APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi SOMO-II Demo

DOCUMENT DATE: **21st May 2019**
DOCUMENT REVISION: **1.1**

Description

This application note shows how to interface a 4D Systems intelligent display to the embedded Sound Module SOMO-II. Before getting started, the following are required:

- Any of the following 4D Picaso display modules:
  gen4-uLCD-24PT
  gen4-uLCD-28PT
  gen4-uLCD-32PT
  uLCD-24PTU
  uLCD-28PTU
  uVGA-III

and other superseded modules which support the ViSi Genie environment

- The target module can also be a Diablo16 display
  gen4-uLCD-24D series
  gen4-uLCD-28D series
  gen4-uLCD-32D series
  gen4-uLCD-35D series
  gen4-uLCD-43D series
  gen4-uLCD-50D series
  gen4-uLCD-70D series
  uLCD-35DT
  uLCD-43D Series
  uLCD-70DT

  Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.

- 4D Programming Cable or µUSB-PA5
- 2 micro-SD (µSD) memory cards
- Workshop 4 IDE (installed according to the installation document)
- SOMO-II module*
- Jumper wires for connecting the SOMO-II to the display
- Speaker
- or has a working knowledge of the topics presented in these recommended application notes.

- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

**Even without a SOMO-II module, it is still possible for the user to run the demo in this application note by using the terminal utility in Workshop. See the section "Simulate the SOMO-II Module using the Terminal".*

Content

Application Overview

It is often difficult to design a graphical display without being able to see the immediate results of the application code. 4D-ViSi is the perfect software tool that allows users to see the instant results of their desired graphical layout. Additionally, there is a selection of inbuilt dials, gauges, and meters (called widgets) that can simply be dragged and dropped onto the simulated display. From here each widget can have its properties edited, and at the click of a button all relevant code is produced in the user program.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

Create a New Project

For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

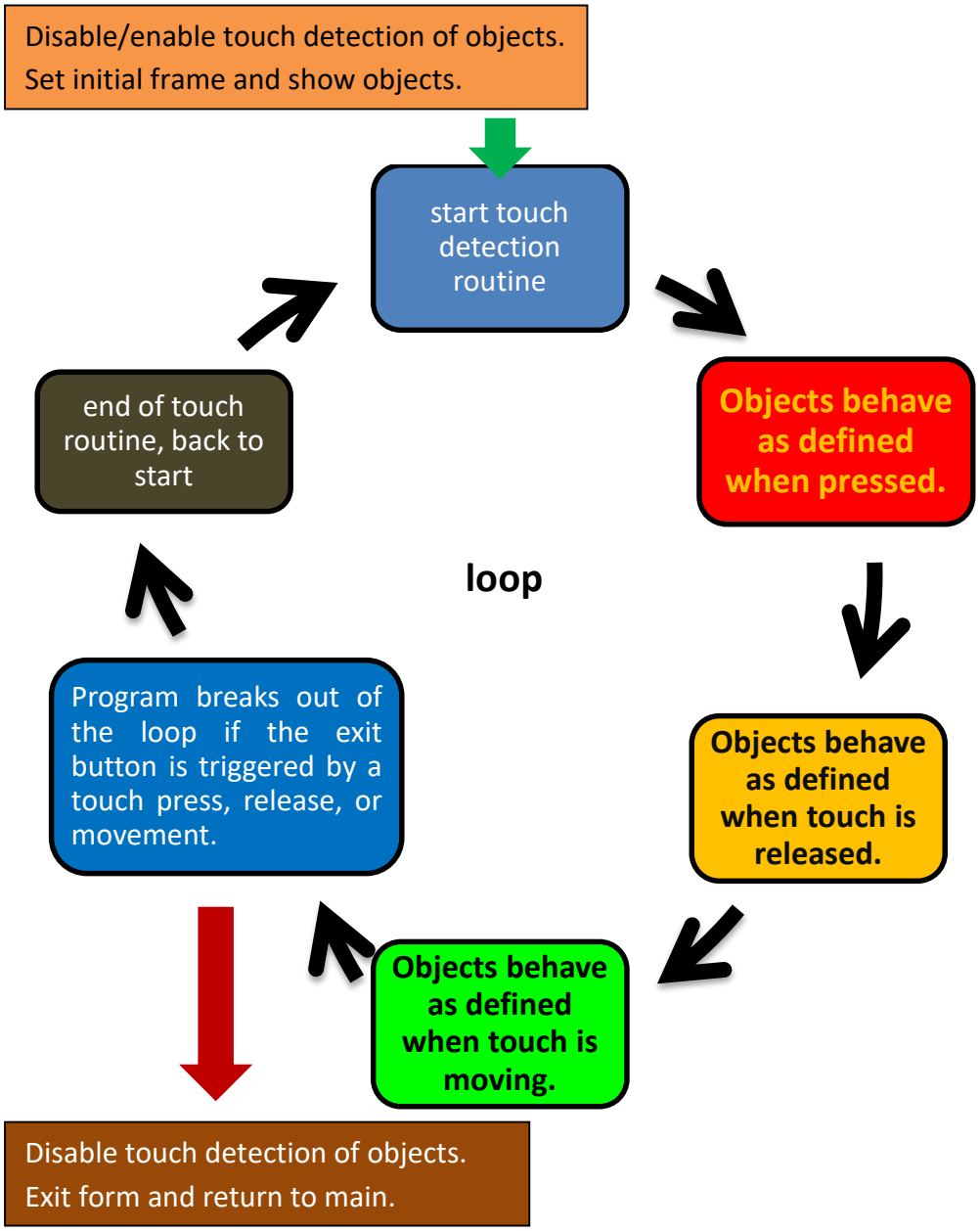
ViSi Getting Started - First Project for Picaso and Diablo16

Design the Project

Writing the Code for a Form

The general instructions below can be followed when writing the 4DGL code for a form that handles graphics and touch detection. The objective is to be able to display objects and control their behaviour when touched.

- Enable touch detection for objects that are meant to respond to touch
- Disable touch detection for objects that are not meant to respond to touch
- Set initial frames of objects then show the objects.
- Start touch detection routine.
- Define behaviour of objects when pressed.
- Define behaviour of objects when touch is released.
- Define behaviour of objects when touch is moving.
- Create an exit button to get out of the loop.
- Loop back to start of touch detection routine.
- Disable touch detection for all objects in the form before returning to main.



In Form1 of **miniSOMOII_demo**, behaviour of objects is defined only when touch status is moving (TOUCH_MOVING). In Form2, behaviour of objects is defined only when the touch status is released (TOUCH_RELEASED). For a more "complete" program, the user has the option of defining the behaviour of objects for all three touch statuses in a loop - "TOUCH_PRESSED", "TOUCH_RELEASED", and "TOUCH_MOVING".

Writing the Code for the Demo

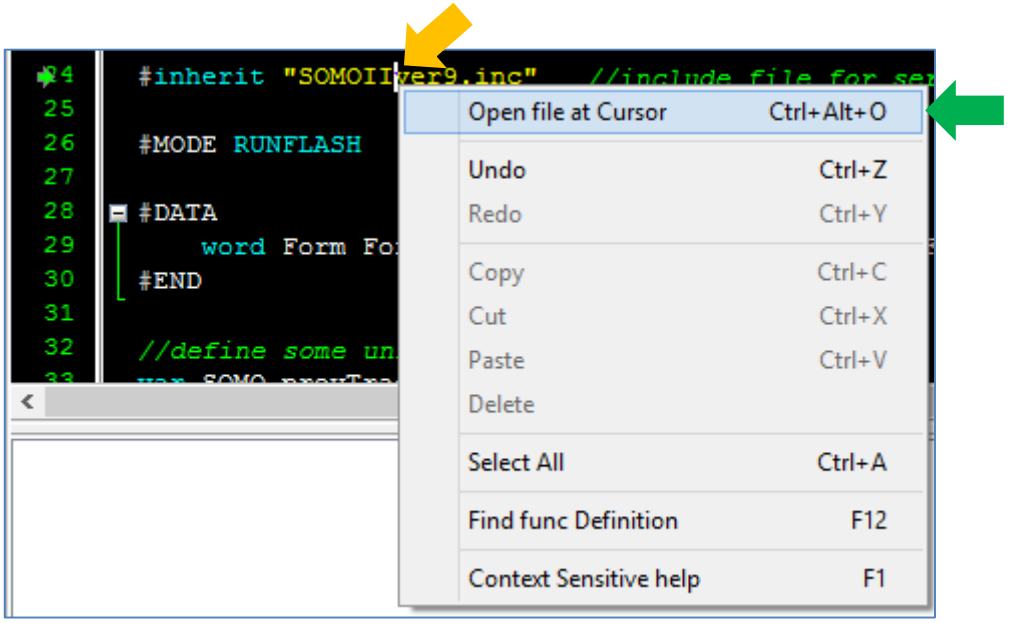
Theoretically, the program for **miniSOMOII_demo** has two tasks – to handle graphics and to manage communication with the SOMO-II module. The task of handling graphics involves the process of displaying objects and defining their behaviour during touch detection. This part is illustrated in the diagram to the left. Beginners may refer to the enumerated application notes on page 4. The task of communicating with the SOMO-II module requires the addition of an include file at the start of the code. In this include file all the high level commands for talking to and listening from the SOMO-II module are defined. These high level commands or functions are called when a form is running. The following paragraphs provide further discussions.

The Include File

The demo has an include file which contains the subroutines for communicating with the display. Note that the file version may change anytime due to changes and/or improvements.

```
24   #inherit "SOMOIIver9.inc"   //include file
```

To view the contents, put the cursor on the include file name text, then click on the right mouse button. Choose the first option, "Open file at Cursor".



The file opens in another tab.

SOMOII | SOMOIIver9.inc

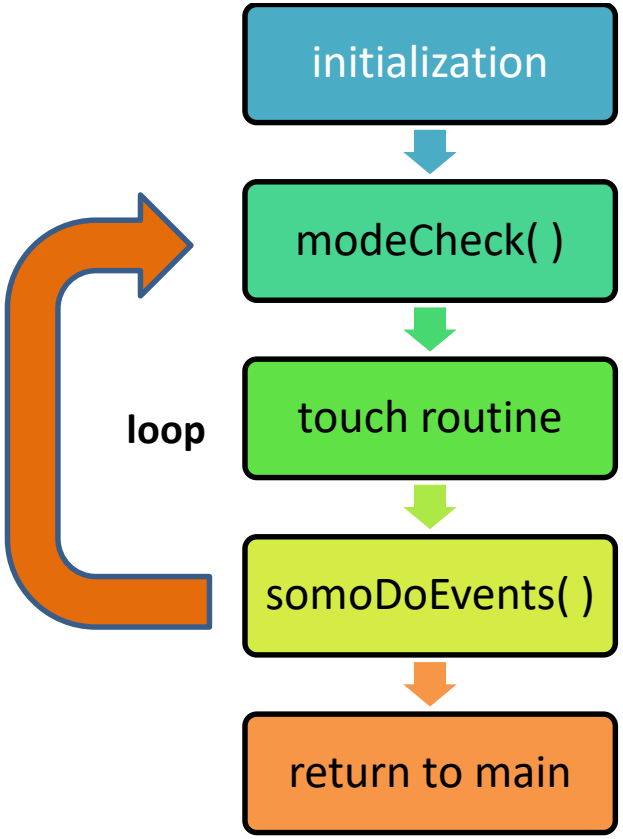
```
//SOMOIIver9.inc
//January 23, 2014

//constants for indicating the status of the SOMO II
#CONST
    SOMO_STAT_ACK               // 0 - Command Succ
    SOMO_STAT_ERR_BUSY          // 1 - Module is bu
    SOMO_STAT_ERR_SLEEP         // 2 - Module is cu
    SOMO_STAT_ERR_SERIAL        // 3 - Serial recei
    SOMO_STAT_ERR_CHKSUM        // 4 - Checksum err
    SOMO_STAT_ERR_SCOPE         // 5 - Beyond scope
    SOMO_STAT_ERR_NOTFOUND      // 6 - Specified fi
    SOMO_STAT_MSG_USB_INSRT     // 7 - USB Flash Dr
    SOMO_STAT_MSG_uSD_INSRT     // 8 - micro SD Car
```

The include file can be edited and improved by adding in more commands. The contents of the include file are further discussed in the section "Functions for Communicating with the SOMO-II module".

Program Flow

Form1 of miniSOMOII_demo follows the model shown below. Note that the graphics handling and communications routines are integrated into this model.



Initialization

Here constants are defined, and variables are declared and initialized. Some of these variables are used as flags. Touch detection is enabled or disabled

for objects depending on their purpose. The initial frames of objects are set and the objects are shown.

modeCheck( )

This subroutine queries the SOMO-II module which track is currently playing. Since querying involves the sending and receiving of messages to and from the SOMO-II module, this subroutine is executed not every time the loop repeats but only when any of the five predefined conditions is true. See lines 165 to 182 of the code.

```
//run modeCheck only if any of the following conditions is true
if(buttonsFlagPrev != buttonsFlagNow) //if any of the media control buttons has been touched
    modeCheck();
    buttonsFlagPrev := buttonsFlagNow;
else if(SOMO_finishedPlayingFlag)     //if a track has just finished playing
    modeCheck();
    SOMO_finishedPlayingFlag := 0;
else if(!TIMER2)                      //if 10 seconds has passed since the first iteration
    modeCheck();
    sys_SetTimer(TIMER2,10000);
else if(!_startFlag)                  //if the user has just navigated from another form
    modeCheck();
    _startFlag := 1;
else if(SOMO_selectTrack)             //if there is a selected track - (see the full version)
    modeCheck();
    SOMO_selectTrack := 0;
endif
```

modeCheck() also sets the initial state of the play/pause button and the initial frame of the album cover user images object depending on the current play mode and other flags. The status of the play/pause button and the frame of the album cover user images object may change each time the loop repeats.

Touch routine

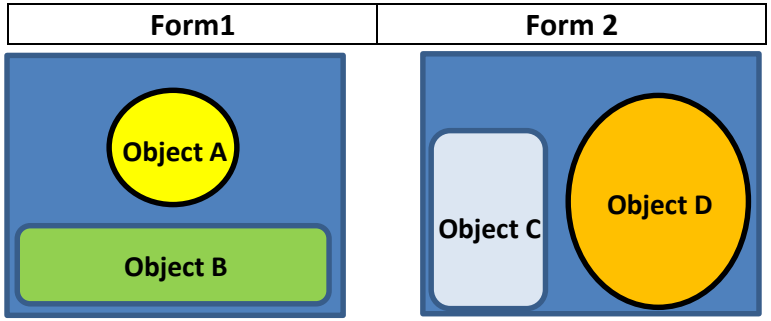
Here the behaviour of objects when they are touched are defined. Commands are also sent to the SOMO-II module depending on the object touched. Note that there are three specific touch status values. The programmer can use one or all of these when coding.

somoDoEvents( )

This subroutine queues messages from the SOMO-II module. If a message is or messages are present in the queue, one is taken out and evaluated. When the message "finished playing a track" has been received, the flag **SOMO_finishedPlayingFlag** will be set so that **modeCheck( )** will know what to do in the next iteration of the loop. This makes the music player "smart", since it knows when to change the status of the play/pause button back to "press-to-play" if a track has finished playing. This is true for the single play mode.

Return to main

If a certain condition has been met (the "exit" button has been pressed for instance), the program breaks out of the loop, disables touch detection for all relevant objects, sets the return value (if present), and returns to main. It is important to disable touch detection of objects in a form before exiting that form since many of the objects in Workshop respond to touch even though they are not shown. To illustrate:



In the image shown, each of the forms has two objects that respond to touch. In Workshop, objects are named as they are added to the project. Suppose the program exits Form1 and executes Form2 without disabling touch detection for objects A and B. Now that the program is in Form2, only objects C and D are shown. Objects A and B still respond to touch however, and this may bring about undesirable or unexpected results. If the lower regions of objects C and D are touched, these objects will not respond. It is actually object B that will respond since it is still active and it was added to the project first before objects C and D. So if, for example, objects C and D will each turn on and off an LED, the user may observe that the LEDs are not responding when objects C and D are touched at the lower regions.

Functions for Communicating with the SOMO-II Module

The following is a table of functions defined in the include file "**SOMOIIver9.inc**".

Command	**initializeSomo(*attempts*)**
Description	Sends the reset command for ***attempts*** times. Display waits for the correct reply.
Return	Returns '0' if failed, '1' if successful.

Command	**somoBufferEvents()**
Description	Queues a maximum of 6 replies from the SOMO-II module. Queued replies are stored in global arrays.
Return	None

Command	**somoDequeue(**reply*)**
Description	Takes the latest message from the queue and evaluates it. The message is stored in an array with the address ***reply***.
Return	Returns '1' if a message has been taken from the queue, '0' if none

Command	**playSong( )**
Description	Play the audio track selected (if selected) else the first track copied on to the media
Return	None

Command	**stopSong( )**
Description	Stop the current playing audio Track. If PLAY command is then sent, the audio track will start from the beginning.
Return	none

Command	**pauseSong( )**
Description	Pause the current playing audio Track. If PLAY command is then sent, the audio track will resume from where it was paused.
Return	none

Command	**nextSong( )**
Description	If no track is currently playing, issuing the NEXT command will start playing the first track copied to the media. If the SOMO-II is currently playing a song or has previously played a song, this will play the next song in the order copied on to the media.
Return	none

Command	previousSong( )
Description	If no track is currently playing, issuing the PREVIOUS command will start playing the last track copied to the media. If the SOMO-II is currently playing a song or has previously played a song, this will play the previous song in the order copied on to the media.
Return	none

Command	volumeSet(*volume*)
Description	Sets the volume level to *volume*. Maximum value is 30.
Return	none

Command	selectuSD( )
Description	This will set the SOMO-II to play from a micro-SD Card
Return	none

Command	queryCurrentTrackuSD(*timeLimit*)
Description	Query the current track playing from the micro-SD Card.
Return	Returns '1' if a valid message has been received within the time limit *timeLimit* (in milliseconds). Returns '0' otherwise. To get the track number, access the fourth element of a private variable, like as shown below.

```
if(queryCurrentTrackuSD(1000))//if successful
    txt_MoveCursor(0,0);
    SOMO_currentTrack := queryCurrentTrackuSD._rcvdMsgSOMO[3];
    print("play track: ",SOMO_currentTrack, "
    //print("\n");
else  //if unsuccessful
    txt_MoveCursor(0,0);
    print("can't determine current track...");
endif
```

Command	queryTracksuSD(*timeLimit*)
Description	Query the number of files present on the uSD card.
Return	Returns '1' if a valid message has been received within the time limit *timeLimit* (in milliseconds). Returns '0' otherwise. To get the number of files present on the media, access the fourth element of a private variable, like as shown below.

```
if(queryTracksuSD(1000))//send a query, with a 1000ms timeout
    _totalTracks := queryTracksuSD._rcvdMsgSOMO[3];
    print("Total number of tracks: ",_totalTracks,"
else
    print("can't determine...                    ");
endif
```

Command	specifyTrack(*trackNumber*)
Description	Specify the number of a track then play it.
Return	none

Command	specifyEQ(*eq*)
Description	Configure the eq setting. Values for *eq*: 0/1/2/3/4/5 Normal, Pop, Rock, Jazz, Classic, Bass. See Form4 of the full demo.
Return	none

Command	sleepSOMO( )
Description	This will put the SOMO-II into a sleep state, which consumes low power. To get the SOMO-II out of sleep, you need to use a PLAY SOURCE command, followed by your next chosen command.
Return	none

Command	**resetSOMO( )**
Description	This will reset the SOMO-II module, to be in its powered-on state.
Return	none

Command	**singlePLAY( )**
Description	This will disable CONTINUOUS, RANDOM TRACK or REPEAT CURRENT modes if previously enabled, which is how the module starts up by default. This will allow one song to play and then stop. Start track with PLAY.
Return	none

Command	**continuousPLAY( )**
Description	This will enable continuous mode (disable RANDOM TRACK and REPEAT CURRENT if previously enabled), which will play all songs on the memory card, one after the other. Start track with PLAY.
Return	none

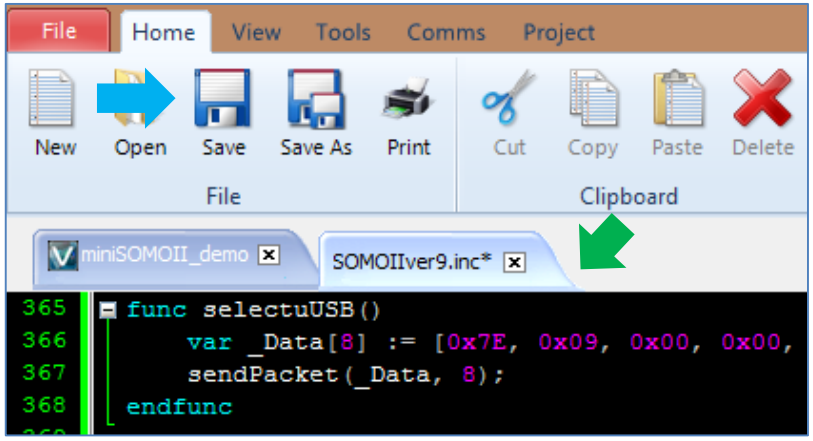
Command	**randomPLAY( )**
Description	This will enable Random Mode (disable CONTINUOUS and REPEAT CURRENT if previously enabled), which plays random tracks one after the other, continuously. Start track with PLAY.
Return	none

Command	**repeatCurrent( )**
Description	This will enable the repeat play mode (disable CONTINUOUS and RANDOM TRACK if previously enabled), which repeats the currently playing track, so it will play over and over continuously. Track must be playing before this command is sent.
Return	none

Command	**setPlayMode(*mode*)**
Description	Set the play mode to ***mode***: 0/1/2/3 – single play/continuous play/random/repeat current
Return	none

Adding Serial Commands to the Include File

The SOMO-II datasheet contains all of the predefined SOMO-II serial commands and their corresponding function and meaning. To add more of these to the include file, simple edit the file and save it.



As an example, the serial command for choosing the USB flash drive as the play source is “7E 09 00 00 01 FF F6 EF”. From the datasheet:

PLAY SOURCE	7E 09 00 00 **01** FF F6 EF	This will set the SOMO-II to play from a USB Flash Drive
	7E 09 00 00 **02** FF F5 EF	This will set the SOMO-II to play from a micro-SD Card

This command can be implemented in 4DGL and added to the include file as shown below.

```
//Select USB Source
func selectuUSB()
    var _Data[8] := [0x7E, 0x09, 0x00, 0x00, 0x01, 0xFF, 0xF6, 0xEF];
    sendPacket(_Data, 8);
endfunc
```

Run the Program

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination (this option is not available for Goldelox displays), and how to compile and download a program, please refer to the section “**Run the Program**” of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

The uLCD-32PTU and uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability