Using the Casio ClassPad 300 Software Development Kit (CpSDK).

by
Anthony S. Pyzdrowski Ph.D.
Department of Mathematics and Computer Science
California University of Pennsylvania
250 University Avenue
California, PA 15419

Contents

Overview

Document Overview

This document is written to supplement the *SDK Programming Tutorial* and the *Programming Guide*. Examples of programming code are presented in order to help introduce the user to programming applications for the *Casio ClassPad 300*.

CpSDK Overview

The *Casio ClassPad Software Development Kit*, referred to here as CpSDK, is a collection of programs which are used to create applications for the *Casio ClassPad 300*. Once loaded, these applications offer new features to the *Casio ClassPad 300*.

CpSDK includes the *BloodShed Dev-C++ Integrated Development Environment (IDE)*, the *Casio ClassPad Add-In Installer*, a program debugger, Casio ClassPad header files and libraries, examples, and documentation.

What is an Add-In

An Add-In is a program written in C++ and compiled for the *Casio ClassPad 300*. Once loaded into the *Casio ClassPad 300* this program becomes a new feature of the *Casio ClassPad 300*. Examples of Add-In's include a spread sheet, an address book, a sketch pad, a scheduler, etc.

Getting Started

Creating a Project

An Add-In is created using a project.
You can create a new project in the CpSDK by selecting the following menu path:

FILE ➜ NEW ➜ PROJECT

Here you will find two default templates: the **ClassPad 300 Add-in** and the **Empty Project**. The ClassPad 300 Add-in template will create a project file with a main.cpp file containing the PegAppInitialize method. The Empty Project template will create a project file only. In this example, we will use the first template. To do this, select **ClassPad 300 Add-in**. Now, specify a project name in the **Name** field (use **example1** for now.) Select **OK**.

After selecting **OK**, the file dialogue window will appear. Navigate to the folder where you want the project and its files to be stored. If the default file name (which would be the project name with a .dev extension) is acceptable, select **SAVE**. You can create a new folder by using a right click and selecting **NEW ➜ FOLDER** in the file folder window, and entering a folder name.

The left window will now show the project, Example1, with all of its files. You may need to click the plus sign (+) to expand the file list. Clicking the minus sign (-) will collapse the file list. The right window is the editor for the selected file, which in this case will be main.cpp

If the edit window is closed, the desired file can be opened for edit by selecting it under the project tree of the left window.

See the CpSDK Programming Tutorial under HELP, SDK REFERENCE MANUAL, SDK PROGRAMMING TUTORIAL (PDF) for detailed instructions on the IDE.

Setting Up an Application

In the main.cpp file, include the "ClassPad.h" header file for function support of the ClassPad.

All Add-ins must contain a PegAppInitialize() method. This is the “main” method for ClassPad applications. Use this method to construct the main frame, windows, etc., which is covered later.

The form for PegAppInitialize() is:

```
void PegAppInitialize(PegPresentationManager *pPresentation)
{
}
```

Also include the following overloaded ExtensionGetLang method.

```
extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
	return "";
}
```

Frames

Add-ins are processed within frames. The primary frame object is called the MainFrame. The MainFrame manages applications; specifically it controls the active application and manages interaction between the applications. The MainFrame also updates the window if data associated with the window changes. The MainFrame contains four areas: menu, tool bar, status, and application.

The layout of the MainFrame from top to bottom is:

Menu	(Default frame menu, merged with the active application menu)
User Interface	(Tool bars, Buttons, etc.)
Application Area	(one or two application windows, or one application window and a virtual keypad)
Status Area	

The Constructor for the MainFrame

CPMainFrame(PegRect rect);

PegRect is an object which identifies the size of the frame rectangle. It is defined as:
PegRect rect = {MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT,
MAINFRAME_BOTTOM};

MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT, and MAINFRAME_BOTTOM are pre-defined values which identify the pixel coordinates of the viewable screen.

Windows

Module Windows are created with the object CPModuleWindow. The CPModuleWindow object contains methods that need to be overloaded to customize the User Interface and the application window.

The Constructor for CPModuleWindow

CPModuleWindow(PegRect rect, CPModuleWindow* invoking_window, CPDocument* doc, CPMainFrame* frame);

The first parameter is the PegRect Object occupied by the window. The second parameter is a pointer to the CPModuleWindow. The third parameter is a pointer to the CPDocument. The last parameter is a pointer to the CPMainFrame for the application.

Draw Method

The Draw() method of the CPModuleWindow is used to display information in the window. This method must be overloaded to add custom features. The form of the Draw method is:

```
void YourSubClass::Draw()
{
        BeginDraw();                    //Begins the draw of the frame
        Invalidate(mClient);            //Invalidate the current frame for re-drawing
                                        //mClient is a protected member of the base class
                                        // PegThing, and corresponds to the rectangle
                                        //occupied by the window
        DrawFrame();                    //Draws the CPMainFrame
        DrawChildren();                 //Draw a children of current frame

        //Add other objects to draw here.
        EndDraw();                      //Closes the draw of the frame
}
```

The draw method needs to be called whenever screen information changes and a re-draw is needed.

Error Message Handling

The ExtensionGetLang() method is used to display error messages in the correct language. This method must be overloaded to handle error messages. The minimal form of the ExtensionGetLang method is:

```
extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
	return "";
}
```

Code Examples

Displaying Text (Example1)

The following example creates a window class and prints “Example 1" in black 20 pixels to the right and 10 pixels down from the upper left corner of the frame.

PegPoint is used to store the location for the text. A PegPoint holds an *x* and *y* coordinate. PegColor is used to set the color to BLACK. DrawTextR is used to display the text. The first parameter in DrawTextR is a PegPoint object which specifies an *x*, *y* position relative to the upper left corner of he frame. The second parameter is the string to be displayed. The third parameter is the PegColor for the text. The last parameter is the Font for the text. DrawText can be used and positions the text in an absolute position instead of a relative position.

See *ClassPad 300 SDK Reference* for details on the method forms and parameter descriptions.

Example 1 main.cpp file:

```
#include "ClassPad.h"

class Example1Window: public CPModuleWindow   //Create the Example1Window class for the application
{
	public:
//	Example1Window constructor, inherit CPModuleWindow
	Example1Window(PegRect rect, CPMainFrame* mf):CPModuleWindow(rect,0,0,mf){}
	~Example1Window(){}                       // destructor

	void Draw()                               // overload the Draw method
	{
		BeginDraw();                      // begin of draw
		Invalidate(mClient);              // invalidate old display
		DrawFrame();                      // draw the frame
		PegPoint pp = {20,10};            // position to pixel column 20 and row 10
		PegColor col = BLACK;             // set the color black
```

```
                DrawTextR(pp,"Example 1",col,PegTextThing::GetBasicFont());
                EndDraw();                              // end of draw
        }
};              //End of Example1Window class

//The main method
void PegAppInitialize(PegPresentationManager *pPresentation)
{
        PegRect Rect;                                           // create a rectangle object for the MainFrame
        // set the rectangle to the usable size of the display
        Rect.Set(MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT,
                MAINFRAME_BOTTOM);
        CPMainFrame *mw = new CPMainFrame(Rect);        // create the MainFrame
        PegRect AppRect;                                        // create a rectangle for the Application Window
        AppRect = mw->FullAppRectangle();               // Size it to the MainFrame
        Example1Window *swin = new Example1Window(AppRect,mw);      // create the Application Window
        mw -> SetTopWindow(swin);                       // load the application window into the MainFrame
        mw -> SetMainWindow(swin);                      // set a main window  for this module
        pPresentation->Add(mw);                         // add the MainFrame to the Presentation Manager
};

//error message handling
extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
        return "";
}
```

Saving the Files

Save the program by selecting the following menu path:

File ➜ Save All

Compile the Project

Compile the program by selecting:

Execute ➜ Rebuild All (Rebuild All compiles all the files in the project)

When finished (assuming you had no errors) select **Close** on the "Compile Progress" window.

Running the Application with the Debugger

The Application can be run in the IDE via the Debugger by selecting the following menu path:

Debug ➜ Debug

A graphic of the *Casio ClassPad 300* will appear on the display, as shown in Figure 1. This Example 1 program simply displays the text "Example 1" on the *Casio ClassPad 300* display. When you are through viewing the application, point to and click the "on/off" button icon on the *Casio ClassPad 300* image. The Debugger can also be terminated by selecting the "Stop

Execution" icon in the Debug toolbar near the bottom of the IDE display. See the *SDK Programming Tutorial* for detailed instructions on the IDE.

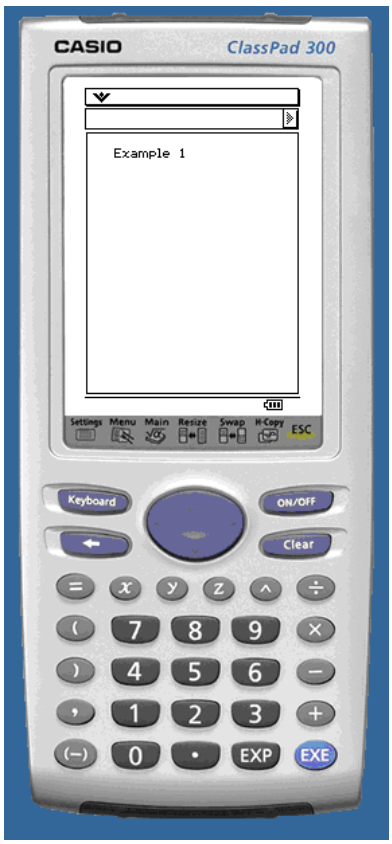

Figure 1

Transferring the Application to the Casio ClassPad 300

The Application can be loaded into the *Casio ClassPad 300* and used as an Add-In Application by following there steps.

First:

1. Plug the communication cable into the side of the ClassPad.
2. Plug the USB" end of the communication cable into a USB port of the computer.
3. The ClassPad will turn on and enter the Link Setup mode with the "Status" window showing Standby.
4. Press the Clear button on the ClasPad keyboard.
5. Select OK for the Terminate on the ClassPad (If this is the first time you are connecting the ClassPad to the computer, the "Found New Hardware" window will open on the computer. Wait for the ClassPad driver to finish installation.)

Second:

On the ClassPad, select:
Link ➜ Install ➜ Add-In ➜ OK
This menu can be found from the Communication icon on the main screen.

Third:

On the Computer, install the Application Example by selecting:
Tools ➜ Add-In Installer

From the Add-In Installer Window, select:
Add-In ➜ Application

Fourth:

1. Navigate to the outputdir directory within the Example1 directory.
2. Select the Example1.cpa file.
   The Application Example1 will transfer from the computer to the ClassPad. When finished the ClassPad will have a Complete message in the status window and the computer will have a Closing Connection to the ClassPad message in the Closing Connection window.
3. Select OK on both devices.
4. Close the "Add-In Installer" window.

Fifth:

On the ClassPad, select
Menu

The Application Example1 icon will be on the bottom of the list.

Select the Example1 icon.

You will see the MainFrame for Example1 with the text “Example 1" displayed.

To remove an Application from the ClassPad, select:
Settings
Delete Application
Select the Application icon
Select **OK**

See the *SDK Programming Tutorial* for detailed instructions on the Add-In Installer.

Input/Output (Example2)

Example 2 modifies the Example1Window class to incorporate changeable output prompts (PegPrompt), input string fields (CPPegString), a button (PegTextButton) and the Status bar (PegStatusBar). The definitions of the class methods are moved outside the class. An init method is added to the constructor to handle initialization of the object. The DrawChildren method is now included in the Draw method. A Message method is added to handle the button event.

Displaying with the PegPrompt

This example displays a changeable message by using the PegPrompt object. Here, the first parameter for the PegPrompt specifies the starting position from the left. The second parameter specifies the starting position from the top. The third parameter specifies the width in pixels. The fourth parameter is the initial text message. The fifth parameter is the message ID if used to trigger an event when the field is clicked. The sixth parameter specifies the style. See Peg Style Flags in the *ClassPad 300 SDK Reference* for details. This example uses no frame border -- FF_NONE, right justification -- TJ_RIGHT, changeable -- TT_COPY, and a transparent appearance -- AF_TRANSPARENT. The last parameter specifies the font. The contents of the PegPrompt can be set by using the DataSet method. The DataSet method accepts a PEGCHAR*.

The constructor for PegPrompt

PegPrompt (SIGNED iLeft, SIGNED iTop, SIGNED iWidth, const PEGCHAR *Text, WORD wId=0, WORD wStyle=FF_NONE|TJ_LEFT|AF_TRANSPARENT, PegFont *pFont=NULL)

Input with the CPPegString

The input field is constructed by a CPPegString object. In this example, the first parameter for the CPPegString specifies the starting position from the left. The second parameter specifies the starting position from the top. The third parameter specifies the width in pixels. The fourth parameter is the initial text message. The fifth parameter is the message ID if used to trigger an event. The sixth parameter specifies the style. The seventh parameter specifies a maximum number of bytes for the string. The data from the CPPegString can be read by using the DataGet method. The data used in this example is a CPString, where the return data needs to be cast from a PEGCHAR* to the desired CPString. The contents of the CPPegString can be set by using the DataSet method. The DataSet method accepts a PEGCHAR*.

The constructor for CPPegString

CPPegString (SIGNED iLeft, SIGNED iTop, SIGNED iWidth, const PEGCHAR *Text=NULL, WORD wId=0, WORD wStyle=FF_RECESSED|AF_ENABLED|EF_EDIT, SIGNED iLen=DEFAULT_PEG_STRING_BYTE_LIMIT)

Selection with a PegTextButton

The button is constructed by a PegTextButton object. A PegRect is used to specify the position for the button. The PegRect will set the left, right, top, and bottom coordinates for the button. By using the TopAppRectangle method from GetMainFrame, the space remaining in the MainFrame is determined. The button will be moved down and centered in this space. In this example, the first parameter for the PegTextButton is a rectangle which specifies the left, top, right, and bottom of the button. The second parameter specifies the text in the button. The third parameter is the message ID which is used when the button generates an event. The last parameter specifies the style. See Peg Style Flags in the *ClassPad 300 SDK Reference* for details. This example enables the viewing of the button -- AF_ENABLED and centers the text in the button -- TJ_CENTER.

The constructor for PegTextButton

PegTextButton (PegRect &rect, const PEGCHAR *text, WORD wId=0, WORD wStyle=AF_ENABLED|TJ_LEFT)

Handling the event with Message()

The overloaded Message method is used to process the event generated by the button. The Message method receives a PegMessage formed by the ID of the generating object and the type of action that triggered the event. In this example, the button has a message ID of CONVERT_ID and the action is PSF_CLICKED. See Peg Signals in the *ClassPad 300 SDK Reference* for details.

Displaying Messages on the Status Bar

The Status Bar is used to display a message in this example. Use the GetStatusBar to initialize the PegStatusBar object with the application's window status bar. The contents of the status bar can be changed by using the SetTextField method. The first parameter is the ID set to 1 and the second parameter is the message as a PEGCHAR*.

Data is entered into the input box. When the button is selected the text moves to the output field and the input box is cleared. The Status Bar will display "Example 2" or an error message for no data input.

Example 2 main.cpp File:

```
#include "ClassPad.h"
// ID code to recognize the button
#define CONVERT_ID 1
//Create the Example2Window class for the application
class Example2Window: public CPModuleWindow
{
//The global PegPrompt and CPPegString objects
        protected:
                PegPrompt* outputPrompt;
                CPPegString* ioString;

//Public methods of the Example2Window class
        public:
                Example2Window(PegRect rect, CPMainFrame* frame);       //constructor module
                ~Example2Window(){}                                     //destructor
                int init();                                             // initialization module
                void Draw();                                            // overload Draw
                SIGNED Message(const PegMessage &Mesg);                 // event handler
                void Process();                                         // method to do the work
};

// constructor
Example2Window::Example2Window(PegRect rect, CPMainFrame* mf):CPModuleWindow(rect,0,0,mf)
{
        init();                                 // call init
}

// initialization method
int Example2Window::init()
{
//formatting for input/output strings
        // no boarder, left justify, transparent, changable
        const WORD wStyle = FF_NONE|TJ_LEFT|AF_TRANSPARENT| TT_COPY;
        // set left margin to 10
        const SIGNED margin = 10;
        // calculate the maximum width that is available.  Get the width from the frame and subtract the margins
        const SIGNED width=mClient.Width()-2*margin;
// create an output prompt positioned at 10,10, with a maximum width, initialized to "Example 2"
        outputPrompt = new PegPrompt(margin,10,width,"Example 2",0,wStyle);
// create an input string positioned at 10, 20 with a maximum width, initialized to ""
```

```
        ioString = new CPPegString(margin, 20, width, "");
// get a rectangle set to the size of the remaining main frame (after ioString)
        PegRect rr = GetMainFrame()->TopAppRectangle();
        rr.wTop += 10;                          //move down 10
        rr.wBottom = rr.wTop+15;                //set bottom 15 after top
        rr.wLeft = (rr.wRight-rr.wLeft)/2-25;   //set left 25 short of center
        rr.wRight = rr.wLeft+50;                //set right 50 from left
// create a text button positioned 10 pixels after ioString and centered with a width of 50 and a height of 15
        PegTextButton* b1 = new PegTextButton(rr, "Move It", CONVERT_ID, AF_ENABLED|TJ_CENTER);
        AddR(b1);                               //add the button to the frame
        AddR(outputPrompt);                     // add the output prompt to the frame
        AddR(ioString);                         // add the input string to the frame
        SetDefaultFocus(ioString);              // set the focus to the ioString
        return 0;
}

void Example2Window::Draw()
{
                BeginDraw();                    // begin of draw
                Invalidate(mClient);            // invalidate old display
                DrawFrame();                    // draw the frame
                DrawChildren();                 // draw the chieldren
                EndDraw();                      // end of draw
}

// decode the triggering signal
SIGNED Example2Window::Message(const PegMessage &Mesg)
{
        switch(Mesg.wType)                                      // determine the ID code of the event
        {
                case SIGNAL(CONVERT_ID, PSF_CLICKED):           // the Button was CLICKED
                        Process();                              // perform the process
                        break;
                default:
                        return CPModuleWindow::Message(Mesg);
        }
        return 0;
}

void Example2Window::Process()
{
        CPString        tempString;                             // a temp string
        CPString        StatusString;                           // a string for the status bar
        PegStatusBar* bar = GetStatusBar();                     // make a Status Bar
        tempString = (CPString)ioString->DataGet();             // read the ioString
        if(tempString == "")                                    //check if nothing was entered
                StatusString = "Type something first!!";        //create a message for the Status Bar
        else
                StatusString = "Example 2";                     //put “Example 2" in the string
        bar->SetTextField(1,StatusString);                      //display StatusString on the Status Bar
        outputPrompt->DataSet(tempString);                      // write the string to the outputPrompt
        ioString->DataSet("");                                  // clear the ioString
        Draw();                                                 // refresh the display
}

void PegAppInitialize(PegPresentationManager *pPresentation)
```

```
{
        PegRect Rect;                                   // create a rectangle object for the MainFrame
        // set the rectangle to the usable size of the display
        Rect.Set(MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT,
                MAINFRAME_BOTTOM);
        CPMainFrame *mw = new CPMainFrame(Rect);        // create the MainFrame

        PegRect AppRect;                                // create a rectangle for the Application Window
        AppRect = mw->FullAppRectangle();               // size it to the MainFrame
        Example2Window *swin = new Example2Window(AppRect,mw);      // create the Application Window
        mw -> SetTopWindow(swin);                       // load the application window into the MainFrame
        mw -> SetMainWindow(swin);                      // set a main window  for this module
        pPresentation->Add(mw);                         // add the MainFrame to the Presentation Manager
}

//error message handling
extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
        return "";
}
```

Compile and run the program. Initially the Status Bar and the input field are blank and the output field displays “Example 2" (see Figure 2). Type something into the input field (see Figure 3). Click the “Move it” button and see that the text moves to the output field, “Example 2" is now displayed in the Status Bar, and the input field is blank (see Figure 4).

Dialog Windows and the Real Data Type, The Temperature Converter (Example 3)

A Temperature Conversion is used in Example 3. This example places the buttons on the Tool Bar, makes use of the pop-up dialog window for messages by overloading the Dialog method, and introduces the real data type for the *Casio ClassPad 300* -- the OBCD. A 45 by 28 pixel monochrome bitmap can be included in the project file list which will be used as the application icon on the *Casio ClassPad 300*.

Using the ToolBar

The Tool Bar is created by using an AddUI method and placing the buttons, as created in example 3, on the Tool Bar with the m_ui->AddToolbarButton method. See the *ClassPad 300 SDK Reference* for more information.

Pop-up Messages

As previously stated, messages are displayed in a pop-up window by overloading the Dialog method. In this example, the Dialog method accepts a title string and a message string. Title, Menus, Toolbar, Statusbar, and selected default buttons are also supported in the Dialog object. The PegMessageWindow object creates the window for the Dialog method. In this example, a rectangle describing the position of the four sides of the window is the first parameter. The second parameter is the title string. The message string is the third parameter, the fourth parameter specifies default buttons that can be included in the window. Here only the OK button

is used. The remaining three parameters are set to 0. See the *ClassPad 300 SDK Reference* for more information.

The constructor for PegMessageWindow

PegMessageWindow (const PegRect &Rect, const PEGCHAR *Title, const PEGCHAR *Message=NULL, WORD wStyle=MW_OK|FF_RAISED, WORD wStyle2=NULL, PegBitmap *pIcon=NULL, PegThing *Owner=NULL)

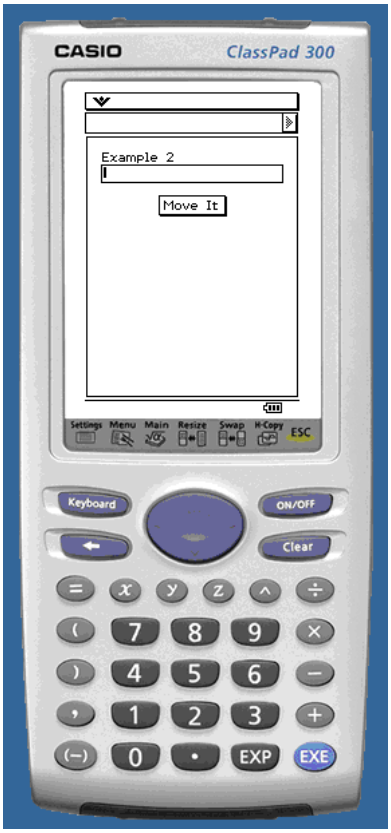

Figure 2

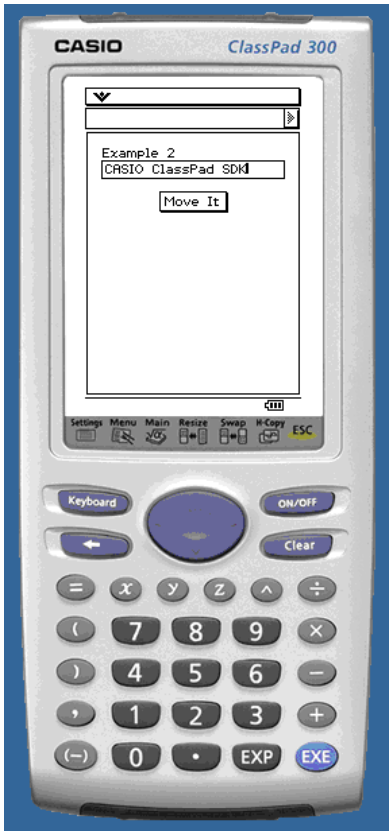

Figure 3

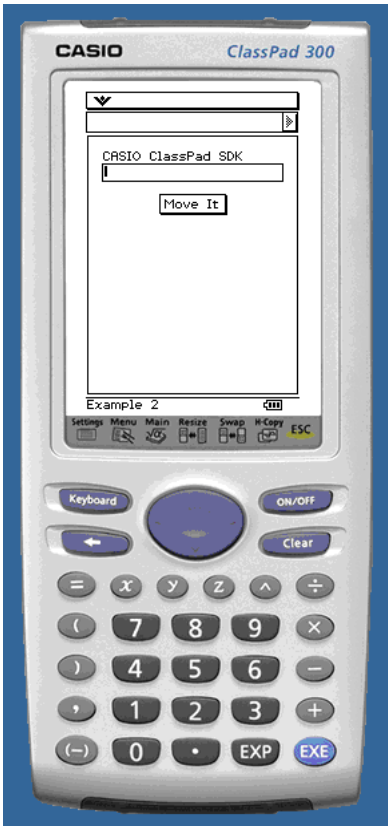

Figure 4

Real Numbers, the OBCD

The *Casio ClassPad 300* uses a data structure of Binary Coded Decimal digits to represent the two real data types. The OBCD represents one real number using a four bit flag, 15 BCD digits, two reserved bytes, a four bit sign flag for the exponent, and three BCD digit exponent (see Figure 5). The number is stored in scientific notation. Positive numbers have a base exponent of 1000 and negative numbers have a base exponent of 6000. The 1 and 6 are in the sign field of the exponent. Numbers which require a left shift in the decimal point, to put the number into scientific notation form, add the shift to the base exponent. Numbers which require a right shift in the decimal point, to put the number into scientific notation form, subtract the shift from the base exponent. A positive 101.5 would be written as 1.015×10^{2}. The exponent would be 1000 + 2 or 1002. A negative 101.5 would be written as $-1 \times 1.015 \times 10^{2}$. The exponent would be 6000 + 2 or 6002. A positive 0.001015 would be written as 1.015×10^{-3}. The exponent would

be 1000 - 3 or 0997. A negative 0.001015 would be written as -1 x 1.015 x 10^{-3}. The exponent would be 6000 - 3 or 5997.

The flag field represents one of six conditions. Zero indicates a number is contained in the structure. Four indicates infinity is contained in the structure, positive and negative is identified by the exponent sign. Eight indicates the number is undefined. Nine indicates true. Ten (0xA) indicates false. Fifteen (0xF) indicates an error occurred. The exponent can be decoded to identify the specific error, see the *Programmers Guide* for details.

This structure can represent 15 digit numbers with an exponent from -999 to +999. The structure can also represent - infinity, + infinity, true, false, and various errors. The CBCD contains two OBCDs to represent a complex number. The *Programmers Guide* is a good source of information on the OBCD data type. The Casio ClassPad 300 DOES NOT SUPPORT any of the C real data types; do not use them.

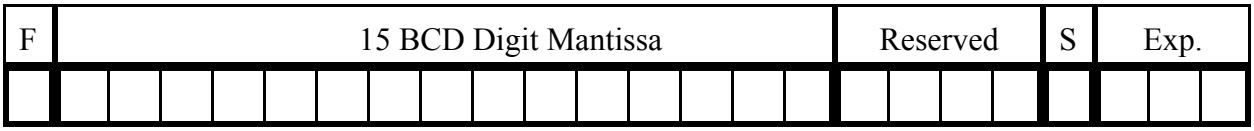

Figure 5

There are several functions in the ClassPad development system to support the use of the OBCD data. These functions are identified with Cal_functionname_OBC. They can be found in the *ClassPad 300 SDK Reference*.

The following example shows how to create an OBCD constant by specifying the nibbles of the number and the nibbles of the exponent. All numbers are stored in scientific notation (shifted to have one integer digit). The sign nibble is 0. The value zero has all digits 0 and the four exponent nibbles 0.

For example:
0 is written as
const OBCD OBCD0 = {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x0000}};
1 is written as 1.00000000000000 * 10 ^ 0
const OBCD OBCD0 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1000}};
-1 is written as 1.00000000000000 * -10 ^ 0
const OBCD OBCD0 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x6000}};
100 is written as 1.00000000000000 * 10 ^ 2
const OBCD OBCD0 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1002}};
-100 is written as 1.00000000000000 * -10 ^ 2
const OBCD OBCD0 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x6002}};
0.001 is written as 1.00000000000000 * 10 ^ 3
const OBCD OBCD0 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x0997}};
-0.001 is written as 1.00000000000000 * -10 ^ -3
const OBCD OBCD0 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x5997}};

CPExpression() can be used to convert a string into an OBCD.

This example displays two input/output fields with labels, one for the temperature in Fahrenheit and one for the temperature in Centigrade. Two buttons are placed on the Tool Bar, are to perform the temperature conversion and one to clear the Fahrenheit and Centigrade fields. Data is entered into one of the fields and the other temperature is calculated and displayed in the empty field when the Convert button is selected. The Clear button clears both fields so another conversion can begin. When an error occurs, a pop-up dialog window appears to display the message. The pop-up window is closed with a close button located on the pop-up window.

Example 3 main.cpp File:

```
#include "ClassPad.h"
// #define 9/5
const OBCD OBCD95 = {{{0x01, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1000}};
// #define 32
const OBCD OBCD32 = {{{0x03, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1001}};

// ID code for button
#define CONVERT_ID 1
#define CLEAR_ID 2

class TempConvertWindow: public CPModuleWindow
{
	protected:
		PegPrompt* Fahrenheit;				// Label for Fahrenheit
		PegPrompt* Centigrade;				// Label for Centigrade
		CPPegString* fString;				// I/O field for Fahrenheit
		CPPegString* cString;				// I/O field for Centigrade
	public:
		TempConvertWindow(PegRect rect, CPMainFrame* frame);	// Constructor
		~TempConvertWindow(){}				// Destructor
		int init();					// Initialization method
		void Draw();					// Draw method
		void AddUI();					// Builds the Toolbar
		SIGNED Message(const PegMessage &Mesg);		// Handles the events
		void Process();					// performs the conversion
		void Clear();					// Clears the fields
		WORD Dialog(char*, char*);			// Displays messages
};

// The main method
void PegAppInitialize(PegPresentationManager *pPresentation)
{
	PegRect Rect;					// create a rectangle object for the MainFrame
	// set the rectangle to the usable size of the display
	Rect.Set(MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT,
		MAINFRAME_BOTTOM);
	CPMainFrame *mw = new CPMainFrame(Rect);	// create the MainFrame
	PegRect AppRect;				// create a rectangle for the Application Window
	AppRect = mw->FullAppRectangle();		// size it to the MainFrame
	TempConvertWindow *swin = new TempConvertWindow(AppRect,mw); // create the Application
Window
	mw -> SetTopWindow(swin);			// load the application window into the MainFrame
	mw -> SetMainWindow(swin);			// set a main window  for this module
	pPresentation->Add(mw);				// add the MainFrame to the Presentation Manager
```

```
}

extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
	return "";
}

TempConvertWindow::TempConvertWindow(PegRect rect, CPMainFrame* mf):CPModuleWindow(rect,0,0,mf)
{
	init();
}

int TempConvertWindow::init()
{
//formatting for input/output strings
	const SIGNED margin = 10;
	const SIGNED width=mClient.Width()-2*margin;
// create a Fahrenheit label positioned at 10,10
	Fahrenheit = new PegPrompt(margin, 10, width, "Fahrenheit");
// create a Fahrenheit field positioned at 10,20
	fString = new CPPegString(margin,20,width,"");
// create a Centigrade label positioned at 10,40
	Centigrade = new PegPrompt(margin, 40, width, "Centigrade");
// create a Centigrade field positioned at 10, 50
	cString = new CPPegString(margin, 50, width, "");
// add the Fahrenheit label & field to the frame
	AddR(Fahrenheit);
	AddR(fString);
// add the iCentigrade label & field to the frame
	AddR(Centigrade);
	AddR(cString);
// set the focus to the ioString
	SetDefaultFocus(fString);
	return 0;
}

void TempConvertWindow::AddUI()
{
// create a text button positioned in the first location of the toolbar
	PegTextButton* b1 = new PegTextButton(1,1, "Convert", CONVERT_ID, AF_ENABLED|TT_COPY);
//Add Button b1 to the Tool Bar
	m_ui->AddToolbarButton(b1);
// create a text button positioned in the first location of the toolbar
	PegTextButton* b2 = new PegTextButton(2,1, "Clear", CLEAR_ID, AF_ENABLED|TT_COPY);
//Add Button b2 to the Tool Bar
	m_ui->AddToolbarButton(b2);
}

void TempConvertWindow::Draw()
{
		BeginDraw();
		Invalidate(mClient);
		DrawFrame();
		DrawChildren();
		EndDraw();
}
```

```
// decode the triggering signal
SIGNED TempConvertWindow::Message(const PegMessage &Mesg)
{
	switch(Mesg.wType)
	{
		case SIGNAL(CONVERT_ID, PSF_CLICKED):	//it was the convert button
			Process();
			break;
		case SIGNAL(CLEAR_ID, PSF_CLICKED):	//it was the clear button
			Clear();
			break;
		default:
			return CPModuleWindow::Message(Mesg);
	}
	return 0;
}

void TempConvertWindow::Clear()
{
//clear the fields
	fString->DataSet("");
	cString->DataSet("");
// refresh the display
	Draw();
}

void TempConvertWindow::Process()
{
	OBCD T;							// temp OBCD
	CPString	ftempString;				// create two temp strings
	CPString	ctempString;
	ftempString = (CPString)fString->DataGet();		// read the fString
	ctempString = (CPString)cString->DataGet();		// read the cString
	if(ftempString != "" && ctempString != "")		//check if both contain data
		Dialog("Error","One must be empty   ");
	else
	{
		if(ftempString == "" && ctempString == "")	//check if both are empty
		{
		   Dialog("Error","Must have a Temp  ");
		}
		else
		{
			if(ftempString != "")			//convert to Centigrade
			{
				T = CPExpression(ftempString);	// convert the string into an OBCD
				T = (T - OBCD32) / OBCD95;	// perform the calculation
				if((T.obcd1.mantissa[0]&0xf0)!= 0x00)	//check for error
				{
					Dialog("Error","Invalid Temperature  ");
				}
				else
				{
					ctempString = CPString(T,0);	//convert the OBCD to a string
					cString->DataSet(ctempString);	// write the answer to the screen
				}
```

```
                        }
                        else            //convert to Fahrenheit
                        {
                                T = CPExpression(ctempString);          // convert the string into an OBCD
                                T = T * OBCD95 + OBCD32;                // perform the calculation
                                if((T.obcd1.mantissa[0]&0xf0)!= 0x00)   //check for error
                                {
                                        Dialog("Error","Invalid Temperature  ");
                                }
                                else
                                {
                                        ftempString = CPString(T,0);    //convert the OBCD to a string
                                        fString->DataSet(ftempString);  // write the answer to the screen
                                }
                        }
                }
        }
        Draw();                                                         // refresh the display
}

WORD TempConvertWindow::Dialog(char* title, char* message)
{
        PegRect rr = GetMainFrame()->TopAppRectangle();         //get the size of the main application
        rr.wBottom -= 30;                                       //make this window 30 from the bottom
        rr.wRight -= 10;                                        //10 from the right
        rr.wTop += 10;                                          //10 down
        rr.wLeft += 10;                                         //and 10 from the left
//build the window with the title and message, include the OK button
        PegMessageWindow *msg = new PegMessageWindow(rr, title, message, MW_OK, 0,0,0);
        return msg->Execute();                                  //display the message window
}
```

Figure 6 shows the screen of the Temp Conversion Example. Enter a temperature in either field and select the convert button. The converted temperature will be displayed in the appropriate field. Select the clear button to clear the fields. When no data is entered or when both fields contain data, the error message dialog window will be displayed, as shown in Figure 7.

Figure 6

Figure 7

Menus, The Measurement Converter (Example4)

Example 4 further uses the OBCD data type, makes use of several OBCD operations, and introduces a two level menu.

Creating Menus

The Menu is created by using the GetMenuDescriptionML method and several PegMenuDescriptionML arrays. Each menu item is described with an entry in a PegMenuDescriptionML, with each entry in a PegMenuDescriptionML having five parameters. The first parameter is the name of the menu item. The second parameter is the ID message, which is used to look up the appropriate message if the first parameter is NULL. Use CMN_NO_ID when you specify the first parameter. The third parameter is the message ID which is used when the menu item generates an event. The fourth parameter specifies the style. See Peg Style Flags in the *ClassPad 300 SDK Reference* for details. The last parameter is a PegMenuDescriptionML pointer. This parameter points to the next level of menu description for this item position. If a next level of menu is used, do not have this menu item generate an event. Typically, it is best to generate the events at the last level of menus. When an item is a terminal item, use NULL for the last parameter.

OBCD Operations

In addition to the Cal_functionname_OBC functions, the OBCDmath.h header file contains the common operators for the OBCD data types. The relational, arithmetic, trigonometric, and miscellaneous operators are available to process OBCD data. Integers can be cast to the OBCD type with the "cast_to_OBCD" function. Likewise, OBCD data can be cast to integers with the "cast_to_int" function. The relational, arithmetic, trigonometric, and miscellaneous operators are overloaded to process OBCD data with integers and integers with OBCD data. See the *ClassPad 300 SDK Reference* for more information.

Example 4 accepts a real number and converts the value to a different unit of measure. The length conversions are made between inches, millimeters, and centimeters. The weight conversions are between ounces, grams, and kilograms.

Example 4 main.cpp File:

```
#include "ClassPad.h"
//Start menu ID's at 2200
#define IDC_CONVERT_ID_BASE 2200

// #define InchToMM 25.4000508
const OBCD InchToMM = {{{0x02, 0x54, 0x00, 0x05, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1001}};
// #define OzToG 28.349527
const OBCD OzToG = {{{0x02, 0x83, 0x49, 0x52, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1001}};
// #define 10
const OBCD OBCD10 = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1001}};
// #define 1000
const OBCD OBCD1K = {{{0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, 0x1003}};

//menu ID's
enum IntIDs
{
        INCH_MM_ID = IDC_CONVERT_ID_BASE,
        INCH_CM_ID,
        MM_INCH_ID,
        MM_CM_ID,
        CM_INCH_ID,
        CM_MM_ID,
        OZ_G_ID,
        OZ_KG_ID,
        G_OZ_ID,
        G_KG_ID,
        KG_OZ_ID,
        KG_G_ID
};

// This is the CM length sub menu
PegMenuDescriptionML LengthMenuCMTo[] = {
        { "mm", CMN_NO_ID, CM_MM_ID, AF_ENABLED, NULL },
        { "inch", CMN_NO_ID, CM_INCH_ID,AF_ENABLED, NULL},
        { "", CMN_NO_ID, 0,0, NULL}
};
```

```
// This is the MM length sub menu
PegMenuDescriptionML LengthMenuMMTo[] = {
        { "cm", CMN_NO_ID, MM_CM_ID, AF_ENABLED, NULL },
        { "inch", CMN_NO_ID, MM_INCH_ID,AF_ENABLED, NULL},
        { "", CMN_NO_ID, 0,0, NULL}
};

// This is the Inch length sub menu
PegMenuDescriptionML LengthMenuInchTo[] = {
        { "cm", CMN_NO_ID, INCH_CM_ID, AF_ENABLED, NULL },
        { "mm", CMN_NO_ID, INCH_MM_ID,AF_ENABLED, NULL},
        { "", CMN_NO_ID, 0,0, NULL}
};

// This is the length menu
PegMenuDescriptionML LengthMenu[] = {
        { "cm", CMN_NO_ID, 0, AF_ENABLED, LengthMenuCMTo},
        { "mm", CMN_NO_ID, 0,AF_ENABLED, LengthMenuMMTo},
        { "inch", CMN_NO_ID, 0,AF_ENABLED, LengthMenuInchTo},
        { "", CMN_NO_ID, 0,0, NULL}
};

// This is the kilogram sub menu
PegMenuDescriptionML WeightMenuKGTo[] = {
        { "g", CMN_NO_ID, KG_G_ID, AF_ENABLED, NULL },
        { "oz", CMN_NO_ID, KG_OZ_ID, AF_ENABLED, NULL },
        { "", CMN_NO_ID, 0, 0, NULL}
};

// This is the gram sub menu
PegMenuDescriptionML WeightMenuGTo[] = {
        { "kg", CMN_NO_ID, G_KG_ID, AF_ENABLED, NULL },
        { "oz", CMN_NO_ID, G_OZ_ID, AF_ENABLED, NULL },
        { "", CMN_NO_ID, 0, 0, NULL}
};

// This is the ounce sub menu
PegMenuDescriptionML WeightMenuOZTo[] = {
        { "kg", CMN_NO_ID, OZ_KG_ID, AF_ENABLED, NULL },
        { "g", CMN_NO_ID, OZ_G_ID, AF_ENABLED, NULL },
        { "", CMN_NO_ID, 0, 0, NULL}
};

// This is the weight menu
PegMenuDescriptionML WeightMenu[] = {
        { "kg", CMN_NO_ID, 0, AF_ENABLED, WeightMenuKGTo },
        { "g", CMN_NO_ID, 0, AF_ENABLED, WeightMenuGTo },
        { "oz", CMN_NO_ID, 0,AF_ENABLED, WeightMenuOZTo},
        { "", CMN_NO_ID, 0, 0, NULL}
};

// This is the main menu
PegMenuDescriptionML MainMenu[] = {
        { "Weight", CMN_NO_ID, 0, AF_ENABLED, WeightMenu },
        { "Length", CMN_NO_ID, 0,AF_ENABLED, LengthMenu},
        { "", CMN_NO_ID, 0, 0, NULL}
```

```
};

//The Example4 window class
class Example4Window: public CPModuleWindow
{
        protected:
                PegPrompt* outputFrom;                          //the from value output field
        PegPrompt* outputTo;                                    //the to value output field
                CPPegString* inputString;                       //the input string
        public:
                Example4Window(PegRect rect, CPMainFrame* frame);
                ~Example4Window(){}
                int init();
                void Draw();
                SIGNED Message(const PegMessage &Mesg);
                PegMenuDescriptionML* GetMenuDescriptionML();
                void Process(OBCD, CPString, CPString);
};

//Example4 constructor
Example4Window::Example4Window(PegRect rect, CPMainFrame* mf):CPModuleWindow(rect,0,0,mf)
{
        init();
}

int Example4Window::init()
{
//formatting for input/output strings
        const WORD wStyle = FF_NONE|TJ_LEFT|AF_TRANSPARENT| TT_COPY;
        const SIGNED margin = 10;
        const SIGNED width=mClient.Width()-2*margin;
// create an input string positioned at 10, 10 with a maximum width
        inputString = new CPPegString(margin, 10, width, "");
// create an output prompt positioned at 10,30 with a maximum width
        outputFrom = new PegPrompt(margin,30,width,"",0,wStyle);
// create an output prompt positioned at 10,40 with a maximum width
        outputTo = new PegPrompt(margin,40,width,"",0,wStyle);
// add the output prompt to the frame
        AddR(outputTo);
        AddR(outputFrom);
        AddR(inputString);                              // add the input string to the frame
        SetDefaultFocus(inputString);                   // set the focus to the ioString
        return 0;
}

void Example4Window::Draw()
{
                BeginDraw();
                Invalidate(mClient);
                DrawFrame();
                DrawChildren();
                EndDraw();
}
```

```
PegMenuDescriptionML* Example4Window::GetMenuDescriptionML()
{
        return MainMenu;
}

// decode the triggering signal
SIGNED Example4Window::Message(const PegMessage &Mesg)
{
   OBCD X;                                                  // temp OBCD
        switch(Mesg.wType)
        {
                case SIGNAL(INCH_MM_ID, PSF_CLICKED):
                        Process(InchToMM, "inch", "mm");
                        break;
                case SIGNAL(INCH_CM_ID, PSF_CLICKED):
                   X = InchToMM / OBCD10;
                        Process(X, "inch", "cm");
                        break;
                case SIGNAL(MM_INCH_ID, PSF_CLICKED):
                        Process(inv(InchToMM), "mm", "inch");
                        break;
                case SIGNAL(MM_CM_ID, PSF_CLICKED):
                        Process(inv(OBCD10), "mm", "cm");
                        break;
                case SIGNAL(CM_INCH_ID, PSF_CLICKED):
                   X = OBCD10 / InchToMM;
                        Process(X, "cm", "inch");
                        break;
                case SIGNAL(CM_MM_ID, PSF_CLICKED):
                        Process(OBCD10, "cm", "mm");
                        break;
                case SIGNAL(OZ_G_ID, PSF_CLICKED):
                        Process(OzToG,"oz","g");
                        break;
                case SIGNAL(OZ_KG_ID, PSF_CLICKED):
                   X = OzToG / OBCD1K;
                        Process(X,"oz","kg");
                        break;
                case SIGNAL(G_OZ_ID, PSF_CLICKED):
                        Process(inv(OzToG),"g","oz");
                        break;
                case SIGNAL(G_KG_ID, PSF_CLICKED):
                        Process(inv(OBCD1K),"g","kg");
                        break;
                case SIGNAL(KG_OZ_ID, PSF_CLICKED):
                   X = OzToG / OBCD1K;
                        Process(inv(X),"kg","oz");
                        break;
                case SIGNAL(KG_G_ID, PSF_CLICKED):
                        Process(OBCD1K,"kg","g");
                        break;
                default:
                        return CPModuleWindow::Message(Mesg);
        }
        return 0;
}
```

```
void Example4Window::Process(OBCD x, CPString uf, CPString ut)
{
	OBCD Z;                                          // temp OBCD
	OBCD X = x;
		//get the converting OBCD
	CPString tempStringTo;                           // create a temp string
	CPString tempStringFrom;
	tempStringFrom = inputString->DataGet();         // read the inputString
	Z = CPExpression(tempStringFrom);                // convert the string into an OBCD
	Z = Z*X;                                         // perform the calculation
	tempStringFrom += " ";                           // add the units and write the string to the outputTo
	tempStringFrom += uf;
	outputFrom->DataSet(tempStringFrom);
	if((Z.obcd1.mantissa[0]&0xf0)!= 0x00)            //check for valid number in string
	{
		tempStringTo = GetLang(GRAPH_ERROR);
	}
	else
	{
		tempStringTo = CPString(Z,0);            //convert the OBCD to a string and add the units
		tempStringTo += " ";
		tempStringTo += ut;
	}
	outputTo->DataSet(tempStringTo);         // write the error message or answer to the screen
	Draw();                                          // refresh the display
}

void PegAppInitialize(PegPresentationManager *pPresentation)
{
	PegRect Rect;
	// create a rectangle object for the MainFrame
	// set the rectangle to the usable size of the display
		Rect.Set(MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT,
MAINFRAME_BOTTOM);
	CPMainFrame *mw = new CPMainFrame(Rect);         // create the MainFrame
	PegRect AppRect;
	// create a rectangle for the Application Window
	AppRect = mw->FullAppRectangle();                //  size it to the MainFrame
	Example4Window *swin = new Example4Window(AppRect,mw);       // create the Application Window
	mw -> SetTopWindow(swin);                        // load the application window into the MainFrame
	mw -> SetMainWindow(swin);                       // set a main window  for this module
	pPresentation->Add(mw);                          // add the MainFrame to the Presentation Manager
}

extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
	return "";
}
```

Figure 8 shows the two level menu Example 4.

Writing and Reading Data and a Data Input Window, The Currency Converter (Example5)

A Currency converter is used in Example 5. This example moves the class definitions to a separate header file, moves the main application to its own file, adds an PSF_FOCUS_RECEIVED event to the CPPegString, incorporates a pop-up window setup data entry, and writes and reads configuration data.

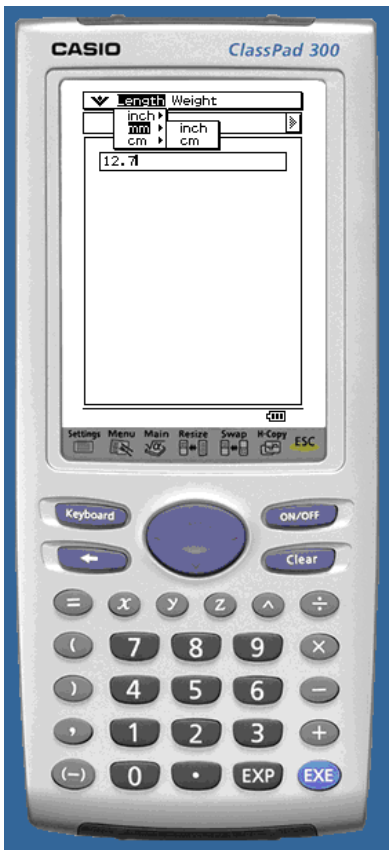

Figure 8

Creating a Header Description File

Following a more traditional object oriented approach, the class definition occurs in a header file. The methods for the class are described in the CPP file. Use the #ifndef UNIQUE_LABEL with the #define UNIQUE_LABEL so the class is defined only once. Enumerated types and #include statements can be included in the header file. End the code with a #endif statement. Include this header file in the CPP file.

Adding a Focus Received Event to the CPPegString

The CPPegString does not have the PSF_FOCUS_RECEIVED ID included by default. (See PegTextSignals under Peg Signals in the *ClassPad 300 SDK Reference* for details) The PSF_FOCUS_RECEIVED of the PegBaseSignals can be enabled by setting the signal for the CPPegString object with the SetSignals(SIGMASK(PSF_FOCUS_RECEIVED)) method. Once set, the CPPegString will generate events when it receives FOCUS.

Adding Input to the Pop-Up Message

A Dialog object is used for a pop-up data input window. PegPrompts are used for the labels and CPPegStrings are used for the data input. PegTextButtons are used for accepting or canceling the data. The CPDialog object is used for the pop-up window. The first parameter is a rectangle that specifies the position and size of the window. The second parameter specifies the title of the window. The third parameter can point to a PegThing for reporting; leave this value NULL. The fourth parameter specifies style flags. The last parameter is used to specify a keyboard. This example uses default styles and brings up the keyboard set to the alpha mode.

Writing Data

The last new feature used in this example is the ability to write the configuration data to a file and read this data when the program starts again. CPWriteMCSFile is used to open a file for writing. The first parameter is the file name. The second parameter is the directory for the file. The last parameter is the data type; use IMU_MCS_TypeMem for data. A file header must be the first set of data written in the file.

The constructor for CPWriteMCSFile

CPWriteMCSFile (const char *name, const char *path=NULL, UCHAR type=0)

The file header is created with the CPMEMFileHeader object. The CPMEMFileHeader constructor has two main parameters. The first is the name representing the application that uses this data. The name is limited to 12 characters. The second parameter is the name of the data. The data name is limited to eight characters. The remaining two parameters specify the major and minor version numbers. These default to one and zero respectively.

The constructor for CPMEMFileHeader

CPMEMFileHeader (const PEGCHAR *AppName, const PEGCHAR *DataName, PEGCHAR MajorVersion=1, PEGCHAR MinorVersion=0)

Once the file is open for reading and the file header is created, the header and data must be written to the file with a Write statement, which will write a string. There are WriteByte, WriteBytes, WriteDouble, WriteInt, and WriteWord statements also. (see the *ClassPad 300 SDK Reference* for details) The first time data is written to the file, only the number of bytes are tabulated. Once all of the data has been written, the size of the file is known and the Realize statement is used to allocate the number of bytes for the file. After the file has been "realized", a second writing of the data actually stores the data in the file. The file is closed with a Close statement.

Reading Data

CPReadMCSFile is used to open a file for reading. The first parameter is the file name. The second parameter is the directory for the file. The last parameter is the data type; use IMU_MCS_TypeMem for data.

The constructor for CPReadMCSFile

CPReadMCSFile (const char *name, const char *path=NULL, UCHAR type=0)

After opening the file, the file header needs to be constructed with the CPMEMFileHeader constructor and read with a Read statement. Two of the errors that can be checked are the FileExists and the IsNotError. The FileExists returns true if the file does exist. The IsNotError statement can be used to verify the file header read matches the desired file header. Proceed to read the data with Read statements. A practice to follow would be to write the number of items written in the file as the first data after the file header. Use this data for a read loop.

Example 5 header File:

```
#ifndef EXAMPLE5_WINDOW_INCLUDED
#define EXAMPLE5_WINDOW_INCLUDED

#include "ClassPad.h"
```

```
//Start menu ID's at 2200
#define IDC_EXAMPLE_ID_BASE 2200

enum IntEIDs
{
	CONVERT_ID = IDC_EXAMPLE_ID_BASE,
	CLEAR_ID,
	RATE_ID,
	FROM_ID,
	TO_ID
};

class Example5Window: public CPModuleWindow
{
	protected:
		PegPrompt*	From;				//Label
		PegPrompt*	To;				//Label
		PegPrompt*	Exchange;			//Label
		CPPegString*	FromString;			//entry field
		CPPegString*	ToString;			//entry field
		CPPegString*	FromText;			//entry field
		CPPegString*	ToText;				//entry field
		CPPegString*	ExchangeRateText;		//entry field
		CPString	ExchangeRateString;		//Exchange Rate Value

	public:
		Example5Window(PegRect, CPMainFrame*);		//Constructor
		~Example5Window(){}				//Destructor
		int init();					//initialization
		void Draw();					//Draw
		void AddUI();					//ToolBar
		SIGNED Message(const PegMessage &Mesg);		//event handler
		void Process();					//perform calculation
		void Clear();					//clear fields
		void Rate();					//load new units and rate
		void Ok();					//process the new rate
		void Save();					//save the data
		bool Load();					//load the data
		WORD Dialog(char*, char*);			//message window
};

#endif
```

Example 5 main.cpp File:

```
#include "Example5Window.h"

// The main method

void PegAppInitialize(PegPresentationManager *pPresentation)
{
	Example5Window *swin;				//create the program as an object
	PegRect Rect;					// create a rectangle object for the MainFrame
	Rect.Set(MAINFRAME_LEFT, MAINFRAME_TOP, MAINFRAME_RIGHT,
		MAINFRAME_BOTTOM);			// set the rectangle to the usable size of the display
```

```
        CPMainFrame *mw = new CPMainFrame(Rect);      // create the MainFrame
        PegRect AppRect;                              // create a rectangle for the Application Window
        AppRect = mw->FullAppRectangle();             // size it to the MainFrame
        swin = new Example5Window(AppRect,mw);        // create the program as an Application Window
        mw -> SetTopWindow(swin);                     // load the application window into the MainFrame
        mw -> SetMainWindow(swin);                    // set a main window  for this module
        pPresentation->Add(mw);                       // add the MainFrame to the Presentation Manager
}
```

Example 5 cpp File:

```
#include "Example5Window.h"

static const PEGCHAR* FILE_NAME = "ExRate";
static const PEGCHAR* FOLDER = "main";
const PEGCHAR* Example5_MEMTYPE_HEADER = "Example 5";
const PEGCHAR* Example5_MEMTYPE_SAVED_STATE = "Data";

void Example5Window::AddUI()
{
// create text buttons for the toolbar
        PegTextButton* b1 = new PegTextButton(1,1, "Convert", CONVERT_ID, AF_ENABLED|TT_COPY);
        PegTextButton* b2 = new PegTextButton(2,1, "Clear", CLEAR_ID, AF_ENABLED|TT_COPY);
        PegTextButton* b3 = new PegTextButton(3,1, "Rate", RATE_ID, AF_ENABLED|TT_COPY);
//position and enable the buttons
        m_ui->AddToolbarButton(b1);
        m_ui->AddToolbarButton(b2);
        m_ui->AddToolbarButton(b3);
}

Example5Window::Example5Window(PegRect rect, CPMainFrame* frame):CPModuleWindow(rect,0,0,frame)
{
  init();
}

int Example5Window::init()
{
//formatting for input/output strings
        const WORD wStyle = FF_NONE|TJ_LEFT|AF_TRANSPARENT| TT_COPY;
        const SIGNED margin = 10;
        const SIGNED width=mClient.Width()-2*margin;
        CPString        StatusString;
// create labels
        From = new PegPrompt(margin, 10, width, "",0,wStyle);
        To = new PegPrompt(margin, 40, width, "",0,wStyle);
        Exchange = new PegPrompt(margin, 70, width, "",0,wStyle);
// create input field
        FromString = new CPPegString(margin,20,width,"",FROM_ID, FF_THIN|AF_ENABLED|EF_EDIT);
        ToString = new CPPegString(margin, 50, width, "",TO_ID, FF_THIN|AF_ENABLED|EF_EDIT);
// force the CPPegStrings to send the PSF_FOCUS_RECEIVED signal
        FromString->SetSignals(SIGMASK(PSF_FOCUS_RECEIVED));
        ToString->SetSignals(SIGMASK(PSF_FOCUS_RECEIVED));
// add the labels and input fields to the frame
        AddR(From);
        AddR(To);
```

```
	AddR(Exchange);
	AddR(FromString);
	AddR(ToString);
// set the focus to the From field
	SetDefaultFocus(FromString);
//Load the conversion data
	if(Load())
	{
//data exists, build the info for message
		StatusString+= " ";
		StatusString = (CPString)From->DataGet();
		StatusString += ExchangeRateString;
		StatusString+= " = ";
		StatusString += (CPString)To->DataGet();
	}
	else
	{
//data does not exist
		StatusString = "No Rate Loaded";
	}
	Exchange ->DataSet(StatusString);
//display the message
	return 0;
}

void Example5Window::Draw()
{
	BeginDraw();
	Invalidate(mClient);
	DrawFrame();
	DrawChildren();
	EndDraw();
}

// decode the triggering signal
SIGNED Example5Window::Message(const PegMessage &Mesg)
{
	switch(Mesg.wType)
	{
		case SIGNAL(FROM_ID, PSF_FOCUS_RECEIVED):	//the from field got focus
			Clear();
			break;
		case SIGNAL(TO_ID, PSF_FOCUS_RECEIVED):		//the to field got focus
			Clear();
			break;
		case SIGNAL(CONVERT_ID, PSF_CLICKED):		//the convert button was selected
			Process();
			break;
		case SIGNAL(CLEAR_ID, PSF_CLICKED):		//the clear button was selected
			Clear();
			break;
		case SIGNAL(RATE_ID, PSF_CLICKED):		//the rate button was selected
			Rate();
			break;
		default:
			return CPModuleWindow::Message(Mesg);
```

```
        }
        return 0;
}

WORD Example5Window::Dialog(char* title, char* message)
{
//determine size and position for the error message window
        PegRect rr = GetMainFrame()->TopAppRectangle();
        rr.wBottom -= 30;
        rr.wRight -= 10;
        rr.wTop += 10;
        rr.wLeft += 10;
        PegMessageWindow *msg = new PegMessageWindow(rr, title, message, MW_OK, 0,0,0);
        return msg->Execute();
}

void Example5Window::Rate()
{
//set up the labels, fields, and buttons
        const SIGNED margin = 4;
        const SIGNED LabelWidth = 30;
        SIGNED rwidth;
        PegRect rrok;
        PegRect rrcancel;
//get space and determine size of window
        PegRect rr = GetMainFrame()->TopAppRectangle();
        rr.wBottom -= 5;
        rr.wRight -= 2;
        rr.wTop += 2;
        rr.wLeft += 2;
//calculate cancel button position and size
        rrcancel = rr;
        rrcancel.wBottom-=5;
        rrcancel.wTop = rrcancel.wBottom-12;
//calculate ok button position and size
        rrok = rrcancel;
        rrok.wLeft += 20;
        rrok.wRight = rrok.wLeft+50;
        rrcancel.wRight -= 20;
        rrcancel.wLeft = rrcancel.wRight-50;
//determine width for fields
        rwidth = rr.wRight-rr.wLeft-2*margin-LabelWidth;
//create the Dialog window with the alpha keyboard
        CPDialog * dlg = new CPDialog(rr,"Exchange Rate",NULL,NULL,CP_KEYPAD_ABC);
//position labels, fields, and buttons
        PegPrompt* FromLabel = new PegPrompt(margin,52,"From");
        FromText = new CPPegString(margin+LabelWidth, 52, rwidth, "");
        PegPrompt* ToLabel = new PegPrompt(margin,69,"To");
        ToText = new CPPegString(margin+LabelWidth, 69, rwidth, "");
        PegPrompt* ExchangeRateLabel = new PegPrompt(margin,86,"Rate");
        ExchangeRateText = new CPPegString(margin+LabelWidth, 86, rwidth, "");
        PegTextButton* b1 = new PegTextButton(rrok, "OK", IDB_OK, AF_ENABLED|TJ_CENTER);
        PegTextButton* b2 = new PegTextButton(rrcancel, "CANCEL", IDB_CANCEL,
                AF_ENABLED|TJ_CENTER);
        dlg->Add(b1);
        dlg->Add(b2);
```

```
	dlg->Add(FromLabel);
	dlg->Add(ToLabel);
	dlg->Add(ExchangeRateLabel);
	dlg->Add(ExchangeRateText);
	dlg->Add(ToText);
	dlg->Add(FromText);
//present the window and return the ok/cancel information
	WORD w = dlg->Execute();
	if (w == IDB_OK)
	{
		Ok();				//process the new data
	}
}

void Example5Window::Clear()
{
//clear the fields
	FromString->DataSet("");
	ToString->DataSet("");
// refresh the display
	Draw();
}

void Example5Window::Ok()
{
	OBCD T;
	CPString	FtempString;
	CPString	TtempString;
	CPString	EtempString;
	CPString	StatusString;
	FtempString = (CPString)FromText->DataGet();		// read the From Currency
	TtempString = (CPString)ToText->DataGet();		// read the To Currency
	EtempString = (CPString)ExchangeRateText->DataGet();	// read the Exchange Rate
	FtempString.UpperCase();
	TtempString.UpperCase();
	if(FtempString == "" && TtempString == "" && EtempString == "")    //make sure all fields contain data
	{
		Dialog("Error","Enter all fields   ");
	}
	else
	{
		if(FtempString == TtempString)			//make sure the currencies are different
		{
		   Dialog("Error","Same currencies  ");
		}
		else
		{
			T = CPExpression(EtempString);		// convert the string into an OBCD
			if((T.obcd1.mantissa[0]&0xf0)!= 0x00)	// check for an error
			{
				Dialog("Error","Invalid exchange rate  ");
			}
			else
			{
				StatusString = FtempString;	//build the new labels
				StatusString += " = ";
```

```
                                        StatusString += EtempString;
                                        StatusString += " ";
                                        StatusString += TtempString;
                                        Exchange ->DataSet(StatusString); //set the data
                                        From ->DataSet(FtempString);
                                        To ->DataSet(TtempString);
                                        ExchangeRateString = EtempString;
                                        Save();                                 //save the data
                                }
                        }
                }
                Draw();
}

void Example5Window::Process()
{
        OBCD T;                                                         // temp OBCD
        OBCD ExchangeRate;
        CPString        FromTempString;                                 // create two temp string
        CPString        ToTempString;
        FromTempString = (CPString)FromString->DataGet();               // read the FromString
        ToTempString = (CPString)ToString->DataGet();                   // read the To
        if(FromTempString != "" && ToTempString != "")                  //check if nothing was entered
        {
                Dialog("Error","One must be empty   ");
        }
        else
        {
                if(FromTempString == "" && ToTempString == "")
                {
                   Dialog("Error","Must have an amount  ");
                }
                else
                {
                        if(FromTempString.Length() != 0)                //check if the from currency was entere
                        {
                                T = CPExpression(FromTempString);       // convert the string into an OBCD
                                if((T.obcd1.mantissa[0]&0xf0)!= 0x00)   // perform the calculation
                                {
                                        Dialog("Error","Invalid Amount  ");
                                }
                                else
                                {
                                        ExchangeRate = CPExpression(ExchangeRateString);
                                        T = T * ExchangeRate;
                                        ToTempString = CPString(T,0);   //convert the OBCD to a string
                                        ToString->DataSet(ToTempString);// write the message to the screen
                                }
                        }
                        else
                        {
                                T = CPExpression(ToTempString); // convert the string into an OBCD
                                if((T.obcd1.mantissa[0]&0xf0)!= 0x00)   // perform the calculation
                                {
                                        Dialog("Error","Invalid Amount  ");
                                }
```

```
                                        else
                                        {
                                                // convert the string into an OBCD
                                                ExchangeRate = CPExpression(ExchangeRateString);
                                                T = T / ExchangeRate;
                                                FromTempString = CPString(T,0);
                                                // write the error message to the screen
                                                FromString->DataSet(FromTempString);
                                        }
                                }
                        }
                }
        Draw();                                                 // refresh the display
}

void Example5Window::Save()
{
        CPString temp;
// open the file for writing
        CPWriteMCSFile f(FILE_NAME, FOLDER,IMU_MCS_TypeMem);
//create the file header
        CPMEMFileHeader header = CPMEMFileHeader(Example5_MEMTYPE_HEADER,
                Example5_MEMTYPE_SAVED_STATE);
        header.Write(f);                                        //write the file header
        temp = (CPString)From->DataGet();                       //get the from unit
        temp.Write(f);                                          //write the from unit
        temp = (CPString)To->DataGet();                         //get the to unit
        temp.Write(f);                                          //write the to unit
        temp = ExchangeRateString;                              //get the exchange rate
        temp.Write(f);                                          //write the exchange rate
        f.Realize();                                            //size the file
        header.Write(f);                                        //write the dat for real
        temp = (CPString)From->DataGet();
        temp.Write(f);
        temp = (CPString)To->DataGet();
        temp.Write(f);
        temp = ExchangeRateString;
        temp.Write(f);
        f.Close();                                              //close the file
}

bool Example5Window::Load()
{
        bool ok = false;
        OBCD T;
        CPString temp;
        //open the file for reading
        CPReadMCSFile f(FILE_NAME, FOLDER,IMU_MCS_TypeMem);
        //create the file header
        CPMEMFileHeader header = CPMEMFileHeader(Example5_MEMTYPE_HEADER,
                Example5_MEMTYPE_SAVED_STATE);
        if (f.FileExists())                                     //check if the file exists
        {
                header.Read(f);                                 //read the file header
                if(f.IsNotError())                              //validate the file header
                {
```

```
                        temp.Read(f);                   //read the from unit
                        if(temp != "")                  //make sure there is data
                        {
                                From ->DataSet(temp);   //set the from unit
                                temp.Read(f);           //read the to unit
                                if(temp != "")          //make sure there is data
                                {
                                        To ->DataSet(temp);     //set the to unit
                                        temp.Read(f);           //read the exchange rate
                                        if(temp != "")          //make sure there is data
                                        {
                                                ExchangeRateString = temp;      //set the exchange rate
                                                T = CPExpression(ExchangeRateString);
                                                if((T.obcd1.mantissa[0]&0xf0)== 0x00)
                                                {
                                                        f.Close();              //close the file
                                                        ok = true;              //read was ok
                                                }
                                        }
                                }
                        }
                }
        }
        return ok;
}

extern "C" char *ExtensionGetLang(ID_MESSAGE MessageNo)
{
        return "";
}
```

Example 5 accepts two currency units and the conversion rate between the two. This information is stored in a file. The user then selects one of the two currency fields and enters a value. When a currency field is selected, the other field is cleared. When the conversion is desired, select the convert button which will show the converted currency in the empty field. The clear button will clear both currency fields. Figure 9 shows the program when started. Figure 10 shows the data input window. A currency rate conversion is shown in Figure 11.

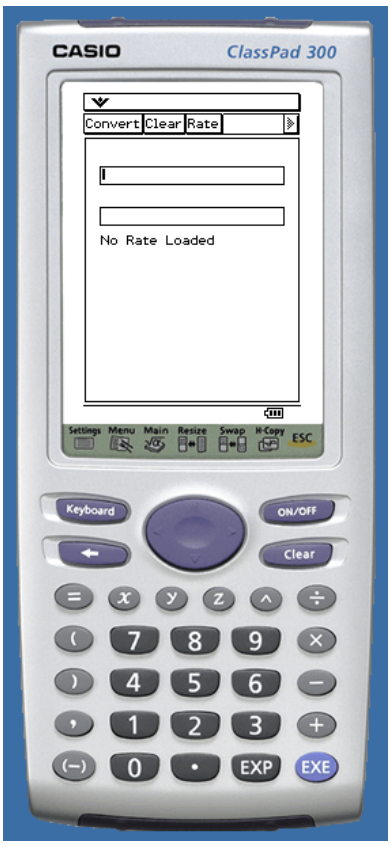

Figure 9

Figure 10

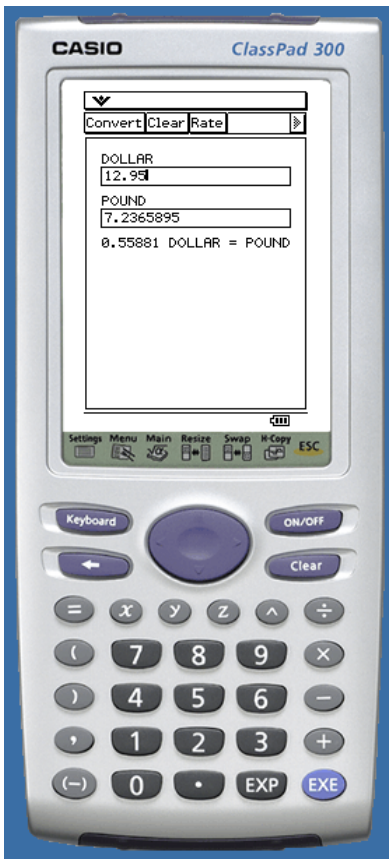

Figure 11

Conclusion

The *Casio ClassPad 300 Software Development Kit* is a useful tool for developing and debugging custom applications for the *Casio ClassPad 300*. Through the use of custom applications, the applications of the *Casio ClassPad 300* is limited only to the creativity of the programmer.

Included here are several sample applications to help you get started. The Length and Weight Conversion Application, the Temperature Conversion Application, the Currency Converter Application, and the Number Base Converter Application are provided for you to use and modify. The CpSDK includes equally useful applications, such as the Address Book Application and the Scribble Application are equally useful Applications. These applications, the examples in this document, the *ClassPad 300 SDK Tutorial*, and the *Programming Guide* provide the user with a good source of information to begin writing custom Applications.

References

CASIO ClassPad 300 SDK Installation Guide

CASIO ClassPad 300 SDK Programing Guide

CASIO ClassPad 300 SDK Reference

ClassPad SDK (Programing) Tutorial