Q u a n t u m   D e s i g n

PPMS Application Note 1070-202

# Interfacing 3rd Party Instruments to the PPMS Software Environment

One of the key features of the Quantum Design PPMS is its capability to interface the automated PPMS MultiVu software with user created application programs. This application note describes how a user's program can respond to advisories and access third party instruments. At this time, Quantum Design has created template programs using Borland's Delphi™, Visual Basic, and Visual C++, which will simplify the process of receiving advisories. This application note provides PPMS users with a tutorial introducing the basic concepts using Delphi to receive advisories and provides a general understanding of how system interfacing can be done on the PPMS.

It should be noted that this application note IS NOT a substitute for learning Delphi nor understanding how to perform general instrument interface using GPIB. We assume that you are familiar with both subject areas and that you only need the specific information pertaining to the PPMS software. This application note only provides general guidelines in understanding the linking procedures for the PPMS.

> *Note: the PPMS also has the ability to trigger third party instruments using the "Digital Line Set", "External Select Output Line", or "Signal Output" commands from the sequence editor. The Digital Line Set provides +24 volts. The External Select Output Line provides a TTL switch. The Signal Output sets the specified BNC connector to a certain fixed output voltage. This allows –10 V to +10 V feedback to other instruments or connection to chart recorders, oscilloscopes, and so on. Refer to the PPMS Hardware Manual for further information on using the external outputs found in the rear of the Model 6000.*

## Tutorial 1: Interfacing User Software to the PPMS:

### Advisories:

The main features of the PPMS software is to give linking capabilities, which makes it possible for the users to connect their program to the PPMS software without having to rewrite the PPMS source code. The linking process is done through what are termed PPMS advisories. An advisory is used when the PPMS software needs to send out a flag throughout the network of running programs such that any program set up to accept advisories can catch them. Thus the advisories act as a software trigger to initiate certain functions of an external application program.

The program language chosen by Quantum Design in this Application Note is Delphi. The Delphi application development environment was chosen for its ease of use and its universal acceptance as a standard applications software language. Delphi takes full advantage of Windows capabilities and is popular among many interface developers. Interfacing the PPMS software with the Model 6000 is done via the IEEE - 488 General Purpose Interface Bus (GPIB). Third party GPIB instruments can be daisy-chained to the PPMS using a standard GPIB interface cable. The GPIB address of the Model 6000 (default 15) can be determined from its front panel by looking at the config #3 selection (see PPMS User's manual for more detail). Third party instruments connected to the PPMS must have a GPIB address different to that of the Model 6000 and the Model 6500 (default 14). Third Party RS-232 instruments can be connected to COM1 or Com2 port at the rear of the PC.

The basic format of the PPMS software architecture revolves around its modular design. Since the Windows operating system makes software multi-tasking possible, different software "modules" can be run in unison when performing a specific application. Such modules include the base PPMS software (PPMS MultiVu), our option software modules (such as ACMS), and utility software (such as Mon6000). Different combinations of the software modules can be chosen for each specific measurement requirement. In the case of a user-developed application, the user-software written in Delphi can also be run simultaneous to the other standard PPMS modules. The base PPMS software will set up the necessary environment such as temperature, field, and time. Once the environment settings are reached, the PPMS sequence can be written to send an advisory, which triggers the user-developed application to perform its necessary tasks. Once the applications program is finished, the PPMS sequence will change the environment to the next set point. This same method is used to run our measurement applications (such as ACMS). More information about advisories is presented below.

## Receiving Advisories:

Before you can start creating your interfacing software, you will need to purchase the necessary application development software. For purposes of this application note, we used Delphi 5.0. There are two versions of Delphi 5.0 available - the standard copy and the professional copy. Though the standard copy is less expensive, the professional copy does have some extra tools that are useful. Either version will allow you to interface correctly to the PPMS. (Note: Delphi 6.0, which is now available, is fully compatible for this application.)

The first step in linking your Delphi application software to the PPMS sequence editor is to give your program the ability to receive advisories from the PPMS software through the server. This capability is provided by the Quantum Design Delphi template program, which is available on our website (www.qdusa.com/resources/ppms3rdparty.html). Visual Basic and Visual C++ template programs and sample programs are also available on our website. Once you have installed Delphi, Visual Basic, or Visual C++ onto the PPMS computer, you can add these files to the computer.

The rest of the tutorial will be separated into three sections. The first section will deal with receiving advisories and sending command strings using the Delphi template program provided by Quantum Design. The second section will demonstrate receiving data strings from the Model 6000. The final section will cover the process of sending and receiving strings from external instruments. As stated earlier you should already be somewhat familiar with creating programs using Delphi. If you are not familiar with the language, there are several instructional texts available to quickly learn the language and its environment. If you are familiar with Delphi then the following steps should be easy and straightforward.

**Hello world 1 (receiving advisories):**

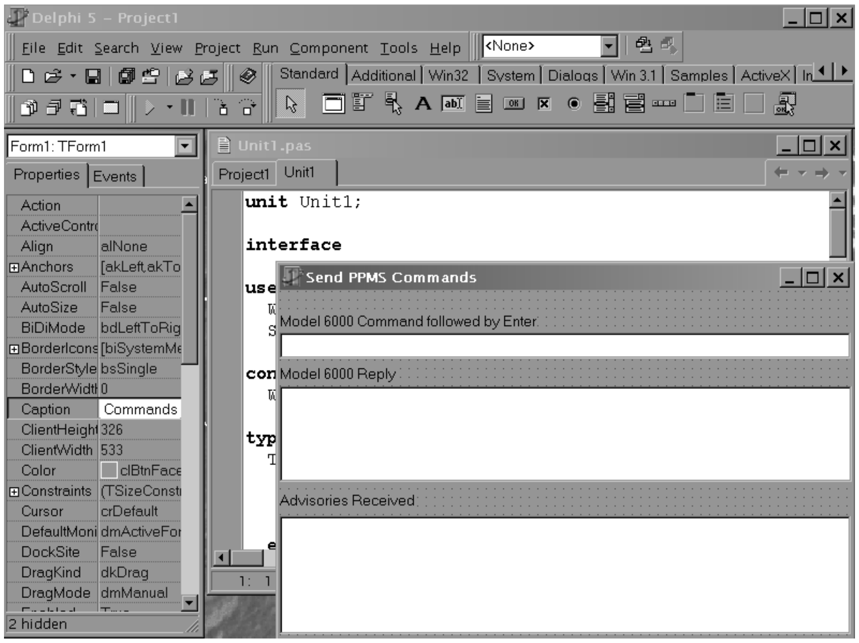
To introduce the process of receiving advisories to trigger Delphi programs, we will start with the generic "Hello World" program. In this situation, we will have the Model 6000 send out an advisory, which will trigger the Delphi Program to print a "Hello World!" text on the computer monitor. This program will be kept very simple so that you can quickly get an understanding of how the Delphi template program accepts advisories. To create this program, follow the step-by-step procedures given below. We will explain the details of the program after the procedures.

- Start Delphi.

  *Delphi will open with several windows on the Desktop. These windows include the Form Designer, Code Editor, Object Inspector and the Toolbar. Close the current Form (form1) and Code Editor (unit1.pas).*

- Select "Open Project" from the "Files" menu. Open the Quantum Design project1.dpr file that you downloaded from our website.

  *Delphi will respond by displaying the PPMS Commands Form and the unit1.pas template on the desktop. (See Figure 1).*


Delphi 5 - Project1
File Edit Search View Project Run Component Tools Help
<None>
Standard Additional Win32 System Dialogs Win 3.1 Samples ActiveX In
Form1: TForm1
Properties Events
Unit1.pas
Project1 Unit1
Send PPMS Commands
Model 6000 Command followed by Enter
Model 6000 Reply
Advisories Received
2 hidden

| | |
|---|---|
| Action | |
| ActiveContr | |
| Align | alNone |
| ⊞Anchors | [akLeft,akTo |
| AutoScroll | False |
| AutoSize | False |
| BiDiMode | bdLeftToRig |
| ⊞BorderIcons | [biSystemMe |
| BorderStyle | bsSingle |
| BorderWidth | 0 |
| Caption | Commands |
| ClientHeight | 326 |
| ClientWidth | 533 |
| Color | clBtnFace |
| ⊞Constraints | (TSizeConst |
| Cursor | crDefault |
| DefaultMoni | dmActiveFor |
| DockSite | False |
| DragKind | dkDrag |
| DragMode | dmManual |

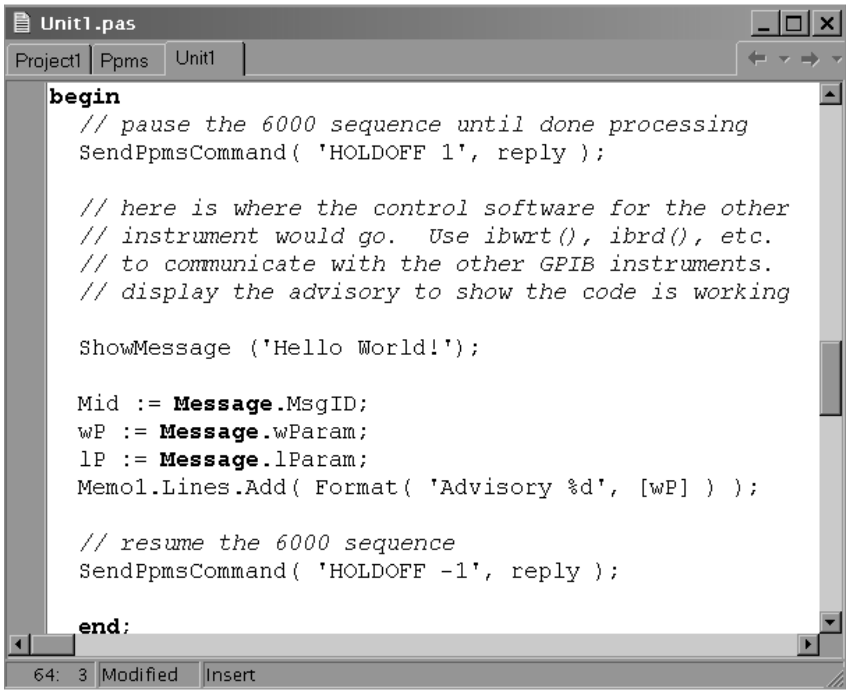
```
unit Unit1;

interface

use
```

**Figure 1**. Delphi template program.

- Click on the unit1.pas window. The Code Editor begins with the declaration of several variables and is set-up to receive advisories. As you scroll down through the code, you will get to the section where there is a holdoff command that will send a command to the Model 6000 to pause the sequence as it executes the application code for your external instrument. This feature gives the ability to put the Model 6000 in a stationary state while the application is being run. Following the holdoff command there is a comment block stating were to put the third party application code. This is where we will add the code for our “Hello World!” program. Type the following code in this section (*See Figure 2)*:

  ShowMessage (‘Hello World!’);

  Note: At the end of this section of code another command string is sent to the Model 6000 to release the holdoff. This will bring the Model 6000 out of its stationary state and it will continue the rest of the PPMS sequence. For a further description of Holdoff commands or other PPMS command strings, see PPMS GPIB commands in the User’s manual.



```
begin
  // pause the 6000 sequence until done processing
  SendPpmsCommand( 'HOLDOFF 1', reply );

  // here is where the control software for the other
  // instrument would go.  Use ibwrt(), ibrd(), etc.
  // to communicate with the other GPIB instruments.
  // display the advisory to show the code is working

  ShowMessage ('Hello World!');

  Mid := Message.MsgID;
  wP := Message.wParam;
  lP := Message.lParam;
  Memo1.Lines.Add( Format( 'Advisory %d', [wP] ) );

  // resume the 6000 sequence
  SendPpmsCommand( 'HOLDOFF -1', reply );

  end;
```

**Figure 2.** “Hello World!” program code.

- Run the program. There should be no errors and the run button will be grayed out indicating that the program is currently running.

- Go to PPMS sequence editor and create a single line sequence “advise 101”.

  *Under Advance Commands in the sequence command menu, select Advise. This will prompt you to enter an advisory number. Enter “101” as an advisory number.*

- Save the sequence as “Advisorytest”.

- Decrease the size of the PPMS MultiVu window so that you can see the Delphi windows also. Run the “Advisorytest”sequence.

When you run the sequence you will see the front panel to the PPMS model 6000 send out the command “Advise101”. Immediately you will see that the PPMS Command Form receives the advisory 101 and a window will appear with “Hello World!” on the computer monitor. Click the **OK** button to close the window and resume the sequence.

This simple program gives you the first step of understanding how the advisories work. First, we set up a program to display “Hello World!” in Delphi. The template program acts as an input device to receive advisories from the PPMS sequence through the server. This program also illustrates the input/output capabilities between Delphi and the PPMS server by sending individual holdoff commands to the Model 6000. We then set up a single line sequence in the PPMS to send out an advisory. When the template program received the advisory of 101, the program wrote “Hello World!” in the textbox. By including your application code in place of the “Hello World!” you can set up an applications program so that it will execute when the template program receives advisories.

**Goodbye World (Receiving Multiple Advisories)**

The following example shows you how events in the Delphi program can be triggered by different advisories. You can tailor the program and the properties of the template program so that only certain functions are triggered by one advisory while other functions are triggered by different advisories. This versatility in the use and adaptation of advisories gives the user the capability of creating complex programs using the PPMS sequence editor and Delphi programs.

- Close the PPMS Command Form to stop running the program.

- In the same section as the previous example, replace the "Hello World!" code in unit1.pas with the following code:

```
if wP = 101 then
   ShowMessage (Hello World!);
If wP = 102 then
  ShowMessage (Goodbye World!);
```

- Run the program.

- Now run the "Advisorytest" sequence.

  *The same will happen as in the previous example. The PPMS will send the Advise 101 and the "Hello World!" window will appear. Once you click* ***OK****, the window will disappear and the sequence will resume. Note: the code for "Goodbye World!" was not executed. The template program was written to display "Goodbye World!" only when it received an advise 102 from the model 6000. In this situation the template program only received an advise 101 which triggered the "Hello World!" text.*

- Go to the PPMS sequence editor and add "Advise 102" to the "Advisorytest" sequence.

```
Set advise number 101
Set advise number 102
End sequence
```

- Save and Run the Sequence. Again the "Hello World!" window opens when the advisory 101 is sent. Once you click **OK**, the PPMS will send the "advise 102" command, which will display another window, "Goodbye World!". Click **OK** to close the window and resume the sequence.

These two example programs illustrate the effect of using advisories. Advisories are essential in making the PPMS sequence work properly with the user created software. The combination of receiving advisories by the template program, the manual holdoff command to pause the model 6000 and the advisory capabilities of the PPMS sequence editor makes it possible to link the user's Delphi software with the automated capabilities of the PPMS.

**GetDat for data acquisition from the model 6000:**

The previous section showed the methods necessary to handle advisories from Delphi. The next example will bring the program closer to a true data acquisition program. In this example the "Hello World!" program will be re-written to query the data from the model 6000.

- Stop the unit1.pas program. Replace the "Hello World!" and the "Goodbye World" with the following code:

```
If wP = 101 then
        SendPpmsCommand ('GetDat? 2', reply);      //sends command string &
                                                    //retrieves data string
        Memo1.Lines.Add (reply);                    //display data string on form
        ShowMessage ('TEMP: ' + reply);            //display data string on window

If wP = 102 then
        SendPpmsCommand('GetDat? 4', reply);
        Memo1.Lines.Add (reply);
        ShowMessage('FIELD: ' + reply);
```

- Run the program.

- Run the "Advisorytest" sequence.

When the sequence is started, the Model 6000 sends the Advise 101 command. From the PPMS Command form, you will see that the program receives the advisory and then displays a string of three numbers separated by commas. Then a window will appear on the computer monitor showing "TEMP: " and the same string of numbers. The third number at the far right is the temperature of the PPMS. You can verify by comparing it with temperature shown on the Model 6000 front panel. The first two numbers are the bit flag (2) and the timestamp, which appear on all GetDat queries. When you press the **OK** button, the sequence resumes and sends the Advise 102 command. The PPMS Command Form will receive the advisory and display another string of data. A window will appear on the computer monitor with "Field: " and the same string of data. The third number is the present field measured in Oersteds. Click the **OK** button to end the sequence.

The data program demonstrates the use of a data window. Unlike the "Hello World!" program, which only printed the "Hello World!" message during the advisory, the Data program actually queried the Model 6000 for the system temperature and magnetic field state. This was done using the GetDat command. The type of data obtained from the model 6000 depends on the bit flag suffix you attach to the GetDat command. For example, the bit flag for the temperature is 2, while the bit flag for the field is 4. If you want to receive both values at once you can specify the bit flag of 6 to the GetDat. The list of bit flags specific to each data item is given in the PPMS User's manual. Remember that all responses will automatically include the bit flag and timestamp.

Once the Delphi program has received the data string from the Model 6000, you can manipulate it using the standard Delphi routines (Once again we suggest you get a book on Delphi if you are not familiar with the language). For your particular application, the data can be transferred to a more permanent variable (such as Temp and Field) and stored in a file using standard Delphi file access. In this way, you can receive data from other instruments via the GPIB while retrieving PPMS information using the server and commands (such as GetDat). For more information on the GetDat and other PPMS command strings, reference the PPMS User's manual.

## Interfacing to Other Instruments

### IEEE GPIB Bus

The PPMS uses a National Instruments GPIB board to provide communication to the Model 6000. The GPIB bus controller resides within the HP computer while the Model 6000 GPIB acts as a slave board for input/output or I/O (similar to any other third party instrument with GPIB capability). Thus, for third party instruments that are connected in parallel to the Model 6000, the user can interface to these instruments by using the standard methods for a National Instruments board.

To interface your software with third party GPIB instruments, you need to send the proper National Instruments commands from Delphi. The support of such an interface is provided by National Instruments using the Dynamic Link Library (DLL). This necessitates learning the command syntax of the third party instrument. To talk to an instrument, your application program must send the correct command strings to the appropriate instrument and have it receive the data string back from that instrument. If you have never used the National Instruments GPIB library to send command strings to outside instruments, you should start by reading the manuals for the National Instruments GPIB board (available from National Instruments) and the GPIB programming section in the manual for the device you wish to use (should be provided by the instrument's manufacturer). The National Instruments manual gives the standard command lines available for controlling the GPIB bus. Examples of such commands include "ibwrt" which sends a command string to an instrument and "ibrd" which receive information. You will also need to know the GPIB address of the instrument with which you wish to communicate. To familiarize yourself with sending National Instruments' command strings to an external instrument (such as a Hewlett Packard multimeter), connect the GPIB port of the instrument to the Model 6000 GPIB connection. Test the GPIB communications by sending single line commands to the instrument. For example, you might try to put the instrument into remote mode or retrieve data from it. When you have learned to control and retrieve data from your instrument using command strings via the GPIB, you can write the command strings into your application programs. Then by linking your application programs to the automated PPMS temperature and magnetic field control system, you can automate complex measurements and sequences. You should, of course, consult the programming section of your third party device to understand its GPIB protocol and instrument specific command strings. The user will still be responsible for creating the software, however with the template program as an example, you can create your own applications programs to receive advisories and communicate with the Model 6000.

**IEEE RS-232 Bus**

To interface the software to the third party RS-232 instrument, one would have to send the proper RS-232 command from Delphi. Windows using the file input/output library and the communications library provides the support of such an interface. This necessitates learning the command syntax of the third party instrument. To talk to an instrument, your application program must send the correct command string to the appropriate instrument and have it receive the data string back from that instrument. If you have never used the Windows communication interface to send command strings to outside instruments, you should start by reading the documentation for the communications library (available from Microsoft).

**Summary of PPMS links:**

This application note has shown you how to link Delphi programs to PPMS sequences using advisories. The Data program also showed how data could be retrieved from the Model 6000 by your applications program. Using these simple tools you can now write application programs that can be triggered from the PPMS advisories to retrieve and store relevant PPMS data into your own file. The PPMS sequence capability combined with the Delphi, Visual Basic, or the Visual C++ template programs and the GPIB interface give you the capability to create automated sequences for your own specific applications. By studying the PPMS User's manual, the Delphi, Visual Basic, or Visual C++ reference guides, and your instrument's GPIB manual, you should be able to write complex programs, which can link your measurement operations to the PPMS automated control system.

## Appendix

## Additional information regarding PPMS applications programming

### 1. Additional Information about GetDat

The GetDat query, unlike the Measure query, directly requests the system for information without placing it into the data memory buffer in the Model 6000. The GetDat command queries the Model 6000 for information and immediately sends this back to the server. In contrast, when a Measure command is sent to the Model 6000, it will create the appropriate string and store it in its internal memory, without sending it back to the server. Thus, the measure command is a specific type of GetDat, which stores the queried information in the internal buffers of the Model 6000. Independent of these specific queries, the Model 6000 is continuing to update its internal information so that when it is queried, this information is current. Some of this information is always being updated (such as the temperature) while others are only updated when they are configured (such as the user bridge channels). Thus, whenever a GetDat query is sent to the Model 6000, there should be no additional information filling up the Model 6000's internal memory buffer. This is why the data % on the front screen of the Model 6000 does not increase as you ask for more data using GetDat.

Depending on the type of GetDat query, one may run into a timing problem between the GetDats and the internal information updates to the Model 6000. A typical GetDat waits for the PPMS to update all appropriate information and then retrieves the information back to the server. Most of the internal information (such as temperature) updates relatively quickly. However other information (such as the user bridge signals) can take some time depending on the experimental conditions. For example, for the user bridge, there are three modes of operation; the fast mode, the standard mode and high-resolution mode. In the fast mode the Model 6000 will immediately respond to a query by giving whatever voltage is being read from the appropriate channel. In the standard mode, the Model 6000 will compare the sample signal with a set of internal precision calibration resistors for a more accurate reading. In the high-resolution mode, the Model 6000 will do several standard mode measurements and average the result. Furthermore, in the high-resolution mode, the number of measurements to average increases as the resistance of the sample decreases. Thus in the worst case scenario, if you have all three bridge channels measuring in the high resolution mode on samples having low resistances that are quickly changing (demanding better feedback current control), such measurements require some time to update (up to several seconds). This may result in a GPIB timeout and you may get a command error. Thus one should always be careful about the timing of the queries to the type of request given to the Model 6000.

## 2. IEEE-488.2 Recommendation

Depending on the type of instrument being connected to the GPIB bus, one may encounter problems with incomplete GPIB protocols from the third party instrument. Although many instruments state GPIB compatibility, not all GPIB capabilities are the same. The current accepted GPIB protocol is IEEE-488.2; the PPMS Model 6000 uses this protocol. Other instruments (especially older models) may only be following the IEEE-488.1 or older protocol. The result of having incomplete protocols may result in the lock up of the GPIB bus such that a complete GPIB reset becomes necessary to release the network. If you feel that you may be having GPIB problems with your program, you can test whether the problem is due to GPIB protocol by modifying your programs so that they are still performing communications to the Model 6000 but are not querying the third party instruments. You should also physically disconnect the instrument from the GPIB network. If the modified program runs, when the third party instrument is physically disconnected and masked from the program, then there may be problems with the third party instrument. If you have several instruments connected to the GPIB bus, you may want to do this with each instrument until you find the instrument at fault. Furthermore, there are several universal command strings, which all IEEE-488.2 instruments accept (such as *IDN?). If the instrument being used is an IEEE-488.2 system it should respond to these universal commands. In summary, if you interface instruments using IEEE-488.1 or older protocol, you should take care when writing your program for proper communication.

## 3. Command Strings and Sequences

When the Model 6000 is running a sequence, any resources being used by the sequence becomes unavailable for interactive operation. For example when the Model 6000 is performing a scan H sequence, the application program cannot send a command string to the Model 6000 to change the field. Conversely, any environment resources not being controlled by the sequence remains available to the user for interactive operation. Of course any read-only parameters (such as GetDat?) can be obtained using command strings, even when it is being used in the sequence.