memo->Lines->SaveToFile (system_configuration_file_name);

    /* Return */
    return (SUCCESS);
}

/* Step to next line */
index++;

/* Scan for subsequent section name or end of file */
while (index < number_of_configuration_lines)
{
    /* Next section name detected? */
    if (POS_Main_Form->System_Configuration_Memo->Lines->Strings[index].c_str()[0] == '[')
    {
        /* End of section found */
        break;
    }

    /* Step to next configuration entry */
    index++;
}

/* End of section found */
end_of_section = index;

/* Search through section for required configuration variable */
for (index = start_of_section; index < end_of_section; index++)
{
    /* Get the next configuration string */
    configuration_string = POS_Main_Form->System_Configuration_Memo->Lines->Strings[index];
    configuration_string = configuration_string.Trim();

    /* Check that configuration string is not null and is not a comment */
    if ((configuration_string != "") &&
        (configuration_string.c_str()[0] != ';'))
    {
        /* Extract tokens from string */
        if (extract_token (configuration_string, &first_token, 0) == FAILURE) return (FAILURE);

        /* Check for configuration variable name */
        if (AnsiUpperCase(first_token) == AnsiUpperCase(configuration_name))
        {

```

```
        /* Configuration variable found - Delete it */
        POS_Main_Form->System_Configuration_Memo->Lines->Delete(index);
        break;
    }
}

/* The configuration variable was deleted or did not exist. */
/* Append a New configuration variable to the end of the section */
POS_Main_Form->System_Configuration_Memo->Lines->Insert(index, new_configuration_variable);

/* Save Configuration File */
POS_Main_Form->System_Configuration_Memo->Lines->SaveToFile (system_configuration_file_name);

/* Success */
return (SUCCESS);
}
```

```

//-----
// Read System Configuration File
//
// This function reads the system configuration file and sets all the system
// variables.
//
// Note: The POS variable is set to UPPER CASE.
//
// Returns: SUCCESS or FAILURE
//-----

int  read_system_configuration_file (void)
{

/* Access [System] Section Variables */
// Note: Computer name is found from the Registry.

POS_application      = Read_configuration ("System", "Application",      "Client");
POS_daylight_savings = Read_configuration ("System", "Daylight_Savings", "On");
POS_sound            = Read_configuration ("System", "Sound",           "OFF");

/* Access [Drives] Section Variables */
POS_server_drive     = Read_configuration ("Drives", "Server",          "C:");
POS_server_name      = Read_configuration ("Drives", "Server_Name",     "");
POS_server_shared_directory = Read_configuration ("Drives", "Server_Shared_Directory", "");

POS_local_drive_low_limit  = Read_configuration ("Drives", "local_drive_low_limit", "2.0");
POS_server_drive_low_limit = Read_configuration ("Drives", "server_drive_low_limit", "2.0");

/* Access [Directory] Section Variables */
POS_project_directory_name = Read_configuration ("Directories", "Project", "PROJ");

/* Fixed directory names */
POS_system_directory_name   = "System";
POS_log_directory_name      = "Log";
POS_license_directory_name   = "License";
POS_inventory_directory_name = "Inventory";
POS_accounts_payable_directory_name = "Accounts_Payable";
POS_accounts_receivable_directory_name = "Accounts_Receivable";
POS_point_of_sale_directory_name = "POS";
POS_general_ledger_directory_name = "General_Ledger";
POS_bank_reconciliation_directory_name = "Bank_Reconciliation";
POS_purchase_order_directory_name = "Purchase_Order";

POS_pdf_directory_name = "PDF";

/* Access [Options] Section Variables */
POS_register = Read_configuration ("Options", "Register", "0");

/* Access [General] Section Variables */
POS_currency_symbol = Read_configuration ("General", "Currency_Symbol", "$");

POS_minimize_screens = Read_configuration ("General", "Minimize_Screens", "OFF");
POS_fixed_forms       = Read_configuration ("General", "Fixed_Forms", "Fixed");
POS_display_network_messages = Read_configuration ("General", "Display_Network_Messages", "OFF");

POS_auto_pos = Read_configuration ("General", "Auto_POS", "OFF");
POS_auto_shutdown = Read_configuration ("General", "Auto_Shutdown", "OFF");
POS_exit_block = Read_configuration ("General", "Exit_Block", "OFF");

/* Access [Programs] Section Variables */
POS_PDF_Reader = Read_configuration ("Programs", "PDF_Reader", "AcroRd32.exe");

/* Success */
return (SUCCESS);
}

```

```
//-----  
// Log  
//  
// This function appends the log_entry to the end of the current system log.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int Log (char * log_entry)  
  
{  
    int file_handle;  
    int bytes_written;  
    char buffer[MAX_STRING_SIZE + 100];  
  
    /* Test if log file operational */  
    if (log_file_status == INACTIVE)  
    {  
        /* Log file is not available */  
        return (FAILURE);  
    }  
  
    /* Does the Log file exist? */  
    if (file_exists (system_log_file_name) == FAILURE)  
    {  
        /* No - Do nothing otherwise infinite loop can occur!!! */  
        return(FAILURE);  
    }  
  
    else  
    {  
        /* Attempt to open the existing training log file */  
        if (file_open (system_log_file_name, &file_handle, OPEN_READ_WRITE_EXCLUSIVE) == FAILURE)  
        {  
            /* Display error message */  
            sprintf (buffer, "Could not open Log File '%s'!!! [errno = %d, %s]",  
                    system_log_file_name,  
                    errno,  
                    _sys_errlist[errno]);  
  
            POS_MessageBox (NULL, buffer, "System Log File Error", MB_OK|MB_ICONEXCLAMATION);  
            return (FAILURE);  
        }  
    }  
  
    /* Prepend the data and time stamp to the log entry */  
    TimeSeparator = ':';  
    LongTimeFormat = "hh:mm:ssam/pm";  
    ShortTimeFormat = "hh:mm:ssam/pm";  
  
    DateSeparator = '/';  
    LongDateFormat = "mm/dd/yyyy";  
    ShortDateFormat = "mm/dd/yyyy";  
  
    /* Create the log file entry */  
    sprintf (buffer, "%s %s", DateTimeToStr(Now()).c_str(), log_entry);  
  
    /* Restore time format to normal setting */  
    LongTimeFormat = "hh:mm:ss am/pm";  
    ShortTimeFormat = "hh:mm:ss am/pm";  
  
    /* Place file pointer at end of file */  
    if (file_seek_end (file_handle) == FAILURE)  
    {  
        sprintf (buffer, "Could not seek to end of Log File '%s'.", system_log_file_name);  
        POS_MessageBox (NULL, buffer, "System Log File Error", MB_OK|MB_ICONEXCLAMATION);  
        return (FAILURE);  
    }  
}
```

```
/* File opened successfully - Append log entry */
if (file_write (file_handle, buffer, strlen(buffer)) == FAILURE)
{
    /* Display error message */
    sprintf (buffer, "Log: Error detected when writing log entry to file '%s'!!! [errno = %d, %s]",
            system_log_file_name,
            errno,
            _sys_errlist[errno]);

    POS_MessageBox (NULL, buffer, "System Log File Error", MB_OK|MB_ICONEXCLAMATION);
    file_close (file_handle);
    return (FAILURE);
}

/* System log entry appended - now append a <CR> */
if (file_write (file_handle, "\n", 1) == FAILURE)
{
    /* No - Display error message */
    sprintf (buffer, "Log: Error detected when writing <CR> to file '%s'!!! [errno = %d, %s]",
            system_log_file_name,
            errno,
            _sys_errlist[errno]);

    POS_MessageBox (NULL, buffer, "System Log File Error", MB_OK|MB_ICONEXCLAMATION);
    file_close (file_handle);
    return (FAILURE);
}

/* All is well, close the file */
file_close (file_handle);

return (SUCCESS);
}
```

```
//-----  
// Start Calculator  
//  
// This function spawns the calculator.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
void start_calculator (void)  
{  
    int result;  
    char buffer [MAX_STRING_SIZE];  
    char executable [MAX_STRING_SIZE];  
  
    AnsiString system_root;  
  
    /* Get System Root environment variable */  
    system_root = GetEnvironmentVariable("SystemRoot");  
  
    /* Build executable file name */  
    strcpy (executable, system_root.c_str());  
    strcat (executable, "\\System32\\calc.exe");  
  
    /* Start the Calculator */  
    result = spawnlp (P_NOWAIT, executable, executable, NULL);  
  
    /* Success? */  
    if (result == INVALID)  
    {  
        /* No - Tell User to start Windows Calculator manually */  
        POS_MessageBox (NULL,  
            "Windows Calculator not found... Please start manually.",  
            "Calculator",  
            MB_OK|MB_ICONEXCLAMATION);  
  
        /* Could not start Windows Calculator */  
        sprintf (buffer,  
            "Could not start 'calc.exe' process [errno = %d, %s].",  
            errno,  
            _sys_errlist[errno]);  
  
        Log_error (1, buffer);  
    }  
}
```

```
//-----  
// Start Windows Explorer  
//  
// This function spawns the Microsoft Windows Explorer.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
void start_windows_explorer (void)  
{  
    int result;  
    char buffer [MAX_STRING_SIZE];  
    char executable [MAX_STRING_SIZE];  
  
    AnsiString system_root;  
  
    /* Get System Root environment variable */  
    system_root = GetEnvironmentVariable("SystemRoot");  
  
    /* Build executable file name */  
    strcpy (executable, system_root.c_str());  
    strcat (executable, "\\explorer.exe");  
  
    /* Start Windows Explorer */  
    result = spawnlp (P_NOWAIT, executable, executable, NULL);  
  
    /* Success? */  
    if (result == INVALID)  
    {  
        /* No - Tell User to start Windows Explorer manually */  
        POS_MessageBox (NULL,  
            "Windows Explorer not found... Please start manually.",  
            "Windows Explorer",  
            MB_OK|MB_ICONEXCLAMATION);  
  
        /* Could not start Windows Explorer */  
        sprintf (buffer,  
            "Could not start 'explorer.exe' process [errno = %d, %s].",  
            errno,  
            _sys_errlist[errno]);  
  
        Log_error (1, buffer);  
    }  
}
```

```
//-----  
// Start Auto Shutdown  
//  
// This function starts the auto shutdown process.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
void start_auto_shutdown (void)  
{  
    int result;  
    char buffer [MAX_STRING_SIZE];  
  
    // !!! Only an Administrator or a user who is part of the Administrators Group can use tsshutdn !!!  
  
    /* Start the shutdown process */  
    result = spawnlp (P_NOWAIT, "tsshutdn.exe", "20", "/powerdown", NULL);  
  
    /* Was an error detected? */  
    if (result == INVALID)  
    {  
        /* Yes - Log error */  
        sprintf (buffer,  
                "Could not start the Auto Shutdown process [errno = %d, %s].",  
                errno,  
                _sys_errlist[errno]);  
  
        Log_error (1, buffer);  
        return;  
    }  
  
    /* Log Auto Shutdown process started */  
    Log_info ("Auto Shutdown process started...");  
}  
  
//-----  
// Play Cash Register Sound  
//  
// This function plays the cash register sound clip.  
//-----  
  
void play_cash_register_sound (void)  
{  
    char sound_file_name [MAX_STRING_SIZE];  
  
    /* Build path using system startup drive and directory */  
    fnmerge (sound_file_name,  
            system_startup_drive.c_str(),  
            system_startup_directory.c_str(),  
            "Cash Register",  
            ".wav");  
  
    try {  
        POS_Register_Form->MediaPlayer1->FileName = sound_file_name;  
        POS_Register_Form->MediaPlayer1->Open();  
        POS_Register_Form->MediaPlayer1->Play();  
    }  
  
    catch (...)  
    {  
        Status_display ("Sound not found...", TIMEOUT_5_SECONDS);  
    }  
}
```

```
//-----  
// Play Computer Says No Sound  
//  
// This function plays "The Computer Says No" sound clip.  
//-----  
  
void play_computer_says_no_sound (void)  
{  
char sound_file_name [MAX_STRING_SIZE];  
  
/* Are we signed on as a SUPERVISOR? */  
if (SUPERVISOR_override)  
{  
/* Yes - Build path using system startup drive and directory */  
fnmerge (sound_file_name,  
system_startup_drive.c_str(),  
system_startup_directory.c_str(),  
"The Computer Says No",  
".wav");  
  
try {  
POS_Register_Form->MediaPlayer1->FileName = sound_file_name;  
POS_Register_Form->MediaPlayer1->Open();  
POS_Register_Form->MediaPlayer1->Play();  
}  
  
catch (...)  
{  
Status_display ("Sound not found...", TIMEOUT_5_SECONDS);  
}  
}  
}  
  
//-----  
// Limit String Size  
//  
// This function shortens the string to the specified limit value. This is  
// used when printing long strings in short fields. If the string is too  
// long it is truncated with the last two string characters being displayed  
// as "...".  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int limit_string_size (char * text, int limit)  
{  
int text_length;  
  
/* Get string length */  
text_length = strlen (text);  
  
/* Determine if text exceeds limit */  
if (text_length > limit)  
{  
/* Abbreviate text string */  
text[limit - 2] = '.';  
text[limit - 1] = '.';  
text[limit] = 0x00;  
}  
  
/* Success */  
return (SUCCESS);  
}
```

```
//-----  
// Limit String Size No Dots  
//  
// This function shortens the string to the specified limit value. This is  
// used when printing long strings in short fields. If the string is too  
// long it is simply truncated.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int limit_string_size_no_dots (char * text, int limit)  
{  
    int text_length;  
  
    /* Get string length */  
    text_length = strlen (text);  
  
    /* Determine if text exceeds limit */  
    if (text_length > limit)  
    {  
        /* Abbreviate text string */  
        text[limit] = 0x00;  
    }  
  
    /* Success */  
    return (SUCCESS);  
}  
  
  
//-----  
// Move Cursor to Home Position  
//  
// This function moves the cursor to the top of the MemoBox.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int move_cursor_to_home_position (TMemo * MemoBox)  
{  
    AnsiString AnsiBuffer;  
  
    /* Abort if memobox does not exist */  
    if (MemoBox == NULL)  
        return (FAILURE);  
  
    /* Display top of report */  
    MemoBox->SelStart = 0;  
  
    /* Success */  
    return (SUCCESS);  
}
```

```
//-----  
// Set Timer  
//  
// This function sets the timer to the specified timeout value. Since the  
// timers are decremented by a separate thread, a locking mechanism is used  
// to ensure correct operation.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
void set_timer (UINT32 * timer, UINT32 timeout)  
{  
/* Obtain sole access to the timers */  
timer_access->Acquire();  
  
/* Initialize the timer */  
*timer = timeout;  
  
/* Release sole access to the timers */  
timer_access->Release();  
}  
  
//-----  
// Get Timer  
//  
// This function gets the timeout count left in the specified timer.  
// Since the timers are decremented by a separate thread, a locking mechanism  
// is used to ensure correct operation.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
UINT32 get_timer (UINT32 * timer)  
{  
UINT32 timer_value;  
  
/* Obtain sole access to the timers */  
timer_access->Acquire();  
  
/* Get the current timer value */  
timer_value = *timer;  
  
/* Release sole access to the timers */  
timer_access->Release();  
  
/* Return the timer value */  
return (timer_value);  
}
```

```
//-----  
// Insert Money Symbol  
//  
// This function takes the specified string and inserts a money symbol at the  
// beginning of the number (if space permits).  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int insert_money_symbol (char * buffer)  
{  
    char    money_symbol;  
    char *  pointer;  
  
    pointer = buffer;  
  
    while (*pointer != 0x00)  
    {  
        /* Test for the first numeric digit */  
        if (*pointer != ASCII_SPACE)  
        {  
            /* Step to the previous character */  
            pointer--;  
  
            /* Are we still within the buffer? */  
            if (pointer >= buffer)  
            {  
                /* Yes - Replace character with a money symbol */  
                money_symbol = POS_currency_symbol.c_str()[0];  
                *pointer = money_symbol;  
                return (SUCCESS);  
            }  
  
            /* No - Buffer string not large enough for money symbol */  
            return (FAILURE);  
        }  
  
        /* Step to next character */  
        pointer++;  
    }  
  
    /* No non-space characters in buffer */  
    return (FAILURE);  
}
```

```
//-----  
// Remove Money Symbol  
//  
// This function takes the specified string and removes the money symbol  
// (if it exists).  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
AnsiString remove_money_symbol (AnsiString money_string)  
{  
    char    input_buffer  [MAX_STRING_SIZE];  
    char    output_buffer [MAX_STRING_SIZE];  
    char    money_symbol;  
  
    AnsiString result_string;  
  
    /* Get the money symbol */  
    money_symbol = POS_currency_symbol.c_str()[0];  
  
    /* Get the money string without additional spaces */  
    strcpy (input_buffer, money_string.Trim().c_str());  
  
    /* Is a money symbol used? */  
    if (input_buffer[0] == money_symbol)  
    {  
        /* Yes - Set the output buffer to the money string without the money symbol */  
        strcpy (output_buffer, &money_string.c_str()[1]);  
    }  
    else  
    {  
        /* No - Set the output buffer to the full money string */  
        strcpy (output_buffer, &money_string.c_str()[0]);  
    }  
  
    /* Convert money string back to an AnsiString */  
    result_string = output_buffer;  
  
    /* No non-space characters in buffer */  
    return (result_string);  
}
```

```
//-----  
// Random Initialize  
//  
// This function random initializes the specified data structure.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int random_initialize (void * data_structure, long size)  
{  
    char * random_structure;  
    long index;  
    time_t random_seed;  
  
    /* Re-Seed the random number generator */  
    srand((unsigned) time(&random_seed));  
  
    /* Initialization */  
    random_structure = (char *) data_structure;  
  
    /* Randomize the entire data structure */  
    for (index = 0; index < size; index++)  
    {  
        /* Randomize each string in the structure */  
        *random_structure++ = (char) RandomRange (0x00, 0xFF);  
    }  
  
    /* Success */  
    return (SUCCESS);  
}
```

```

//-----
// Determine Days In Month
//
// This function determines the number of days in a given month [1..12].
//
// Returns: Number of days in the specified month.
//          FAILURE - Date is not valid.
//-----

int  determine_days_in_month (int  month, int  year)
{
int  days_in_month;
int  days_in_february;

/*****
/* Determine days in february */
*****/

/* Is this a leap year? */
if ((year % 4) == 0)
{
/* Yes - Is this a new century? */
if ((year % 100) == 0)
{
/* Yes - Is this century divisible by 400? */
if ((year % 400) == 0)
{
/* Yes - It is a true leap year */
days_in_february = 29;
}
else
{
/* No - It is not a true leap year */
days_in_february = 28;
}
}
else
{
/* Not a new century but still a true leap year */
days_in_february = 29;
}
}
else
{
/* Not a leap year */
days_in_february = 28;
}

/* Determine days in month */
switch (month)
{
case 1: days_in_month = 31; break; // Jan
case 2: days_in_month = days_in_february; break; // Feb
case 3: days_in_month = 31; break; // Mar
case 4: days_in_month = 30; break; // Apr
case 5: days_in_month = 31; break; // May
case 6: days_in_month = 30; break; // Jun
case 7: days_in_month = 31; break; // Jul
case 8: days_in_month = 31; break; // Aug
case 9: days_in_month = 30; break; // Sep
case 10: days_in_month = 31; break; // Oct
case 11: days_in_month = 30; break; // Nov
case 12: days_in_month = 31; break; // Dec
default: days_in_month = 0;
}

/* Return number of days in the month */
return (days_in_month);
}

```

```
//-----  
// Convert To Double  
//  
// This function converts the text in an EditBox into a double number. If  
// no text is present then the number is set to the default value.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int Convert_To_Double (TEdit * EditBox, double * number, double default_value)  
{  
    char    buffer [MAX_STRING_SIZE];  
    double  result;  
  
    AnsiString  text_number;  
  
    /* Convert the EditBox text to a double number */  
    try {  
        /* Remove leading and trailing spaces */  
        text_number = EditBox->Text.Trim();  
  
        /* Is this a null string? */  
        if (EditBox->Text.Length() == 0)  
        {  
            /* Yes - No text entered - set to default */  
            *number = default_value;  
            return (SUCCESS);  
        }  
  
        /* Is the first text character a currency symbol? */  
        if (text_number.c_str()[0] == POS_currency_symbol.c_str()[0])  
        {  
            /* Yes - Remove any leading currency symbol */  
            strcpy (buffer, &text_number.c_str()[1]);  
            text_number = buffer;  
        }  
  
        /* Try to convert text into a number */  
        result = text_number.ToDouble();  
        *number = result;  
        return (SUCCESS);  
    }  
  
    catch (...)  
    {  
        /* Invalid number - Inform the user */  
        return (FAILURE);  
    }  
}
```

```
//-----  
// Convert To Int  
//  
// This function converts the text in an EditBox into a integer number.  If  
// no text is present then the number is set to the default value.  
//  
// Returns:  SUCCESS or FAILURE  
//-----  
  
int  Convert_To_Int (TEdit * EditBox, double *  number, int  default_value)  
{  
char  buffer [MAX_STRING_SIZE];  
int   result;  
  
AnsiString  text_number;  
  
/* Convert the EditBox text to an integer number */  
try {  
    /* Remove leading and trailing spaces */  
    text_number = EditBox->Text.Trim();  
  
    /* Is this a null string? */  
    if (EditBox->Text.Length() == 0)  
    {  
        /* Yes - No text entered - set to default */  
        *number = default_value;  
        return (SUCCESS);  
    }  
  
    /* Is the first text character a currency symbol? */  
    if (text_number.c_str()[0] == POS_currency_symbol.c_str()[0])  
    {  
        /* Yes - Remove any leading currency symbol */  
        strcpy (buffer, &text_number.c_str()[1]);  
        text_number = buffer;  
    }  
  
    /* Try to convert text into a number */  
    result = text_number.ToInt();  
    *number = result;  
    return (SUCCESS);  
}  
  
catch (...)  
{  
    /* Invalid number - Inform the user */  
    return (FAILURE);  
}  
}
```

```
//-----  
// Get Version Numbers  
//  
// This function reads the version information table.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int get_version_numbers (char * file_name,  
                        char * version_numbers)  
{  
    char * version_table;  
    char * version_number_string;  
    DWORD version_table_size;  
    unsigned int parameter_length;  
    AnsiString version_search_string;  
    AnsiString file_version;  
  
    /* Initialization */  
    strcpy (version_numbers, "0.0.0.0");  
  
    /* Get the version table size */  
    version_table_size = GetFileVersionInfoSize(file_name, &version_table_size);  
  
    /* Does the version table exist? */  
    if (version_table_size == 0)  
    {  
        /* No - Exit */  
        return (FAILURE);  
    }  
  
    /* Allocate dynamic memory for the version table */  
    version_table = (char *) malloc(version_table_size);  
  
    /* Was the version table memory allocated okay? */  
    if (version_table == NULL)  
    {  
        /* No - Exit */  
        return (FAILURE);  
    }  
  
    /* Build version table search string */  
    version_search_string = "StringFileInfo\\040904E4\\";  
    version_search_string += "FileVersion";  
  
    /*Read the Version Table */  
    if (GetFileVersionInfo(file_name, 0, version_table_size, version_table) == 0)  
    {  
        /* Failed - Exit */  
        free (version_table);  
        return (FAILURE);  
    }  
  
    /* Obtain the version numbers */  
    if (VerQueryValue(version_table, version_search_string.c_str(), (void **)&version_number_string,  
&parameter_length))  
    {  
        /* Get the version number string */  
        strcpy (version_numbers, version_number_string);  
    }  
  
    /* Release version table memory */  
    free (version_table);  
  
    /* Success */  
    return (SUCCESS);  
}
```

```
//-----  
// Extract Quad Numbers  
//  
// This function extracts four integer numbers from a quad string such as  
// "1234.2.34341.2".  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int extract_quad_numbers (char * quad_number_string,  
                          int * number1,  
                          int * number2,  
                          int * number3,  
                          int * number4)  
{  
char * string_ptr;  
char * decimal_point_ptr;  
  
/*****  
/* First Number */  
*****/  
  
/* Initialize */  
string_ptr = quad_number_string;  
  
/* Find first decimal point */  
decimal_point_ptr = StrPos (string_ptr, ".");  
  
if (decimal_point_ptr == NULL)  
    return (FAILURE);  
  
/* Set decimal point to string terminator */  
*decimal_point_ptr = 0x00;  
  
/* Extract first number */  
*number1 = atoi (string_ptr);  
  
/*****  
/* Second Number */  
*****/  
  
/* Set string ptr to start of second number */  
string_ptr = ++decimal_point_ptr;  
  
/* Find next decimal point */  
decimal_point_ptr = StrPos (string_ptr, ".");  
  
if (decimal_point_ptr == NULL)  
    return (FAILURE);  
  
/* Set decimal point to string terminator */  
*decimal_point_ptr = 0x00;  
  
/* Extract second number */  
*number2 = atoi (string_ptr);
```

```
/* Third Number */
/* Set string ptr to start of third number */
string_ptr = ++decimal_point_ptr;

/* Find next decimal point */
decimal_point_ptr = StrPos (string_ptr, ".");

if (decimal_point_ptr == NULL)
    return (FAILURE);

/* Set decimal point to string terminator */
*decimal_point_ptr = 0x00;

/* Extract third number */
*number3 = atoi (string_ptr);

/* Fourth Number */
/* Set string ptr to start of fourth number */
string_ptr = ++decimal_point_ptr;

/* Extract fourth number */
*number4 = atoi (string_ptr);

/* Success */
return (SUCCESS);
}
```

```
//-----  
// Get Major Version Number  
//  
// This function reads the version information table and extracts the major  
// version number.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int get_major_version_number (char * file_name,  
                             int * major_version_number)  
{  
    int number1;  
    int number2;  
    int number3;  
    int number4;  
    char version_numbers [MAX_STRING_SIZE];  
  
    /* Initialization */  
    *major_version_number = 0;  
  
    /* Obtain version numbers */  
    if (get_version_numbers (file_name, version_numbers) == FAILURE)  
        return (FAILURE);  
  
    /* Extract the four independent version numbers */  
    if (extract_quad_numbers (version_numbers, &number1, &number2, &number3, &number4) == FAILURE)  
        return (FAILURE);  
  
    /* Extract major version number */  
    *major_version_number = number1;  
  
    /* Success */  
    return (SUCCESS);  
}
```

```
//-----  
// Get Minor Version Number  
//  
// This function reads the version information table and extracts the minor  
// version number.  
//  
// Returns: SUCCESS or FAILURE  
//-----  
  
int get_minor_version_number (char * file_name,  
                             int * minor_version_number)  
{  
    int number1;  
    int number2;  
    int number3;  
    int number4;  
    char version_numbers [MAX_STRING_SIZE];  
  
    /* Initialization */  
    *minor_version_number = 0;  
  
    /* Obtain version numbers */  
    if (get_version_numbers (file_name, version_numbers) == FAILURE)  
        return (FAILURE);  
  
    /* Extract the four independent version numbers */  
    if (extract_quad_numbers (version_numbers, &number1, &number2, &number3, &number4) == FAILURE)  
        return (FAILURE);  
  
    /* Extract minor version number */  
    *minor_version_number = number2;  
  
    /* Success */  
    return (SUCCESS);  
}
```

```

//-----
// Print MemoBox
//
// This function prints the specified MemoBox on to the printer.
//
// Returns: SUCCESS or FAILURE
//-----

int print_memobox (TPrinter * OUT_Printer, int orientation, TStrings * Document, char * Title)
{
    int index;
    int line_count;
    int max_pages;
    int page_count;
    int page_width;
    int page_height;
    int text_height;
    int max_document_lines;
    int max_document_pages;
    int max_printed_lines_per_page;
    int max_document_lines_per_page;
    int number_of_document_lines;
    int number_of_document_pages;
    char buffer [MAX_STRING_SIZE];
    struct date d;
    struct time t;

    /* Get number of lines in document */
    number_of_document_lines = Document->Count;

    /* Is the document empty? */
    if (number_of_document_lines == 0)
        return (FAILURE);

    /* Get current Date and Time Information */
    getdate(&d);
    gettime(&t);

    /* Determine maximum number of lines to the page */
    page_width = OUT_Printer->PageWidth - PRINTER_LEFT_MARGIN; // - PRINTER_RIGHT_MARGIN;
    page_height = OUT_Printer->PageHeight - PRINTER_TOP_MARGIN; // - PRINTER_BOTTOM_MARGIN;

    OUT_Printer->Canvas->Font->Name = "Courier";
    OUT_Printer->Canvas->Font->Pitch = fpFixed;
    OUT_Printer->Canvas->Font->Size = 10;

    text_height = OUT_Printer->Canvas->TextHeight(Document->Strings[0]);

    if (text_height == 0)
    {
        for (index = 0; index < number_of_document_lines; index++)
        {
            /* Get the text height of another line */
            text_height = OUT_Printer->Canvas->TextHeight(Document->Strings[index]);

            /* Is the text height > 0 ? */
            if (text_height > 0)
                break;
        }
    }

    /* Did we obtain a valid text height? */
    if (text_height == 0)
    {
        /* No - Cancel print */
        return (FAILURE);
    }
}

```

```

/* Is Portrait orientation required? */
if (orientation == PORTRAIT)
{
    /* Yes - Print page in Portrait Format */
    OUT_Printer->Orientation = poPortrait;

    /* Determine printing parameters */
    max_printed_lines_per_page = page_height / text_height;
    max_document_lines_per_page = max_printed_lines_per_page - 4;    //2 lines for header and
                                                                    //2 lines for footer

    number_of_document_pages = (number_of_document_lines / max_document_lines_per_page) + 1;
}
else
{
    /* No - Print page in Landscape Format */
    OUT_Printer->Orientation = poLandscape;

    /* Determine printing parameters */
    max_printed_lines_per_page = page_width / text_height;
    max_document_lines_per_page = max_printed_lines_per_page - 4;    //2 lines for header and
                                                                    //2 lines for footer

    number_of_document_pages = (number_of_document_lines / max_document_lines_per_page) + 1;
}

/* Initialization */
line_count = 0;
page_count = 1;

/* Start Printing */
OUT_Printer->Title = Title;
OUT_Printer->BeginDoc();

/* Output each line */
for (index = 0; index < number_of_document_lines; index++)
{
    /* Print next line of document */
    OUT_Printer->Canvas->TextOut (PRINTER_LEFT_MARGIN,
                                PRINTER_TOP_MARGIN + (line_count++ * text_height),
                                Document->Strings[index]);

    /* Print document and footer */
    if (line_count >= max_document_lines_per_page)
    {
        /* Print Blank Line */
        OUT_Printer->Canvas->TextOut (PRINTER_LEFT_MARGIN,
                                    PRINTER_TOP_MARGIN + (line_count++ * text_height),
                                    "");

        /* Print Footer */
        sprintf (buffer, "Date: %4d-%02d-%02d   Time: %02d-%02d-%02d   Page: %d / %d",
                d.da_year, d.da_mon, d.da_day, t.ti_hour, t.ti_min, t.ti_sec, page_count,
                number_of_document_pages);

        OUT_Printer->Canvas->TextOut (PRINTER_LEFT_MARGIN,
                                    PRINTER_TOP_MARGIN + (line_count * text_height),
                                    buffer);

        /* Reset line counter */
        line_count = 0;

        /* Increment page counter */
        page_count++;

        /* Still more to print? */
        if (index + 1 < number_of_document_lines)
        {
            /* Yes - Start a new page */
            OUT_Printer->NewPage();
        }
    }
}

```

```
    }
  }
}

/* Write blank lines until end of page */
for (index = line_count; index <= max_document_lines_per_page; index++)
{
  /* Print Blank Line */
  OUT_Printer->Canvas->TextOut (PRINTER_LEFT_MARGIN,
                                PRINTER_TOP_MARGIN + (line_count++ * text_height),
                                "");
}

/* Print Footer */
sprintf (buffer,
        "Date: %4d-%02d-%02d   Time: %02d-%02d-%02d   Page: %d / %d",
        d.da_year,
        d.da_mon,
        d.da_day,
        t.ti_hour,
        t.ti_min,
        t.ti_sec,
        page_count,
        number_of_document_pages);

OUT_Printer->Canvas->TextOut (PRINTER_LEFT_MARGIN,
                              PRINTER_TOP_MARGIN + (line_count * text_height),
                              buffer);

/* Stop Printing */
OUT_Printer->EndDoc();

/* Success */
return (SUCCESS);
}
```