MetaMatters

Volume 3
Issue 3
June 2011

The Scripting Issue

One of the powerful new features available in the new MetaMorph® NX Software is the integrated IronPython scripting environment. IronPython is Microsoft's open-source implementation of the Python programming language. With the scripting tool, you can write and execute code that automates or customizes operations in the MetaMorph NX Software. The scripting language can be used both to perform simple repetitive tasks or bring unique customized functionality to the MetaMorph NX Software interface. Scripts you write are portable and can be shared with other users.

In this issue of *MetaMatters*, we will show you a couple of examples of the scripting environment in action. Be sure to check out the MetaMorph NX Software on-line help for further information. In addition, you can find the scripts discussed in this issue on our knowledge base.

Current Events

Webinars

July 15, 2011: Dataset Management and Measurement in MetaMorph® NX Software

August 18, 2011: Introduction to IronPython in MetaMorph NX Software

Register for Webinars on-line here.

Promotions

We are currently offering some great incentives to upgrade to MetaMorph NX Software. See our incentive promotion page for details.

Twitter

For all the latest news and information fromMolecular Devices, follow us on Twitter at@Moldev.

Adding a Trigger Dialog with IronPython

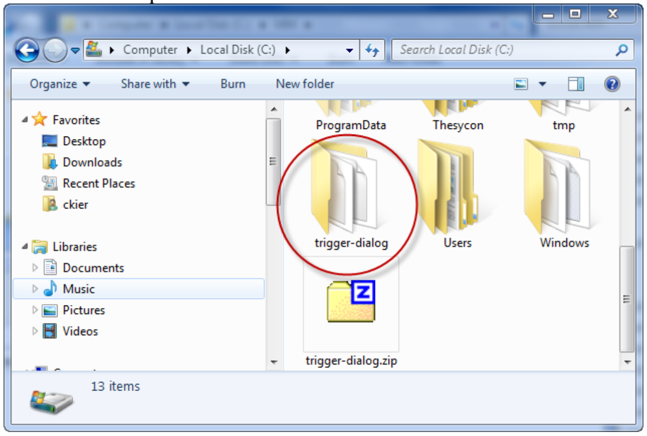
It is often desired to have simple triggering control over an external device. The device may be a micro-injector, voltage stimulator, liquid handler, and so on. To accomplish this you must have the ability to send an output trigger signal from the computer to the external device. To do this we will employ the inexpensive USB-6501 from National Instruments. This is a basic digital output module capable of generating the types of voltages we will need to trigger an external device.

Follow these steps to get started:

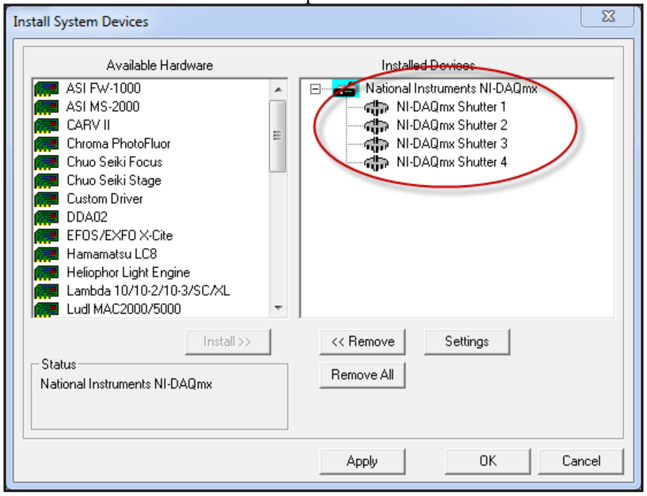
1. Install the USB drivers that came with the USB-6501.

2. Connect the USB-6501 to a USB port on your computer.

3. Download the IronPython script files from our support website (download files here). After downloading the files unzip them to a convenient location.



4. Follow the on-line help to setup the USB-6501. The driver to install is the "National Instruments NI-DAQmx". This will create four "shutter" outputs. We will use the shutter lines to deliver the pulse to the external device.


Install System Devices

Available Hardware

- ASI FW-1000
- ASI MS-2000
- CARV II
- Chroma PhotoFluor
- Chuo Seiki Focus
- Chuo Seiki Stage
- Custom Driver
- DDA02
- EFOS/EXFO X-Cite
- Hamamatsu LC8
- Heliophor Light Engine
- Lambda 10/10-2/10-3/SC/XL
- Ludl MAC2000/5000

Installed Devices

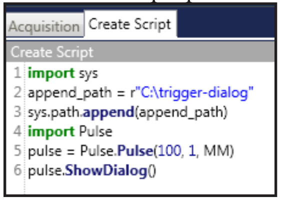
- National Instruments NI-DAQmx
  - NI-DAQmx Shutter 1
  - NI-DAQmx Shutter 2
  - NI-DAQmx Shutter 3
  - NI-DAQmx Shutter 4

Status: National Instruments NI-DAQmx

5. Select the "Script" tab in MetaMorph NX Software user interface, and press the "Load" button at the bottom of the window. Navigate to the folder location for the documents downloaded in step 3 above and select "run.py"

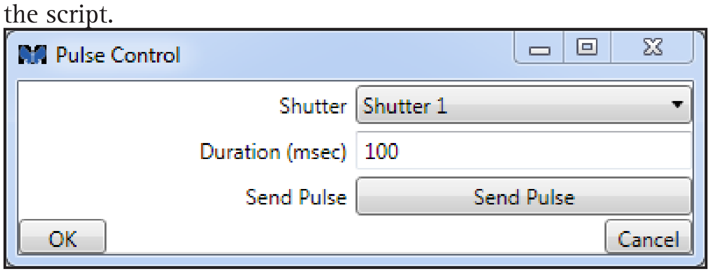
Molecular Devices

6. You will see the contents of the “run.py” script listed in the “Create Script” pane. Modify the path in line 2 to the location of the files you previously downloaded in step 3 above. Note, the letter “r” prepended to the path is required.



```
1 import sys
2 append_path = r"C:\trigger-dialog"
3 sys.path.append(append_path)
4 import Pulse
5 pulse = Pulse.Pulse(100, 1, MM)
6 pulse.ShowDialog()
```

7. Press the run button at the bottom of the window to start the script.



Shutter: Shutter 1
Duration (msec): 100
Send Pulse: Send Pulse

The dialog presents a couple of options. The drop-down menu allows you to select which shutter to use for sending the trigger output. The text field allows you to set the duration of the trigger pulse in milliseconds. Finally, the “Send Pulse” button is used to well... send the pulse.

For those interested, here is a line by line run down of the script statements and their function as shown in step 6.

Line 1: `import sys` is used to gain access to the sys library. The `sys` library provides a set of standard functions for manipulating the underlying file system.

Line 2: `append_path = r”C:\trigger-dialog”` sets the value of `append_path` to the location of the downloaded script files described above.

Line 3: `sys.path.append(append_path)` allows the downloaded files to be found by the embedded IronPython interpreter.

Line 4: `import Pulse` is used to gain access to the Pulse library. The Pulse library provides the functions we will use next to setup the “Pulse Control” dialog.

Line 5: `pulse = Pulse.Pulse(100, 1, MM)` creates a default pulse. You can open the file Pulse.py with a text editor for more details.

Line 6: `pulse.ShowDialog()` pops-up the dialog.

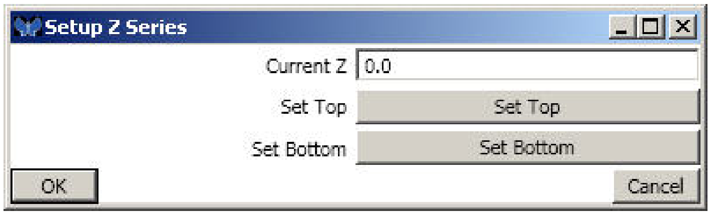
Scripts associated with this article can be downloaded from our knowledge base.

Building Dialogs with MetaMorph NX Scripting

Creating a Python Script that Displays a Dialog for Setting the Top and Bottom of a Z Series in the MetaMorph NX Software

To set up a Z series in MetaMorph NX Software, you must first locate the center focal position of the Z series and then define the range around it. However, this is not always easy to do. For example, your sample may not be evenly distributed around the center focal position. In this case, the Z series would be configured more accurately if you could locate the top and bottom of the Z series, and then let the software calculate the center and range of the Z series for you. Using the Python scripting tool, we can create this functionality. We can write a script that displays a dialog which lets the user set the top and bottom of the Z series, and then the software can use that information to calculate the center focal position and range.

The z-series script and the DialogUtilities module it uses can be downloaded from our knowledge base. To follow along with the article it may be helpful to download these files. The script is also reproduced here on page 5.



Current Z: 0.0
Set Top: Set Top
Set Bottom: Set Bottom

Figure 1: Input dialog generated by the z-series script.

How the Script Works

At first glance, this script may seem complicated. However, the way the script works is summarized in the six steps that are labeled a through f:

a. We wanted the user to be able to see the changes in focus, so the first thing we did was to call the `SetupUserInterface()` function which sets up the ribbon and starts a Live acquisition.

b. Next, we wanted to ask the user to locate the top and bottom of the Z Series. To do this, we used the `GatherTopAndBottom()` function which displays a dialog to gather the information and then records the results in a variable. The next section of this article describes how to create the dialog that this function uses.

c. To undo any changes we may have made to the user interface in step a, we called `CleanUpUserInterface()` to reverse those changes (in this case, we are stopping the Live acquisition).

d. With the user input collected, we can start to set up the Z series. After retrieving the `zTop` and `zBottom` out of the results variable, step d uses those values to calculate the `zRange` and `zCenter`. This is simple math performed in the `CalculateZRange()` function.

e. As mentioned earlier, the MetaMorph NX software sets up a Z series based on the center position and range of the Z series. Now that we have that information we can use the ConfigureZSeries() function to provide appropriate values for the software to use. The ConfigureZSeries() function contains an example of how the software uses these values.

f. It is often good practice to provide some feedback on how the script executed. Step f uses the built-in MM.Print() function to print the Z Series parameters used in the script in the script's Output text area.

How to Create the Dialog

In Step b above we used the GatherTopAndBotton() function to display a dialog that asks the user for input. To help script writers generate highly functional dialogs without a lot of effort, we created a Python module called 'DialogUtilities'.

Downloading the DialogUtilities module

Before you can use the DialogUtilities module, you must place the module file in the correct directory on your hard disk. The module should be placed in the MetaMorph NX Software installation directory. This is usually C:\ProgramFiles\Molecular Devices\MetaMorph NX\. To use the module you must import it into the script using the import statement.

```
import DialogUtilities
```

Implementing the dialog used in the z-series script

In the z-series script, we wanted three inputs in the dialog:

- A field that displays the current Z focus position, allows the user to set the focus position, and is updated as the focus changes
- A button that records the current Z position to be used as the top of the Z series
- A button that records the current Z position to be used as the bottom of the Z series

Creating a dialog for user input involves four steps. These steps are labeled 1 to 4 in the GatherTopAndBottom() function. The steps for creating the dialog used in the z-series script are as follows:

1. Create a DialogInputs object.

```
inputs = DialogInputs()
```

DialogInputs is a collection of the values you want to obtain and their types. You create the DialogInputs before you display the dialog to tell the dialog which fields you want and how you want them displayed. After the user closes the dialog, the same DialogInputs object will contain the results (the values selected or entered by the user).

2. In the DialogInputs collection, add fields for the inputs that you want the user to provide. An input represents one field on the dialog. You need to provide a name, type, and default value for the field.

```
inputs["Current Z"] = ("Current Z",
DialogInputTypes.LinkedNumber, MM.StageAndZ.ZPosition)
```

- The name in this example is "Current Z" and it is located both in the square brackets to the left of the equal sign and the inside the parentheses to the right. The name is shown on the label in the dialog.
- The type of the input is the second parameter on the right of the equal sign – in the example above it is DialogInputTypes.LinkedNumber. A linked number is a field that stays in synch with a value from the MetaMorph NX software. That is, the software immediately updates the value in the field as the value in the software changes. Other DialogInputTypes are discussed later in this article.
- The final parameter to the right of the equal sign is the "default value." What this value is depends on the type of the input. For a LinkedNumber type of input, the value should be a MetaMorph NX Property and it is the value that is kept in synch between the input window and the underlying application.

In addition to the above example input, two DialogInputTypes.Button fields were added to the dialog to display buttons for setting the top and bottom of the Z Series.

3. Display the dialog and wait for the user to provide values.

```
inputMade = DialogUtilities.GetUserInput(inputs,
"Setup Z Series", HandleErrors)
```

The DialogUtilities.GetUserInput() function displays the dialog with the inputs you define. It takes three parameters: the DialogInputs object configured in steps 1 and 2, a title to display on the input dialog, and a function used to handle errors if any occur (the example z-series script includes a minimalist version of a HandleErrors() function). The GetUserInput() function waits for the user to close the dialog, and will return False if the user closed the dialog without pressing the OK button.

4. Use the results. Typically, the values entered by the user would be available via the DialogInputs object you created in steps 1 and 2. For example:

```
finalZPosition = inputs["Current Z"].Value
```

In this example we extracted the value stored in the input using the input's name in the square brackets ("Current Z") and using .Value to retrieve the value the user provided. In the z-series script we held the requested data a different way because we used buttons to set the top and bottom of the Z series instead of input fields.

There are a number of different types of inputs that we can gather from the user. The DialogInputTypes class defines the types of inputs and fields that you can use with the DialogUtilities module. These types are as follows:

- DialogInputTypes.String is for simple text input. The user will be presented with a text box to type in a value. The default value (third parameter when defining the input in step 2) would be String which would populate the field before the user types anything. The default value is optional. The returned value will be the text that is displayed in the dialog when the user closes the dialog, or an empty string if no text was displayed.
- DialogInputTypes.Number is for input of any type of number (integer or floating point). The Number field will allow any numerical field to be entered, and is displayed as a text box that does not allow letters or special characters to be

entered. The default value would be a number which would populate the field before the user inputs any numbers. The default value is optional. The value returned when the dialog closes is the number that is displayed when the dialog closes.

- `DialogInputTypes.Checkbox` is for an On/Off or True/False option. The user will be presented with a simple check box which can be selected or cleared. The default value would be True or False, and if True would start the dialog with the check box selected. The default value is optional. The value returned will be True if the check box was left selected when the dialog was closed or False otherwise.

- `DialogInputTypes.SingleSelectList` is used to allow the user to select one of several options. It presents the list to the user as a "combo" box. The default value is a list of strings and defines what values should appear in the list of options. The default value is optional, and if not present the combo box will be empty. The returned value will be a string with the name of the option the user had selected when the dialog closed.

- `DialogInputTypes.MultiSelectList` is used to allow the user to select more than one of several provided options. It presents the list as a text area, with each option displayed as a separate, selectable line (you can use Shift+Click or CNTRL+Click to select multiple options). The default value is a list of strings to show as selectable options. The default value is optional and if not present will display no options. The returned value will be a list of all the strings for each option the user selected, or an empty list if none were selected when the dialog closed.

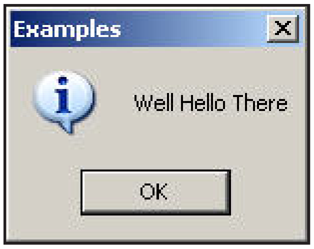
- `DialogInputTypes.Button` is used to provide a button to the user which will perform some action when the button is pressed. The text on the button will be the same as the name of the input. The default value needs to be a function which will be called when the button is pressed (it defines the "action" that will be performed when the user presses the button). The default value is *mandatory*, and an error will occur if the function is not present. There is no return value for a button input (the value will return the function passed in as the default value).

- `DialogInputTypes.LinkedNumber` is used to display a number in the dialog which will be kept in synch with a value in the underlying application. When the value in the application changes, the value that is displayed in the text box will also change. Likewise, when the user changes the value in the dialog, the value changes in the underlying application. The dialog allows only numerical input (including decimal and negative numbers). The default value is the MetaMorph NX Property which holds the value to be synched. The default value is *mandatory* and will result in errors if not present. The return value will be the same Property which the input is synched with (not the value of the Property, but the Property itself).

Using the Other Dialogs in the DialogUtilities Module

In addition to a dialog for gathering user input, the DialogUtilities module has several other dialogs for informing or alerting users.

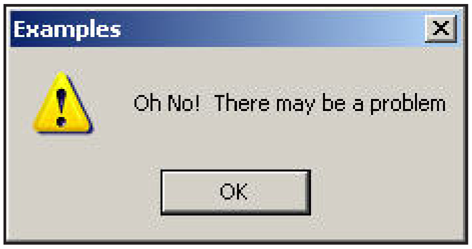
`DialogUtilities.ShowMessage()`: Displays a simple message to the user and an OK button.



Well Hello There

```
DialogUtilities.ShowMessage("Well Hello There", "Examples")
```

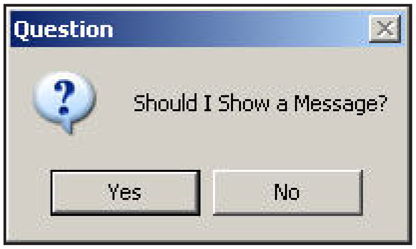
`DialogUtilities.ShowWarning()`: Displays a message with a warning icon to indicate its importance.



Oh No! There may be a problem

```
DialogUtilities.ShowWarning("Oh No!  There may be a problem", 
"Examples")
```

`DialogUtilities.ShowYesNo()`: Displays a dialog which allows the user to answer a simple Yes/No question. The response is returned as a True (if the user clicks Yes) or False (if the user clicks No or exits the dialog without clicking Yes or No).



Should I Show a Message?

```
response = DialogUtilities.ShowYesNo("Should I Show a Mes-
sage?", "Question")
if response:
DialogUtilities.ShowMessage("Here is the Message!", "Answer")
else:
MM.Print("No message shown.")
```

Example: Set Up Z Series Script

```
# This script uses the DialogUtilities module to get user input
import DialogUtilities

# Specifically, we want to use the DialogInputs and DialogInputTypes
# from inside the DialogUtilities module.
from DialogUtilities import DialogInputs, DialogInputTypes

'''
Display a dialog which allows the user to adjust the Z position to record
for the Top and the Bottom Of the Z series.  The Z motor can be changed
either via software or hardware.

Returns the Top and the Bottom values (Top first, Bottom second) or an
empty Tuple if the user canceled.
'''
def GatherTopAndBottom():
    # This list will hold the Z Positions for the Top and Bottom.
              # Defaults to current location
    zPositions = [MM.StageAndZ.ZPosition.Value,
                  MM.StageAndZ.ZPosition.Value]

    # These functions will be used to set the top and bottom values when
    # buttons get pressed.
    def TopSet():
        zPositions[0] = MM.StageAndZ.ZPosition.Value
    def BottomSet():
        zPositions[1] = MM.StageAndZ.ZPosition.Value

    #####
    # 1.  Create the DialogInputs object
    #####
    inputs = DialogInputs()

    #####
    # 2.  Add Inputs (user controls) for user input
    #####
    # Display the current Z position
    inputs[“Current Z”] = (“Current Z”, DialogInputTypes.LinkedNumber,
                                    MM.StageAndZ.ZPosition)

    # Create two buttons, one to set the top (using TopSet()
    # to do the work) and the other to set the bottom.
    inputs[“Set Top”] = (“Set Top”, DialogInputTypes.Button, TopSet)
    inputs[“Set Bottom”] = (“Set Bottom”, DialogInputTypes.Button,
                               BottomSet)

    #####
    # 3. Display the dialog to get user input
    #####
    inputMade = DialogUtilities.GetUserInput(inputs, “Setup Z Series”,
                                              HandleErrors)

    #####
    # 4. Use results.
    #####
    if inputMade:
        # If the user presses OK, return Top and Bottom
        return zPositions[0], zPositions[1]
    else:
        return ()

'''
This function is used to set up the GUI in preparation to allow the user
to setup the Z series. It will make sure Ribbons are in the correct
position and will start live mode, for example.
'''
def SetupUserInterface():
    MM.Acquisition.AcquisitionMode.Value = 0
    MM.ZSeries.IsSeriesUsed.Value = True

    UI.Ribbon.SelectActiveTabByName(‘Mode: Multidimensional’)
    MM.Camera.StartLiveMode()

'''
Undoes any changes to the User Interface which need undoing after the
script
runs, for example it stops live mode.
'''
def CleanUpUserInterface():
    MM.Camera.StopLiveMode()

'''
Calculates the center and range of the Z Series, and provides the two
values needed to configure NX

Returns the Range and Center of the Z Series (Range first, Center second)
'''
def CalculateZRange(zTop, zBottom):
    if zTop > zBottom :
        zRange = (zTop - zBottom)
        zCenter = (zTop - (zRange/2))
    else :
        zRange = (zBottom - zTop)
        zCenter = (zBottom - (zRange/2))

    return zRange,zCenter

'''
Setups appropriate MM NX Properties for the Z Series
'''
def ConfigureZSeries(zRange, zCenter):
    MM.StageAndZ.ZPosition.Value = zCenter
    MM.ZSeries.Range.Value = zRange

'''
This example error handler simply re-raises any error which may occur so
it gets handled by NX.
'''
def HandleErrors(error):
    raise error

'''
Note: The rest of this code is not in a function call, which means it
gets executed automatically when the script is loaded.  We use it to
define the order in which the other functions are called.
'''

# These lines will handle the User Interface related to getting
# the Top and Bottom of the Z Series by calling the appropriate
# functions.
SetupUserInterface()                                          # a.
results = GatherTopAndBottom()                                # b.
CleanUpUserInterface()                                        # c.

#If the user pressed OK on the dialog configure the Z series with the
# appropriate range and center.
if results:
    zTop, zBottom = results
    zRange, zCenter = CalculateZRange(zTop,zBottom)           # d.
    ConfigureZSeries(zRange,zCenter)                          # e.
    MM.Print(“(Top,Bottom)  (Range,Center):
            (“ + str(zTop) + “,” + str(zBottom) + “)
            ( “ + str(zRange) + “,” + str(zCenter) + “)”)     # f.
```

MetaMorph NX

Molecular Devices

Contact Us
Molecular Devices Inc.
402 Boot Road
Downingtown, PA 19335
USA

Phone Toll Free:	(800) 635-5577
Phone Intl.:	(610) 873-5610
Fax:	(610) 873-5492
Support:	(800) 635-5577 x1820 (408) 747-1700 x1820

On the web: www.moleculardevices.com

Sales:	meta.admin@moldev.com
Orders:	om-meta@moldev.com
Support:	support.dtn@moldev.com
Training:	training.dtn@moldev.com

MetaMorph® Software… making imaging easy!

Important Web Links

Product Information:	
Overview of available MetaMorph Software products	http://www.moleculardevices.com/Products/Software/Meta-Imaging-Series.html
Comparison of available MetaMorph Software products	http://www.moleculardevices.com/Products/Software/Meta-Imaging-Series/Comparison-Table.html
Application modules available for MetaMorph Software	http://www.moleculardevices.com/Products/Software/Meta-Imaging-Series/Application-Modules.html
List of distribution partners	http://www.moleculardevices.com/Company/Who-We-Are/Contact-Us/Distributors/Meta-Morph.html
Support:	
MetaMorph Software updates and upgrades	http://www.meta.moleculardevices.com/updates/
MetaMorph NX Software updates and upgrades	http://www.meta.moleculardevices.com/updatesnx
MetaMorph Software supported hardware database	http://support.meta.moleculardevices.com/hardware/hardware.php
MetaMorph Software supported operating systems	http://support.meta.moleculardevices.com/info/os-compatibility.php
Technical and application notes for MetaMorph Software	http://mdc.custhelp.com/app/home
Back issues of MetaMatters newsletter	http://mdc.custhelp.com/app/answers/detail/a_id/18689
Information and registration for MetaMorph Software training courses and webinars	http://www.moleculardevices.com/Support/Training-Center/Research-Imaging.html

Upcoming Training, Courses, and Exhibitions

Exhibitions

November 12-16, 2011
Society for Neuroscience Annual Meeting
Washington, DC

December 3-7, 2011
ASCB Annual Meeting
Denver, CO

Training

July 15, 2011
Webinar: Dataset Management and Measurement in MetaMorph NX Software

August 19, 2011
Webinar: An introduction to IronPython in MetaMorph NX Software

September 19, 2011
MetaMorph NX Software Introductory Training Course
Downingtown, PA

September 20-21, 2011
Metamorph® Software Basic Training Course
Downingtown, PA

September 22-23, 2011
Metamorph Software Advanced Training Course
Downingtown, PA

September 30, 2011
Webinar: TBD
November 18, 2011
Webinar: TBD

Supported Courses

October 11-21, 2011
OMIBS
MBL, Woods Hole, MA

October 19-November 1, 2011
In-situ Hybridization & Live Cell Imaging
CSHL, Cold Spring Harbor, NY

FOR RESEARCH USE ONLY. NOT FOR USE IN DIAGNOSTIC PROCEDURES.