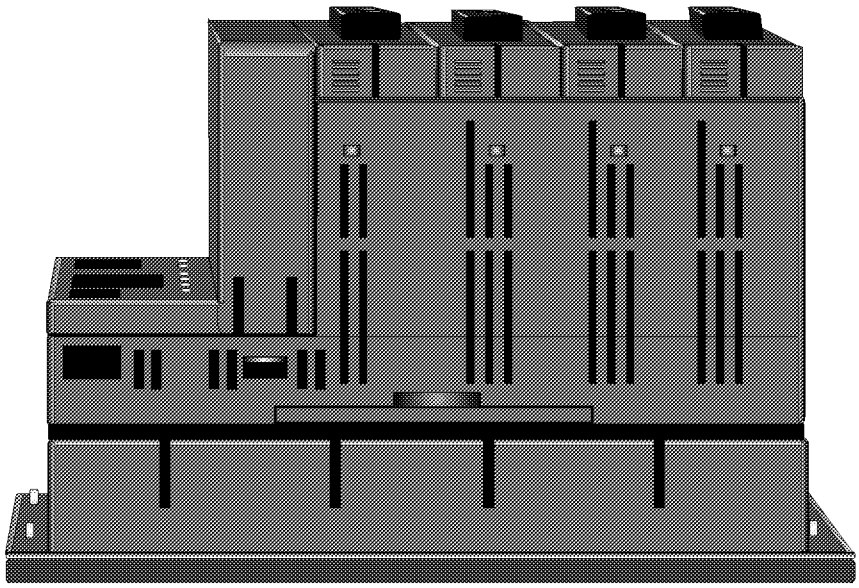
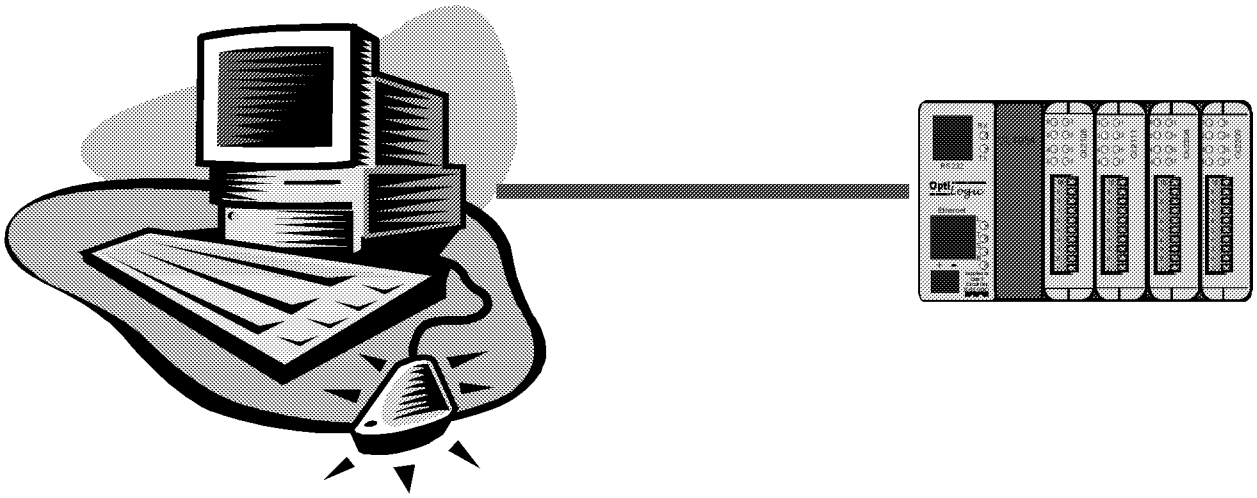
Opti*Logic* Series



Opti*Logic*

Software Interface Definition

(for use with Visual Basic)



Optimation

Optimal Automation for Industry

Optimation, Inc.
www.optimate.com
(256)883-3050

WARNING

Thank you for purchasing industrial control products from Optimation, Inc. We want your new system to operate safely. Anyone who installs or uses this equipment should read this manual (and any other relevant publication) before installing or operating the system.

To minimize the risk of potential safety problems, you should follow all applicable local and national codes that regulate the installation and operation of your system. These include the National Fire Code, National Electric Code, and other codes of the National Electrical Manufacturer's Association (NEMA). There may be local regulatory or governmental offices that can help determine which codes and standards apply to your situation. It is your responsibility to determine which codes should be followed, and to verify that the equipment, installation, and operation is in compliance with the latest revision of these codes. If you have any questions concerning the installation and operation of Optimation products, please call us at (256)883-3050.

All Optimation products are warranted against defects in materials and workmanship for a period of one year from the date of shipment. Warranty applies to unmodified product under normal and proper use and service. Optimation's sole obligation under this warranty shall be limited to either, at Optimation's option, repairing or replacing defective product. The cost of freight to and from Optimation will be borne by the customer. No other warranty is given or implied.

This publication is based on information that was available at the time it was printed. We constantly strive to improve our products and services, so we reserve the right to make changes to the products and/or publications at any time without notice and without any obligation. This publication may also discuss features that may not be available in certain revisions of the product.

Trademarks

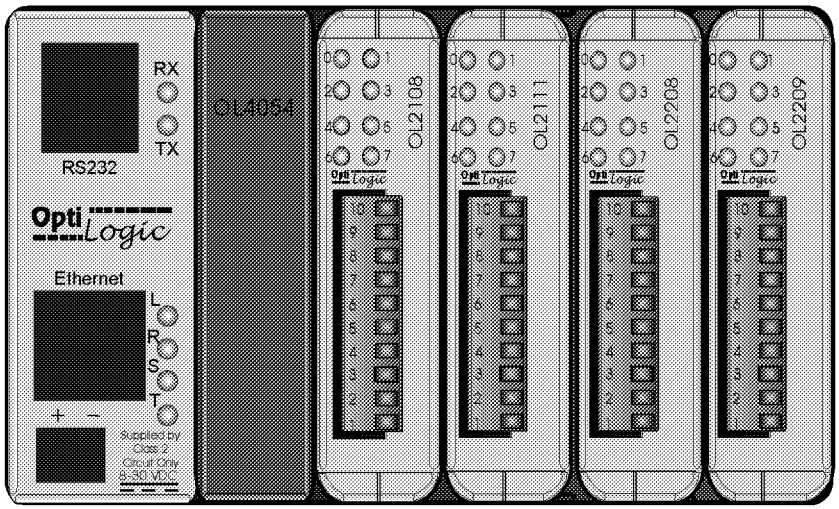
This publication may contain references to products produced and/or offered by other companies. These products and company names may be trademarked and are the sole property of the respective owners. Optimation disclaims any proprietary interest in the marks and names of others.

Table of Contents

Revision History

Issue	Date	Pages	Description
1.0	11/99	1 - 94	Original release
1.1	04/01	multiple	Added ReadLatchedInput, ReadOuput, OL2258 and OL2304

OptiLogic Software Interface Definition



Introduction

Optimation's OptiLogic™ remote terminal units provide point of use I/O and operator panel capabilities with a high speed link to a PC. They are designed to be a flexible, high performance, low cost I/O subsystem for PC based data acquisition and control systems.

One of many available OptiLogic operator panels

OptiLogic ethernet remote terminal units are designed to be easily integrated for use with user application software. Standard Win32 DLLs (dynamically linked libraries) are available from Optimation, which will allow user programs to interface OptiLogic RTUs over ethernet connections through a simple set of program calls. This document defines this interface and provides some simple examples to help you get started.

OptiLogic RTUs are modular in design. They allow you to plug together any combination of analog and digital inputs and outputs that will fit in the available slots. The card cage base snaps onto standard DIN rail for back panel mounting. If an operator panel is required, the base snaps onto any of a variety of available OptiLogic operator panels - which can, in turn, be panel mounted. The ethernet connection provides a 10BaseT (10 MBPS) connection to the network.

The DLLs are designed to make the low level details of the ethernet system operation transparent to the application programmer. By use of these DLLs, you will be able to plug your computer and associated OptiLogic RTUs into an ethernet link and deal with operation on a strictly logical level. Standard function calls are available for initialization, RTU identification, direct I/O calls, buffered I/O operation and network tuning.

The following pages should provide all the information necessary for a software professional to integrate OptiLogic RTUs into an application.

General Overview

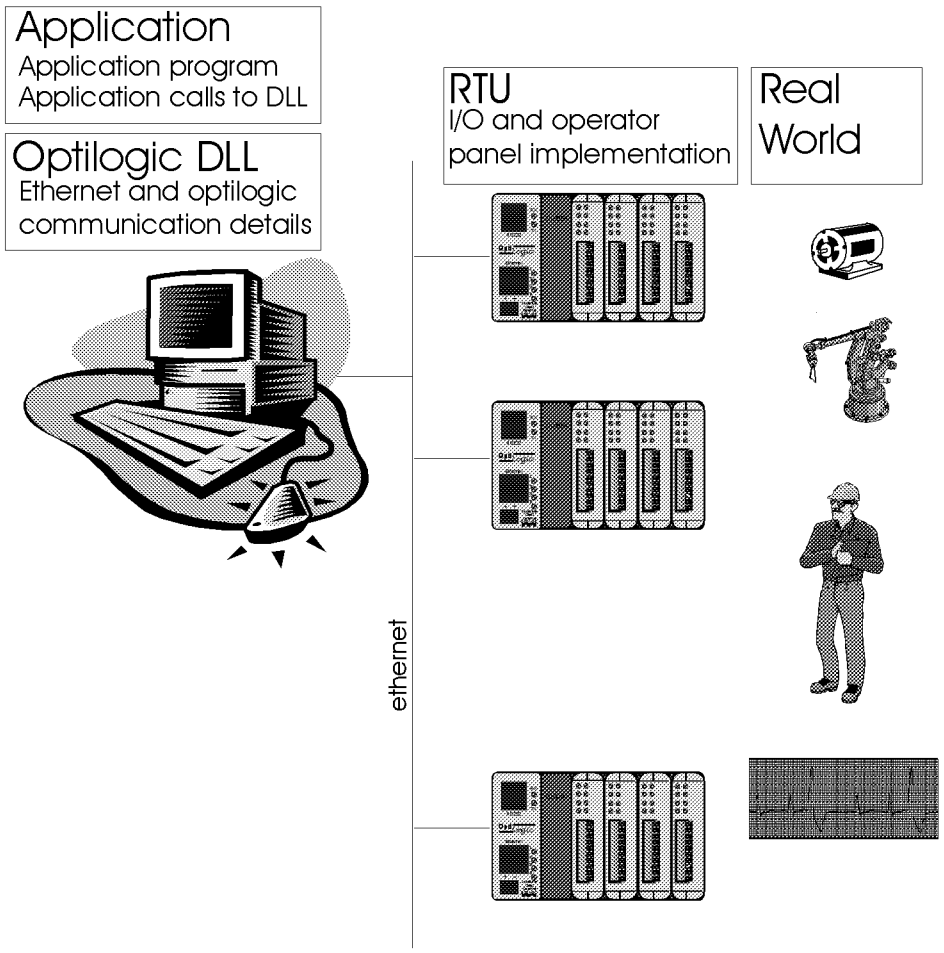
Before we discuss details of the operation, we'll look at the basic system configuration and architecture. The figure below illustrates a typical system.

From a system designer's perspective, the two important areas of the figure are the application program, running on the PC, and the real world. Everything in between is just details. The OptiLogic system is designed to allow you, the system designer, to focus on just those two areas. Everything else is transparent. The details are handled by the OptiLogic DLL, the OptiLogic RTUs and the data link (cabling, ethernet board, etc.).

The OptiLogic DLL is divided into the following five basic function classes.

- **Session group** - Established network communications
- **Node group** - Identifies the attached nodes and the modules available in each.
- **Immediate mode I/O group** - Immediate command and polling operation with modules plugged into an OptiLogic RTU's card cage.
- **Immediate mode Operator Panel group** - Immediate command and polling operation with OptiLogic Operator Panels
- **Buffered mode I/O group** - Interface between the application program and the ethernet communications which uses a buffer in the PC. Recommended for larger applications.
- **Buffered mode Operator Panel group** - Buffered mode for operator panels.
- **Housekeeping group** - Handles errors, timeouts, etc.

The following pages detail each of these groups of functions.



OptiLogic System Builder CD

The OptiLogic System Builder CD contains a number of tools, files and examples designed to bring you on line quickly. Included on this CD are the following.

- A **Setup** utility which installs the OptiLogic DLL and a test utility that can be run to verify communications with OptiLogic remotes and allow you to operate any of the RTU's modules.
- A **Sample** directory which contains header files that must be included in your C++ or Visual Basic application and example code in C++ and Visual Basic

Setup Utility

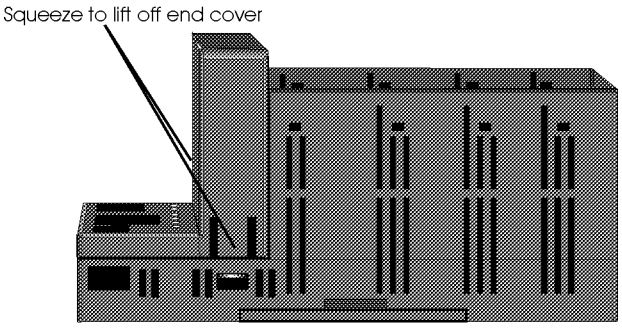
The Setup utility performs the following tasks.

- Installs Visual Basic support files in the Windows directory
- Installs the "OptiLogicProtocol.DLL" in the Windows System directory
- Creates a directory "c:\Program Files\OptiLogic QuickStart" - which will be used for the test utility. (The user has the option of selecting a different directory name and path)
- Installs OL_QuickStart.exe in the directory just created
- Adds uninstall information to the system registry
- Adds an icon called OptiLogic QuickStart to the programs menu

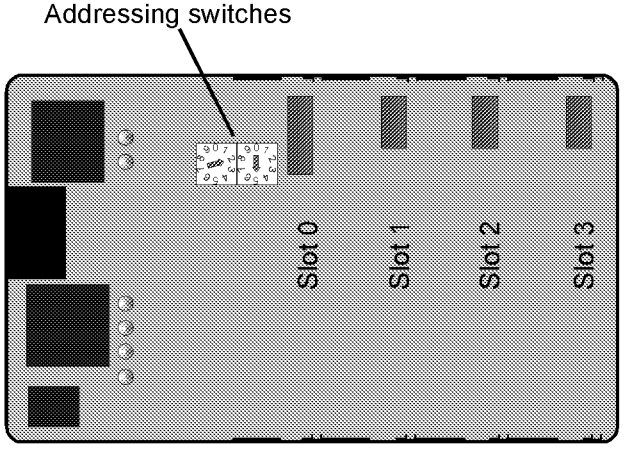
Now you are ready to run the test on an OptiLogic RTU. You also now have the DLL installed for use by your own programs.

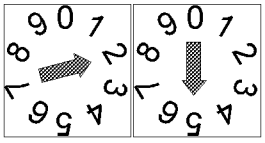
OptiLogic QuickStart

The first thing you should do, before running the OptiLogic QuickStart, is to set the RTU address. To do so, remove the snap-on end cover from the card cage by pressing the top and bottom latches in and lifting the cover off. This will expose two rotary



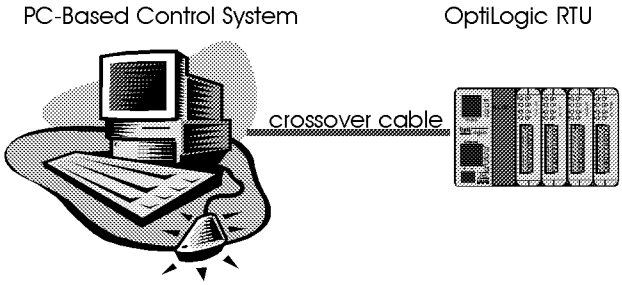
address switches. The switch on the left is the 'tens' digit. The switch on the right is the 'ones' digit. Dial in your desired address (we recommend a low number for the first test).



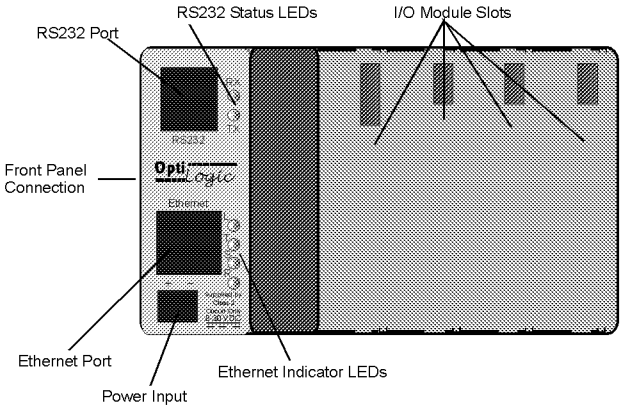


Exploded view

Next you should have the OptiLogic RTU connected either directly to your PC's ethernet board (through a crossover ethernet cable) or through a hub to the ethernet board (through standard, uncrossed ethernet cables). Power (12VDC or 24VDC) should be applied to the OptiLogic RTU power connector.



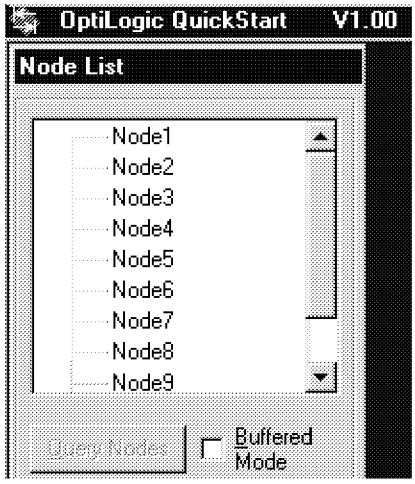
After a two-second start up check, two LEDs will indicate operational status. The "S", or Select LED next to the ethernet connector, should be on - indicating that the RTU is powered on, the firmware is executing, and the processor is talking to the ethernet circuit. The "L", or Link LED, should be on - indicating the ethernet connection is good and the RTU is receiving a period "link" pulse. If the "S" LED is not on, check power. If power is OK call Optimation tech support at (256)883-3050. If the "S" LED is on and the "L" LED is not, you do not have a good ethernet connection. Check your cabling, hub, and PC ethernet board and setup.



Now you're ready to run the test software.

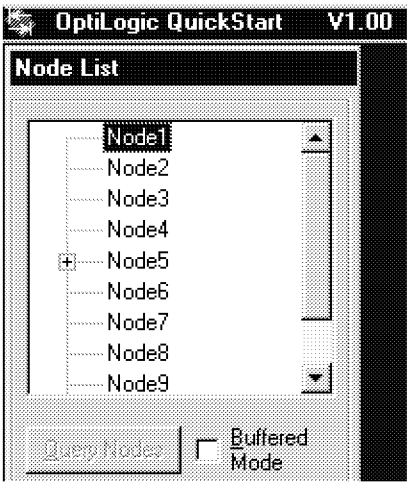
The test program OptiLogic QuickStart can be selected through program selection via the Windows START button.

When OptiLogic QuickStart starts up, the screen will display a window like the one shown below.

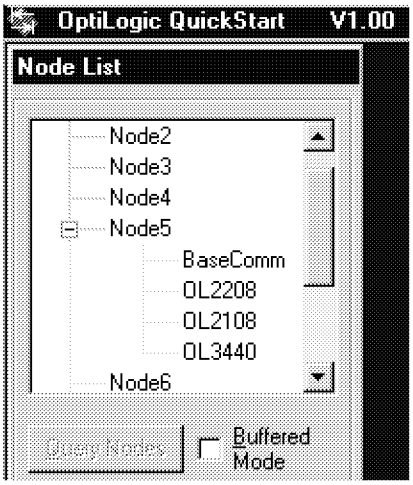


Select "Query Nodes". This will start the process of finding out what is attached. The following two screens will allow you to define the range of addresses you want to query. You can select any address range between 1 and 99.

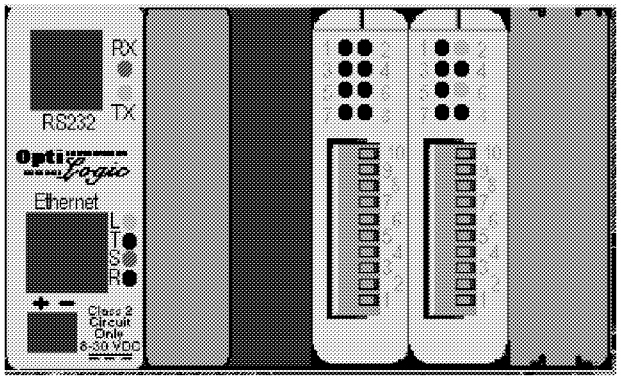
While the query is being performed, you should see the RTU's Receive light come on for every RTU address being polled. The RTU's Transmit light will flash when it receives it's own address. At the end of the poll a screen like the one shown below will be displayed.



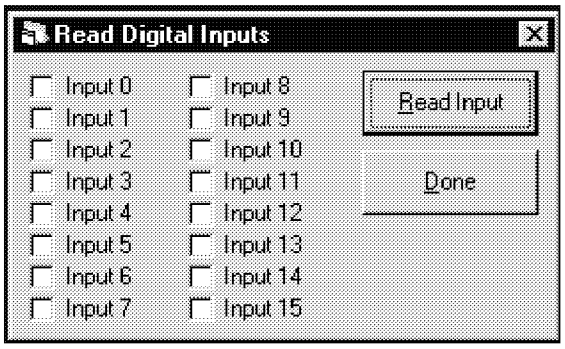
Notice the'+' sign by address 5. A plus sign will display next to every address in the selected range that has responded and has I/O modules or operator panels. When you click on the node, the tree will expand to show you which I/O boards and



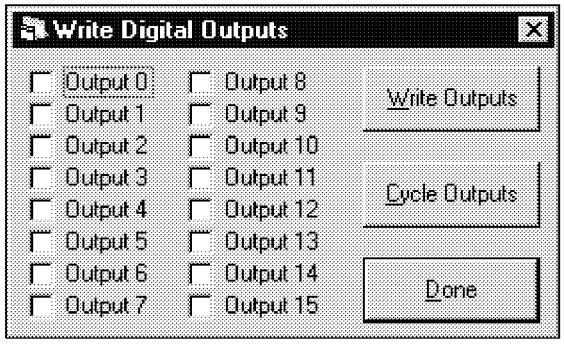
operator panels are present. A separate display, similar to the one shown below will show you a pictorial view of the node's card cage.



Click on an I/O board (try the OL2208 digital input board). A screen like the one shown below will be displayed. For digital I/O, 16 points will be shown regardless of the number that are actually on the board. For example, for an 8 channel digital input board, only the first column of input indicators has any meaning. Every time you click the "Read Input" button, the status indication for the inputs will be updated. When you are done, click the "Done" button.

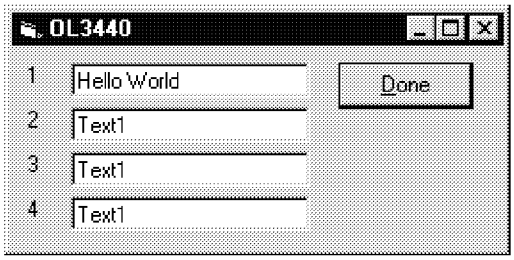


Click to select a digital output board. A display like the one shown below will come up.



To write to an output, click the selected point, or points. All click boxes that are checked will be activated the next time you click the "Write Outputs" button. To cycle through all of the outputs, click the "cycle outputs" button (don't do this for long with relay boards - you'll wear out the relays). Again, click the "Done" button when you are done.

For an operator panel, click it to select. Depending on the type of operator panel plugged in, a display similar to the one shown below will come up.



To write text to an LCD display, simply click your cursor on the required line and type. Other operator panel functions are similar to those touched on for the I/O boards. They are also pretty self explanatory.

Poke around with this test software and get a feel for how this system operates - and how fast!

Functional Overview

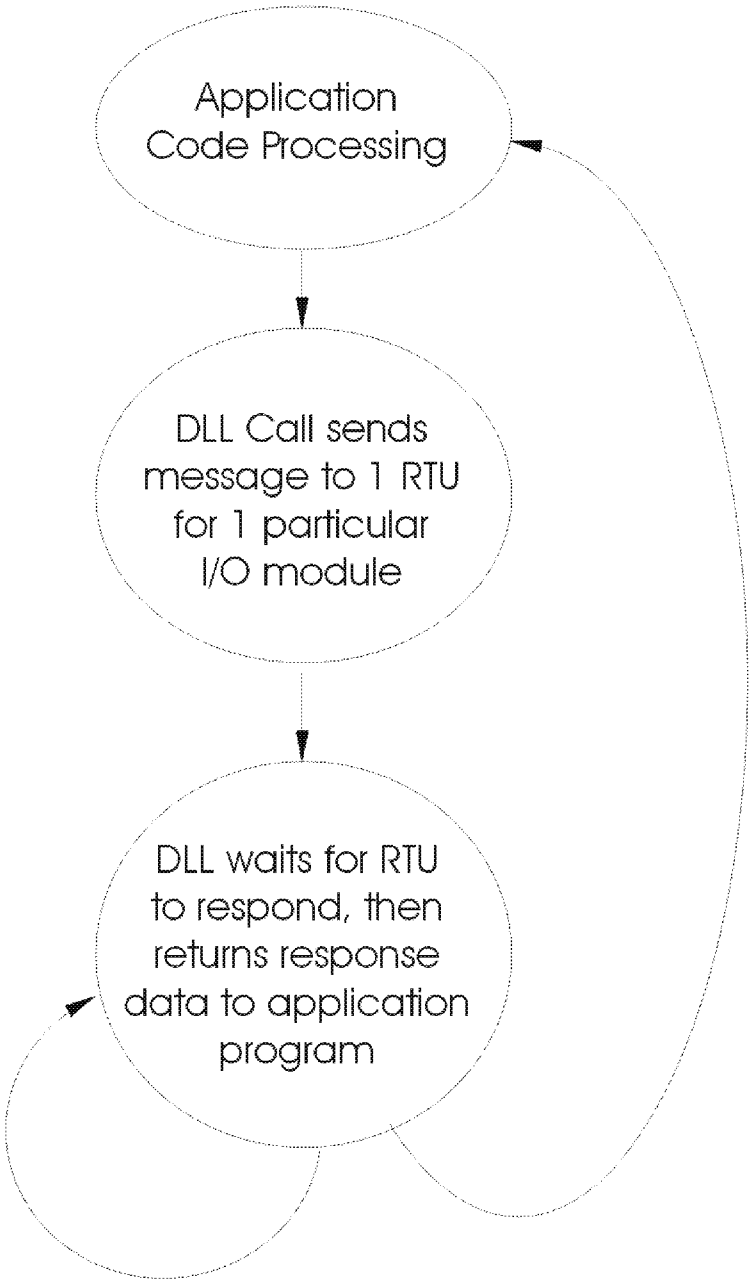
The OptiLogic DLL is a tool that you can use to interface your program to one or more OptiLogic RTUs. The DLL allows you to choose one of two basic modes of operation - immediate or buffered. The particular mode that you use depends on the application. It is even possible to use both modes - **but be careful!!!**

Immediate Mode

The figure at the right illustrates immediate mode operation. At any point in your program you may issue a call to an OptiLogic DLL function to read or write to a particular module at a particular RTU. The DLL will immediately transmit a message and wait for a response. The typical time involved is 2-6 milliseconds (may vary based on the PC).

Immediate mode is well-suited for small programs with limited remote I/O. In such applications, the delay waiting for the ethernet link is not significant. As the application size grows you will be more likely to use the buffered mode.

Immediate mode functions may also be used in conjunction with buffered mode. This may be desirable in cases where very rapid updates are necessary. However, when mixing modes for output, make sure that you use the buffered mode function call, in addition to the immediate mode function call. This will prevent the buffered mode update from changing the output status back to the state it was in prior to the immediate mode command.

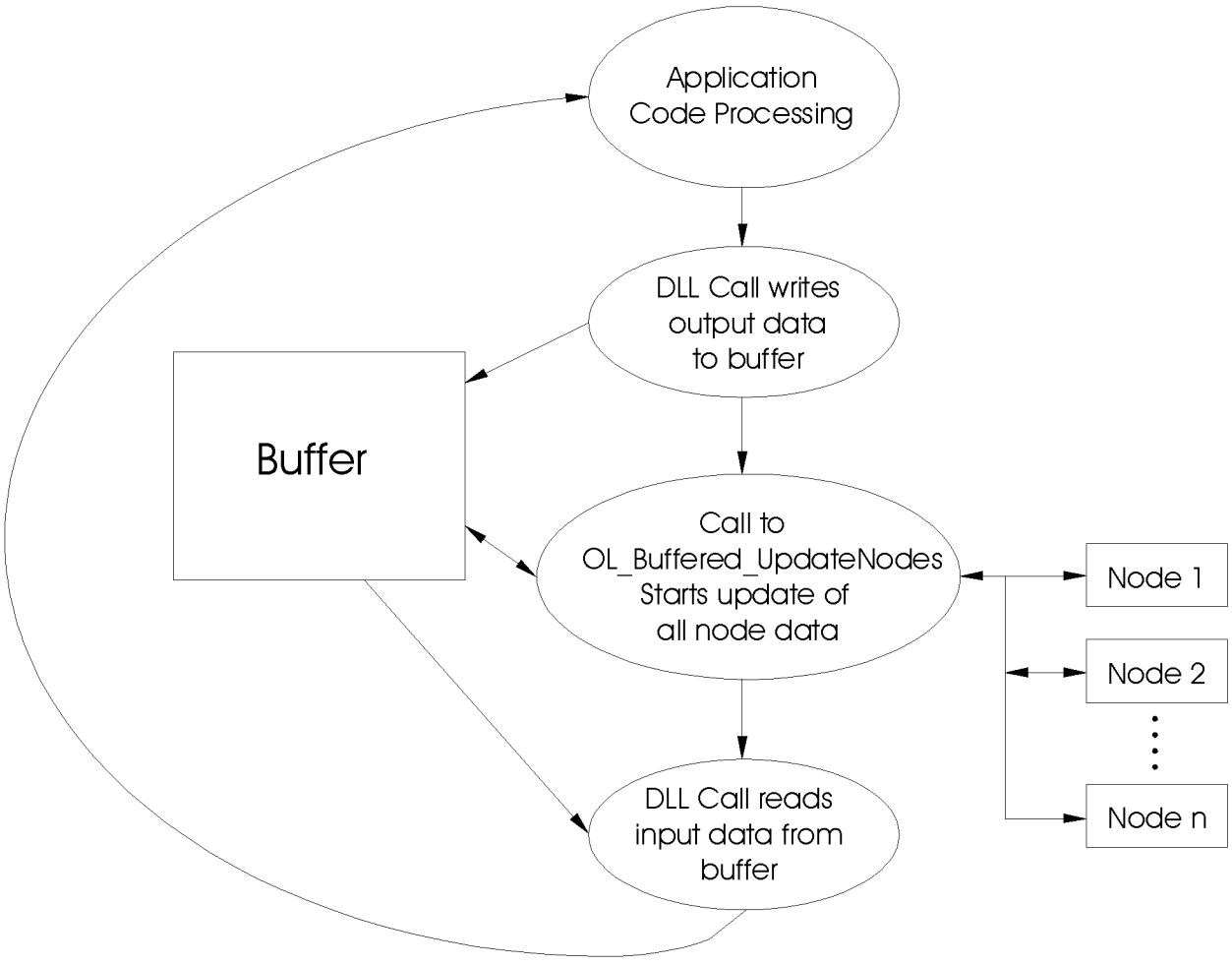


Immediate Mode Operation

Buffered Mode

Buffered mode operation is the mode that is recommended for larger applications. This mode, shown below, allows your application program to access the latest input data and send the required output data without waiting for the ethernet link. It does so by using an intermediate buffer.

All application accesses are to the buffer. On every pass through the program loop an OptiLogic DLL call must be made to function OL_Buffered_UpdateNodes() to cause the PC to communicate with all RTUs. The OL_Buffered_UpdateNodes() sends the output data to the RTUs and requests input data from the RTUs. It waits for the response from the RTU. It will retry a message depending on the timeout value specified and will continue to retry based on a value sent from the application program.



Buffered Mode Operation

Samples

It is instructive to look through the example code and header files. The same functional files are provided for both Microsoft C++ and Microsoft Visual Basic. The code examples are discussed in greater detail in the subsequent pages.

The header files - optilogic.h for C++ or DLLdefinitions.bas for Visual Basic - must be included in your applications.

The Session Group – Opening and Closing the Link

The session group consists of two functions. One function opens the network link. The other function closes it.

Initializing the Network OL_NetworkInit()

The function OL_NetworkInit() must be used to initialize the network. This function initializes the PC network and gets it ready to communicate with the nodes.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_NetworkInit Lib "OPTILOGICPROTOCOL.DLL" (ByVal port As Integer,
        ByVal protocoltype As Integer, ByVal Timeout As Integer, ByVal retry As Integer) As
        Integer
```

Usage :

```
Dim timeout_value as Long
Dim num_retries As Byte

timeout_value = 100    ' retry period in ms
num_retries = 3        ' number of retries before stopping

Call OL_NetworkInit(OPTISOCKET, OL_IPX, timeout_value, num_retries)
```

Where OPTISOCKET and OL_IPX (and OL_UDP) are defined in DLLdefinitions.bas. Use OL_IPX to select IPX communication or OL_UDP to select UDP/IP communications.

Closing the Network OL_NetworkClose()

The function OL_NetworkClose() will close the socket responsible for communications and do the cleanup work necessary to gracefully shutdown communications. It requires no parameters.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Sub OL_NetworkClose Lib "OPTILOGICPROTOCOL.DLL" ()
```

Usage :

```
Call OL_NetworkClose()
```

Node Group - Who's There?

The Node Group of functions is used to determine what RTUs are connected to the network and what modules are contained in each. These routines work by sending out a broadcast message asking a node to respond to the modules it contains.

Add an IP Search Address to the Query List OL_NetworkAddIPSearchAddress()

This function is necessary to specify which IP network address(es) to broadcast a QueryNode() packet to. If all of the RTUs are in the same local network as the PC, then this function does not need to be called. This function call is necessary under the following conditions: (1) if using the IP protocol and if you have multiple networks containing RTUs, and (2) if using the IP protocol and if your PC is in a different local network from the RTUs.

This function adds the input IP address to the search list for the QueryNode() function, therefore, it must be called before the QueryNode() function is called. Up to 9 IP search addresses may be added to the list. The function OL_NetworkAddIPSearchAddress() will return a '1' if the IP address has been added to the search list. It will return a '0' if the address cannot be added to the search list. If a '0' is returned, the address passed into the function might not be in the proper format or the IP address search list could be full.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_NetworkAddIPSearchAddress Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        strIPAddress As String) As Integer
```

where : strIPAddress - the Class C *directed broadcast* IP Address for a particular network (e.g. "208.170.106.255").

Note: The last octet is always a 255 for the directed broadcast IP address.

Usage :

```
Dim IPAddress As String      ' the IP Address string
Dim status As Integer        ' the value returned from the function call

IPAddress = "215.150.127.255"      ' the IP search address for a particular network
status = OL_NetworkAddIPSearchAddress(IPAddress)
If status = 0 Then
        ' The specified address could not be added to the search list.
        ' Perform error checking to determine the cause of the error.
End If
```

Remove an IP Search Address from the Query List OL_NetworkRemoveIPSearchAddress()

This function is necessary if you choose to remove an IP network address from the search list. The function will remove the IP address specified from the search list that's used by the QueryNode() function. The OL_NetworkRemoveIPSearchAddress() function will return a '1' if the IP address has been removed from the search list. It will return a '0' if the specified address cannot be removed from the search list. If a '0' is returned, the address passed into the function might no be in the proper format or the specified IP address might not be in the search list.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_NetworkRemoveIPSearchAddress Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal strIPAddress As String) As Integer
```

where : strIPAddress - the Class C *directed broadcast* IP Address for the network (e.g. "208.170.106.255").

Usage :

```
Dim IPAddress As String      ' the IP Address string
Dim status As Integer        ' the value returned from the function call

IPAddress = "215.150.127.255"      ' the IP search address for a particular network
status = OL_NetworkRemoveIPSearchAddress(IPAddress)
If status = 0 Then
        ' The specified address could not be removed from the search list.
        ' Perform error checking to determine the cause of the error.
 End If
```

Query a Particular Node
OL_QueryNode()

This function can be used to query a particular node to determine (1) if the node with the rotary switch address exists, (2) what version of software it is running and (3) what modules are plugged in.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_QueryNode Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As
    Integer, ByRef BaseType As Byte, ByRef moduletypes As Byte, ByRef subtypes As
    Byte, ByRef moduleVersions As Byte, ByRef minversion As Byte, ByRef majversion As
    Byte) As Integer
```

To use this function you must first define the variables that it uses.

- BaseType - a byte variable whose address is passed to the function. The function places the "Base Type" in this variable. Base types are defined in a table in Appendix A.
- moduletypes - an array of bytes which will be used to hold the "type" information of all modules installed in the base. This array must be sized to handle the largest number of modules that any base can handle (currently 9). The value that is placed in each array position is the "type" number for the module connected to that slot. In other words, if an 8 digital output module (type 9) is plugged into the first slot, the OL_QueryNode() function will place a 9 in moduletypes(0). The last slot location is for the operator panel. For a 4-slot base the operator panel data is placed in moduletypes(4). For an 8-slot base, it is placed in moduletypes(8). Module types and subtypes are defined in a table in Appendix B
- subtypes - an array of bytes which will be used to hold the "subtype" information of all modules installed in the base. This array must be sized to handle the largest number of modules that any base can handle (currently 9). The value that is placed in each array position is the "subtype" number for the module connected to that slot. In other words, if a relay output module (subtype 1) is plugged into the first slot, the OL_QueryNode() function will place a 1 in subtypes(0).
- moduleVersions - an array of bytes which will be used to hold the "version" information of all modules installed in the base.
- minversion - minor version number of base. For software version 3.7, the minor version is 7.
- majversion - major version number or base. For software version 3.7, the major version is 3.

The returned long (the Long) is the number of slots (plus the operator panel) available in the node. For a 4-slot RTU, this number would be 5 (4 slots + operator panel). A 0 returned indicates no answer from the address selected (normally meaning no RTU exists at this address).

Usage :

```
Dim slots As Integer
Dim mtype(9) As Byte, stype(9) As Byte, mver(9) As Byte, minver As Byte, majver As Byte
Dim basetype As Byte
Dim nodeaddr As Integer
nodeaddr = 1
slots = OL_QueryNode(nodeaddr, basetype, mtype(0), stype(0), mver(0), minver, majver)
```

Query the Nodes in Sequence OL_GetFirst() and OL_GetNext()

As an alternative to the OL_QueryNode() function just described, two other functions are provided. The OL_GetFirst() function will query nodes, starting at address 1, until it finds one, then return that node's data. OL_GetNext() will continue, starting at the node following the last one reported and poll until it finds the next node. When it finds the next node, it returns node data. In both cases the function will terminate when the node address reaches 100.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_GetFirst Lib "OPTILOGICPROTOCOL.DLL" (ByRef NODEADDR As
        Integer, ByRef BaseType As Byte, ByRef moduletypes As Byte, ByRef subtypes As
        Byte, ByRef moduleVersions As Byte, ByRef minversion As Byte, ByRef majversion As
        Byte) As Integer

Declare Function OL_GetNext Lib "OPTILOGICPROTOCOL.DLL" (ByRef NODEADDR As
        Integer, ByRef BaseType As Byte, ByRef moduletypes As Byte, ByRef subtypes As
        Byte, ByRef moduleVersions As Byte, ByRef minversion As Byte, ByRef majversion As
        Byte) As Integer
```

The definitions are very similar to the one just defined for OL_QueryNode(). The difference is in the first parameter. With OL_QueryNode(), you pass in the address number that you want polled. With OL_GetFirst() and OL_GetNext() you don't pass in the address, it is returned to you to denote the node that was found.

Usage :

The following usage example will find up to 25 RTUs and store their definitions in an array. It calls OL_GetFirst() to find the first node, stores the node data then goes into a loop. The loop will continue to find nodes, using OL_GetNext() until either 25 total nodes are found or until node address 100 is reached (return value of 0).

```
Dim     address(25) As Integer
Dim     basetype(25) As Byte, majorver(25) As Byte, minorver(25) As Byte,
        modtype(25)(9) As Byte, subtype(25)(9) As Byte, modver(25)(9) As Byte

Dim     slots As Integer, nodeaddr As Integer
Dim     mtype(9) As Byte,  stype(9) As Byte, mver(9) As Byte, minver As Byte, majver As Byte,
        btype As Byte


slots = OL_GetFirst(nodeaddr, btype, mtype(0), stype(0), mver(0), minver, majver)

If slots > 0 Then
        ' Store data for first RTU
        address(0) = nodeaddr
        basetype(0) = btype
        majorver(0) = majver
        minorver(0) = minver
```

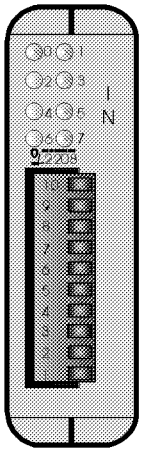
(continued on next page)

```
        For j = 0 To 9
                modtype(0)(j) = mtype[j]
                subtype(0)(j) = stype[j]
                modver(0)(j) = mver[j]
        Next j

        ‘ Find more RTUs
        For i = 1 to 25
                slots = OL_GetNext(nodeaddr, btype, mtype(0), stype(0), mver(0), minver,
                        majver)
                If slots > 0 Then
                        ‘ Store data for RTU
                        address(i) = nodeaddr
                        basetype(i) = btype
                        majorver(i) = majver
                        minorver(i) = minver
                        For j = 0 To 9
                                modtype(i)(j) = mtype[j]
                                subtype(i)(j) = stype[j]
                                modver(i)(j) = mver[j]
                        Next j

                Else
                        i = 30   ‘Node address 100 reached so break out of For loop
                End If

        Next i

End If
```

Immediate Mode I/O Group

The Immediate Mode Group of functions is used to directly and immediately communicate with an RTU. The general concepts have been covered in the previous pages. All of these functions return an integer indicator. A "1" returned indicates the function was successful. If a "0" is returned, the error code can be retrieved by calling OL_GetLastErrorCode() - defined in the Housekeeping group.

Read Digital Inputs
OL_ReadDigitalInput()



This function can be used to read a digital input module's inputs at a particular node address. The returned data is placed in a "long" variable (32 bits) with the status of each input point being indicated by a bit within the variable. A "1" in the bit position indicates active. The bits are in sequence starting at bit 0. If the input module has less than 32 inputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ReadDigitalInput Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR
        As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef data As Long) As
        Integer
```

Usage :

The following example will check input 3 on the input module residing in slot 2 at node 23.

```
Dim dig_data As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 23
slot = 2
modtype = 1             ' 8 channel digital inputs are type 1

status = OL_ReadDigitalInput(nodeaddr, slot, modtype, dig_data)
If status = 1 Then
        If (dig_data And &H08) Then
                ' *********************
                ' application processing
                ' *********************
         End If
End If
```

Read Latched Digital Inputs
OL_ReadLatchedDigitalInput()

This function can be used to see if a digital input module's inputs have turned ON at any time. Suppose that there is an input that may have turned ON and then back OFF in between calls to the routine OL_ReadDigitalInput(). In those cases the input signal will be missed by your program, but you may need to know that it turned ON, if only briefly. In that case, call the routine OL_ReadLatchedDigitalInput(). The OptiLogic RTU stores the "latched" status of each input. If the input ever turns ON, the "latched" status will be a "1" regardless of the current input state. The status will stay at "1" until read by the routine OL_ReadLatchedDigitalInput(). If the input is OFF when the routine is called, it will be reset to "0". If it is still ON, the "latched" input bit will stay at "1".

The data returned from the routine call is placed in a "long" variable (32 bits) with the status of each input point being indicated by a bit within the variable. A "1" in the bit position indicates active. The bits are in sequence starting at bit 0. If the input module has less than 32 inputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ReadLatchedDigitalInput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef data
        As Long) As Integer
```

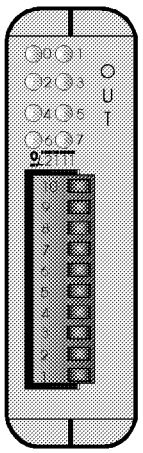
where data = status of inputs (input 0 --> bit 0, input 1 --> bit 1 ...)

Usage :

The following example will check to see if input 3 on the input module residing in slot 2 at node 23 has latched ON.

```
Dim dig_data As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 23          ' node address 23
slot = 2               ' input card resides in slot 2
modtype = 1            ' 8 channel digital inputs are type 1

' Read the Latched Input Status
status = OL_ReadLatchedDigitalInput(nodeaddr, slot, modtype, dig_data)
If status = 1 Then
        If (dig_data And &H08) Then
                ' ********************
                ' input 3 is ON, process data
                ' ********************
        End If
End If
```

Write Digital Outputs
OL_WriteDigitalOutput()



The OL_WriteDigitalOutput() function can be used to write required output state data to a digital output module at a particular node address. The command sends a long (32 bits) to the module. Each bit, starting with the LSB, indicates the required state for one output point. A "1" in a bit position indicates the required output state is "active", i.e. relay closed, transistor on, etc.. If the output module has less than 32 outputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_WriteDigitalOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByVal data As
        Long) As Integer
```

Usage :

The following example will turn outputs 3, 5 and 6 on, and all other points off at the output module residing in slot 3 at node 23. The returned boolean is not used in the first case, it is used in the second.

```
Dim dig_outs As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 23
slot = 3
modtype = 9                    ' 8 channel digital outputs are type 9

dig_outs = &H68                ' turn on outputs 3, 5 and 6

' First case:  Just send outputs, ignore the response
status = OL_WriteDigitalOutput(nodeaddr, slot, modtype, dig_outs)



' Second case:  Send outputs, and use the response
status = OL_WriteDigitalOutput(nodeaddr, slot, modtype, dig_outs)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If
```

Set Up Digital Output Fail Safe Mode OL_SetUpDigitalOutputFailSafe()

The OL_SetUpDigitalOutputFailSafe() function is used to set the state of outputs on an output module in the event of a communication failure between the RTU and the host. This command can be used in 3 ways: (1) it can be set to turn all outputs off, (2) it can be set to turn all outputs to a set pattern or (3) it can be set to leave the outputs in the last known state.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_SetUpDigitalOutputFailSafe Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByVal byModuleType As Byte, ByVal
    fsType As Byte, ByVal fsPattern As Long, ByVal fsTime As Byte) As Integer
```

where: byModuleType	=	&H08 (4 digital outputs) &H09 (8 digital outputs) &H0A (16 digital outputs) &H0B (24 digital outputs) &H0C (32 digital outputs)
fsType	=	Fail Safe Type 1 - fail safe to all outputs off 2 - fail safe to the pattern fsPattern 3 - fail safe to the last known state
fsPattern	=	Fail Safe Pattern (32 - bits) "1" = ON, "0" = OFF
fsTime	=	Time from communication failure before RTU goes into the Fail Safe State. (In tenths of a second.)

Continued on next page.

Usage :

The following example will turn OFF all outputs of an OL2111 residing in slot 3 of Node 30 when a communication failure occurs.

```
Dim nodeaddr As Integer          ' address of RTU
Dim slot As Integer              ' slot of output card
Dim ModuleType As Byte           ' output card type
Dim fsType As Byte               ' fail safe type
Dim fsPattern As Long            ' fail safe pattern
Dim fsTime As Byte               ' fail safe timeout (in tenths of a secretary.)
Dim status As Integer            ' OL routine return value

nodeaddr = 30                    ' node 30
slot = 3                         ' the card resides in the last slot
ModuleType = &H09                ' 8 channel digital outputs are type 9
fsType = 1                       ' Turn all outputs off
fsPattern = 0                    ' Pattern
fsTime = 40                      ' wait 4 seconds after comm failure before implementing

status = OL_SetUpDigitalOutputFailSafe(nodeaddr, slot, ModuleType, fsType, fsPattern,
        fsTime)
```

The routine call should be repeated for each output card in each RTU that you want to set to a Fail Safe Pattern. This needs to be sent to the RTU after any power loss by the RTU. It should also be placed in a system initialization routine.

Read Digital Outputs
OL_ReadDigitalOutput()

The OL_ReadDigitalOutput() function can be used to read the current output state from a digital output module. The "data" variable returns a long (32 bits) that represents the current state of the outputs. Each bit, starting with the LSB, indicates the required state for one output point. A "1" in a bit position indicates the required output state is "active", i.e. relay closed, transistor ON, etc.. If the output module has less than 32 outputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ReadDigitalOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef data As
        Long) As Integer
```

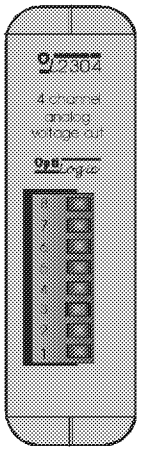
where data = status of outputs (output 0 --> bit 0, output 1 --> bit 1 ...)

Usage :

The following example will read the status of each output on an 8 point output module. The module resides at node 5 in slot 0. Then the example checks to see if output 5 is ON.

```
Dim out_data As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 5          ' output card resides at node address 5
slot = 2              ' output card resides in slot 2 of node 5
modtype = 9           ' 8 channel digital outputs are type 9

status = OL_ReadDigitalOutput(nodeaddr, slot, modtype, out_data)
If status = 1 Then
        If (out_data And &H20) Then
                ' ********************
                ' output 5 is ON, therefore process application
                ' ********************
        End If
End If
```

OL2304 Analog Output Module



The OL2304 analog output module has 4 channels. Each channel is independently configured as either 0-5V, 0-10V, +/-5V or +/-10V. Each channel has a 12-bit resolution (1 in 4096) with a scale of 0 - 4095.

To write the output data you must first decide the voltage range of each channel. Next, you must configure the OL2304. Then, define an array of integers (16-bit) with an index of 4, place the output data in the array and write the data to the output module. The maximum output value is 4095. Any value over 4095 will be automatically set to 4095.

The following function calls are available for the OL2304 analog output module.

- OL_ConfigAnalogOutput() - A single call configures the analog output range for each channel.
- OL_WriteAnalogOutput() - Writes the analog output values to each channel.

Configure Analog Outputs
OL_ConfigAnalogOutput()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ConfigAnalogOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByVal range1
    As Byte, ByVal range2 As Byte, ByVal range3 As Byte, ByVal range4 As Byte) As
    Integer
```

Where: range 1 - voltage output range for channel 1
range 2 - voltage output range for channel 2
range 3 - voltage output range for channel 3
range 4 - voltage output range for channel 4

Each range is configured in the following manner:

rangex = 0 : 0-5 Volt range
1 : 0-10 Volt range
2 : +/- 5 Volt range
3 : +/- 10 Volt range

rangex = range1, range2, range3 or range4

Note: The channels have to be reconfigured every time the OptiLogic base has it's power cycled.
If you change the configuration of a channel "on the fly", make sure you set the channel's output to 0 or unexpected results may occur with your device.

Continued on next page.

Write Analog Outputs
OL_WriteAnalogOutput()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_WriteAnalogOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef data
    As Integer) As Integer
```

where: data is an array (index = 4) to place the output values into.
The data should be in the range of 0 - 4095.

Conversion of Analog Output Voltage to Equivalent Output Value

0-5 Volt or 0-10 Volt Range

To convert an output voltage to the equivalent value, use the following formula:

x = (y * 4095) / z

where x = equivalent value in the 0-4095 range to output
y = output voltage value
z = output range maximum (5 or 10 depending on configuration)

Example: A channel is configured for 0-10 Volts and a 3.3 Volt output is required, the value will be calculated as follows:

x = (3.3 * 4095) / 10 = **1351**

+/-5 Volt or +/-10 Volt Range

To convert an output voltage to the equivalent value, use the following formula:

x = ((z + y) / (2 * z)) * 4095

where x = equivalent value in the 0-4095 range to output
y = output voltage value
z = output range maximum (+5 or +10 depending on configuration)

Example: A channel is configured for +/-5 Volt and a +2.5 Volt output is required, the value will be calculated as follows.

x = ((5 + (+2.5)) / (2 * 5)) * 4095 = **3071**

Example: A channel is configured for +/-10 Volts and a -2.5 Volt output is required, the value will be calculated as follows.

x = ((10 + (-2.5)) / (2 * 10)) * 4095 = **1536**

Continued on next page.

Usage :

Example A: The following example will calculate the equivalent output value (0 - 4095 range) when the output voltage range is 0-5VDC.

```
Dim AnalogOut_Val(4) As Integer          'number between 0 and 4095 that is equivalent to
                                         'output voltage value
Dim VoltageOut_Val(4) As Single          'voltage value to output

' initialize output values
VoltageOut_Val(0) = 5
VoltageOut_Val(1) = 2.5
VoltageOut_Val(2) = 1.22
VoltageOut_Val(3) = 3.3

'Convert to equivalent output value and place in an array.
For i = 0 to 3
        ' use the formula x = (y * 4095) / 5
        ' Notice that in the next line CInt() is used to round the decimal point.
        AnalogOut_Val(i) = CInt((VoltageOut_Val(i) * 4095) / 5)
Next i

' The results from the above calculations are as follows:
'       AnalogOut_Val(0) = 4095          ' 5VDC output
'       AnalogOut_Val(1) = 2048          ' 2.5VDC output
'       AnalogOut_Val(2) = 999           ' 1.22VDC output
'       AnalogOut_Val(3) = 2702          ' 3.3VDC output
' End of example
```

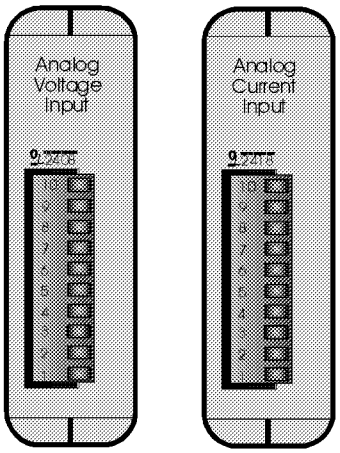
Example B: The following example will configure each channel for the 0-5VDC range on an OL2304 residing at node 21in slot 2.

```
Dim range1 As Byte, range2 As Byte, range3 As Byte, range4 As Byte
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 21                ' node address
slot = 2                     ' slot 2
modtype = 25                 ' 4 channel analog outputs are type 25

range1 = 0                   ' 0 - 5 Volt range
range2 = 0
range3 = 0
range4 = 0

status = OL_ConfigAnalogOutput(nodeaddr, slot, modtype, range1, range2, range3, range4)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If
' End of Example
```

Example C: The following example will cause a specified voltage to output from each channel of an OL2304 residing at node 21in slot 2.

```
Dim analog_val(4) as Integer
Dim modtype As Byte
Dim slot As Integer, nodeaddr as Integer, status as Integer

nodeaddr = 21                ' node address
slot = 2                     ' OL2304 resides in slot 2
modtype = 25                 ' 4 channel analog outputs are type 25

' The card has been configured for an output range of 0 - 5VDC (see Example B for
' configuration)

' As calculated in Example A, the equivalent output values are as follows:

analog_val(0) = 4095         ' 5VDC output
analog_val(1) = 2048         ' 2.5VDC output
analog_val(2) = 999          ' 1.22VDC output
analog_val(3) = 2702         ' 3.3VDC output

status = OL_WriteAnalogOutput(nodeaddr, slot, modtype, analog_val(0))
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If
```

Read Analog Inputs
OL_ReadAnalogInput()



This function can be used to read all of the analog inputs of a particular analog input module at a particular node address. The returned data is placed in an array. The values are straight unscaled readings from the input module. To use this function you must define an array of integers (16 bit) at least as large as the number of analogs on the module being read.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ReadAnalogInput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef data As
        Integer) As Integer
```

where: data is an array holding the analog input values

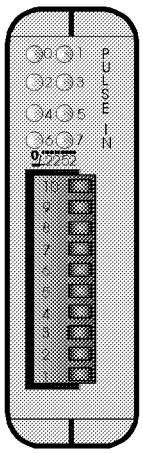
Usage :

The following example will read inputs from an 8 channel analog input module residing in slot 0 at node 23.

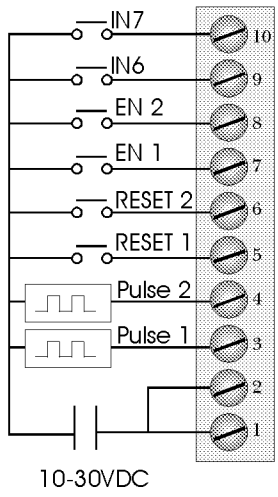
```
Dim analog_val(8) as Integer
Dim modtype As Byte
Dim slot As Integer, nodeaddr as Integer, status as Integer

nodeaddr = 23
slot = 0
modtype = 18                    ' 8 channel analog inputs are type 18

status = OL_ReadAnalogInput(nodeaddr, slot, modtype, analog_val(0))
If status = 1 Then
        ' ********************
        ' Handle analog values here
        ' ********************
Else
        ' ********************
        ' Place Error handling code here
        ' ********************
End If
```

The 8 analog values will be placed in the array analog_val. Analog input 0 will reside in analog_val(0), input 1 in analog_val(1), etc..

OL2252 Dual High Speed Pulse Counter



The OL2252 Dual High Speed Pulse Counter module has two 0-15KHz pulse counter inputs. It also has four other digital inputs. Four of these other inputs (two for each channel) can be configured as control inputs for the pulse counter.



The figure on the left illustrates the pulse counter interface. Input 0, attached to terminal 3, is the first pulse input channel. EN 1 (Input 4, terminal 7) can be configured for use as an enable counting input for Pulse 1. RESET 1 (Input 2, terminal 5) can be configured for an external input to reset the count to 0. EN 2 and RESET 2 can likewise be configured as external control signals for Pulse input 2. Any input not used for its count related function can be used as a general purpose input.

In addition to the hardware reset and enable signals, each pulse counter can be sent a message from the host computer for "reset" and "enable".

The following function calls are available for the OL2252 pulse counter module.

- OL_Configure_Counter() - Define which, if any, hardware controls are to be used, as well as a debounce count associated with each channel.
- OL_Read_Counter() - Sends control signals (reset & enable) to each counter and reads current count values.

Configure Counter
OL_Configure_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Configure_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal ctlFlags As Byte, ByVal dbSet1
        As Byte, ByVal dbSet2 As Byte) As Integer
```

where : ctlFlags - control flags
- bit 0 - use channel 1 hardware enable
- bit 1 - use channel 1 hardware reset
- bit 4 - use channel 2 hardware enable
- bit 5 - use channel 2 hardware enable

dbSet1 - debounce count for channel 1 (establishes the max pulse frequency that the channel will count)
- dbset1 = 2 - 15 KHz
- = 4 - 10 KHz
- = 8 - 5 KHz
- = 16 - 2.5 KHz
- = 40 - 1 KHz

dbSet2 - debounce count for channel 2

OL2252 continued

Send Counter Controls and Read Counts OL_Read_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Read_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR
        As Integer, ByVal SlotNo As Integer, ByVal ctlFlags As Byte, ByRef countData As Byte,
        ByRef inData As Byte) As Integer
```

where : ctlFlags - control flags
- bit 0 - channel 1 enable
- bit 1 - channel 1 reset
- bit 4 - channel 2 enable
- bit 5 - channel 2 reset

countData - an array of bytes that hold 2 32-bit count values. The 1st element holds the 8 most significant bits of the channel 1 count. The second array element holds the next 8 bits. The third element holds the next 8 bits and the fourth element holds the 8 least significant bits of the channel 1 count. The count for the second channel follows in the same format. The 5th array element holds the 8 most significant bits of the channel 2 count. The 6th array element holds the next 8 bits of channel 2. The 7th element holds the next 8 bits and the 8th element holds the 8 least significant bits of the channel 2 count.

inData- a variable that holds the input status of all 8 input lines. All input points can be used as general purpose inputs, if so desired.

OL2252 continued

Usage :

The following example configures the first channel to use the hardware enable and reset. It configures the second channel to use *neither* the hardware reset nor enable. The program checks both channel counts. When a count of 400,000 is reached on channel 1, it will send an output signal to a device attached to the RTU. It depends on the external device to reset and re-enable the count. The second channel is checked for a value above 1,000,000. When that value is reached, the program will disable and reset the channel 2 count and perform an operation. When the operation is complete, it will start the process again.

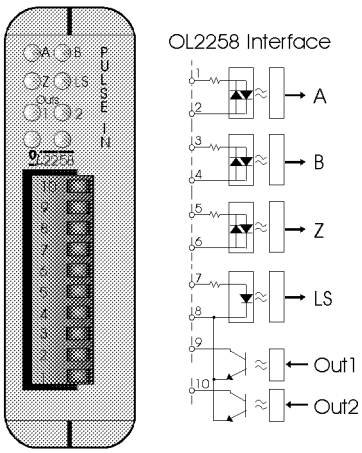
```
Dim slot As Integer, nodeaddr As Integer, status As Integer
Dim hw_control As Byte, sw_control As Byte, debounce1 As Byte, debounce2 As Byte,
        in_data As Byte, count_val(8) As Byte
Dim ch1_val As Long, ch2_val As Long
Dim ch_str As String, temp As String

slot = 3
nodeaddr = 7
hw_control = &H03    ' Enable ch 1 hardware control, disable ch 2 hw control
debounce1 = 2        ' 20KHz max rate for both channels
debounce2 = 2
status = OL_Configure_Counter (nodeaddr, slot, hw_control, debounce1, debounce2)


While (1)                ' loop forever
        sw_control = &H11    'enable channel 1 and channel 2
        status = OL_Read_Counter(nodeaddr, slot, sw_control, count_val(0), in_data)
                'Convert channel 1 data to a long value
                For i = 0 To 3
                        temp = CStr(Hex(count_val(i)))
                        If Len(temp) = 1 Then          ' if value only has 1 digit, pad with a 0
                                ch_str =  ch_str + "0" + temp
                        Else
                                ch_str = ch_str + temp
                        End If
                Next i
                ch_str = "&H" + ch_str
                ch1_val = CLng(ch_str)        ' channel 1 count value

                If ch1_val  > 400000 Then
                        '****************
                        ' Send output signal
                        '****************
                Else
                        '****************
                        ' clear output signal
                        '****************
                End If
```

(continued on next page)

OL2252 continued

```
'Convert channel 2 data to a long value
For i = 4 To 7
        temp = CStr(Hex(count_val(i)))
        If Len(temp) = 1 Then           ' if value only has 1 digit, pad with a 0
                ch_str =  ch_str + "0" + temp
        Else
                ch_str = ch_str + temp
        End If
Next i
ch_str = "&H" + ch_str
ch2_val = CLng(ch_str)          ' channel 2 count value

If ch2_val  > 1000000
        sw_control = &H21     'reset and disable channel 2, keep channel 1 enabled
        '*****************
        ' Reset count
        '*****************
        OL_Read_Counter(nodeaddr, slot, sw_control, count_val(0), in_data)
        '*****************
        ' Perform required operation
        '*****************
End If

DoEvents        ' process any system events that may occur

Wend
```

OL2258 High Speed Pulse Counter



The OL2258 High Speed Pulse Counter is designed to interface to a variety of standard pulse encoder devices. The electrical interface is shown on the right. Differential, sourcing or sinking inputs can be interfaced to the OL2258.

The OL2258 is configurable. It can be used with pulse and direction, quadrature or up/down count type pulse encoders. The OL2258 maintains the current cumulative count as a 32 bit integer value. It also makes frequency snapshot data available over the most recent count of 1 second or 200 milliseconds. The Z and LS inputs can be used to automatically reset the count to a user defined "preset" value. Each transistor output can be configured to turn on when the count value is within a specified range.

For more detailed information on the features of the OL2258 High Speed Pulse Counter, see the section on the OL2258 in the **OptiLogic Input/Output Modules** manual.

The following function calls are available for the OL2258 High Speed Pulse Counter module.

- OL_Configure_HS_Counter() - Defines count type (pulse and direction, quadrature or up/down.), configures preset inputs, frequency period and preset value.
- OL_HS_SendOutput_Range_Counter() - Configures minimun and maximun range to turn ON Output 1 or 2.
- OL_Read_HS_Counter() - Returns input status, output status, count type, current count value and frequency count over selected time period.

Term	Label	Description
1	A1	Pulse input A (quadrature)/ Pulse input (pulse & direction)/ Up pulse (up/down count)
2	A2	
3	B1	Pulse input B (quadrature)/ Pulse input (pulse & direction)/ Up pulse (up/down count)
4	B2	
5	Z1	Z input (optional)
6	Z2	
7	LS	Limit Switch input (optional)
8	COM	Commom for limit switch and the two outputs.
9	Out1	Open collector output1
10	Out1	Open collector output2

Continued on next page.

Configure High Speed Counter
OL_Configure_HS_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Configure_HS_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByVal ctlFlags As Byte, ByVal
    cfgFlags As Byte, ByRef presetVal As Long) As Integer
```

where: ctlFlags = bits 0, 4-7 : Unused
bit 1 : Count type - Pulse & Direction
bit 2 : Count type - Up/Down Count
bit 3 : Count type - Quadrature

cfgFlags = bit 0 : Output 1 range enabled - enable output 1 if in range
bit 1 : Output 2 range enabled - enable output 2 if in range
bit 2 : Unused
bit 3 : Z preset enabled - when Z input ON, force to preset value
bit 4 : LS preset enabled - when LS input ON, force to preset value
bit 5 : force preset - when set, force count to preset value
bit 6 : hold count - when set, hold count at current value
bit 7 : frequency period selection
0 = 1second count (good up to 30KHz)
1 = 200 msec count (for over 30KHz)

presetVal = 32 bit value to force pulse count to when a preset enabled or force preset bit is set.

Note: Only turn ON one bit in ctlFlags, ensure that the rest are OFF

Send Output Range
OL_HS_SendOutput_Range_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_HS_SendOutput_Range_Counter Lib "OPTILOGICPROTOCOL.DLL"
    (ByVal NODEADDR As Integer, ByVal SlotNo As Integer, ByVal output As Byte,  ByRef
    minValue As Long, ByRef maxValue As Long) As Integer
```

where: output = output to configure (1 or 2)
minValue = minimum value for range to turn ON specified output
maxValue = maximum value for range to turn ON specified output

Continued on next page.

Read High Speed Counter
OL_Read_HS_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Read_HS_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByRef countData As Byte, ByRef
    freqData As Byte, ByRef inStat As Long, ByRef outStat As Long) As Integer
```

where: countData = current count, 32 bit signed value
formatted as an array 4 bytes long
countData(0) - hi byte … to … countData(3) - low byte

freqData = pulse count in the most recent frequency period
formatted as an array 2 bytes long
freqData(0) - hi byte
freqData(1) - low byte

inStat = input status (input ON = 1, input OFF = 0) & frequency rate status
bit 0 : in_A
bit 1 : in_B
bit 2 : in_Z
bit 3: in_LS
bits 4, 5 and 6 : Unused
bit 7 : frequency selection rate
1 = 1 second count
0 = 200 msec count

outStat = output status (output ON = 1, output OFF = 0) & count type status
bit 0 : output 1 status
bit 1 : output 2 status
bits 2 - 4 : Unused
bit 5 : Count type - Pulse & Direction
bit 6 : Count type - Up/Down Count
bit 7 : Count type - Quadrature

Continued on next page.

Usage :

Example A: The configuration routine OL_Configure_HS_Counter() will tell the OL2258 in what mode(s) it will be used. The following example shows a call which will configure the OL2258 for Quadrature type counting. It will enable Output 1 and the LS preset and set the preset count to 12867. The OL2258 resides at node 2 in slot 3.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer
Dim configFlags As Byte, controlFlags As Byte
Dim presetCntValue As Long

slot = 3                              ' The OL2258 is in slot 3.
nodeaddr = 2                          ' The OL2258 is at node 2.

controlFlags =      &H08              ' Quadrature type counting
configFlags =       &H11              ' Output 1 range enabled and LS preset enabled
presetCntValue =  12867               ' set preset count value to 12867

status = OL_Configure_HS_Counter(nodeaddr, slot, controlFlags, configFlags, presetCntValue)
If status = 0  Then
        '*************************
        ' process error
        '*************************
End If
```

Example B: The following example will configure the output range for output 1 on the OL2258. When the count value is within the range of 83592 to 135000, Output 1 will be ON. If it's not within the range, Output 1 will be OFF. As in Example A, the OL2258 resides at node 2 in slot 3.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer
Dim SendOutput As Byte
Dim MinRangeValue As Long, MaxRangeValue As Long


slot =              3             ' The OL2258 is in slot 3.
nodeaddr =          2             ' The OL2258 is at node 2.
SendOutput =        1             ' Configure range for Output 1
MinRangeValue =     83597         ' Configure minimum limit for 83597
MaxRangeValue =     135000        ' Configure maximum limit for 135000

status = OL_HS_SendOutput_Range_Counter(nodeaddr, slot, SendOutput, MinRangeValue,
        MaxRangeValue)
If status = 0  Then
        '*************************
        ' process error
        '*************************
End If
```

Continued on next page.

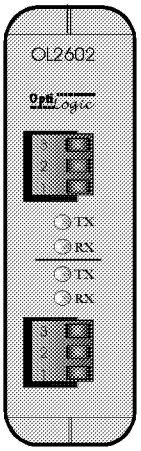
Example C: The following example will read the current count value, input status and output status on the OL2258. It will convert the current count value to a long and the frequency count to a frequency value in Hertz. As in Examples A and B, the OL2258 resides at node 2 in slot 3.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer, Frequency As Integer
Dim countValue(4) As Byte, freqCount(2) As Byte
Dim inputStatus As Long, outputStatus As Long, CurrentCount As Long
Dim ch_str As String, temp As String


slot =          3               ' The OL2258 is in slot 3.
nodeaddr =      2               ' The OL2258 is at node 2.

status = OL_Read_HS_Counter(nodeaddr, slot, countValue(0), freqCount(0), inputStatus,
        outputStatus)
If status = 0  Then
        '*************************
        ' process error
        '*************************
Else
        '*****Convert the countValue to a long *********

        For i = 0 to 3
                temp = CStr(Hex(countValue(i)))
                If Len(temp) = 1 Then           ' if temp only has 1 digit, pad it with a leading 0
                        ch_str = ch_str + "0" + temp
                Else
                        ch_str = ch_str + temp
                End If
        Next i

        ch_str = "&H" + ch_str
        CurrentCount = CLng(ch_str)             ' count value as a long

        '*****Convert the frequency count to an integer and then to frequency in Hz*********

        For i = 0 to 1
                temp = CStr(Hex(freqCount(i)))
                If Len(temp) = 1 Then           ' if value only has 1 digit, pad it with a leading 0
                        ch_str = ch_str + "0" + temp
                Else
                        ch_str = ch_str + temp
                End If
        Next i

        ch_str = "&H" + ch_str
        Frequency = CInt(ch_str)                ' frequency count value as an integer
```

Continued on next page.

```
        '*****Depending on time period, calculate frequency in Hz *********

        If inputStatus And &H80 Then          ' if time period = 200ms
                Frequency = Frequency / 0.2
                '****if time period = 1 second, Frequency already calculated****
        End If


End If


'End of Example
```

OL2602 Dual RS232 and Base Serial Comm Port



The OL2602 module is a Dual RS232 communications module. Each of the ports operates independently at 1200 -19200 baud, 7 or 8 data bits, configurable parity and stop bits. The serial communications port on the OL4054 base operates with the same parameters as the OL2602 except it can handle a range of baud rates from 300 -19200.

The serial ports operate as a typical general purpose communication port with a fair-sized buffer. The RTU transmit buffers are 48 bytes and receive buffers are 48 bytes each. Any message can be sent or received. Messages longer than 48 bytes must be buffered within your program in such a way that they do not overflow the transmit buffer. Also, the host program must be careful to communicate with the RTU often enough that the receive ports do not overflow.

With the OL2602 configuration of either port can mean the loss of transmit and receive data in both ports. Therefore, configuration should occur once, at initialization. If reconfiguration is required during operation, realize that a message on the alternate port may be garbled.

The following function calls are available for the OL2602 Dual RS232 module and the OL4054 base serial communications port.

- OL_ConfigureSerialPort() - Set up port operating parameters
- OL_ReadSerialData() - Read receive data from a port
- OL_WriteSerialData() - Send data to be transmitted from a port

OL_ConfigureSerialPort()
Configure a Communication Port

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ConfigureSerialPort Lib "OPTILOGICPROTOCOL.DLL" (ByVal glbNodeId
        As Integer, ByVal SlotNo As Integer, ByVal modtype As Byte, ByVal PortNo As Integer,
        ByVal BaudRate As Integer, ByVal DataBits As Integer, ByVal Parity As Integer, ByVal
        StopBits As Integer) As Integer
```

The following parameters are defined in the file DLLdefinitions.bas

where :	SlotNo	- 0, 1 for OL2602 or 81 for Base Comm Port
	modtype	- 112 for OL2602 or 0 for Base Comm Port (see DLLdefinitions.bas)
	PortNo	- 0 (base), 1 (OL2602 - top) or 2 (OL2602 - bottom)
	BaudRate	- see DLLdefinitions.bas
	DataBits	- see DLLdefinitions.bas
	Parity	- see DLLdefinitions.bas
	StopBits	- see DLLdefinitions.bas

Use the selections listed in the see DLLdefinitions.bas file.

Note: The port configuration is not stored in nonvolatile memory. The configuration message (for each port) must be sent each time power is lost by the RTU base.

Read Received Data
OL_ReadSerialData()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ReadSerialData Lib "OPTILOGICPROTOCOL.DLL" (ByVal glbNodeId As
    Integer, ByVal SlotNo As Integer, ByVal modtype As Byte, ByVal PortNo As Integer,
    ByRef numbytes As Byte, ByRef flags As Byte, ByRef message As Byte) As Integer
```

where :	SlotNo	- 0, 1 for OL2602 or 81 for Base Comm Port
	modtype	- 112 for OL2602 or 0 for Base Comm Port (see DLLdefinitions.bas)
	PortNo	- 0 (base), 1 (OL2602 - top), or 2 (OL2602 - bottom)
	numbytes	- Variable which will contain the # of receive bytes returned.
	flags	- Variable to store returned error flags associated with received bytes
	message	- Array containing received data

Write to Serial Port
OL_WriteSerialData()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_WriteSerialData Lib "OPTILOGICPROTOCOL.DLL" (ByVal glbNodeId As
    Integer, ByVal SlotNo As Integer, ByVal modtype As Byte, ByVal PortNo As Integer,
    ByVal numbytes As Byte, ByRef message As Byte, ByRef Leftover As Byte) As Integer
```

where :	SlotNo	- 0, 1 for OL2602 or 81 for Base Comm Port
	modtype	- 112 for OL2602 or 0 for Base Comm Port (see DLLdefinitions.bas)
	PortNo	- 0 (base), 1 (OL2602 - top), or 2 (OL2602 - bottom)
	numbytes	- Variable which will contain the # of receive bytes returned.
	message	- Array containing data to transmit
	left	- Variable in which the # of bytes left in the transmit buffer will be returned

OL2602 continued

Usage :

Port Configuration

To configure a serial port the OL_ConfigureSerialPort() routine should be called during the program initialization process.

The following example configures the top port on an OL2602 for 9600 baud, 8 data bits, no parity and 1 stop bit. The node address is 7 and the OL2602 is in slot 0 of the RTU.

```
Dim slot as Integer, nodeaddr as Integer, status as Integer

slot = 0
nodeaddr = 7
status = OL_ConfigureSerialPort(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
                OL_9600_BAUD, OL_8_DATA_BITS, OL_NO_PARITY, OL_1_STOP_BIT)
```

Reading data from a Serial Port

To read data from a serial port, the routine OL_ReadSerialData() must be called. The routine has to be called often enough so that data loss does not occur. If the routine is not called often enough, the receive buffer (48 bytes) will overflow and you will lose data. Once the port is read, the data will clear from the buffer.

The best way to read the data is to start a timer and read the serial port each time the timer timeout occurs. The interval should be set low enough to keep the data from overflowing. The following example assumes that a timer has been created in Visual Basic with the name tmrRx1. The interval for the timer may vary depending on your application. You should adjust the interval within the program until you get consistent results.

```
'The following 2 variables are global and should be dimensioned in a .bas module

Public   glb_RxBuffer(500) As Byte     'This array can be used to hold the current message
                                       'received by the serial port so that it can be used for later
                                       'processing. The actual size of this buffer depends on
                                       'your application.
Public   glb_RxByteCount  As Integer   'the total number of bytes received for a message

Private Sub tmrRx1_Timer()

        Dim numbytes As Byte, flags As Byte, message(48) As Byte
        Dim slot As Integer, nodeaddr As Integer, status As Integer

        slot = 0
        nodeaddr = 7
        tmrRx1.Enabled = False        'disable receive timer
```

(continued on next page)

OL2602 continued

```
status = OL_ReadSerialData(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
        numbytes, flags, message(0))

For i = 0 To (numbytes - 1)             ' buffer received data in global receive buffer
        glb_RxBuffer(glb_RxByteCount) = message(i)
        glb_RxByteCount = glb_RxByteCount + 1    ' increment global receive buffer
                                                 ' count
Next i

' Logic can be placed here to determine if a complete message has been received.
' Requirements for a complete message may change depending on your application.  It
' could be a fixed number of bytes, a certain character, etc.  When a complete message
' has been received, you can set a flag to notify another portion of your program or you
' can check the message in this routine.

tmrRx1.Enabled = True       ' This is optional, you might want to enable your timer in a
                            ' different place in the code.

End Sub
```

Writing data to a Serial Port

The following routine is an example that can be used to transmit data from the top port of an OL2602. The node address of the RTU is 7 and it is in slot 0.

```
Private Sub TransmitData(TransmitLength As Byte, TransmitBuffer() As Byte)

Dim BytesTransmitted As Integer, status As Integer
Dim message(48) As Byte, BytesLeft As Byte, NumberToTransmit As Byte

'TransmitBuffer() = the array containing the data to be transmitted
'TransmitLength = the total number of bytes to transmit
'BytesTransmitted = the total number of bytes that have been transmitted
'BytesLeft = # of available bytes left in tx buffer of OL2602
'NumberToTransmit = number of bytes placed into transmit buffer

BytesTransmitted = 0   'initialize to 0
BytesLeft = 48         'initialize to size of OL2602 transmit buffer
slot = 0
nodeaddr = 7

If TransmitLength > 48 Then    'if message longer than the OL2602
                               'transmit buffer, send it in different pieces
                               'to prevent an overflow of the transmit buffer
```

(continued on next page)

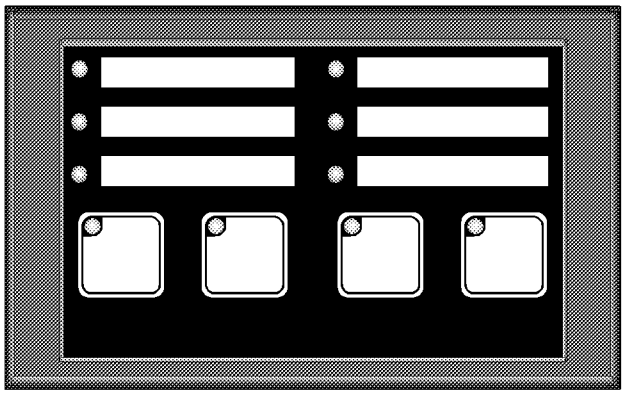
OL2602 continued

```
        While (BytesTransmitted < TransmitLength)  'loop until entire message sent
                DoEvents
                NumberToTransmit = BytesLeft / 2     'send half the available bytes
                For i = 0 To NumberToTransmit - 1    'copy into transmit buffer
                        message(i) = TransmitBuffer(BytesTransmitted)
                        BytesTransmitted = BytesTransmitted + 1 'increment total byte counter
                Next i
                status = OL_WriteSerialData(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
                                NumberToTransmit, message(0), BytesLeft)
        Wend

Else      'if message smaller than transmit buffer size, buffer entire message
        For i = 0 To TransmitLength - 1
                message(i) = TransmitBuffer(i)
        Next i
        status = OL_WriteSerialData(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
                        TransmitLength, message(0), BytesLeft)

End If

End Sub
```

Immediate Mode Operator Panel Group

Operator panels tend to be more complex than typical I/O modules. In most cases there are several messages associated with each operator panel.

OL3406 Pushbutton/Indicator Panel



The OL3406 Operator Panel has 4 pushbuttons and 6 LED indicators. The buttons can be configured for either momentary or alternate-action operation. The button LEDs normally reflect button status. The momentary buttons can also be configured for LED separation (direct on/off control over the ethernet). The indicator LEDs can be turned on, off, or flashed. See the OL3406 manual for further details.

The following function calls are available for the OL3406 operator panel.

- OL_OL3406_StatusControl() - Read button status and control the LEDs
- OL_OL3406_ForceButtons() - Force selected alternate-action button(s) to the desired state
- OL_OL3406_ConfigurePanel() - Define the panel configuration
- OL_OL3406_ReadPanelConfiguration() - Read the configuration of the panel

Read the Pushbutton Status and Control the LEDs OL_OL3406_StatusControl()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3406_StatusControl Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByRef databuffer As Byte, ByRef pbstatus As Byte) As Integer
```

where : databuffer is a 4 byte array with the following definition:

databuffer(0) = on/off control of the 4 inset button LEDs. This will only affect a button LED if it is momentary and configured for LED separation. Bits are in sequence starting at bit 0.

databuffer(1) = on/off control of indicator light LEDs. Bits are in sequence starting at bit 0.

databuffer(2) = flash control of 4 inset button LEDs. LED must be on to flash. Bits are in sequence starting at bit 0.

databuffer(3) = flash control of 6 indicator light LEDs. LED must be on to flash. Bits are in sequence starting at bit 0.

pbstatus - current status of panel buttons will be placed in this byte. Bits are in sequence starting at bit 0. A "1" indicates button active.

Force Alternate-Action Button Status
OL_OL3406_ForceButtons()

Three types of button force operations are available for the OL3406 panel alternate-action buttons. Buttons can be selectively forced on, or forced off. Also, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3406_ForceButtons Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal ButtonFlags As Byte, ByVal ButtonData As Byte) As
        Integer
```

where : ButtonData defines the button which will be affected. Bits are in button sequence starting with bit 0.

ButtonFlags defines the type of force as follows (defined in DLLdefinitons.bas)

ButtonFlags = OL3406_BUTTON_ALL ; force the buttons to a state matching ButtonData (force state)

= OL3406_BUTTON_OR ; logically OR current button state with ButtonData (force on)

= OL3406_BUTTON_AND ; logically AND current button state with the complement of ButtonData (force off)

OL3406 continued

Configure OL3406 Panel
OL_OL3406_ConfigurePanel()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3406_ConfigurePanel Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal conflags As Byte) As Integer
```

where : conflags = bits 0 - 3 defines button operation of the 4 buttons.
"0" = momentary, "1" = alternate action
= bit 7, if "1" enables LED separation on momentary button LEDs
if "0" disables LED separation

Read OL3406 Configuration
OL_OL3406_ReadPanelConfiguration()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3406_ReadPanelConfiguration Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByRef conflags As Byte) As Integer
```

where : conflags = bits 0 - 3 indicates button type.
"0" = momentary, "1" = alternate action
= bit 7, if "1" indicates LED separation on momentary button LEDs
if "0" indicates LED separation is disabled

OL3406 continued

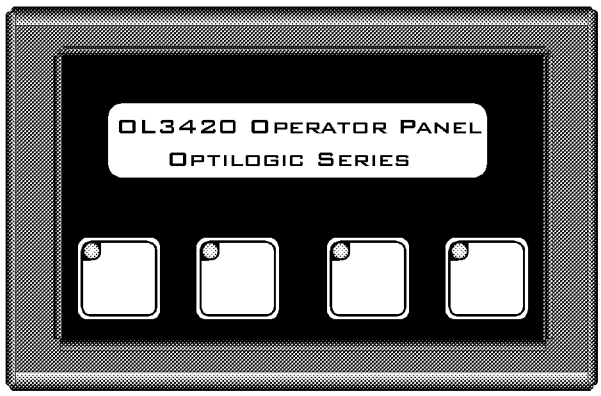
Usage :

The following example reads the panel configuration. If the configuration is not as required, it sends the configuration to the panel. It then goes into a loop. Whenever the first button is sensed active, it lights indicator lights 3 and 4, flashes light 4 and performs an application process. At the end of the process, it forces the first button off.

```
Dim    config As Byte              ' Configuration data
Dim    RequiredConfig As Byte      ' Required configuration
Dim    LEDstate(4) As Byte         ' LED state array
Dim    ButtonStatus As Byte        ' Button status
Dim    nodeaddr As Integer         ' Node address of the RTU
Dim    status As Integer           ' Return status of RTU routine calls, can be used for error
                                   '   checking

nodeaddr = 30                  ' node address of RTU is 30
RequiredConfig = &H07          ' Required configuration = buttons 1 - 3 alternate action
                               '   button 4 momentary, no LED separation

LEDstate(0) = 0                ' Start with all lights out
LEDstate(1) = 0
LEDstate(2) = 0
LEDstate(3) = 0

'Check button config, if it isn't correct then send configuration message
status = OL_OL3406_ReadPanelConfiguration(nodeaddr, config)
If   config <> RequiredConfig   Then
        status = OL_OL3406_ConfigurePanel(nodeaddr, RequiredConfig)
End If

While (1)
        ' Forever loop
        DoEvents

        ' Is the first button active?
        status = OL_OL3406_StatusControl(nodeaddr, LEDstate(0), ButtonStatus)
        If  ButtonStatus  And  &H01  Then
                ' If so, turn light 3 and 4 on, flash light 4
                LEDstate(1) = LEDstate(1) Or &H0C          'lights 3 and 4 on
                LEDstate(3) = LEDstate(3) Or &H08          'flash light 4
                status = OL_OL3406_StatusControl(nodeaddr, LEDstate(0), ButtonStatus)

                ' Do function processing

                ' After function processing done, Force the first button OFF
                status = OL_OL3406_ForceButtons(nodeaddr, OL3406_Button_AND, &H01)
        End If
Wend
```

Note: The 'While' loop can be replaced by a Timer routine to update the status at a fixed interval.

OL3420 Operator Terminal



The OL3420 Operator Terminal has 4 pushbuttons and a 2 line x 20 character LCD display. The buttons can be configured for either momentary or alternate-action operation. The button LEDs normally reflect button status. The momentary buttons can also be configured for LED separation (direct on/off control over the ethernet). The status LEDs can be turned on, off, or flashed. See the OL3420 manual for further details.

The following function calls are available for the OL3420 operator terminal.

- OL_OL3420_StatusRequest() - Reads button status (and controls the LEDs if LED separation)
- OL_OL3420_ForceButtons() - Force selected alternate-action button(s) to the desired state
- OL_OL3420_SendTextDisplayMessage() - Send text to the display
- OL_OL3420_ConfigurePanel() - Define the panel configuration
- OL_OL3420_ReadConfiguration() - Read the configuration of the panel

Read the Pushbutton Status and Control Momentary Button LEDs OL_OL3420_StatusRequest()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3420_StatusRequest Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal LEDState As Byte, ByRef ButtonStatus As Byte) As
        Integer
```

where : LEDState - applies only if LED separation is active
bits 0 - 3 turns LED ON if "1" (for buttons 1 - 4 in sequence)
bits 4 - 7 flashes LED if "1" (for buttons 1 - 4 in sequence), button LEDs must be ON to flash

ButtonStatus - current status of panel buttons will be placed in this byte. Bits are in sequence starting at bit 0 (button 1 is bit 0). A "1" indicates button active.

OL3420 continued

Force Alternate-Action Button Status
OL_OL3420_ForceButtons()

Three types of button force operations are available for the OL3420 panel alternate-action buttons. Buttons can be selectively forced on, or forced off. Also, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3420_ForceButtons Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal ButtonFlags As Byte, ByVal ButtonData As Byte) As
        Integer
```

where : ButtonData defines the button which will be affected. Bits are in button sequence starting with bit 0.
ButtonFlags defines the type of force as follows -
ButtonFlags = OL3420_BUTTON_ALL ; force the buttons to a state matching ButtonData (force state)
= OL3420_BUTTON_OR ; logically OR current button state with ButtonData (force on)
= OL3420_BUTTON_AND ; logically AND current button state with the complement of ButtonData (force off)

Send Text to OL3420 Display
OL_OL3420_SendTextDisplayMessage()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3420_SendTextDisplayMessage Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte) As Integer
```

where : line - is the line number (0 = top, 1 = bottom) to send text to
message - is message text (20 character array containing ASCII equivalent of each character in message)

OL3420 continued

Configure OL3420 Terminal OL_OL3420_ConfigurePanel()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_OL3420_ConfigurePanel Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal buttonstate As Byte) As Integer

where : buttonstate = bits 0 - 3 defines button operation of the 4 buttons.
"0" = momentary, "1" = alternate action
= bit 7, if "1" enables LED separation on momentary button LEDs
if "0" disables LED separation

Read OL3420 Configuration OL_OL3420_ReadConfiguration()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_OL3420_ReadConfiguration Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByRef buttonstate As Byte) As Integer

where : buttonstate = bits 0 - 3 indicates button type.
"0" = momentary, "1" = alternate action
= bit 7, if "1" indicates LED separation on momentary button LEDs
if "0" indicates LED separation is disabled

OL3420 continued

Usage :

The following example reads the panel configuration. If the configuration is not as required, it sends the configuration to the panel. It then goes into a loop. Whenever the second button is sensed active, it sends "Process Active" to the top line of the display and performs an application process. At the end of the process, it forces the second button off.

```
Dim     config As Byte              ' Configuration data
Dim     RequiredConfig As Byte      ' Required configuration
Dim     PBLEDstate As Byte          ' Pushbutton LED state
Dim     ButtonStatus As Byte        ' Button status
Dim     LineNo As Byte              ' Line number for message  0 = top, 1 = bottom
Dim     message(20) As Byte         ' Message string array
Dim     nodeaddr As Integer         ' Node address of the RTU
Dim     status As Integer           ' Return status of RTU routine calls, can be used for error
                                    '   checking
Dim     DisplayMessage As String    ' String containing message string to place into buffer

RequiredConfig = &H07               ' Required configuration = buttons 1 - 3 alternate action
                                    '   button 4 momentary, no LED separation
PBLEDstate = 0                      ' Start with all lights out (unused because no LED
                                    '   separation)
DisplayMessage = "Process Active"

For  i = 0 To 19
        message(i) = &H20           'Fill message buffer with blank spaces
Next i
For i = 0 to Len(DisplayMessage) - 1
        message(i) = Asc(Mid$(DisplayMessage, i + 1, 1))  'parse message string to get ASCII
                                                          'equivalent of each character
Next i

'Check button config, if it isn't correct then send configuration message
status = OL_OL3420_ReadConfiguration(nodeaddr, config)
If  config <> RequiredConfig  Then
        status = OL_OL3420_ConfigurePanel(nodeaddr, RequiredConfig)
End If

While (1)
        ' Forever loop
        DoEvents                ' Process any system events that may occur
        ' Is the second button active ?
        status = OL_OL3420_StatusRequest(nodeaddr, PBLEDstate, ButtonStatus)
        If  ButtonStatus And &H02  Then
                ' If so, send message to the display
                LineNo = 0
                status = OL_OL3420_SendTextDisplayMessage(nodeaddr, LineNo, message(0))

                ' Do function processing
```

OL3420 continued

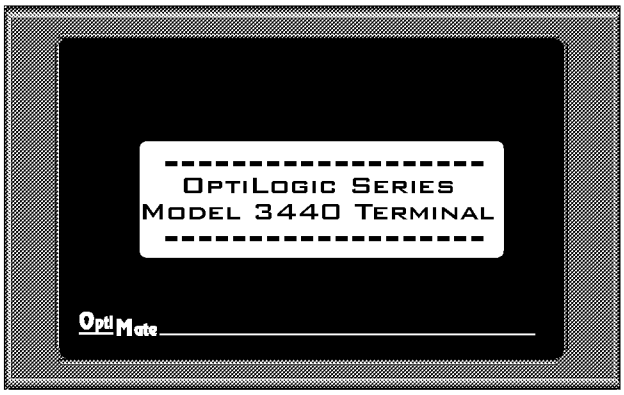
```
            ' Force the second button OFF
            status = OL_OL3420_ForceButtons(nodeaddr, OL3420_BUTTON_AND, &H02)
        End If
Wend
```

Notes:

Notice that the first 'For' loop put the ASCII equivalent of a space into the message buffer. This should be done each time a new message is sent to clear old character out of the message buffer.
The second 'For' loop takes the message string and places it into the message buffer via the Len, Asc and Mid Visual Basic commands. The message string has to be converted in this manner so that each character in the string can be converted to its ASCII equivalent.
The 'While' loop can be replaced by a Timer routine to read the button status and update the message at a fixed interval.

OL3440 Display Panel

The OL3440 Display Panel is 4 line by 20 character display. You can send any text to any line.



Send Text to OL3440 Display OL_OL3440_SendTextDisplayMessage()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3440_SendTextDisplayMessage Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal glbNodeId As Integer, ByVal index As Byte, ByRef message As Byte) As Integer
```

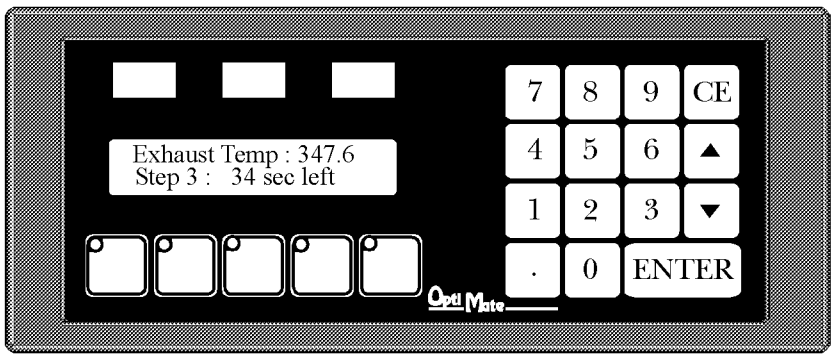
where : index is line number (0 - 3 from top to bottom)
message is 20 character array of ASCII text

Usage

The following will send the message "Text message..Abcd" to the third line from the top on the OL3440 panel present at node address 6.

```
Dim     LineNo As Byte              ' Line number for message: 0 = top, 1 = second, 2 = third,
                                    '  3 = bottom
Dim     message(20) As Byte         ' Message string array
Dim     nodeaddr As Integer         ' Node address of the RTU
Dim     status As Integer           ' Return status of RTU routine calls, can be used for error
                                    '  checking
Dim     DisplayMessage As String    ' String containing message string to place into buffer

DisplayMessage = "Text message..Abcd"
nodeaddr = 6

For  i = 0 To 19
        message(i) = &H20           'Fill message buffer with blank spaces
Next i
For i = 0 to Len(DisplayMessage) - 1
        message(i) = Asc(Mid$(DisplayMessage, i + 1, 1))  ' parse message string to get ASCII
                                                          ' equivalent of each character, then
                                                          ' place into message buffer
Next i

' send message to the display
LineNo = 2
status = OL_OL3440_SendTextDisplayMessage(nodeaddr, LineNo, message(0))
```

OL3850 Operator Terminal



The OL3850 Operator Terminal has a two line x 20 character LCD display, a numeric keypad, five user-definable function keys and three user-definable LED indicator light bars. This panel can be used to display messages, accept numeric data input, display status indications and select functions.

The following function calls are available for the OL3850 Operator Terminal:

- OL3850_StatusRequest() - Control lights, read button status and retrieve entered data
- OL3850_SendTextDisplayMessage() - Send a text message to one of the two display lines
- OL3850_ConfigurePanel() - Configure the operation of the five function keys
- OL3850_ReadConfiguration() - Read the configuration of the five function keys
- OL3850_ForceButtons() - Force the state of selected "alternate-action" buttons
- OL3850_SendKeypadMessage() - Send a message with a field for the entry of a number input from the keypad
- OL3850_SendArrowMessage() - Send a message with a field for and initial value to be adjusted by the arrow keys

OL3850 continued

Send Light Controls and Read Status information OL_OL3850_StatusRequest()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3850_StatusRequest Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal LEDState As Byte, ByVal ledflash As Byte, ByRef
        ButtonStatus As Byte, ByRef returndata As Double) As Integer
```

where : LEDState - bits 0 - 2 are on/off controls for the three light bars. Bit 0 is the left most light bar, etc.
bits 3 - 7 are controls for the indicator LED inset in the five general purpose function buttons. They only have an effect if the panel is configured for LED separation (otherwise the LEDs reflect button status) for all LEDs and light bars, "1" = on , "0" = off

ledflash - flash controls for the same LEDs and light bars mentioned for LEDState. To flash, the light must be on.

ButtonStatus - Current status of the five function buttons will be placed in this byte. Bits 0 - 4 are associated with the buttons, with the left most button in bit 0. A "1" indicates button active. Bit 7 is a flag indicating if data has been entered. A "1" indicates data has been entered.

returndata - Data entry value will be returned in this variable.

Send a Text Message to the Display OL_OL3850_SendTextDisplayMessage()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3850_SendTextDisplayMessage Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte) As Integer
```

where : line - the line number (0 = top, 1 = bottom) to send text to
message - 20 character array of ASCII text

OL3850 continued

Configure OL3850 Function Buttons
OL_OL3850_ConfigurePanel()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3850_ConfigurePanel Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal buttonstate As Byte) As Integer
```

where : buttonstate = bits 0-4 defines the operation of the 5 function buttons.
"0" = momentary, "1" = alternate action
= bit 7, if "1" enables LED separation on momentary button LEDs
if "0" disables LED separation

Read OL3850 Configuration
OL_OL3850_ReadConfiguration()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3850_ReadConfiguration Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByRef buttonstate As Byte) As Integer
```

where : buttonstate = bits 0 - 4 indicates button type.
"0" = momentary, "1" = alternate action
= bit 7, if "1" indicates LED separation on momentary button LEDs
if "0" indicates LED separation is disabled

Force Alternate-Action Button Status
OL_OL3850_ForceButtons()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_OL3850_ForceButtons Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal ButtonFlags As Byte, ByVal ButtonData As Byte) As
        Integer
```

where : ButtonData defines the button which will be affected. Bits are in button sequence starting with bit 0 for the leftmost button.
ButtonFlags defines the type of force as follows -
= 0x80 ; force the buttons to a state matching ButtonData (force state)
= 0x40 ; logically OR current button state with ButtonData (force on)
= 0x20 ; logically AND current button state with the complement of ButtonData (force off)

OL3850 continued

Send Keypad Data Entry Message OL_OL3850_SendKeypadMessage()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_OL3850_SendKeypadMessage Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte, ByVal startNum As Double, ByVal decimalPoint As Byte) As Integer

where : line - the line number (0 = top, 1 = bottom) to display message on
message - 20 character array of ASCII text. Carets "^" should be used as placeholders for the data entry field.
startNum - starting value of keypad entry data
decimalPoint - A beginning value can be passed, which the user can adjust using the arrow key, or start a new data entry with the keypad. The numeric startNum is passed as a double. The decimalPoint is the number of digits after the decimal. In other words, if startNum = 4567 and decimalPoint = 2, the starting value displayed is 45.67.

Send Arrow Adjust Message OL_OL3850_SendArrowMessage()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_OL3850_SendArrowMessage Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte, ByVal startNum As Long, ByVal decimalPoint As Byte, ByVal lowLimit As Long, ByVal highLimit As Long) As Integer

where : line - the line number (0 = top, 1 = bottom) to display message on.
message - 20 character array of ASCII text. Carets "^" should be used as placeholders for the data entry field.
startNum - starting value of arrow adjust data
decimalPoint - A beginning value must be passed, which the user can adjust using the arrow key. The numeric startNum is passed as a long. The decimalPoint is the number of digits after the decimal. In other words, if startNum = 4567 and decimalPoint = 2, the starting value displayed is 45.67.
lowLimit - The minimum value allowed for adjustment (the same decimalPoint value applies)
highLimit - The maximum value allowed for adjustment (the same decimalPoint value applies)

For example, if startNum = 4567, decimalPoint = 2, lowLimit = 1000, and highLimit = 9000, the starting value of 45.67 can be adjust by the user to any value between 10.00 and 90.00.

OL3850 continued

Usage

The following example shows various functions using an OL3850 Operator Terminal attached to an OptiLogic RTU with a node address of 30. The program will verify the pushbutton configuration and reconfigure them if necessary. It will read/update the general status of the OL3850 based on the interval of a timer, given the name *tmrOL3850* for this example. Then based on pushbutton status and the conditions of 2 flags, it will display various types of messages and wait for user interaction with the terminal.

The program contains the following five global variables. The first two are used as flags for active message types, the third one is used to lock out the arrow message after button 4 is forced off and the other two are used to store data values from keypad entry and arrow adjust messages. Be sure to initialize the global flags to 0 when the program starts.

```
Private glb_KeypadActive As Byte    ' flag denoting a keypad message is active on the
                                    ' display
Private glb_ArrowActive As Byte     ' flag denoting an arrow adjust message is active on the
                                    ' display
Private glb_ForceActive As Byte     ' flag used to prevent the arrow adjust message from
                                    ' reappearing on the display after button 4 is forced OFF
Private glb_KeypadValue As Double   ' variable that stores the keypad entry result
Private glb_ArrowValue As Long      ' variable that stores the current arrow adjust value
```

This subroutine is used to convert a string passed into it into an array of bytes containing the ASCII equivalent of each character in the string. It passes that array back to the calling routine for transmission to the OL3850.

```
Private Sub SetUpMessage(MessageString As String, MsgBuffer() As Byte)

    ' place blank characters in buffer
    For i = 0 To 19
        MsgBuffer(i) = &H20
     Next i

    ' get ASCII equivalent of each character in message string and place into buffer
    For i = 0 To Len(MessageString) - 1
        MsgBuffer(i) = Asc(Mid$(MessageString, i + 1, 1))
    Next i

End Sub
```

OL3850 continued

This subroutine is the body of the program. When the timer tmrOL3850 times out, this routine performs several tasks.
(1) First, it reads the configuration of the pushbuttons. If they are not as required by the program, then the program will reconfigure them.
(2) Next, the program reads/updates the general status (lamps, LEDs, button status and current data entry value).
Next, the program takes action depending on certain parameters.
(3) If button 3 is pressed then the top line will display a text message and the bottom line a keypad entry message. Next, a flag is set indicating that a keypad message is active on the panel.
(4) Else if the keypad message is active and if the data available bit (most significant bit of the button status byte) is set, the keypad data is stored into a variable, the top and bottom lines of the LCD are updated with text messages and the keypad message active flag is cleared.
(5) Else if button 4 is active and the force lockout flag inactive, the program can perform several tasks:

(1) If an arrow adjust message is not active, then it sends a text message to the top line and an arrow adjust message to the bottom line.
(2) If an arrow adjust message is active, it continuously copies the arrow adjust data into its storage variable.
(2a) If an arrow adjust message is active and button 2 is pressed, it will clear the arrow adjust message by placing new messages on both lines of the LCD, force button 4 off, clear the arrow adjust message active flag and set the force lockout flag.

(6) Else if button 4 has cleared and the force lockout flag is active, reset the force lockout flag.
(7) The last thing the subroutine does is restart the timer tmrOL3850.

```
Private Sub tmrOL3850_Timer()

Dim RequiredConfig As Byte ' required button configuration
Dim config As Byte         ' configuration read from RTU
Dim LEDState As Byte       ' byte holding Lamp and LED on/off states
Dim LEDflash As Byte       ' byte holding Lamp and LED flash states
Dim ButtonStatus As Byte   ' byte containing button status as returned from the RTU
Dim message(20) As Byte    ' array holding message to display on 3850 LCD
Dim DataValue As Double    ' current data value returned from RTU
Dim status As Integer      ' result of a DLL function call
Dim DisplayMessage As String  ' string to display on the 3850's LCD

tmrOL3850.Enabled = False  ' disable timer
LEDState = &H0             ' turn all lights off
LEDflash = &H0             ' turn all flash bits off
RequiredConfig = &H19      ' buttons 1, 4, 5 alternate  -  2, 3 momentary
' read the general configuration of the 3850
status = OL_OL3850_ReadConfiguration(30, config)
' if the required config doesn't match the config read from the RTU, configure
If RequiredConfig <> config Then
        status = OL_OL3850_ConfigurePanel(30, RequiredConfig)
End If
```

OL3850 continued

```
' read and update the general status of lamps, LEDs, buttons, and the current data entry value
status = OL_OL3850_StatusRequest(30, LEDState, LEDflash, ButtonStatus, DataValue)
‘ if StatusRequest successful
If status Then
        ' if button 3 is pressed
        If (ButtonStatus And &H4) Then
                ' send text message to top line
                DisplayMessage = "Key Data,Press ENTER"
                Call SetUpMessage(DisplayMessage, message)
                status = OL_OL3850_SendTextDisplayMessage(30, 0, message(0))
                ' send keypad message to bottom line, initial data and decimal point 0, carets “^”
                ‘       are placeholders for the data
                DisplayMessage = "Keypad Entry ^^^^^"
                Call SetUpMessage(DisplayMessage, message)
                status = OL_OL3850_SendKeypadMessage(30, 1, message(0), 0, 0)
                If status Then
                        glb_KeypadActive = 1  ' set keypad message active flag
                End If

        ' if keypad message active and the data available bit set, read the data
        ElseIf ((glb_KeypadActive = 1) And (ButtonStatus And &H80)) Then
                glb_KeypadValue = DataValue     ' store entered value
                ' blank top line
                DisplayMessage = ""
                Call SetUpMessage(DisplayMessage, message)
                status = OL_OL3850_SendTextDisplayMessage(30, 0, message(0))
                ' send message to bottom line
                DisplayMessage = "Keypad Entry Done"
                Call SetUpMessage(DisplayMessage, message)
                status = OL_OL3850_SendTextDisplayMessage(30, 1, message(0))
                If status Then
                        glb_KeypadActive = 0            ' reset keypad message active flag
                End If
        ' if button 4 active and the force lockout flag inactive
        ElseIf ((ButtonStatus And &H8) And (glb_ForceActive = 0)) Then
                ' if arrow adjust message not active send it
                If (glb_ArrowActive = 0) Then
                        ' send message to top line
                        DisplayMessage = "Press F2 When Done"
                        Call SetUpMessage(DisplayMessage, message)
                        status = OL_OL3850_SendTextDisplayMessage(30, 0, message(0))
                        ' send arrow adjust message to bottom line, initial value is 3686.75,
                        ‘       adjustment range is 3000.00 - 4000.00
                        DisplayMessage = "Adjust Value ^^^^^^"
                        Call SetUpMessage(DisplayMessage, message)
                        status = OL_OL3850_SendArrowMessage(30, 1, message(0), 368675, 2,
                                300000, 400000)
                        If status Then
                                glb_ArrowActive = 1  'set arrow adjust message active
                        End If
```

OL3850 continued

```
                ' if arrow adjust active
                ElseIf (glb_ArrowActive) Then
                        glb_ArrowValue = DataValue  'update arrow data as its adjusted
                        ' if button 2 is pressed, reset arrow message
                        If (ButtonStatus And &H2) Then
                                ' blank top line
                                DisplayMessage = ""
                                Call SetUpMessage(DisplayMessage, message)
                                status =  OL_OL3850_SendTextDisplayMessage(30, 0,
                                        message(0))
                                ' send message to bottom line
                                DisplayMessage = "Arrow Adjust Done"
                                Call SetUpMessage(DisplayMessage, message)
                                status = OL_OL3850_SendTextDisplayMessage(30, 1,
                                        message(0))
                                ' force button 4 off
                                status = OL_OL3850_ForceButtons(30, &H20, &H8)
                                If status Then
                                        glb_ArrowActive = 0   ' reset arrow adjust message active
                                        glb_ForceActive = 1   ' set force lockout to 1
                                End If
                        End If
                End If
        ' if button 4 status has cleared and force lockout flag active, reset flag
        ElseIf (((ButtonStatus And &H8) = 0) And (glb_ForceActive = 1)) Then
                glb_ForceActive = 0             ' reset force lockout
        End If
End If

tmrOL3850.Enabled = True                ' restart timer

End Sub
```

Buffer Mode I/O Group

The Buffered Mode Group of functions is used to interface the RTUs indirectly, through a buffer, and handle communications more in a block format and as a background task. The general concepts have been covered in the previous pages. All of these functions return a boolean indicator. A "1" returned indicates the function was successful. If a "0" is returned, the error code can be retrieved by calling OL_Buffered_GetErrorCode() - defined in the Housekeeping group.

Buffered Mode Housekeeping

There are a number of "housekeeping" type functions associated with the buffered mode.

Update Nodes
OL_Buffered_UpdateNodes()

The OL_Buffered_UpdateNodes() function is used to update all specified nodes in buffered mode. Any specified node will be updated and any return information (input status, comm port receive data, etc.) will be reported from the specified node(s) and stored within the memory buffer of the host.

The function is called by passing in an array containing the node address of each RTU that you want to update. A value containing the total number of nodes that you want to update (i.e. number of elements in the array) also has to be passed in to the function.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_UpdateNodes Lib "OPTILOGICPROTOCOL.DLL" (ByRef
    nodearray As Integer, ByVal arraysize As Long) As Integer
```

where:	nodearray	= an array containing each active node address
	arraysize	= the number of active nodes in the array

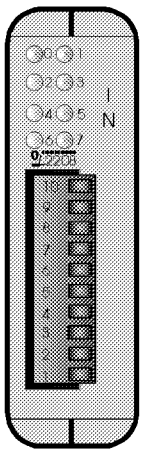
Usage

The following example shows nodes 1 - 99 being queried. If the node is found, it is added to the node list. Then a call to OL_Buffered_UpdateNodes() is shown.

```
'Global variables

Public glb_NodeQueryList(99) As Integer          ' array containing list of active nodes
Public glb_NodeQuerySize As Long                 ' total # of nodes found



' This code is in a routine which only has to be called once. It checks each possible node to see
' if an RTU exists at that address. If it exists, then it stores the node address into an array and
' increments a counter to keep track of the total number of nodes.

Dim noslots As Integer, i As Integer
Dim mtype(9) As Byte, stype(9) As Byte, mver(9) As Byte, minver As Byte, majver As Byte
Dim basetype As Byte

glb_NodeQuerySize = 0
For i = 1 to 99
        noslots = OL_QueryNode(i, basetype, mtype(0), stype(0), mver(0), minver, majver)
        If noslots > 0 Then     ' If a node is found add it to the node list
                glb_NodeQueryList(glb_NodeQuerySize) = i        ' add node to array for
                                                                ' buffered mode
                glb_NodeQuerySize = glb_NodeQuerySize + 1       ' increment number of
                                                                ' acitve nodes found
        End If
Next i



' This code can be in another routine such as a timer to read and update the status of each
' node at a fixed interval.

Dim status As Integer

status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)
```

Note: Always remember to call OL_QueryNode() or OL_GetFirst() and OL_GetNext() to find each node in the network before using Buffered Mode. If you don't query a node before OL_Buffered_UpdateNodes() is called, then the routine call will return an error condition and it will not attempt communications with the node(s).

Read Digital Inputs
OL_Buffered_ReadDigitalInput()



This function can be used to read a digital input module's inputs at a particular node address. The returned data is placed in a "long" variable (32 bits) with the status of each input point being indicated by a bit within the variable. A "1" in the bit position indicates active. The bits are in sequence starting at bit 0. If the input module has less than 32 inputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ReadDigitalInput Lib "OPTILOGICPROTOCOL.DLL"  (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef  data
        As Long) As Integer
```

Usage :

The following example will check input 3 on the input module residing at slot 2 at node 23.

```
Dim digins As Long          ' input status returned from RTU
Dim slot As Integer         ' slot input card resides in
Dim modtype As Byte         ' digital input type
Dim NODEADDR As Integer     ' node address

nodeaddr = 23               ' node address 23
slot = 2                    ' slot 2
modtype = 1                 ' 8 channel digital inputs are type 1

status = OL_Buffered_ReadDigitalInput(NODEADDR, slot, modtype, digins)
If status = 0 Then
        ' ********************
        ' place error processing here
        ' ********************
End If

' Somewhere later in the loop if input 3 on, process
If digins And  &H08 Then
        ' ********************
        ' application processing
        ' ********************
End If
```

Read Latched Digital Inputs
OL_Buffered_ReadLatchedDigitalInput()

This function can be used to see if a digital input module's inputs have turned ON at any time. Suppose that there is an input that may have turned ON and then back OFF in between calls to the routine OL_Buffered_ReadDigitalInput(). In those cases the input signal will be missed by your program, but you may need to know that it turned ON, if only briefly. In that case, call the routine OL_Buffered_ReadLatchedDigitalInput(). The OptiLogic RTU stores the "latched" status of each input. If the input ever turns ON, the "latched" status will be a "1" regardless of the current input state. The status will stay at "1" until read by the routine OL_Buffered_ReadLatchedDigitalInput(). If the input is OFF when the routine is called, it will be reset to "0". If it is still ON, the "latched" input bit will stay at "1".

The data returned from the routine call is placed in a "long" variable (32 bits) with the status of each input point being indicated by a bit within the variable. A "1" in the bit position indicates active. The bits are in sequence starting at bit 0. If the input module has less than 32 inputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ReadLatchedDigitalInput Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef
        data As Long) As Integer
```

where data = status of inputs (input 0 --> bit 0, input 1 --> bit 1 ...)

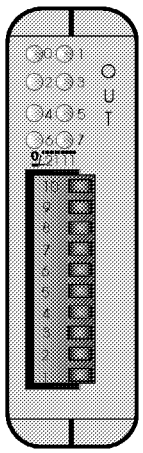
Continued on next page.

Usage :

The following example will check to see if input 3 on the input module residing in slot 2 at node 23 has latched ON.

```
Dim dig_data As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 23          ' node address 23
slot = 2               ' input card resides in slot 2
modtype = 1            ' 8 channel digital inputs are type 1

' Read the Latched Input Status
status = OL_Buffered_ReadLatchedDigitalInput(nodeaddr, slot, modtype, dig_data)
If status = 1 Then
        If (dig_data And &H08) Then
                ' *********************
                ' input 3 is ON, process data
                ' *********************
        End If
End If
                                .
                                .
                                .
                                .

' update node 23 and any other nodes in the system

status = OL_Buffered_UpdateNodes(NodeArray(0), NumNodes)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If



'End of Example
```

Write Digital Outputs
OL_Buffered_WriteDigitalOutput()



The OL_WriteDigitalOutput() function can be used to write required output state data to a digital output module at a particular node address. The command sends an unsigned long (32 bits) to the module. Each bit, starting with the LSB, indicates the required state for one output point. A "1" in a bit position indicates the required output state is "active", i.e. relay closed, transistor on, etc.. If the output module has less than 32 outputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_WriteDigitalOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByVal data As
        Long) As Integer
```

Usage :

The following example will turn on outputs 3, 5 and 6, and all other points off at the output module residing at slot 3 at node 23. The returned integer is not used in the first case, it is used in the second.

```
Dim dig_outs As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 23                ' address of RTU
slot = 3                     ' card resided in slot 3
modtype = 9                  ' 8 channel digital outputs are type 9

dig_outs = &H68              ' turn on outputs 3,5 and 6

' First case:  Just send outputs, ignore the response
status = OL_Buffered_WriteDigitalOutput(nodeaddr, slot, modtype, dig_outs)


' Second case:  Send outputs, and use the response
status = OL_Buffered_WriteDigitalOutput(nodeaddr, slot, modtype, dig_outs)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If
```

Read Digital Outputs
OL_Buffered_ReadDigitalOutput()

The OL_Buffered_ReadDigitalOutput() function can be used to read the current output state from a digital output module. The "data" variable returns a long (32 bits) that represents the current state of the outputs. Each bit, starting with the LSB, indicates the required state for one output point. A "1" in a bit position indicates the required output state is "active", i.e. relay closed, transistor ON, etc.. If the output module has less than 32 outputs (most cases), the higher order bits are unused.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ReadDigitalOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByRef data As
        Long) As Integer
```

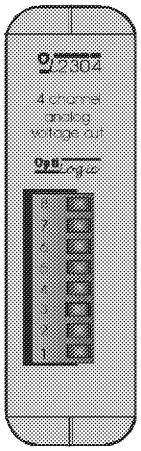
where data = status of outputs (output 0 --> bit 0, output 1 --> bit 1 ...)

Usage :

The following example will read the status of each output on an 8 point output module. The module resides at node 5 in slot 0. Then the example checks to see if output 5 is ON.

```
Dim out_data As Long
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 5          ' output card resides at node address 5
slot = 2              ' output card resides in slot 2 of node 5
modtype = 9           ' 8 channel digital outputs are type 9

status = OL_Buffered_ReadDigitalOutput(nodeaddr, slot, modtype, out_data)
If status = 1 Then
        If (out_data And &H20) Then
                ' *********************
                ' output 5 is ON, process application
                ' *********************
        End If
End If


'******************* somewhere in the program, call UpdateNodes ****************************
' update node 5 and any other nodes in the system

status = OL_Buffered_UpdateNodes(NodeArray(0), NumNodes)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If

'End of Example
```

OL2304 Analog Output Module



The OL2304 analog output module has 4 channels. Each channel is independently configured as either 0-5V, 0-10V, +/-5V or +/-10V. Each channel has a 12-bit resolution (1 in 4096) with a scale of 0 - 4095.

To write the output data you must first decide the voltage range of each channel. Next, you must configure the OL2304. Then, define an array of integers (16-bit) with an index of 4, place the output data in the array and write the data to the output module. The maximum output value is 4095. Any value over 4095 will be automatically set to 4095.

The following buffered mode function calls are available for the OL2304 analog output module.

- OL_Buffered_ConfigAnalogOutput() - A single call configures the analog output range for each channel.
- OL_Buffered_WriteAnalogOutput() - Writes the analog output values to each channel.

Configure Analog Outputs
OL_Buffered_ConfigAnalogOutput()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ConfigAnalogOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByVal IOType As Byte, ByVal range1
    As Byte, ByVal range2 As Byte, ByVal range3 As Byte, ByVal range4 As Byte) As
    Integer
```

Where: range 1 - voltage output range for channel 1
range 2 - voltage output range for channel 2
range 3 - voltage output range for channel 3
range 4 - voltage output range for channel 4

Each range is configured in the following manner:

rangex = 0 : 0-5 Volt range
1 : 0-10 Volt range
2 : +/- 5 Volt range
3 : +/- 10 Volt range

rangex = range1, range2, range3 or range4

Note: The channels have to be reconfigured every time the OptiLogic base has it's power cycled.
If you change the configuration of a channel "on the fly", make sure you set the channel's output to 0 or unexpected results may occur with your device.

Continued on next page.

Write Analog Outputs
OL_Buffered_WriteAnalogOutput()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_WriteAnalogOutput Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByRef data As Integer, ByVal IOType
    As Byte, ByVal numValues As Byte) As Integer
```

where: IOType - 25 (OL2304 has 4 analog outputs, therefore the type is 25)
data - an array (index = 4) to place the output values into.
The data should be in the range of 0 - 4095.
NumValues - # of analog outputs (numValues = 4 for an OL2304)

Conversion of Analog Output Voltage to Equivalent Output Value

0-5 Volt or 0-10 Volt Range

To convert an output voltage to the equivalent value, use the following formula:

x = (y * 4095) / z

where x = equivalent value in the 0-4095 range to output
y = output voltage value
z = output range maximum (5 or 10 depending on configuration)

Example: A channel is configured for 0-10 Volts and a 3.3 Volt output is required, the value will be calculated as follows.

x = (3.3 * 4095) / 10 = **1351**

+/-5 Volt or +/-10 Volt Range

To convert an output voltage to the equivalent value, use the following formula:

x = ((z + y) / (2 * z)) * 4095

where x = equivalent value in the 0-4095 range to output
y = output voltage value
z = output range maximum (+5 or +10 depending on configuration)

Example: A channel is configured for +/-5 Volts and a +2.5 Volt output is required, the value will be calculated as follows.

x = ((5 + (+2.5)) / (2 * 5)) * 4095 = **3071**

Example: A channel is configured for +/-10 Volts and a -2.5 Volt output is required, the value will be calculated as follows.

x = ((10 + (-2.5)) / (2 * 10)) * 4095 = **1536**

Continued on next page.

Usage :

Example A: The following example will calculate the equivalent output value (0 - 4095 range) when the output voltage range is 0-5VDC.

```
Dim AnalogOut_Val(4) As Integer          'number between 0 and 4095 that is equivalent to
                                         'output voltage value
Dim VoltageOut_Val(4) As Single          'voltage value to output

' initialize output values
VoltageOut_Val(0) = 5
VoltageOut_Val(1) = 2.5
VoltageOut_Val(2) = 1.22
VoltageOut_Val(3) = 3.3

'Convert to equivalent output value and place in an array.
For i = 0 to 3
        ' use the formula x = (y * 4095) / 5
        ' Notice that in the next line CInt() is used to round the decimal point.
        AnalogOut_Val(i) = CInt((VoltageOut_Val(i) * 4095) / 5)
Next i

' The results from the above calculations are as follows:
'       AnalogOut_Val(0) = 4095          ' 5VDC output
'       AnalogOut_Val(1) = 2048          ' 2.5VDC output
'       AnalogOut_Val(2) = 999           ' 1.22VDC output
'       AnalogOut_Val(3) = 2702          ' 3.3VDC output
' End of example
```

Example B: The following example will configure each channel for the 0-5VDC range on an OL2304 residing at node 21in slot 2.

```
Dim range1 As Byte, range2 As Byte, range3 As Byte, range4 As Byte
Dim modtype As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 21                ' node address
slot = 2                     ' slot 2
modtype = 25                 ' 4 channel analog outputs are type 25

range1 = 0                   ' 0 - 5 Volt range
range2 = 0
range3 = 0
range4 = 0

status = OL_Buffered_ConfigAnalogOutput(nodeaddr, slot, modtype, range1, range2, range3,
range4)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If

' End of Example
```

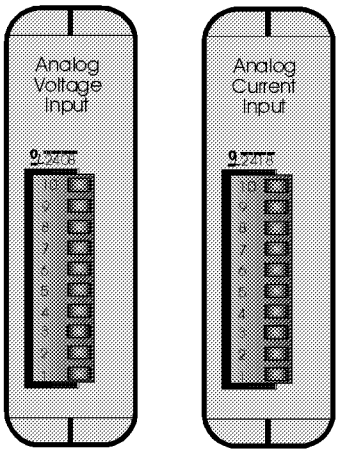
Example C: The following example will cause a specified voltage to output from each channel of an OL2304 residing at node 21in slot 2.

```
Dim analog_val(4) as Integer
Dim modtype As Byte
Dim slot As Integer, nodeaddr as Integer, status as Integer

nodeaddr = 21                ' node address
slot = 2                     ' OL2304 resides in slot 2
modtype = 25                 ' 4 channel analog outputs are type 25

' The card has been configured for an output range of 0 - 5VDC (see Example B for
' configuration)

' As calculated in Example A, the equivalent output values are as follows:

analog_val(0) = 4095         ' 5VDC output
analog_val(1) = 2048         ' 2.5VDC output
analog_val(2) = 999          ' 1.22VDC output
analog_val(3) = 2702         ' 3.3VDC output
status = OL_Buffered_WriteAnalogOutput(nodeaddr, slot, modtype, analog_val(0))
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If

'End of Example
```

Example D: The following example will update node 21. It will write output type data to the node that has had the an output type routine called (OL_Buffered_WriteAnalogOutput(), etc.). It will also update all input values so that when an input routine (OL_Buffered_ReadDigitalInput(), etc.) is called. The value returned will be the current status.

```
Dim nodeArray(1) as Integer  ' array containing active node addresses
Dim NumNodes as Long         ' number of active nodes
Dim status as Integer

NodeArray(0) = 21            ' array containing active node addresses
NumNodes = 1                 ' update only 1 node
nodeaddr = 21                ' update node 21

' update node 21
status = OL_Buffered_UpdateNodes(NodeArray(0), NumNodes)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If

'End of Example
```

Read Analog Inputs
OL_Buffered_ReadAnalogInputs()



This function can be used to read all of the analog inputs at a particular analog input module at a particular node address. The returned data is placed in an array. The values are straight unscaled readings from the input module. To use this function you must define an array of unsigned shorts (16 bit) at least as large as the number of analogs on the module being read.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ReadAnalogInputs Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByRef data As Integer, ByVal IOType
        As Byte, ByVal numValues As Byte) As Integer
```

where : data - an array holding the analog input values
IOType - the module type for the analog input (See Appendix B)
numValues - number of inputs to read

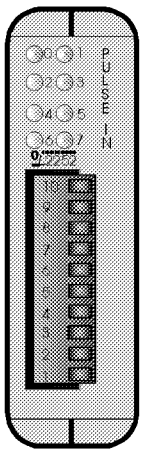
Usage :

The following example will read inputs from an 8 channel analog input module residing in slot 0 at node 23.

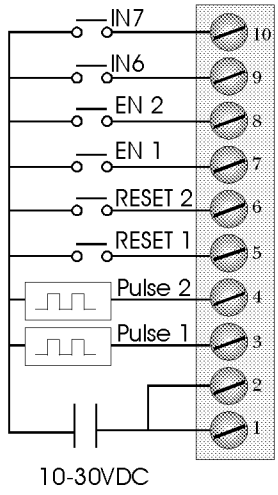
```
Dim analog_val(8) As Integer
Dim modtype As Byte, numChannels As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

nodeaddr = 23                ' RTU is address 23
slot = 0                     ' card resides in slot 0
modtype = 18                 ' 8 channel analog inputs are type 18
numChannels = 8              ' 8 channels

status = OL_Buffered_ReadAnalogInputs(nodeaddr, slot, analog_val(0), modtype,
        numChannels )
If status = 1 Then
        ' ********************
        ' Handle analog values here
        ' ********************
Else
        ' ********************
        ' Place Error handling code here
        ' ********************
End If
```

The 8 analog values will be placed in the array analog_val. Analog input 0 will reside in analog_val(0), input 1 in analog_val(1), etc.

OL2252 Dual High Speed Pulse Counter



The OL2252 Dual High Speed Pulse Counter module has two 0-15KHz pulse counter inputs. It also has four other digital inputs. Four of these other inputs (two for each channel) can be configured as control inputs for the pulse counter.



The figure on the left illustrates the pulse counter interface. Input 0, attached to terminal 3, is the first pulse input channel. EN 1 (Input 4, terminal 7) can be configured for use as an enable counting input for Pulse 1. RESET 1 (Input 2, terminal 5) can be configured for an external input to reset the count to 0. EN 2 and RESET 2 can likewise be configured as external control signals for Pulse input 2. Any input not used for its count related function can be used as a general purpose input.

In addition to the hardware reset and enable signals, each pulse counter can be sent a message from the host computer for "reset" and "enable".

The following buffered mode function calls are available for the OL2252 pulse counter module.

- OL_Buffered_Configure_Counter() - Define which, if any, hardware controls are to be used, as well as a debounce count associated with each channel.
- OL_Buffered_Read_Counter() - Sends control signals (reset & enable) to each counter and reads current count values.

Configure Counter
OL_Buffered_Configure_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_Configure_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal ctlFlags As Byte, ByVal dbSet1
        As Byte, ByVal dbSet2 As Byte) As Integer
```

where : ctlFlags - control flags
- bit 0 - use channel 1 hardware enable
- bit 1 - use channel 1 hardware reset
- bit 4 - use channel 2 hardware enable
- bit 5 - use channel 2 hardware enable

dbSet1 - debounce count for channel 1 (establishes the max pulse frequency that the channel will count)
- dbset1 = 2 - 15 KHz
- = 4 - 10 KHz
- = 8 - 5 KHz
- = 16 - 2.5 KHz
- = 40 - 1 KHz

dbSet2 - debounce count for channel 2

OL2252 continued

Send counter controls and read counts
OL_Buffered_Read_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_Read_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer, ByVal ctlFlags As Byte, ByRef
        countData As Byte, ByRef inData As Byte) As Integer
```

where : ctlFlags - control flags
- bit 0 - channel 1 enable
- bit 1 - channel 1 reset
- bit 4 - channel 2 enable
- bit 5 - channel 2 reset

countData - an array of bytes that hold 2 32-bit count values. The 1st element holds the 8 most significant bits of the channel 1 count. The second array element holds the next 8 bits. The third element holds the next 8 bits and the fourth element holds the 8 least significant bits of the channel 1 count. The count for the second channel follows in the same format. The 5th array element holds the 8 most significant bits of the channel 2 count. The 6th array element holds the next 8 bits of channel 2. The 7th element holds the next 8 bits and the 8th element holds the 8 least significant bits of the channel 2 count.

inData- a variable that holds the input status of all 8 input lines. All input points can be used as general purpose inputs, if so desired.

OL2252 continued

Usage :

The following example configures the first channel to use the hardware enable and reset. It configures the second channel to use *neither* the hardware reset nor enable. The program checks both channel counts. When a count of 400,000 is reached on channel 1, it will send an output signal to a device attached to the RTU. It depends on the external device to reset and re-enable the count. The second channel is checked for a value above 1,000,000. When that value is reached, the program will disable and reset the channel 2 count and perform an operation. When the operation is complete, it will start the process again.

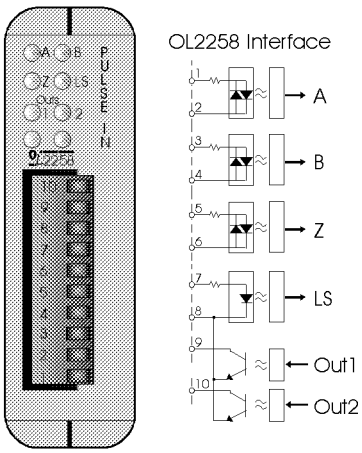
```
Dim slot As Integer, nodeaddr As Integer, status As integer
Dim hw_control As Byte, sw_control As Byte, debounce1 As Byte, debounce2 As Byte,
            in_data As Byte, count_val(8) As Byte
Dim ch1_val As Long, ch2_val As Long
Dim ch_str As String, temp As String

slot = 3
nodeaddr = 7
hw_control = &H03    ' Enable ch 1 hardware control, disable ch 2 hw control
debounce1 = 2        ' 20KHz max rate for both channels
debounce2 = 2
status = OL_Buffered_Configure_Counter (nodeaddr, slot, hw_control, debounce1, debounce2)


While (1)                ' loop forever
    sw_control = &H11    'enable channel 1 and channel 2
    status = OL_Buffered_Read_Counter(nodeaddr, slot, sw_control, count_val(0), in_data)
    'Convert channel 1 data to a long value
        For i = 0 To 3
            temp = CStr(Hex(count_val(i)))
            If Len(temp) = 1 Then          ' if value only has 1 digit, pad with a 0
                ch_str =  ch_str + "0" + temp
            Else
                ch_str = ch_str + temp
            End If
        Next i
        ch_str = "&H" + ch_str
        ch1_val = CLng(ch_str)        ' channel 1 count value

    If ch1_val  > 400000 Then
        '*****************
        ' Send output signal
        '*****************
    Else
        '*****************
        ' clear output signal
        '*****************
    End If
```

(continued on next page)

OL2252 continued

```
'Convert channel 2 data to a long value
        For i = 4 To 7
                temp = CStr(Hex(count_val(i)))
                If Len(temp) = 1 Then           ' if value only has 1 digit, pad with a 0
                        ch_str =  ch_str + "0" + temp
                Else
                        ch_str = ch_str + temp
                End If
        Next i
        ch_str = "&H" + ch_str
        ch2_val = CLng(ch_str)          ' channel 2 count value

If ch2_val  > 1000000
        sw_control = &H21     'reset and disable channel 2, keep channel 1 enabled

        '*****************
        ' Reset count
        '*****************
        OL_Buffered_Read_Counter(nodeaddr, slot, sw_control, count_val(0), in_data)

        '*****************
        ' Perform required operation
        '*****************
End If

' update all nodes
status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)

DoEvents        ' process any system events that may occur

Wend
```

OL2258 High Speed Pulse Counter



The OL2258 High Speed Pulse Counter is designed to interface to a variety of standard pulse encoder devices. The electrical interface is shown on the right. Differential, sourcing or sinking inputs can be interfaced to the OL2258.

The OL2258 is configurable. It can be used with pulse and direction, quadrature or up/down count type pulse encoders. The OL2258 maintains the current cumulative count as a 32 bit integer value. It also makes frequency snapshot data available over the most recent count of 1 second or 200 milliseconds. The Z and LS inputs can be used to automatically reset the count to a user defined "preset" value. Each transistor output can be configured to turn on when the count value is within a specified range.

For more detailed information on the features of the OL2258 High Speed Pulse Counter, see the section on the OL2258 in the **OptiLogic Input/Output Modules** manual.

The following buffered mode unction calls are available for the OL2258 High Speed Pulse Counter module.

- OL_Buffered_Configure_HS_Counter() - Defines count type (pulse and direction, quadrature or up/down), configures preset inputs, frequency period and preset value.
- OL_HS_SendOutput_Range_Counter() - Configures minimum and maximum range to turn ON Output 1 or 2.
- OL_Buffered_Read_HS_Counter() - Returns input status, output status, count type, current count value and frequency count over selected time period.

Term	Label	Description
1	A1	Pulse input A (quadrature)/ Pulse input (pulse & direction)/ Up pulse (up/down count)
2	A2	
3	B1	Pulse input B (quadrature)/ Pulse input (pulse & direction)/ Up pulse (up/down count)
4	B2	
5	Z1	Z input (optional)
6	Z2	
7	LS	Limit Switch input (optional)
8	COM	Common for limit switch and the two outputs.
9	Out1	Open collector output1
10	Out1	Open collector output2

Continued on next page.

Configure High Speed Counter
OL_Buffered_Configure_HS_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_Configure_HS_Counter Lib "OPTILOGICPROTOCOL.DLL"
    (ByVal NODEADDR As Integer, ByVal SlotNo As Integer, ByVal ctlFlags As Byte, ByVal
    cfgFlags As Byte, ByRef presetVal As Long) As Integer
```

where: ctlFlags = bits 0, 4-7 : Unused
bit 1 : Count type - Pulse & Direction
bit 2 : Count type - Up/Down Count
bit 3 : Count type - Quadrature

cfgFlags = bit 0 : Output 1 range enabled - enable output 1 if in range
bit 1 : Output 2 range enabled - enable output 2 if in range
bit 2 : Unused
bit 3 : Z preset enabled - when Z input ON, force to preset value
bit 4 : LS preset enabled - when LS input ON, force to preset value
bit 5 : force preset - when set, force count to preset value
bit 6 : hold count - when set, hold count at current value
bit 7 : frequency period selection
0 = 1second count (good up to 30KHz)
1 = 200 msec count (for over 30KHz)

presetVal = 32 bit value to force pulse count to when a preset enabled or force preset bit is set.

Note: Only turn ON one bit in ctlFlags, ensure that the rest are OFF

Send Output Range
OL_Buffered_HS_SendOutput_Range_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_HS_SendOutput_Range_Counter Lib
    "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal SlotNo As
    Integer, ByVal output As Byte, ByRef minValue As Long, ByRef maxValue As Long) As
    Integer
```

where: output = output to configure (1 or 2)
minValue = mininum value for range to turn ON specified output
maxValue = maximum value for range to turn ON specified output

Continued on next page.

Read High Speed Counter
OL_Buffered_Read_HS_Counter()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_Read_HS_Counter Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer, ByVal SlotNo As Integer, ByRef countData As Byte, ByRef
    freqData As Byte, ByRef inStat As Long, ByRef outStat As Long) As Integer
```

where: countData = current count, 32 bit signed value
formatted as an array 4 bytes long
countData(0) - hi byte … to … countData(3) - low byte

freqData = pulse count in the most recent frequency period
formatted as an array 2 bytes long
freqData(0) - hi byte
freqData(1) - low byte

inStat = input status (input ON = 1, input OFF = 0) & frequency rate status
bit 0 : in_A
bit 1 : in_B
bit 2 : in_Z
bit 3: in_LS
bits 4, 5 and 6 : Unused
bit 7 : frequency selection rate
1 = 1 second count
0 = 200 msec count

outStat = output status (output ON = 1, output OFF = 0) & count type status
bit 0 : output 1 status
bit 1 : output 2 status
bits 2 - 4 : Unused
bit 5 : Count type - Pulse & Direction
bit 6 : Count type - Up/Down Count
bit 7 : Count type - Quadrature

Continued on next page.

Usage :

Example A: The configuration routine OL_Buffered_Configure_HS_Counter() will tell the OL2258 in what mode(s) it will be used. The following example shows a call which will configure the OL2258 for Quadrature type counting. It will enable Output 1 and the LS preset and set the preset count to 12867. The OL2258 resides at node 2 in slot 3.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer
Dim configFlags As Byte, controlFlags As Byte
Dim presetCntValue As Long

slot = 3                              ' The OL2258 is in slot 3.
nodeaddr = 2                          ' The OL2258 is at node 2.

controlFlags =      &H08              ' Quadrature type counting
configFlags =       &H11              ' Output 1 range enabled and LS preset enabled
presetCntValue =  12867               ' set preset count value to 12867

status = OL_Buffered_Configure_HS_Counter(nodeaddr, slot, controlFlags, configFlags,
        presetCntValue)
If status = 0  Then
        '*************************
        ' process error
        '*************************
End If
```

Example B: The following example will configure the output range for output 1 on the OL2258. When the count value is within the range of 83592 to 135000, Output 1 will be ON. If it's not within the range, Output 1 will be OFF. As in Example A, the OL2258 resides at node 2 in slot 3.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer
Dim SendOutput As Byte
Dim MinRangeValue As Long, MaxRangeValue As Long


slot =                3               ' The OL2258 is in slot 3.
nodeaddr =            2               ' The OL2258 is at node 2.
SendOutput =          1               ' Configure range for Output 1
MinRangeValue =       83597           ' Configure minimum limit for 83597
MaxRangeValue =       135000          ' Configure maximum limit for 135000

status = OL_Buffered_HS_SendOutput_Range_Counter(nodeaddr, slot, SendOutput,
        MinRangeValue, MaxRangeValue)
If status = 0  Then
        '*************************
        ' process error
        '*************************
End If
```

Continued on next page.

Example C: The following example will read the current count value, input status and output status on the OL2258. It will convert the current count value to a long and the frequency count to a frequency value in Hertz. As in Examples A and B, the OL2258 resides at node 2 in slot 3.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer, Frequency As Integer
Dim countValue(4) As Byte, freqCount(2) As Byte
Dim inputStatus As Long, outputStatus As Long, CurrentCount As Long
Dim ch_str As String, temp As String


slot =          3               ' The OL2258 is in slot 3.
nodeaddr =      2               ' The OL2258 is at node 2.

status = OL_Buffered_Read_HS_Counter(nodeaddr, slot, countValue(0), freqCount(0),
        inputStatus, outputStatus)
If status = 0  Then
        '*************************
        ' process error
        '*************************
Else
        '*****Convert the countValue to a long *********

        For i = 0 to 3
                temp = CStr(Hex(countValue(i)))
                If Len(temp) = 1 Then           ' if temp only has 1 digit, pad it with a leading 0
                        ch_str = ch_str + "0" + temp
                Else
                        ch_str = ch_str + temp
                End If
        Next i

        ch_str = "&H" + ch_str
        CurrentCount = CLng(ch_str)             ' count value as a long

        '*****Convert the frequency count to an integer and then to frequency in Hz*********

        For i = 0 to 1
                temp = CStr(Hex(freqCount(i)))
                If Len(temp) = 1 Then           ' if value only has 1 digit, pad it with a leading 0
                        ch_str = ch_str + "0" + temp
                Else
                        ch_str = ch_str + temp
                End If
        Next i

        ch_str = "&H" + ch_str
        Frequency = CInt(ch_str)                ' frequency count value as an integer
```

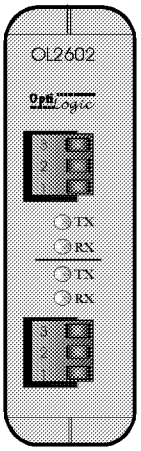
Continued on next page.

```
'*****Depending on time period, calculate frequency in Hz *********

If inputStatus And &H80 Then          ' if time period = 200ms
        Frequency = Frequency / 0.2
        '****if time period = 1 second, Frequency already calculated****
End If


End If

'End of Example
```

Example D: Update Node 2 - The following example will update node 2. It will write data to the node that has had the an output type routine called (OL_Buffered_HS_SendOutput_Range_Counter(), etc.). It will also update all input values so that when an input routine (OL_Buffered_Read_HS_Counter(), etc.) is called, the value returned will be the current status.

```
Dim nodeArray(1) as Integer  ' array containing active node addresses
Dim NumNodes as Long         ' number of active nodes
Dim status as Integer

NodeArray(0) = 2             ' array containing active node addresses
NumNodes = 1                 ' update only 1 node

' update node 2
status = OL_Buffered_Read_HS_Counter(NodeArray(0), NumNodes)
If status = 0 Then
        ' ********************
        ' Place Error handling code here
        ' ********************
End If

'End of Example
```

OL2602 Dual RS232 and Base Serial Comm Port



The OL2602 module is a Dual RS232 communications module. Each of the ports operates independently at 1200 -19200 baud, 7 or 8 data bits, configurable parity and stop bits. The serial communications port on the OL4054 base operates with the same parameters as the OL2602 except it can handle a range of baud rates from 300 -19200.

The serial ports operate as a typical general purpose communication port with a fair-sized buffer. The RTU transmit buffers are 48 bytes and receive buffers are 48 bytes each. Any message can be sent or received. Messages longer than 48 bytes must be buffered within your program in such a way that they do not overflow the transmit buffer. Also, the host program must be careful to communicate with the RTU often enough that the receive ports do not overflow.

With the OL2602 configuration of either port can mean the loss of transmit and receive data in both ports. Therefore, configuration should occur once, at initialization. If reconfiguration is required during operation, realize that a message on the alternate port may be garbled.

The following buffered mode function calls are available for the OL2602 Dual RS232 module and the OL4054 base serial communications port.

- OL_Buffered_ConfigureSerialPort() - Set up port operating parameters
- OL_Buffered_ReadSerialData() - Read receive data from a port
- OL_Buffered_WriteSerialData() - Send data to be transmitted from a port

Configure a Communication Port OL_Buffered_ConfigureSerialPort()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ConfigureSerialPort Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    glbNodeId As Integer, ByVal SlotNo As Integer, ByVal modtype As Byte, ByVal PortNo
    As Integer, ByVal BaudRate As Integer, ByVal DataBits As Integer, ByVal Parity As
    Integer, ByVal StopBits As Integer) As Integer
```

The following parameters are defined in the file DLLdefinitions.bas

where :	SlotNo	- 0, 1 for OL2602 or 81 for Base Comm Port
	modtype	- 112 for OL2602 or 0 for Base Comm Port (see DLLdefinitions.bas)
	PortNo	- 0 (base), 1 (OL2602 - top) or 2 (OL2602 - bottom)
	BaudRate	- see DLLdefinitions.bas
	DataBits	- see DLLdefinitions.bas
	Parity	- see DLLdefinitions.bas
	StopBits	- see DLLdefinitions.bas

Use the selections listed in the see DLLdefinitions.bas file.

Read received Data
OL_Buffered_ReadSerialData()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_ReadSerialData Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    glbNodeId As Integer, ByVal SlotNo As Integer, ByVal modtype As Byte, ByVal PortNo
    As Integer, ByRef numbytes As Byte, ByRef flags As Byte, ByRef message As Byte) As
    Integer
```

where :	SlotNo	- 0, 1 for OL2602 or 81 for Base Comm Port
	modtype	- 112 for OL2602 or 0 for Base Comm Port (see DLLdefinitions.bas)
	PortNo	- 0 (base), 1 (OL2602 - top), or 2 (OL2602 - bottom)
	numbytes	- Variable which will contain the # of receive bytes returned.
	flags	- Variable to store returned error flags associated with received bytes
	message	- Array containing received data

Write to Serial Port
OL_Buffered_WriteSerialData()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_WriteSerialData Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    glbNodeId As Integer, ByVal SlotNo As Integer, ByVal modtype As Byte, ByVal PortNo
    As Integer, ByVal numbytes As Byte, ByRef message As Byte, ByRef Leftover As Byte)
    As Integer
```

where :	SlotNo	- 0, 1 for OL2602 or 81 for Base Comm Port
	modtype	- 112 for OL2602 or 0 for Base Comm Port (see DLLdefinitions.bas)
	PortNo	- 0 (base), 1 (OL2602 - top), or 2 (OL2602 - bottom)
	numbytes	- Variable which will contain the # of receive bytes returned.
	message	- Array containing data to transmit
	left	- Variable in which the # of bytes left in the transmit buffer will be returned

OL2602 continued

Usage :

Port Configuration

To configure a serial port the OL_Buffered_ConfigureSerialPort() routine should be called during the program initialization process.

The following example configures the top port on an OL2602 for 9600 baud, 8 data bits, no parity and 1 stop bit. The node address is 7 and the OL2602 is in slot 0 of the RTU. Ensure that the OL_Buffered_UpdateNodes() function is called at some point after the call to OL_Buffered_ConfigureSerialPort() or the configuration message will not be sent to the RTU.

```
Dim slot As Integer, nodeaddr As Integer, status As Integer

slot = 0
nodeaddr = 7
status = OL_Buffered_ConfigureSerialPort(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
        OL_9600_BAUD, OL_8_DATA_BITS, OL_NO_PARITY, OL_1_STOP_BIT)
' OL_Buffered_UpdateNodes has to be called before the data will be sent to the RTU
' update all nodes
status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)
```

Reading data from a Serial Port

To read data from a serial port in buffered mode, the routine OL_Buffered_UpdateNodes() must be called. Then the routine OL_Buffered_ReadSerialData() must be called. The routines have to be called often enough so that data loss does not occur. If the routine is not called often enough, the receive buffer (48 bytes) will overflow and you will lose data. Once the port is read, the data clears from the buffer. The best way to read the data is to start a timer and read the serial port each time the timer timeout occurs. The interval should be set low enough to keep the data from overflowing. The following example assumes that a timer has been created in Visual Basic with the name tmrRx1. The interval for the timer may vary depending on your application. You should adjust the interval within the program until you get consistent results.

```
'The following 2 variables are global and should be dimensioned in a .bas module

Public   glb_RxBuffer(1024) As Byte ' This array can be used to hold the current message
                                    ' received by the serial port so that it can be used for later
                                    ' processing. The actual size of this buffer depends on
                                    ' your application.
Public   glb_RxByteCount  As Integer  'the total number of bytes received for a message
```

(continued on next page)

OL2602 continued

```
Private Sub tmrRx1_Timer()

Dim numbytes As Byte, flags As Byte, message(48) As Byte
Dim slot As Integer, nodeaddr As Integer, status As Integer

        slot = 0
        nodeaddr = 7
        tmrRx1.Enabled = False        ' disable receive timer
        ' read the data from the RTU and store in the memory of the host PC
        ' update nodes
        status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)
        ' read the data from the PC memory buffer into the application program
        status = OL_Buffered_ReadSerialData(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
                numbytes, flags, message(0))

        For i = 0 To (numbytes - 1)              ' buffer received data in global receive buffer
                glb_RxBuffer(glb_RxByteCount) = message(i)
                glb_RxByteCount = glb_RxByteCount + 1    ' increment global receive buffer
                                                         ' count
        Next i


        ' Logic can be placed here to determine if a complete message has been received.
        ' Requirements for a complete message may change depending on your application.  It
        ' could be a fixed number of bytes, a certain character, etc.  When a complete message
        ' has been received, you can set a flag to notify another portion of your program or you
        ' can check the message in this routine.

        tmrRx1.Enabled = True        ' This is optional, you might want to enable your timer in a
                                     ' different place in the program.

End Sub
```

(continued on next page)

Writing data to a Serial Port

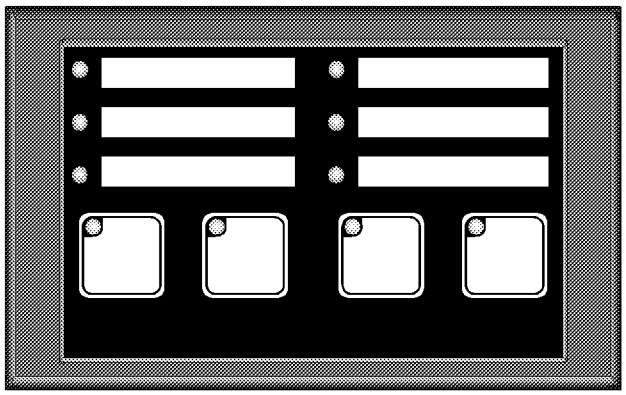
The following routine is an example that can be used to transmit data from the top port of an OL2602 in buffered mode. This example is better suited to immediate mode but it can be related to buffered mode. The node address of the RTU is 7 and it is in slot 0.

```
Private Sub TransmitData(TransmitLength As Byte, TransmitBuffer() As Byte)

Dim BytesTransmitted As Integer, status As Integer
Dim message(48) As Byte, BytesLeft As Byte, NumberToTransmit As Byte

'*********************************************************************
'TransmitBuffer() = the array containing the data to be transmitted
'TransmitLength = the total number of bytes to transmit
'BytesTransmitted = the total number of bytes that have been transmitted
'BytesLeft = # of available bytes left in tx buffer of OL2602
'NumberToTransmit = number of bytes placed into transmit buffer
'*********************************************************************

BytesTransmitted = 0   'initialize to 0
BytesLeft = 48         'initialize to size of OL2602 transmit buffer
slot = 0
nodeaddr = 7

If TransmitLength > 48 Then     ' if message longer than the OL2602
                                ' transmit buffer, send it in different pieces
                                ' to prevent an overflow of the transmit buffer

        While (BytesTransmitted < TransmitLength)  'loop until entire message sent
                DoEvents
                NumberToTransmit = BytesLeft / 2     'send half the available bytes
                For i = 0 To NumberToTransmit - 1    'copy into transmit buffer
                        message(i) = TransmitBuffer(BytesTransmitted)
                        BytesTransmitted = BytesTransmitted + 1 'increment total byte counter
                Next i
                status = OL_Buffered_WriteSerialData(nodeaddr, slot, OL2602,
                          OL_2602_TOP_PORT, NumberToTransmit, message(0), BytesLeft)
                ' update the node
                status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0),
                                glb_NodeQuerySize)
        Wend

Else      'if message less than or equal to transmit buffer size, buffer entire message
        For i = 0 To TransmitLength - 1
                message(i) = TransmitBuffer(i)
        Next i
        status = OL_Buffered_WriteSerialData(nodeaddr, slot, OL2602, OL_2602_TOP_PORT,
                                TransmitLength, message(0), BytesLeft)
        ' The Update Nodes routine doesn't have to be called in this location if your transmit
        ' data is 48 bytes or less.  It can be called elsewhere in the program.
End If
End Sub
```

Buffer Mode Operator Panel Group

OL3406 Pushbutton/Indicator Panel

The OL3406 Operator Panel has 4 pushbuttons and 6 LED indicators. The buttons can be configured for either momentary or alternate action operation. The button LEDs normally reflect button status. The momentary buttons can also be configured for LED separation (direct on/off control over the ethernet). The status LED can be turned on, off, or flashed. See the OL3406 manual for further details.



The following function buffered mode calls are available for the OL3406 operator panel.

- OL_Buffered_OL3406_StatusControl() - Read button status and control the LEDs
- OL_Buffered_OL3406_ForceButtons() - Force selected alternate-action button(s) to the desired state
- OL_Buffered_OL3406_ConfigurePanel() - Define the panel configuration
- OL_Buffered_OL3406_ReadPanelConfiguration() - Read the configuration of the panel

Read the Pushbutton Status and Control the LEDs OL_Buffered_OL3406_StatusControl()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3406_StatusControl Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByRef databuffer As Byte, ByRef pbstatus As Byte) As
        Integer
```

where : databuffer is a 4 byte array with the following definition:

databuffer(0) = on/off control of the 4 inset button LEDs. This will only affect a button LED if it is momentary and configured for LED separation. Bits are in sequence starting at bit 0.

databuffer(1) = on/off control of indicator light LEDs. Bits are in sequence starting at bit 0.

databuffer(2) = flash control of 4 inset button LEDs. LED must be on to flash. Bits are in sequence starting at bit 0.

databuffer(3) = flash control of 6 indicator light LEDs. LED must be on to flash. Bits are in sequence starting at bit 0.

pbstatus - current status of panel buttons will be placed in this byte. Bits are in sequence starting at bit 0. A "1" indicates button active.

Force Alternate-Action Button Status
OL_Buffered_OL3406_ForceButtons()

Three types of button force operations are available for the OL3406 panel alternate-action buttons. Buttons can be selectively forced on, or forced off. Also, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3406_ForceButtons Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal ButtonFlags As Byte, ByVal ButtonData As Byte)
        As Integer
```

where : ButtonData defines the button which will be affected. Bits are in button sequence starting with bit 0.

ButtonFlags defines the type of force as follows (defined in DLLdefinitions.bas)

ButtonFlags = OL3406_BUTTON_ALL ; force the buttons to a state matching ButtonData (force state)

= OL3406_BUTTON_OR ; logically OR current button state with ButtonData (force on)

= OL3406_BUTTON_AND ; logically AND current button state with the complement of ButtonData (force off)

OL3406 continued

Configure OL3406 Panel
OL_Buffered_OL3406_ConfigurePanel()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3406_ConfigurePanel Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal conflags As Byte) As Integer
```

where : conflags = bits 0 - 3 defines button operation of the 4 buttons.
"0" = momentary, "1" = alternate action
= bit 7, if "1" enables LED separation on momentary button LEDs
if "0" disables LED separation

Read OL3406 Configuration
OL_Buffered_OL3406_ReadPanelConfiguration()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3406_ReadPanelConfiguration Lib
        "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByRef conflags As
        Byte) As Integer
```

where : conflags = bits 0 - 3 indicates button type.
"0" = momentary, "1" = alternate action
= bit 7, if "1" indicates LED separation on momentary button LEDs
if "0" indicates LED separation is disabled

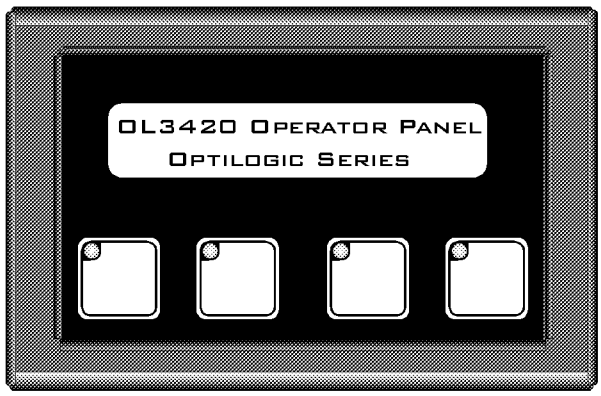
Usage :

The following example reads the panel configuration. If the configuration is not as required, it sends the configuration to the panel. It then goes into a loop. Whenever the first button is sensed active, it lights indicator lights 3 and 4, flashes light 4 and performs an application process. At the end of the process, it forces the first button off.

```
Dim    config As Byte              ' Configuration data
Dim    LEDstate(4) As Byte         ' LED state array
Dim    ButtonStatus As Byte        ' Button status
Dim    nodeaddr As Integer         ' Node address of the RTU
Dim    status As Integer           ' Return status of RTU routine calls, can be used for error
                                   '  checking
nodeaddr = 30                  ' node address of RTU is 30
config = &H07                  ' configure buttons, 1-3 alternate, 4, 5 momentary
LEDstate(0) = 0                ' Start with all lights out
LEDstate(1) = 0
LEDstate(2) = 0
LEDstate(3) = 0
ButtonStatus = 0

' configure pushbuttons
status = OL_Buffered_OL3406_ConfigurePanel(nodeaddr, config)
While (1)
        ' Forever loop
        DoEvents               ' process any system events that may occur

        ' Is the first button active?
        If  ButtonStatus  And  &H01  Then
                ' If so, turn light 3 and 4 on, flash light 4
                LEDstate(1) = LEDstate(1) Or &H0C          'lights 3 and 4 on
                LEDstate(3) = LEDstate(3) Or &H08          'flash light 4
                FunctionActive = True
                FunctionDone = False
        End If
        If FunctionActive Then
                ' Do function processing
                If FunctionDone Then
                        ' after function processing done, Force the first button OFF
                        status = OL_Buffered_OL3406_ForceButtons(nodeaddr,
                                OL3406_Button_AND, &H01)
                        If (Not(ButtonStatus And &H01)) Then
                                FunctionActive = False
                        End If
                End If
        End If
        ' read/write status and update the node
        status = OL_Buffered_OL3406_StatusControl(nodeaddr, LEDstate(0), ButtonStatus)
        ' update the node
        status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)
Wend
```

Note: The 'While' loop can be replaced by a Timer routine to update the status at a fixed interval.

OL3420 Operator Terminal



The OL3420 Operator Terminal has 4 pushbuttons and a 2 line x 20 character LCD display. The buttons can be configured for either momentary or alternate-action operation. The button LEDs normally reflect button status. The momentary buttons can also be configured for LED separation (direct on/off control over the ethernet). The status LED can be turned on, off, or flashed. See the OL3420 manual for further details.

The following buffered mode function calls are available for the OL3420 operator terminal.

- OL_Buffered_OL3420_StatusRequest() - Reads button status (and controls the LEDs if LED separation)
- OL_Buffered_OL3420_ForceButtons() - Force selected alternate-action button(s) to the desired state
- OL_Buffered_OL3420_SendTextDisplayMessage() - Send text to the display
- OL_Buffered_OL3420_ConfigurePanel() - Define the panel configuration
- OL_Buffered_OL3420_ReadConfiguration() - Read the configuration of the panel

Read the Pushbutton Status and Control Momentary Button LEDs OL_Buffered_OL3420_StatusRequest()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3420_StatusRequest Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal LEDState As Byte, ByRef ButtonStatus As Byte)
        As Integer
```

where : LEDState - applies only if LED separation is active
bits 0 - 3 turns LED ON if "1" (for buttons 1 - 4 in sequence)
bits 4 - 7 flashes LED if "1" (for buttons 1 - 4 in sequence), button LEDs must be ON to flash

ButtonStatus - current status of panel buttons will be placed in this byte. Bits are in sequence starting at bit 0 (button 1 is bit 0). A "1" indicates button active.

OL3420 continued

Force Alternate-Action Button Status
OL_Buffered_OL3420_ForceButtons()

Three types of button force operations are available for the OL3420 panel alternate-action buttons. Buttons can be selectively forced on, or forced off. Also, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3420_ForceButtons Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal ButtonFlags As Byte, ByVal ButtonData As Byte)
        As Integer
```

where : ButtonData defines the button which will be affected. Bits are in button sequence starting with bit 0.
ButtonFlags defines the type of force as follows -
ButtonFlags = OL3420_BUTTON_ALL ; force the buttons to a state matching ButtonData (force state)
= OL3420_BUTTON_OR ; logically OR current button state with ButtonData (force on)
= OL3420_BUTTON_AND ; logically AND current button state with the complement of ButtonData (force off)

Send Text to OL3420 Display
OL_Buffered_OL3420_SendTextDisplayMessage()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3420_SendTextDisplayMessage Lib
        "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal line As Byte,
        ByRef message As Byte) As Integer
```

Where : line - is the line number (0 = top, 1 = bottom) to send text to
message - is message text (20 character array containing ASCII equivalent of each character in message)

Configure OL3420 Terminal
OL_Buffered_OL3420_ConfigurePanel()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3420_ConfigurePanel Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal buttonstate As Byte) As Integer
```

where : buttonstate = bits 0 - 3 defines button operation of the 4 buttons.
"0" = momentary, "1" = alternate action
= bit 7, if "1" enables LED separation on momentary button LEDs
if "0" disables LED separation

Read OL3420 Configuration
OL_Buffered_OL3420_ReadConfiguration()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3420_ReadConfiguration Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByRef buttonstate As Byte) As Integer
```

where : buttonstate = bits 0 - 3 indicates button type.
"0" = momentary, "1" = alternate action
= bit 7, if "1" indicates LED separation on momentary button LEDs
if "0" indicates LED separation is disabled

OL3420 continued

Usage :

The following example configures the OL3420 panel. It then goes into a loop. Whenever the second button is sensed active, it sends "Process Active" to the top line of the display and performs an application process. At the end of the process, it forces the second button off.

```
Dim     RequiredConfig As Byte      ' Required configuration
Dim     PBLEDstate As Byte          ' Pushbutton LED state
Dim     ButtonStatus As Byte        ' Button status
Dim     LineNo As Byte              ' Line number for message  0 = top, 1 = bottom
Dim     message(20) As Byte         ' Message string array
Dim     nodeaddr As Integer         ' Node address of the RTU
Dim     status As Integer           ' Return status of RTU routine calls, can be used for error
                                    '   checking
Dim     DisplayMessage As String    ' String containing message string to place into buffer

RequiredConfig = &H07               ' Required configuration = buttons 1 - 3 alternate action
                                    '   button 4 momentary, no LED separation
PBLEDstate = 0                      ' Start with all lights out (unused because no LED
                                    '   separation)
DisplayMessage = "Process Active"

For  i = 0 To 19
        message(i) = &H20           'Fill message buffer with blank spaces
Next i
For i = 0 to Len(DisplayMessage) - 1
        message(i) = Asc(Mid$(DisplayMessage, i + 1, 1))  'parse message string to get ASCII
                                                          'equivalent of each character
Next i

'Configure Panel
status = OL_Buffered_OL3420_ConfigurePanel(nodeaddr, RequiredConfig)
While (1)
        ' Loop forever
        DoEvents                ' Process any system events that may occur
        ' Is the second button active ?
        If  ButtonStatus And &H02  Then
                ' If so, send message to the display
                LineNo = 0
                status = OL_Buffered_OL3420_SendTextDisplayMessage(nodeaddr, LineNo,
                        message(0))
                FunctionAcitve = True
                FunctionDone = False
        End If
        If FunctionActive Then
                ' Do function processing
```

(continued on next page)

```
            If FunctionDone Then
                ' Force the second button OFF
                status = OL_Buffered_OL3420_ForceButtons(nodeaddr,
                            OL3420_BUTTON_AND, &H02)
            End If
        End If
        'read and update the status
        status = OL_Buffered_OL3420_StatusRequest(nodeaddr, PBLEDstate, ButtonStatus)
        ' update all nodes
        status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)

Wend
```

Notes:

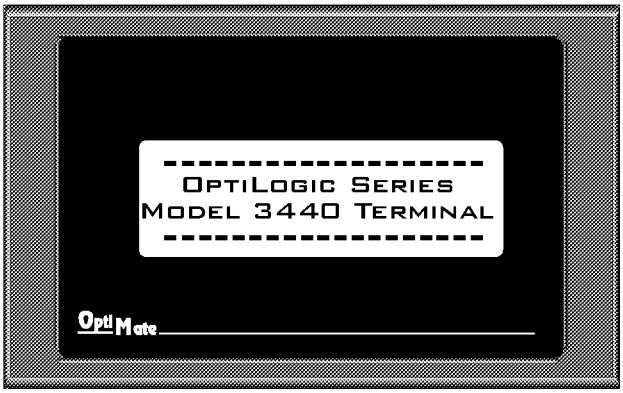
Notice that the first 'For' loop put the ASCII equivalent of a space into the message buffer. This should be done each time a new message is sent to clear old character out of the message buffer.

The second 'For' loop takes the message string and places it into the message buffer via the Len, Asc and Mid Visual Basic commands. The message string has to be converted in this manner so that each character in the string can be converted to its ASCII equivalent.

The 'While' loop can be replaced by a Timer routine to read the button status and update the message at a fixed interval.

OL3440 Display Panel

The OL3440 Display Panel is 4 line by 20 character display. You can send any text to any line.



Send Text to OL3440 Display OL_Buffered_OL3440_SendTextDisplayMessage()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3440_SendTextDisplayMessage Lib
        "OPTILOGICPROTOCOL.DLL" (ByVal glbNodeId As Integer, ByVal index As Byte,
        ByRef message As Byte) As Integer
```

where : index is line number (0 - 3 from top to bottom)
message is 20 character array of ASCII text

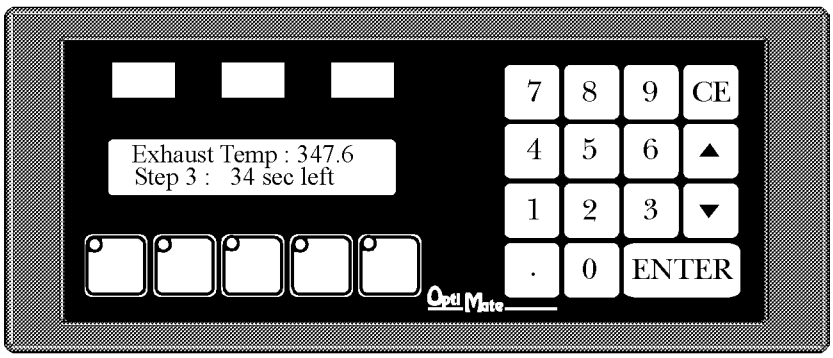
Usage

The following will send the message "Text message..Abcd" to the third line from the top on the OL3440 panel present at node address 6.

```
Dim    LineNo As Byte              ' Line number for message: 0 = top, 1 = second, 2 = third,
                                   '  3 = bottom
Dim    message(20) As Byte         ' Message string array
Dim    nodeaddr As Integer         ' Node address of the RTU
Dim    status As Integer           ' Return status of RTU routine calls, can be used for error
                                   '  checking
Dim    DisplayMessage As String    ' String containing message string to place into buffer

DisplayMessage = "Text message..Abcd"
nodeaddr = 6
For  i = 0 To 19
       message(i) = &H20           'Fill message buffer with blank spaces
Next i
For i = 0 to Len(DisplayMessage) - 1
       message(i) = Asc(Mid$(DisplayMessage, i + 1, 1))  ' parse message string to get ASCII
                                                         ' equivalent of each character, then
                                                         ' place into message buffer
Next i          ' send message to the display
LineNo = 2
status = OL_Buffered_OL3440_SendTextDisplayMessage(nodeaddr, LineNo, message(0))
```

Note: Remember to call OL_Buffered_UpdateNodes() once each pass.

OL3850 Operator Terminal



The OL3850 Operator Terminal has a two line x 20 character LCD display, a numeric keypad, five user-definable function keys and three user-definable LED indicator light bars. This panel can be used to display messages, accept numeric data input, display status indications and select functions.

The following buffered mode function calls are available for the OL3850 operator terminal

- OL_Buffered_OL3850_StatusRequest() - Control lights, read button status and retrieve entered data
- OL_Buffered_OL3850_SendTextDisplayMessage() - Send a text message to one of the two display lines
- OL_Buffered_OL3850_ConfigurePanel() - Configure the operation of the five function keys
- OL_Buffered_OL3850_ReadConfiguration() - Read the configuration of the five function keys
- OL_Buffered_OL3850_ForceButtons() - Force the state of selected “alternate-action” buttons
- OL_Buffered_OL3850_SendKeypadMessage() - Send a message with a field for the entry of a number input from the keypad
- OL_Buffered_OL3850_SendArrowMessage() - Send a message with a field for and initial value to be adjusted by the arrow keys

OL3850 continued

Send Light Controls and Read Status information OL_Buffered_OL3850_StatusRequest()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_Buffered_OL3850_StatusRequest Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal LEDState As Byte, ByVal ledflash As Byte, ByRef ButtonStatus As Byte, ByRef returndata As Double) As Integer

where : LEDState - bits 0 - 2 are on/off controls for the three light bars. Bit 0 is the left most light bar, etc.
bits 3 - 7 are controls for the indicator LED inset in the five general purpose function buttons. They only have an effect if the panel is configured for LED separation (otherwise the LEDs reflect button status) for all LEDs and light bars, "1" = on , "0" = off

ledflash - flash controls for the same LEDs and light bars mentioned for LEDState. To flash, the light must be on.

ButtonStatus - Current status of the five function buttons will be placed in this byte. Bits 0 - 4 are associated with the buttons, with the left most button in bit 0. A "1" indicates button active. Bit 7 is a flag indicating if data has been entered. A "1" indicates data has been entered.

returndata - Data entry value will be returned in this variable.

Send a Text Message to the Display OL_Buffered_OL3850_SendTextDisplayMessage()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_Buffered_OL3850_SendTextDisplayMessage Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte) As Integer

where : line - the line number (0 = top, 1 = bottom) to send text to
message - 20 character array of ASCII text

OL3850 continued

Configure OL3850 Function Buttons
OL_Buffered_OL3850_ConfigurePanel()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3850_ConfigurePanel Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal buttonstate As Byte) As Integer
```

where : buttonstate = bits 0-4 defines the operation of the 5 function buttons.
"0" = momentary, "1" = alternate action
= bit 7, if "1" enables LED separation on momentary button LEDs
if "0" disables LED separation

Read OL3850 Configuration
OL_Buffered_OL3850_ReadConfiguration()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3850_ReadConfiguration Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByRef buttonstate As Byte) As Integer
```

where : buttonstate = bits 0 - 4 indicates button type.
"0" = momentary, "1" = alternate action
= bit 7, if "1" indicates LED separation on momentary button LEDs
if "0" indicates LED separation is disabled

Force Alternate-Action Button Status
OL_Buffered_OL3850_ForceButtons()

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_OL3850_ForceButtons Lib "OPTILOGICPROTOCOL.DLL"
        (ByVal NODEADDR As Integer, ByVal ButtonFlags As Byte, ByVal ButtonData As Byte)
        As Integer
```

where : ButtonData defines the button which will be affected. Bits are in button sequence starting with bit 0 for the leftmost button.
ButtonFlags defines the type of force as follows -
= 0x80 ; force the buttons to a state matching ButtonData (force state)
= 0x40 ; logically OR current button state with ButtonData (force on)
= 0x20 ; logically AND current button state with the complement of ButtonData (force off)

Send Keypad Data Entry Message
OL_Buffered_OL3850_SendKeypadMessage()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_Buffered_OL3850_SendKeypadMessage Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte, ByVal startNum As Double, ByVal decimalPoint As Byte) As Integer

where : line - the line number (0 = top, 1 = bottom) to display message on
message - 20 character array of ASCII text. Carets "^" should be used as placeholders for the data entry field.
startNum - starting value of keypad entry data
decimalPoint - A beginning value can be passed, which the user can adjust using the arrow key, or start a new data entry with the keypad. The numeric startNum is passed as a double. The decimalPoint is the number of digits after the decimal. In other words, if startNum = 4567 and decimalPoint = 2, the starting value displayed is 45.67.

Send Arrow Adjust Message
OL_Buffered_OL3850_SendArrowMessage()

Prototype : (defined in DLLdefinitions.bas)

Declare Function OL_Buffered_OL3850_SendArrowMessage Lib "OPTILOGICPROTOCOL.DLL" (ByVal NODEADDR As Integer, ByVal line As Byte, ByRef message As Byte, ByVal startNum As Long, ByVal decimalPoint As Byte, ByVal lowLimit As Long, ByVal highLimit As Long) As Integer

where : line - the line number (0 = top, 1 = bottom) to display message on.
message - 20 character array of ASCII text. Carets "^" should be used as placeholders for the data entry field.
startNum - starting value of arrow adjust data
decimalPoint - A beginning value must be passed, which the user can adjust using the arrow key. The numeric startNum is passed as a long. The decimalPoint is the number of digits after the decimal. In other words, if startNum = 4567 and decimalPoint = 2, the starting value displayed is 45.67.
lowLimit - The minimum value allowed for adjustment (the same decimalPoint value applies)
highLimit - The maximum value allowed for adjustment (the same decimalPoint value applies)

For example, if startNum = 4567, decimalPoint = 2, lowLimit = 1000, and highLimit = 9000, the starting value of 45.67 can be adjust by the user to any value between 10.00 and 90.00.

OL3850 continued

Usage

The following example shows various functions using an OL3850 Operator Terminal attached to an OptiLogic RTU with a node address of 30. The program will verify the pushbutton configuration and reconfigure them if necessary. It will read/update the general status of the OL3850 based on the interval of a timer, given the name *tmrOL3850* for this example. Then based on pushbutton status and the conditions of 2 flags, it will display various types of messages and wait for user interaction with the terminal.

The program contains the following five global variables. The first two are used as flags for active message types, the third one is used to lock out the arrow message after button 4 is forced off and the other two are used to store data values from keypad entry and arrow adjust messages. Be sure to initialize the global flags to 0 when the program starts.

```
Private glb_KeypadActive As Byte    ' flag denoting a keypad message is active on the
                                    ' display
Private glb_ArrowActive As Byte     ' flag denoting an arrow adjust message is active on the
                                    ' display
Private glb_ForceActive As Byte     ' flag used to prevent the arrow adjust message from
                                    ' reappearing on the display after button 4 is forced OFF
Private glb_KeypadValue As Double   ' variable that stores the keypad entry result
Private glb_ArrowValue As Long      ' variable that stores the current arrow adjust value
```

This subroutine is used to convert a string passed into it into an array of bytes containing the ASCII equivalent of each character in the string. It passes that array back to the calling routine for transmission to the OL3850.

```
Private Sub SetUpMessage(MessageString As String, MsgBuffer() As Byte)

    ' place blank characters in buffer
    For i = 0 To 19
    MsgBuffer(i) = &H20
    Next i

    ' get ASCII equivalent of each character in message string and place into buffer
    For i = 0 To Len(MessageString) - 1
    MsgBuffer(i) = Asc(Mid$(MessageString, i + 1, 1))
    Next i

End Sub
```

This subroutine is the body of the program. When the timer tmrOL3850 times out, this routine performs several tasks.
(1) First, it reads the configuration of the pushbuttons. If they are not as required by the program, then the program will reconfigure them.
(2) Next, the program reads/updates the general status (lamps, LEDs, button status and current data entry value).
Next, the program takes action depending on certain parameters.
(3) If button 3 is pressed then the top line will display a text message and the bottom line a keypad entry message. Next, a flag is set indicating that a keypad message is active on the panel.
(4) Else if the keypad message is active and if the data available bit (most significant bit of the button status byte) is set, the keypad data is stored into a variable, the top and bottom lines of the LCD are updated with text messages and the keypad message active flag is cleared.
(5) Else if button 4 is active and the force lockout flag inactive, the program can perform several tasks:

(1) If an arrow adjust message is not active, then it sends a text message to the top line and an arrow adjust message to the bottom line.
(2) If an arrow adjust message is active, it continuously copies the arrow adjust data into its storage variable.

(2a) If an arrow adjust message is active and button 2 is pressed, it will clear the arrow adjust message by placing new messages on both lines of the LCD, force button 4 off, clear the arrow adjust message active flag and set the force lockout flag.

(6) Else if button 4 has cleared and the force lockout flag is active, reset the force lockout flag.
(7) The last thing the subroutine does is restart timer tmrOL3850.

```
Private Sub tmrOL3850_Timer()

Dim RequiredConfig As Byte ' required button configuration
Dim config As Byte         ' configuration read from RTU
Dim LEDState As Byte       ' byte holding Lamp and LED on/off states
Dim LEDflash As Byte       ' byte holding Lamp and LED flash states
Dim ButtonStatus As Byte   ' byte containing button status as returned from the RTU
Dim message(20) As Byte    ' array holding message to display on 3850 LCD
Dim DataValue As Double    ' current data value returned from RTU
Dim status As Integer      ' result of a DLL function call
Dim DisplayMessage As String  ' string to display on the 3850's LCD

tmrOL3850.Enabled = False  ' disable timer
LEDState = &H0             ' turn all lights off
LEDflash = &H0             ' turn all flash bits off
RequiredConfig = &H19      ' buttons 1, 4, 5 alternate  -  2, 3 momentary
' read the general configuration of the 3850
status = OL_Buffered_OL3850_ReadConfiguration(30, config)
' if the required config doesn't match the config read from the RTU, configure
If RequiredConfig <> config Then
        status = OL_Buffered_OL3850_ConfigurePanel(30, RequiredConfig)
End If
```

OL3850 continued

```
' read and update the general status of lamps, LEDs, buttons, and the current data entry value
status = OL_Buffered_OL3850_StatusRequest(30, LEDState, LEDflash, ButtonStatus,
        DataValue)
‘ if StatusRequest successful
If status Then
        ' if button 3 is pressed
        If (ButtonStatus And &H4) Then
                ' send text message to top line
                DisplayMessage = "Key Data,Press ENTER"
                Call SetUpMessage(DisplayMessage, message)
                status = OL_Buffered_OL3850_SendTextDisplayMessage(30, 0, message(0))
                ' send keypad message to bottom line, initial data and decimal point 0, carets “^”
                ‘       are placeholders for the data
                DisplayMessage = "Keypad Entry ^^^^^^"
                Call SetUpMessage(DisplayMessage, message)
                status = OL_Buffered_OL3850_SendKeypadMessage(30, 1, message(0), 0, 0)
                If status Then
                        glb_KeypadActive = 1  ' set keypad message active flag
                End If

        ' if keypad message active and the data available bit set, read the data
        ElseIf ((glb_KeypadActive = 1) And (ButtonStatus And &H80)) Then
                glb_KeypadValue = DataValue     ' store entered value
                ' blank top line
                DisplayMessage = ""
                Call SetUpMessage(DisplayMessage, message)
                status = OL_Buffered_OL3850_SendTextDisplayMessage(30, 0, message(0))
                ' send message to bottom line
                DisplayMessage = "Keypad Entry Done"
                Call SetUpMessage(DisplayMessage, message)
                status = OL_Buffered_OL3850_SendTextDisplayMessage(30, 1, message(0))
                If status Then
                        glb_KeypadActive = 0            ' reset keypad message active flag
                End If
        ' if button 4 active and the force lockout flag inactive
        ElseIf ((ButtonStatus And &H8) And (glb_ForceActive = 0)) Then
                ' if arrow adjust message not active send it
                If (glb_ArrowActive = 0) Then
                        ' send message to top line
                        DisplayMessage = "Press F2 When Done"
                        Call SetUpMessage(DisplayMessage, message)
                        status = OL_Buffered_OL3850_SendTextDisplayMessage(30, 0,
                                message(0))
                        ' send arrow adjust message to bottom line, initial value is 3686.75,
                        ‘       adjustment range is 3000.00 - 4000.00
                        DisplayMessage = "Adjust Value ^^^^^^"
                        Call SetUpMessage(DisplayMessage, message)
                        status = OL_Buffered_OL3850_SendArrowMessage(30, 1, message(0),
                                368675, 2, 300000, 400000)
```

OL3850 continued

```
                If status Then
                        glb_ArrowActive = 1  'set arrow adjust message active
                End If

                ' if arrow adjust active
                ElseIf (glb_ArrowActive) Then
                        glb_ArrowValue = DataValue  'update arrow data as its adjusted
                        ' if button 2 is pressed, reset arrow message
                        If (ButtonStatus And &H2) Then
                                ' blank top line
                                DisplayMessage = ""
                                Call SetUpMessage(DisplayMessage, message)
                                status =  OL_Buffered_OL3850_SendTextDisplayMessage(30, 0,
                                        message(0))
                                ' send message to bottom line
                                DisplayMessage = "Arrow Adjust Done"
                                Call SetUpMessage(DisplayMessage, message)
                                status = OL_Buffered_OL3850_SendTextDisplayMessage(30, 1,
                                        message(0))
                                ' force button 4 off
                                status = OL_Buffered_OL3850_ForceButtons(30, &H20, &H8)
                                If status Then
                                        glb_ArrowActive = 0   ' reset arrow adjust message active
                                        glb_ForceActive = 1   ' set force lockout to 1
                                End If
                        End If
                End If
        ' if button 4 status has cleared and force lockout flag active, reset flag
        ElseIf (((ButtonStatus And &H8) = 0) And (glb_ForceActive = 1)) Then
                glb_ForceActive = 0          ' reset force lockout
        End If
End If

' update all nodes
status = OL_Buffered_UpdateNodes(glb_NodeQueryList(0), glb_NodeQuerySize)

tmrOL3850.Enabled = True             ' restart timer

End Sub
```

Housekeeping Group (Immediate Mode)

The Housekeeping Group, as the name implies, consists of functions that perform network housekeeping. Two functions, OL_NetworkRetryCount() and OL_NetworkTimeout(), set up how the network operates when talking to an RTU. OL_NetworkTimeout() sets the amount of time allowed before the host considers that the RTU has not responded. The OL_NetworkRetryCount() sets the number of times the host will reissue a message and wait for a response before it gives up and sets an error. OL_IsNetworkInitialized() can be called to check to see if the network port has been initialized. OL_GetNodeRetryCount() can be called to get the total number of message retries for a particular node and OL_ResetNodeRetryCount() can be called to set the retry count for a node to 0.

The rest of the Housekeeping functions have to do with detecting & retrieving errors.

The Immediate Mode housekeeping functions all perform direct action and immediately return a result. They do not perform actual communications with the RTUs. Buffered Mode housekeeping functions all have to do with checking errors. They also return results without communicating with the RTUs.

Set the Number of Retries
OL_NetworkRetryCount()

This function sets the number of times that the DLL will retry a message before it considers a response to have failed. It is used in conjunction with the timeout. Typically, a message will be sent. The system will wait for a response within the timeout period. If there is no response, it will retry. The number of times this will repeat before giving up is set via this function.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Sub OL_NetworkRetryCount Lib "OPTILOGICPROTOCOL.DLL"(ByVal nRetryCount As
        Integer)
```

where : nRetryCount is the number of times to retry

Usage:

The following will set the retry count for a message to 5.

```
Call OL_NetworkRetryCount(5)                ' sets the retry count for a message to 5
```

Set the Network Timeout
OL_NetworkTimeout()

This function sets the amount of time (in milliseconds) to allow for a response from an RTU. Works in conjunction with OL_NetworkRetryCount (see above).

Prototype : (defined in DLLdefinitions.bas)

```
Declare Sub OL_NetworkTimeout Lib "OPTILOGICPROTOCOL.DLL" (ByVal Timeout As Long)
```

where : Timeout is the amount of time allowed, in milliseconds, for a response.

Usage:

The following will set the message timeout value to 100 milliseconds.

```
Call OL_NetworkTimeout(100)          ' sets the timeout value for a message to 100ms.
```

Check if to see the Network is Initialized
OL_IsNetworkInitialized()

This is a simple function that can be called to check if the network is initialized.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_IsNetworkInitialized Lib "OPTILOGICPROTOCOL.DLL" () as Integer
```

The returned flag will be a "1" if the network is initialized or a "0" if it is not.

Usage:

The following will check to see if the network port has been initialized on the host PC.

```
status = OL_IsNetworkInitialized()          ' Is the network initialized?
If status = 0 Then
        'initialize network here
End If
```

Get the Total Retry Count for a Node OL_GetNodeRetryCount()

This function can be called to determine the total number of message retries for a node (since the last time it was reset). It can be useful in determining communication problems within a network.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_GetNodeRetryCount Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer) As Long
```

The return value will contain the total number of retries for the node since the last count reset was performed.

Usage:

The following line will get the total retry count for an RTU with a node address of 30.

```
TotalRetryCount = OL_GetNodeRetryCount(30)
```

Reset the Total Retry Count for a Node OL_ResetNodeRetryCount()

This function can be called to set the total number of message retries for a node to 0.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_ResetNodeRetryCount Lib "OPTILOGICPROTOCOL.DLL" (ByVal
    NODEADDR As Integer) As Integer
```

This function will reset to total retry count for a node.

Usage:

The following line will reset the total retry count for an RTU with a node address of 30.

```
status = OL_ResetNodeRetryCount(30)
```

Get the Last Network Error Code
OL_GetLastErrorCode()

This function gets the error code of the last, and only the last error on the network.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_GetLastErrorCode Lib "OPTILOGICPROTOCOL.DLL" () As Long
```

Usage:

The following call will get the error code of the last error on the network.

```
Dim ErrorCode As Long

ErrorCode = OL_GetLastErrorCode()
```

See Appendix C for a List of Error codes.

Get the Last Network Error Code String
OL_GetLastErrorCodeString()

This function gets the error code *string* of the last, and only the last error on the network. This can be useful by displaying it so the operator can view it.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_GetLastErrorCodeString Lib "OPTILOGICPROTOCOL.DLL" () As String
```

Usage:

The following call will get the error code string of the last error on the network.

```
Dim ErrorString As String

ErrorString = OL_GetLastErrorCodeString()
```

See Appendix C for a List of Error strings.

Housekeeping Group (Buffered Mode)

The Buffered Mode Housekeeping Group consists of 2 function calls to help with error checking.

Get the Last Network Error Code OL_Buffered_GetErrorCode()

This function gets the error code of the specified slot in a node. If there is no error for that slot, the function will return a 0. The error code also is reset after it has been read.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_GetErrorCode Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer) As Long
```

Usage:

The following call will get the error code of the card plugged into slot 2 of node 30.

```
Dim ErrorCode As Long
Dim status As Integer
Dim i As Integer, NodeAddress(1) As Integer

NodeAddress(0) = 30
' update node 30
status = OL_Buffered_UpdateNodes(NodeAddress(0), 1)
If status <> 1 Then     'check the error code
        For i = 0 to 3
                ErrorCode = OL_Buffered_GetErrorCode(30, i)
                ' if ErrorCode is not = to 0 then there is an error in this slot
                If ErrorCode <> 0 Then
                        ' handle error condition
                End If
        Next i
End If
```

See Appendix C for a List of Error codes.

Get the Last Network Error Code String OL_Buffered_GetErrorCodeString()

This function gets the error code *string* of the specified slot in a node. It can be useful for display on the screen to notify the operator.

Prototype : (defined in DLLdefinitions.bas)

```
Declare Function OL_Buffered_GetErrorCodeString Lib "OPTILOGICPROTOCOL.DLL" (ByVal
        NODEADDR As Integer, ByVal SlotNo As Integer) As String
```

Usage:

The following call will get the error code string of the card plugged into slot 2 of node 30.

```
Dim ErrorCode As Long
Dim ErrorString As String
Dim status As Integer
Dim i As Integer, NodeAddress(1) As Integer

NodeAddress(0) = 30
' update the node
status = OL_Buffered_UpdateNodes(NodeAddress(0), 1)
If status <> 1 Then     'check the error code
        For i = 0 to 3
                ErrorCode = OL_Buffered_GetErrorCode(30, I)
                ' if ErrorCode is not = to 0 then there is an error in this slot
                If ErrorCode <> 0 Then
                        ' handle error condition
                        ErrorString = OL_Buffered_GetErrorCodeString(30, i)
                        ' place error code string in the text box txtTextBox1
                        txtTextBox1.Text = ErrorString
                End If
        Next i
End If
```

See Appendix C for a List of Error strings.

Appendix A
Definition of Different Base Types

Base Type	Definition
OL4054 4-slot RTU Base	1
OL4058 8-slot RTU Base	2

Appendix B
Definition of Module Types and Sub-Types

Module	Type Definition	Sub-Type Definition
OL2104	8	1
OL2108	9	1
OL2109	9	2
OL2111	9	3
OL2201	1	3
OL2205	5	1
OL2208	1	1
OL2211	1	2
OL2252	80	0
OL2258	82	1
OL2304	25	1
OL2408	18	1
OL2418	18	2
OL2602	112	1
OL3406	128	-
OL3420	136	-
OL3440	137	-
OL3850	185	-

Appendix C
Error Codes and Error Strings

Error Code	Equivalent Error String
0 (sent from RTU)	"No Error"
1 (sent from RTU)	"Module message was the improper length."
2 (sent from RTU)	"Improper module command."
3 (sent from RTU)	"Module not present."
40960	"Unable to MALLOC memory for object."
40961	"General socket error."
40962	"Error sending data to output socket."
40963	"Error receiving data from input socket."
40964	"Too many messages in packet for a single node...max is 100"
40965	"Error socket receive timeout occurred."
40966	"Maximum number of IP Search Addresses Exceeded!"
40967	"IP Address is too long (max 15 chars)"
40968	"IP Address not FOUND!"
40969	"Error sending data to output socket."
40970	"Error receiving data from input socket."
40971	"OL_NetworkInit - Create Mutex"
40972	"OL_NetworkInit - Release Mutex"
40973	"Error calling WSAStartup"
40974	"OL_NetworkInit - Invalid protocol type given"
40975	"The protocol family you selected is not installed on your machine."

Appendix C continued

Error Code	Equivalent Error String
40976	"Error creating input socket."
40977	"Error setting input socket to broadcast."
40978	"Error binding IPX input socket."
40979	"Error binding UDP input socket."
41216	"Digital Input type does not exist."
41217	"Digital Output type does not exist."
41218	"Analog Input type does not exist."
41219	"Analog Output type does not exist."
41472	"Requested node not found in DLL memory."
41473	"Requested module not found in DLL memory."
41474	"Global memory location could not be changed."
41475	"Node has not responded to last message sent."