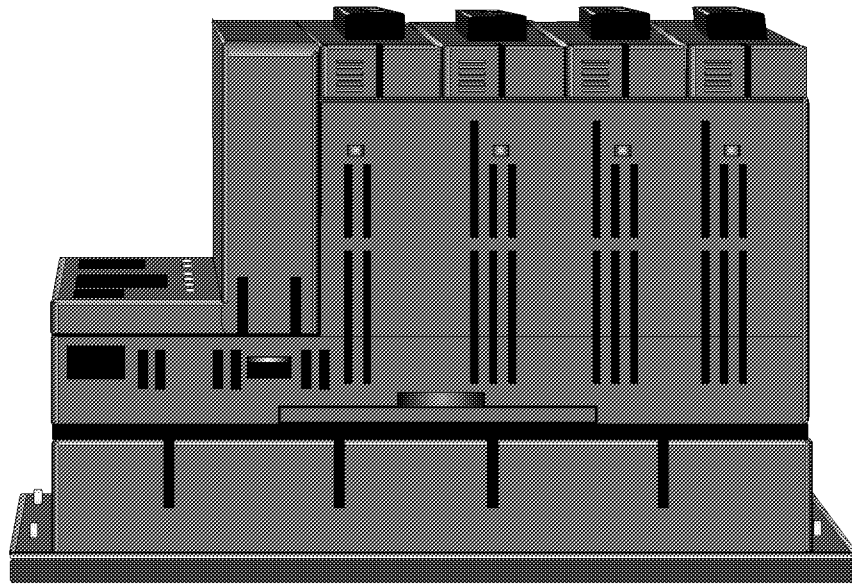


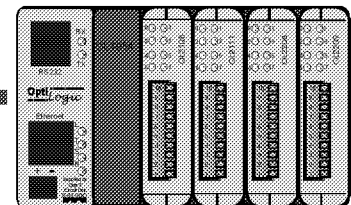
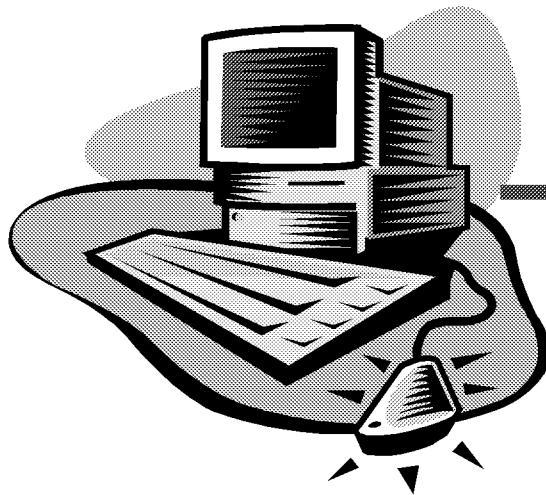
OptiLogicSeries



OptiLogic

Software Interface Definition

(for use with C or C++)



Optimization
Optimal Automation for Industry

Optimization, Inc.

(256)883-3050
www.optimize.com

WARNING

Thank you for purchasing industrial control products from Optimization, 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 Optimization products, please call us at (256)883-3050.

All Optimization 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. Optimization's sole obligation under this warranty shall be limited to either, at Optimization's option, repairing or replacing defective product. The cost of freight to and from Optimization 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. Optimization disclaims any proprietary interest in the marks and names of others.

Copyright 1999, Optimization, Inc.
All rights reserved

No part of this document shall be copied, reproduced or transmitted in any way without the prior, written consent of Optimization, Inc. Optimization retains the exclusive rights to all information included in this document.

Table of Contents

Introduction	1
General Overview	2
OptiLogic System Builder CD	3
Setup Utility	3
OptiLogic QuickStart	3
Functional Overview	7
Immediate Mode	7
Buffered Mode	8
Samples	9
The Session Group – Opening and Closing the Link	10
Initializing the Network	
OL_NetWorkInit()	10
Closing the Network	
OL_NetworkClose()	10
Node Group - Who's There?	11
Add an IP Search Address to the Query List	
OL_NetworkAddIPSearchAddress()	11
Remove an IP Search Address from the Query List	
OL_NetworkRemoveIPSearchAddress()	12
Query a Particular Node	
OL_QueryNode()	13
Query the Nodes in Sequence	
OL_GetFirst() and OL_GetNext()	14
Immediate Mode I/O Group	16
Read Digital Inputs	
OL_ReadDigitalInput()	16
Read Latched Digital Inputs	
OL_ReadLatchedDigitalInput()	17
Write Digital Outputs	
OL_WriteDigitalOutput()	18
Set Up Digital Output Fail Safe	
OL_SetUpDigitalOutputFailSafe()	19
Read Digital Outputs	
OL_ReadDigitalOutput()	21
OL2252 Dual High Speed Pulse Counter	22
Configure Counter	
OL_Configure_Counter()	22
Send Counter Controls and Read Counts	
OL_Read_Counter()	23
OL2258 High Speed Pulse Counter	25

Configure High Speed Counter	
OL_Configure_HS_Counter()	26
Send Output Range	
OL_HS_SendOutput_Range_Counter()	26
Read High Speed Counter	
OL_Read_HS_Counter()	27
OL2304 Analog Output Module	30
Configure Analog Outputs	
OL_ConfigAnalogOutput()	30
Write Analog Outputs	
OL_WriteAnalogOutput()	31
Conversion of Analog Output Voltage to Equivalent Output Value	31
Read Analog Inputs	
OL_ReadAnalogInput()	34
OL2602 Dual RS232 and Base Serial Comm Port	35
Configure a Communication Port	
OL_ConfigureSerialPort()	35
Read received Data	
OL_ReadSerialData()	36
Write to Serial Port	
OL_WriteSerialData()	36
Immediate Mode Operator Panel Group	40
OL3406 Pushbutton/Indicator Panel	40
Read the Pushbutton Status and Control the LEDs	
OL_OL3406_StatusControl()	40
Force Alternate-Action Button Status	
OL_OL3406_ForceButtons()	41
Configure OL3406 Panel	
OL_OL3406_ConfigurePanel()	41
Read OL3406 Configuration	
OL_OL3406_ReadPanelConfiguration()	41
OL3420 Operator Terminal	43
Read the Pushbutton Status and Control Momentary Button LEDs	
OL_OL3420_StatusRequest()	43
Force Alternate-Action Button Status	
OL_OL3420_ForceButtons()	44
Send Text to OL3420 Display	
OL_OL3420_SendTextDisplayMessage()	44
Configure OL3420 Terminal	
OL_OL3420_ConfigurePanel()	45
Read OL3420 Configuration	
OL_OL3420_ReadConfiguration()	45
OL3440 Display Panel	47
Send Text to OL3440 Display	
OL_OL3440_SendTextDisplayMessage()	47
OL3850 Operator Terminal	48

Send Light Controls and Read Status information	
OL_OL3850_StatusRequest()	49
Send a Text Message to the Display	
OL_OL3850_SendTextDisplayMessage()	49
Configure OL3850 Function Buttons	
OL_OL3850_ConfigurePanel()	50
Read OL3850 Configuration	
OL_OL3850_ReadConfiguration()	50
Force Alternate-Action Button Status	
OL_OL3850_ForceButtons()	50
Send Keypad Data Entry Message	
OL_OL3850_SendKeypadMessage()	51
Send Arrow Adjust Message	
OL_OL3850_SendArrowMessage()	51
Buffer Mode I/O Group	56
Buffered Mode Housekeeping	56
Update Nodes	
OL_Buffered_UpdateNodes()	56
Read Digital Inputs	
OL_Buffered_ReadDigitalInput()	58
Read Latched Digital Inputs	
OL_Buffered_ReadLatchedDigitalInput()	59
Write Digital Outputs	
OL_Buffered_WriteDigitalOutput()	60
Read Digital Outputs	
OL_Buffered_ReadDigitalOutput()	61
OL2252 Dual High Speed Pulse Counter	62
Configure Counter	
OL_Buffered_Configure_Counter()	62
Send Counter Controls and Read Counts	
OL_Buffered_Read_Counter()	63
OL2258 High Speed Pulse Counter	65
Configure High Speed Counter	
OL_Buffered_Configure_HS_Counter()	66
Send Output Range	
OL_Buffered_HS_SendOutput_Range_Counter()	66
Read High Speed Counter	
OL_Buffered_Read_HS_Counter()	67
OL2304 Analog Output Module	70
Configure Analog Outputs	
OL_Buffered_ConfigAnalogOutput()	70
Write Analog Outputs	
OL_Buffered_WriteAnalogOutput()	71
Conversion of Analog Output Voltage to Equivalent Output Value	71
Read Analog Inputs	
OL_Buffered_ReadAnalogInput()	74
OL2602 Dual RS232 and Base Serial Comm Port	75

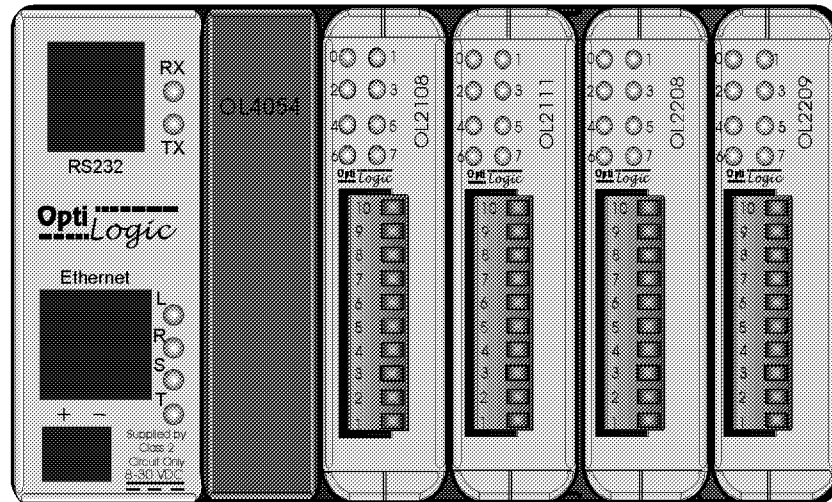
Configure Serial Port	
OL_Buffered_ConfigureSerialPort()	75
Read Serial Data	
OL_Buffered_ReadSerialData()	76
Write Serial Data	
OL_Buffered_WriteSerialData()	76
Buffer Mode Operator Panel Group	80
OL3406 Pushbutton/Indicator Panel	80
Read the Pushbutton Status and Control the LEDs	
OL_Buffered_OL3406_StatusControl()	80
Force Alternate-Action Button Status	
OL_Buffered_OL3406_ForceButtons()	81
Configure OL3406 Panel	
OL_Buffered_OL3406_ConfigurePanel()	81
Read OL3406 Configuration	
OL_Buffered_OL3406_ReadPanelConfiguration()	81
OL3420 Operator Terminal	83
Read the Pushbutton Status and Control Momentary Button LEDs	
OL_Buffered_OL3420_StatusRequest()	83
Force Alternate-Action Button Status	
OL_Buffered_OL3420_ForceButtons()	84
Send Text to OL3420 Display	
OL_Buffered_OL3420_SendTextDisplayMessage()	84
Configure OL3420 Terminal	
OL_Buffered_OL3420_ConfigurePanel()	85
Read OL3420 Configuration	
OL_Buffered_OL3420_ReadConfiguration()	85
OL3440 Display Panel	87
Send Text to OL3440 Display	
OL_Buffered_OL3440_SendTextDisplayMessage()	87
OL3850 Operator Terminal	88
Send Light Controls and Read Status information	
OL_Buffered_OL3850_StatusRequest()	89
Send a Text Message to the Display	
OL_Buffered_OL3850_SendTextDisplayMessage()	89
Configure OL3850 Function Buttons	
OL_Buffered_OL3850_ConfigurePanel()	90
Read OL3850 Configuration	
OL_Buffered_OL3850_ReadConfiguration()	90
Force Alternate-Action Button Status	
OL_Buffered_OL3850_ForceButtons()	90
Send Keypad Data Entry Message	
OL_Buffered_OL3850_SendKeypadMessage()	91
Send Arrow Adjust Message	
OL_Buffered_OL3850_SendArrowMessage()	91
Housekeeping Group (Immediate Mode)	95
Set the Number of Retries	
OL_NetworkRetryCount()	95

Set the Network Timeout OL_NetworkTimeout()	96
Check to see if the Network is Initialized OL_IsNetworkInitialized()	96
Get the Total Retry Count for a Node OL_GetNodeRetryCount()	97
Reset the Total Retry Count for a Node OL_ResetNodeRetryCount()	97
Get the Last Network Error Code OL_GetLastErrorCode()	98
Get the Last Network Error Code String OL_GetLastErrorCodeString()	98
Housekeeping Group (Buffered Mode)	99
Get the Last Network Error Code OL_Buffered_GetErrorCode()	99
Get the Last Network Error Code String OL_Buffered_GetErrorCodeString()	100
Appendix A Definition of Different Base Types	101
Appendix B Definition of Module Types and Sub -Types	102
Appendix C Error Codes and Error Strings	103

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

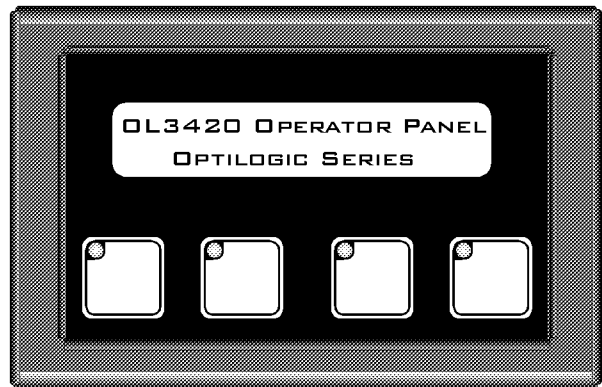


Introduction

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.

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.



One of many available OptiLogic operator panels

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

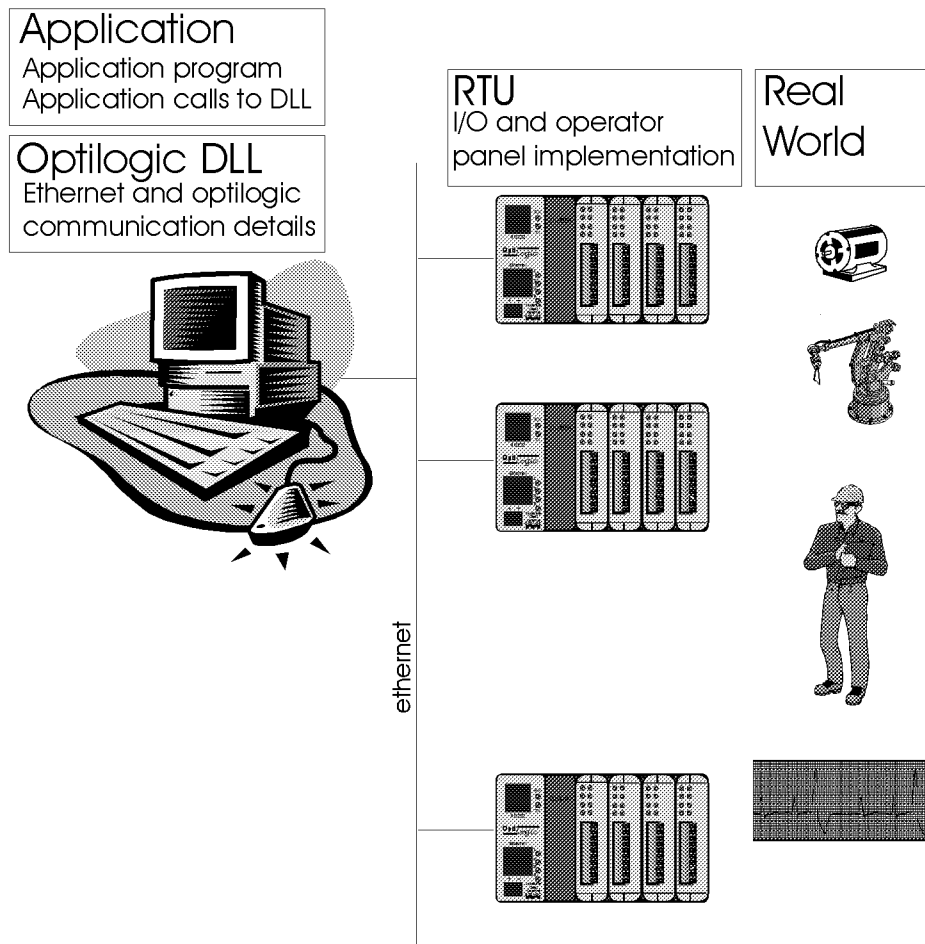
Before we discuss details of the operation, we'll look at the basic system configuration and architecture. The figure below illustrates a typical system.

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.



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.

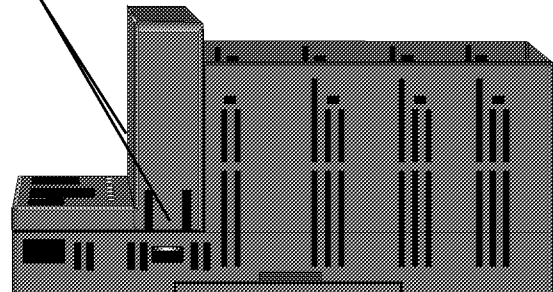
- Installs Visual Basic support files in the Windows directory
- Installs the "OptiLogicProtocol.DLL" in the Windows System directory
- Creates a directory "c:\Program Files\OptiLogic QuickStart" - which will be used for the test utility. (The user has the option of selecting a different directory name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

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

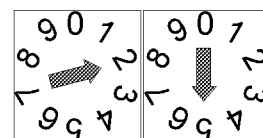
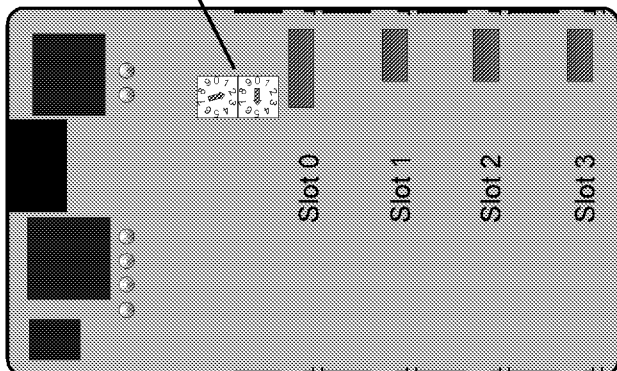
The first thing you should do, before running the OptiLogic QuickStart, is to set the RTU address. To do so, remove the snap-on end cover from the card cage by pressing the top and bottom latches in and lifting the cover off. This will expose two rotary

Squeeze to lift off end cover



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).

Addressing switches



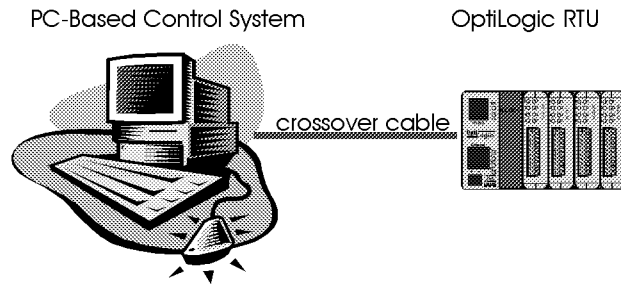
Exploded view

OptiLogic Series

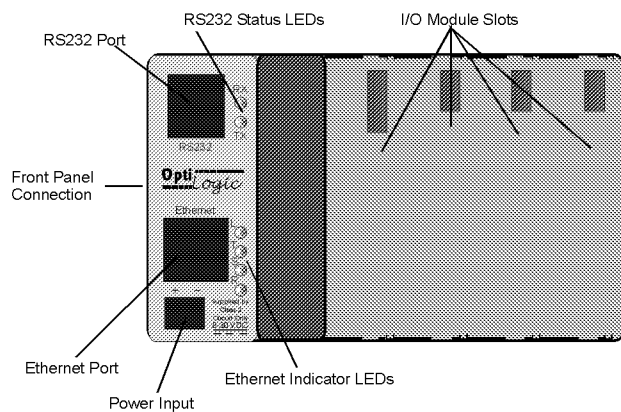
Next, you should have the OptiLogic RTU connected either directly to your PC's ethernet board (through a crossover ethernet cable) or through a hub to the ethernet board (through standard, uncrossed ethernet cables). Power (12VDC or 24VDC) should be applied to the OptiLogic RTU power connector.

Now you're ready to run the OL_QuickStart software.

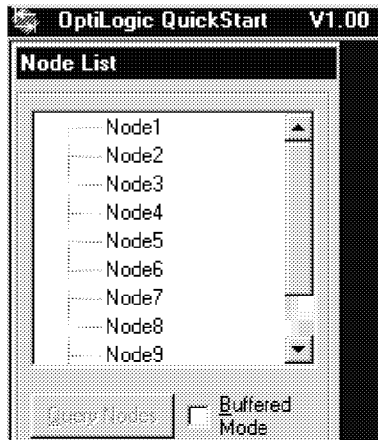
The test program can be selected through program selection via the Windows START button.



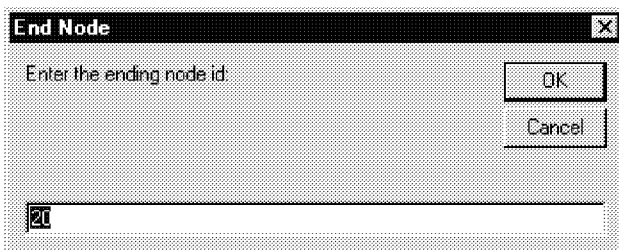
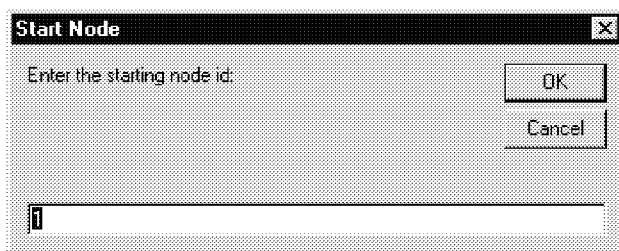
After a two-second start up check, two LEDs will indicate operational status. The "S", or Select LED next to the ethernet connector, should be on - indicating that the RTU is powered on, the firmware is executing, and the processor is talking to the ethernet circuit. The "L", or Link LED, should be on - 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 Optimization 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.



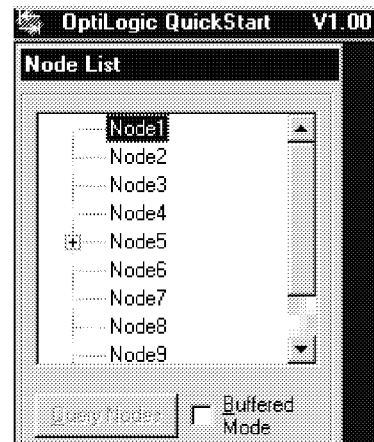
When OptiLogic QuickStart begins, the screen will display a window like the one shown below.



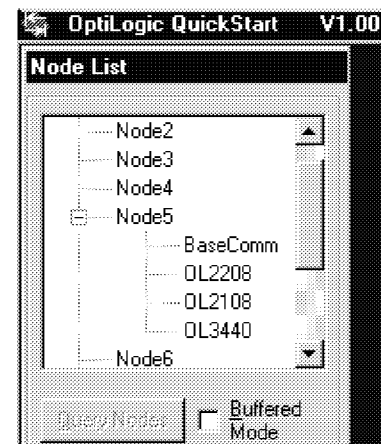
Select "Query Nodes". This will start the process of finding out what is attached. The following two screens will allow you to define the range of addresses you want to query. You can select any address range between 1 and 99.



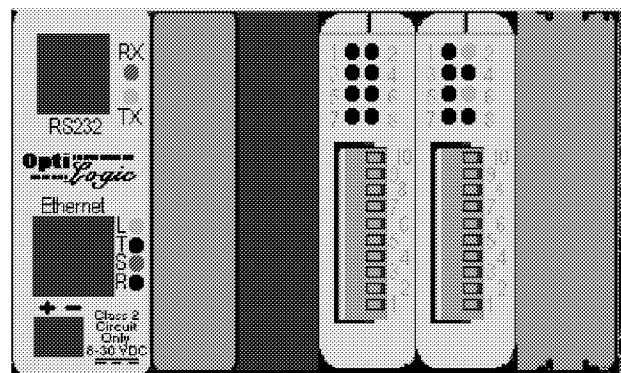
While the query is being performed, you should see the RTU's Recieve light come on for every RTU address being polled. The RTU's Transmit light will flash when it receives it's own address. At the end of the poll a screen like the one shown below will be displayed.



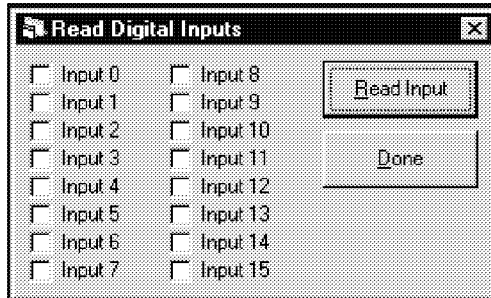
Notice the '+' sign by address 5. A plus sign will display next to every address in the selected range that has responded and has I/O modules or operator panels. When you click on the name "Node 5", the tree will expand to show you which I/O boards and operator panels are present.



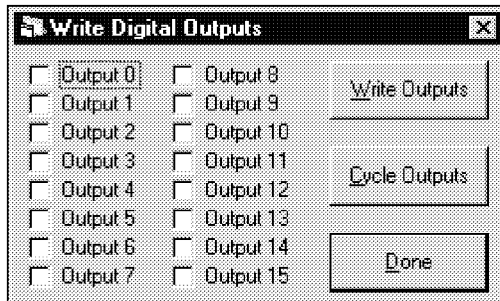
A separate display, similar to the one shown below will show you a pictorial view of the node's card cage.



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 Inputs” button, the status indication for the inputs will be updated. When you are finished, click the “Done” button.

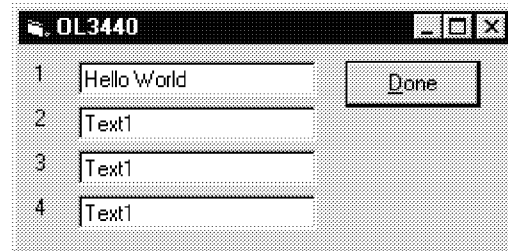


Click to select a digital output board. A display like the one shown below will come up.



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 QuickStart software and get a feel for how this system operates - and how fast!

Functional Overview

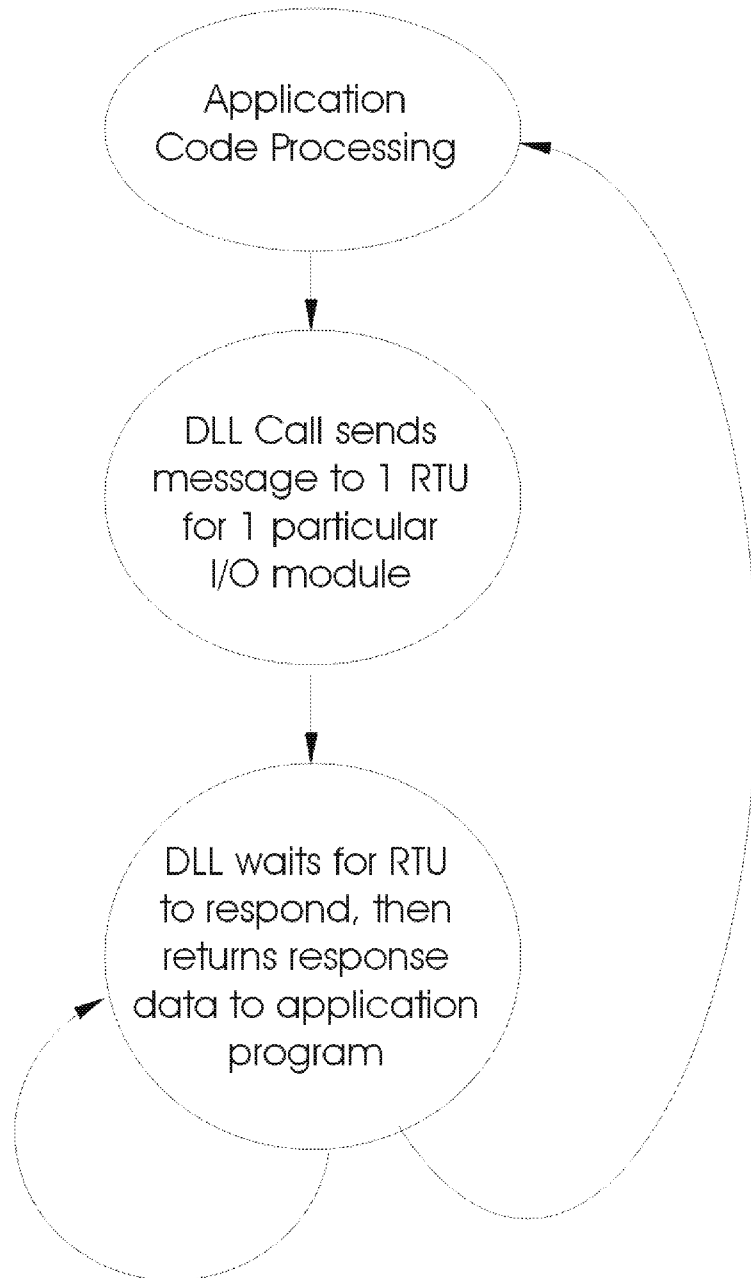
The OptiLogic DLL is a tool that you can use to interface your program to one or more OptiLogic RTUs. The DLL allows you to choose one of two basic modes of operation - immediate or buffered. The particular mode that you use depends on the application. It is even possible to use both modes - **but be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

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



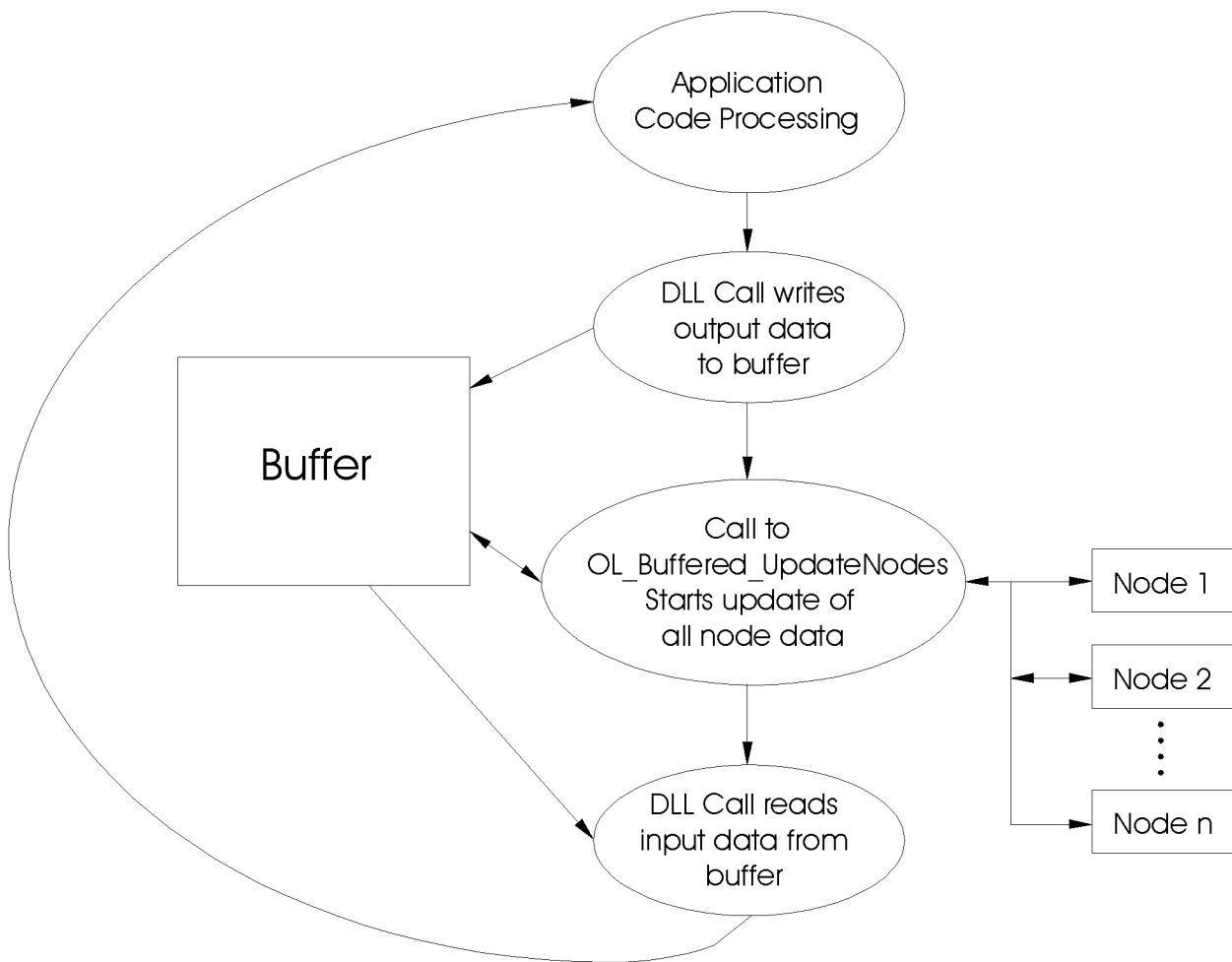
Immediate Mode Operation

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 optilogic.h)

```
BOOL OL_NetworkInit(NETWORKPORT port, BYTE protocol, int OL_TIMEOUT,
                    intMaxRetries)
```

where:	port =	network port address (defined in optilogic.h)
	protocol =	communication protocol (defined in optilogic.h)
	OL_Timeout =	retry timeout period in milliseconds
	MaxRetries =	maximum number of times to retry a message before returning an error

Usage :

```
OL_NetworkInit(OPTISOCKET, OL_IPX, 100, 3);
```

Note: Optisocket in OL_IPX (OL_UDP) are defined in optilogic.h
Use OL_IPX to select IPX communications or OL_UDP to select UDP/IP communications.
100 denotes a retry timeout value of 100ms.
3 denotes 3 retries before giving up on that message.

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 optilogic.h)

```
void OL_NetworkClose(void)
```

Usage :

```
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 with 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 optilogic.h)

```
BOOL __stdcall OL_NetworkAddIPSearchAddress(char *strIPAddress);
```

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

strIPAddress is in the form of a character array of length 16.

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

Usage :

```
#include <string.h> /* this file should be included in the program so the strcpy() function can  
be used */
```

```
char IPAddress[16]; /* pointer to the IP Address string */  
int status;         /* the value returned from the function call */
```

```
/* copy IP Address into character array*/  
strcpy(IPAddress, "215.150.127.255");  
status = OL_NetworkAddIPSearchAddress(&IPAddress);  
if (status == 0)  
{  
    /* The specified address could not be added to the search list.  
    Perform error checking to determine the cause of the error. */  
}
```

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 not be in the proper format or the specified IP address might not be in the search list.

Prototype : (defined in optilogic.h)

```
BOOL __stdcall OL_NetworkRemoveIPSearchAddress(char *strIPAddress);
```

where : strIPAddress - the Class C *directed broadcast* IP Address for the network
(e.g. "208.170.106.255").
strIPAddress is in the form of a character array of length 16.

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

Usage :

```
#include <string.h> /* this file should be included in the program so the strcpy() function can  
be used */
```

```
char IPAddress[16]; /* pointer to the IP Address string */  
int status;         /* the value returned from the function call */
```

```
/* copy IP Address into character array*/  
strcpy(IPAddress, "215.150.127.255");  
status = OL_NetworkRemoveIPSearchAddress(&IPAddress);  
if (status == 0)  
{  
    /* The specified address could not be added to the search list.  
    Perform error checking to determine the cause of the error.    */  
}
```


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 optilogic.h)

```
UINT OL_QueryNode(NODEADDR nodeaddress, BYTE *basetype, BYTE *moduletypes,
                  BYTE *subtypes, BYTE *modversions, BYTE *minversion, BYTE *majversion)
```

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 - pointer to 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 - pointer to 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].
- modversions - pointer to 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 of base. For software version 3.7, the major version is 3.

The returned unsigned integer (the UINT) 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 :

```
unsigned int  slots ;
BYTE         mtype[9], stype[9], mver[9], minver, majver, basetype ;
NODEADDR     nodeaddr ;
```

```
nodeaddr = 1 ;
slots = OL_QueryNode(nodeaddr, &basetype, &mtype, &stype, &mver, &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 0, 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 optilogic.h)

```
UINT OL_GetFirst(NODEADDR *nodeaddress, BYTE *basetype, BYTE moduletypes[],
                BYTE subtypes[], BYTE modversions[], BYTE *minversion,
                BYTE *majversion)
```

```
UINT OL_GetNext(NODEADDR *nodeaddress, BYTE *basetype, BYTE moduletypes[],
                BYTE subtypes[], BYTE modversions[], BYTE *minversion,
                BYTE *majversion)
```

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_GetLast() you pass a pointer to indicate the location of the address of the RTU.

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 all node address 100 is reached (return value of 0).

```
#define MAX_RTUS 25
```

```
typedef struct
```

```
{
    NODEADDR  address ;
    BYTE      basetype ;
    BYTE      majorver ;
    BYTE      minorver ;
    BYTE      modtype[9] ;
    BYTE      subtype[9] ;
    BYTE      modver[9] ;
} RTU_TYPE ;
```

```
RTU_TYPE  rtu[MAX_RTUS] ;
```

```
unsigned int  slots ;
```

```
byte          mtype[9], stype[9], mver[9], minver, majver, basetype ;
```

```
NODEADDR  nodeaddr ;
```

(continued on next page)

```
slots = OL_GetFirst(&nodeaddr, &basetype, mtype, stype, mver, &minver, &majver) ;
if (slots > 0)
{
    /* Store data for first RTU */
    rtu[0].address = nodeaddr ;
    rtu[0].basetype = basetype ;
    rtu[0].majorver = majver ;
    rtu[0].minorver = minver ;
    for (j = 0 ; j < 9 ; j++)
    {
        rtu[0].modtype[j] = mtype[j] ;
        rtu[0].subtype[j] = stype[j] ;
        rtu[0].modver[j] = mver[j] ;
    }

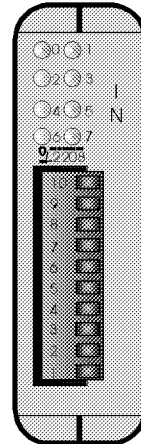
    /* Find more RTUs */
    for (i = 1 ; i < MAX_RTUS ; i++)
    {
        slots = OL_GetNext(&nodeaddr,&basetype,mtype,stype,mver,&minver,&majver) ;
        if (slots > 0)
        {
            /* Store data for RTU */
            rtu[i].address = nodeaddr ;
            rtu[i].basetype = basetype ;
            rtu[i].majorver = majver ;
            rtu[i].minorver = minver ;
            for (j = 0 ; j < 9 ; j++)
            {
                rtu[i].modtype[j] = mtype[j] ;
                rtu[i].subtype[j] = stype[j] ;
                rtu[i].modver[j] = mver[j] ;
            }
        }
        else
        {
            break ;
        }
    }
}
```

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 a boolean indicator. A TRUE returned indicates the function was successful. If a FALSE is returned, the error code can be retrieved by calling `OL_GetLastError()` - 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 `optilogic.h`)

```
BOOL OL_ReadDigitalInput(NODEADDR Addr, USHORT modno, BYTE type,
                        ULONG *inputdata)
```

where:	Addr	-	node address
	modno	-	RTU slot
	type	-	input type number, 4 digital in, 8 digital in, etc.
	inputdata-		input status returned from the RTU

Usage :

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

```
USHORT    modno;
ULONG     digins;
BYTE      modtype;
NODEADDR  nodeaddr;

nodeaddr = 23;
modno = 2;
modtype = 1;          /* 8 channel digital inputs are type 1 */

if (OL_ReadDigitalInput(nodeaddr, modno, modtype, &digins))
{
    if (digins & 0x08)
    {
        /* application processing */
    }
}
```

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 optilogic.h)

```
BOOL __stdcall OL_ReadLatchedDigitalInput(NODEADDR Addr, USHORT modno, BYTE type,
ULONG *inputdata);
```

where: Addr - node address
modno - input slot
type - input type number, 4 digital in, 8 digital in, etc.
inputdata - input status returned from the RTU status of inputs
(input 0 --> bit 0, input 1 --> bit 1 ...)

Usage :

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

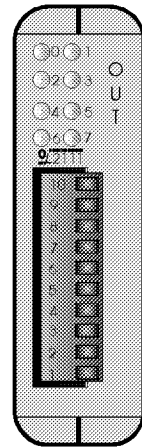
```
USHORT slot;
ULONG digins;
BYTE modtype;
NODEADDR nodeaddr;

nodeaddr = 23;
slot = 2;
modtype = 1; /* 8 channel digital inputs are type 1 */

if (OL_ReadLatchedDigitalInput(nodeaddr, slot, modtype, &digins))
{
    if (digins & 0x08)
    {
        /* application processing */
    }
}
```

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 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”, ie. 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 optilogic.h)

```
BOOL OL_WriteDigitalOutput(NODEADDR addr, USHORT modno, BYTE type,
                           ULONG data)
```

where:	addr	-	node address
	modno	-	RTU slot
	type	-	output type number, 4 digital out, 8 digital out, etc.
	data	-	output state

Usage :

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

```
USHORT    modno;
ULONG     digouts;
BYTE      modtype;
NODEADDR  nodeaddr;

nodeaddr = 23;
modno = 3;
modtype = 9;          /* 8 channel digital outputs are type 9 */

digouts = 0x68;

/* Just send outputs, ignore the boolean response */
OL_WriteDigitalOutput(nodeaddr, modno, modtype, digouts);

/* Send outputs, and use the boolean response */
if (!OL_WriteDigitalOutput(nodeaddr, modno, modtype, digouts))
{
    /* Error handling code */
}
```

Set Up Digital Output Fail Safe 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 optilogic.h)

BOOL OL_SetUpDigitalOutputFailSafe (NODEADDR addr, USHORT modno,
BYTE byModuleType, BYTE fsType, DWORD fsPattern, BYTE fsTime)

where:	byModuleType	=	0x08(4 digital outputs) 0x09(8 digital outputs) 0x0A (16 digital outputs) 0x0B (24 digital outputs) 0x0C (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 last 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 at Node 30 when a communication failure occurs.

```
NODEADDR address    ; /* address of RTU */
USHORT  slot        ; /* slot of output card */
BYTE    ModuleType  ; /* output card type */
BYTE    fsType       ; /* Fail Safe Type */
DWORD   fsPattern    ; /* Fail Safe Pattern */
BYTE    fsTime       ; /* Fail Safe Timeout Value (in tenths of a second) */

address    = 30    ; /* node 30 */
slot       = 3     ; /* card resides in slot 3 */
ModuleType = 0x09 ; /* 8 Digital Output */
fsType     = 0x01 ; /* Turn all outputs off */
fsPattern  = 0x00 ; /* Pattern */
fsTime     = 40    ; /* wait 4 seconds after Comm. Failure before implementing */
```

```
OL_SetUpDigitalOutputFailSafe(address, 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 optilogic.h)

```
BOOL _stdcall OL_ReadDigitalOutput(NODEADDR addr, USHORT modno, BYTE type,
    ULONG *data);
```

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.

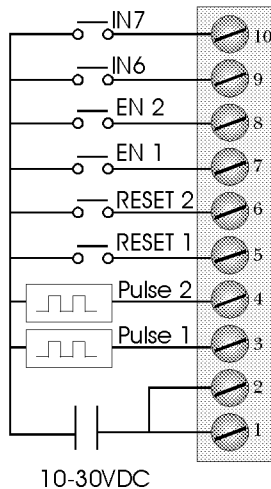
```
USHORT    slot;           /* slot output card resides in */
ULONG     digouts;        /* output status data */
BYTE      modtype;        /* output card type */
NODEADDR  nodeaddr;       /* node address */

nodeaddr = 23;            /* nodes address is 23 */
slot = 3;                 /* card resides in slot 3 */
modtype = 9;              /* 8 channel digital outputs are type 9 */

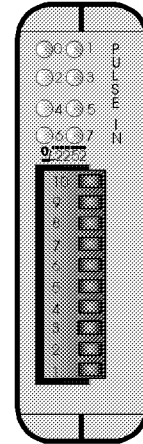
/* Send outputs, and use the boolean response */
if (OL_ReadDigitalOutput(nodeaddr, slot, modtype, &digouts))
{
    /* if out 5 is on, process data */
    if (digouts & 0x20)
    {
        /* output 5 is ON, process data here */
    }
}
else
{
    /* Error handling code */
}
```

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 `optilogic.h`)

```
BOOL OL_Configure_Counter(NODEADDR Addr, USHORT modno, BYTE ctlflags,
                           BYTE dbSet1, BYTE dbSet2)
```

where : <code>ctlflags</code> -	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
<code>dbSet1</code> -	debounce count for channel 1 (establishes the max pulse frequency that the channel will count)
	<code>dbset1 = 2</code> - 15 KHz
	<code>= 4</code> - 10 KHz
	<code>= 8</code> - 5 KHz
	<code>= 16</code> - 2.5 KHz
	<code>= 40</code> - 1 KHz
<code>dbSet2</code> -	debounce count for channel 2

OL2252 continued

Send Counter Controls and Read Counts OL_Read_Counter()

BOOL OL_Read_Counter (NODEADDR Addr, USHORT modno, BYTE flags,
BYTE *countdata, ULONG *inData)

where : flags - control flags

bit 0 - channel 1 enable

bit 1 - channel 1 reset

bit 4 - channel 2 enable

bit 5 - channel 2 reset

*countdata - pointer to 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.

*inData- pointer to 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.

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.

```

BYTE      mod_no, hw_control, sw_control;
NODEADDR  nodeaddr;
BYTE      BYTE      debounce1, debounce2;
ULONG     digins;
struct count {
    ULONG     dat1;
    ULONG     dat2;
}
union count_info {
    BYTE countdata[8];
    struct count count;
}
union count_info    pulsar;

mod_no = 3;
nodeaddr = 7;
hw_control = 0x03;           // Enable ch 1 hardware control, disable ch 2 hw control
debounce1 = debounce2 = 2;   // 20KHz max rate
OL_Configure_Counter (nodeaddr, mod_no, hw_control, debounce1, debounce2);
while (1)
{
    sw_control = 0x11;
    OL_Read_Counter(nodeaddr, modno, sw_control, &pulsar, &digins);
    if (pulsar.count.dat1 > 400000)
    {
        // Send output signal
    }
    else
    {
        // Clear output signal
    }
    if (pulsar.count.dat2 > 1000000)
    {
        sw_control = 0x21;
        // Reset count
        OL_Read_Counter(nodeaddr, modno, sw_control, &pulsar, &digins);
        // Perform required operation
    }
}

```

OL2258 High Speed Pulse Counter

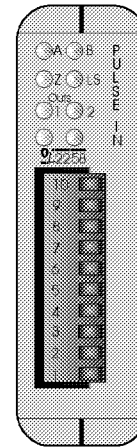
The OL2258 High Speed Pulse Counter is designed to interface to a variety of standard pulse encoder devices. The electrical interface is shown on the right. Differential, sourcing or sinking inputs can be interfaced to the OL2258.

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.

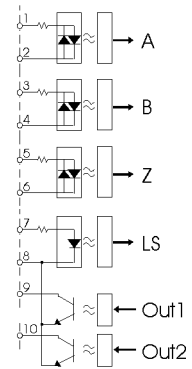
For more detailed information on the features of the OL2258 High Speed Pulse Counter, see the section on the OL2258 in the **OptiLogic Input/Output Modules** manual.

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 minimum and maximum 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.



OL2258 Interface



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	

Continued on next page.

Configure High Speed Counter OL_Configure_HS_Counter()

Prototype : (defined in optilogic.h)

**BOOL OL_Configure_HS_Counter (NODEADDR Addr, USHORT modno, BYTE ctlflags, BYTE
cfgflags, BYTE *preSet);**

where: modno =	slot that card resides in (0 - 3)
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)
preSet =	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 optilogic.h)

**BOOL OL_HS_SendOutput_Range_Counter (NODEADDR Addr, USHORT modno, BYTE
output, BYTE *minValue, BYTE *maxValue);**

where: modno =	slot that card resides in (0 - 3)
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 optilogic.h)

BOOL OL_Read_HS_Counter(NODEADDR Addr, USHORT modno, BYTE *countdata, BYTE *freqdata, ULONG *inStat, ULONG *outStat);

where: modno = slot that card resides in (0 - 3)
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.

```
USHORT slot ;
NODEADDR node ;
BOOL status ;
BYTE configFlags, controlFlags, presetCntValue ;

slot = 3 ;                /* The OL2258 is in slot 3. */
node = 2 ;                /* The OL2258 is at node 2. */

controlFlags = 0x08 ;     /* Quadrature type counting */
configFlags = 0x11 ;     /* Output 1 range enabled and LS preset enabled */
presetCntValue = 12867 ;  /* set preset count value to 12867 */

status = OL_Configure_HS_Counter(node, slot, controlFlags, configFlags, &presetCntValue) ;
if (!status)
{
    /* *****
    process error
    ***** */
}
```

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.

```
USHORT slot ;
NODEADDR node ;
BOOL status ;
BYTE SendOutput, MinRangeValue, MaxRangeValue ;

slot = 3 ;                /* The OL2258 is in slot 3 */
node = 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(node, slot, SendOutput, &MinRangeValue,
&MaxRangeValue) ;
If (!status)
{
    /* *****
    process error
    ***** */
}
```

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 in Hertz. As in Examples A and B, the OL2258 resides at node 2 in slot 3.

```
USHORT slot ;
NODEADDR node ;
BOOL status ;
BYTE countValue[4], freqCount[2] ;
ULONG inputStatus, outputStatus, currentCount ;
unsigned int frequency, freq_in_Hz ;

slot =      3 ;           /* The OL2258 is in slot 3. */
node =     2 ;           /* The OL2258 is at node 2. */

status = OL_Read_HS_Counter(node, slot, countValue, freqCount, &inputStatus,
    &outputStatus) ;
if (!status)
{
    /* *****
    process error
    ***** */
}
else
{
    /* *****Convert the countValue array to a long ***** */

    currentCount = (countValue[0] << 24) + (countValue[1] << 16) + (countValue[2] << 8) +
        countValue[3] ;

    /* *****Convert the frequency count to an integer ***** */
    frequency = (freqCount[0] << 8) + freqCount[1] ;

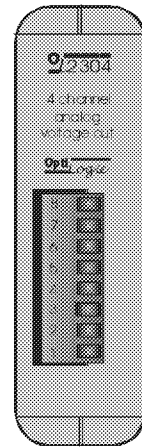
    /* *****Depending on time period, calculate frequency in Hz ***** */
    if (inputStatus & 0x80)
    {
        /* if time period = 1 second , f = (counts / 1 second) */
        freq_in_Hz = frequency ;
    }
    else
    {
        /* time period = .2 seconds, f = (counts / .2 seconds) */
        freq_in_Hz = frequency / 0.2 ;
    }
}

/* 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 (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 automatically output the value of itself minus 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 optilogic.h)

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 optilogic.h)

```
BOOL _stdcall OL_Buffered_WriteAnalogOutput(NODEADDR addr, USHORT modno, BYTE  
type, USHORT *buffer);
```

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 21 in 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 21 in 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
```

```
  ' *****
```

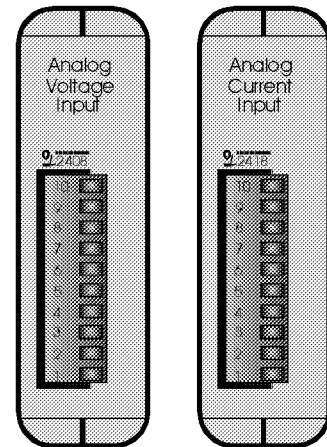
```
  ' Place Error handling code here
```

```
  ' *****
```

```
End If
```

Read Analog Inputs OL_ReadAnalogInput()

This function can be used to read all of the analog inputs 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 optilogic.h)

```
BOOL OL_ReadAnalogInput(NODEADDR addr, USHORT modno, BYTE modtype,
                        USHORT *buffer)
```

where : buffer is a pointer to an array

Usage :

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

```
USHORT    modno ;
USHORT    anaval[8] ;
BYTE      modtype ;
NODEADDR  nodeaddr ;

nodeaddr = 23 ;
modno = 0 ;
modtype = 18 ;           /* 8 channel analog inputs are type 18 */

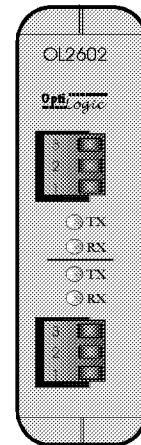
if (OL_ReadAnalogInput(nodeaddr, modno, modtype, &anaval))
{
    /* process readings */
}
else
{
    /* error processing */
}
```

The 8 analog values will be placed in the array anaval[.].

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.

- 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

Configure a Communication Port OL_ConfigureSerialPort()

Prototype : (defined in optilogic.h)

```
BOOL OL_ConfigureSerialPort (NODEADDR addr, USHORT modno, BYTE type,
                             enum OL_SERIAL_PORTS, enum OL_SERIAL_BAUD_RATES,
                             enum OL_SERIAL_DATA_BITS, enum OL_SERIAL_PARITY,
                             enum OL_SERIAL_STOP_BITS)
```

where : modno = 0 or 1 for OL2602 or 81 for Base Comm Port
 type = 112 for OL2602 or 0 for Base Comm Port
 OL_SERIAL_PORTS = 0 (base), 1 (OL2602 - top) or 2 (OL2602- bottom)
 OL_SERIAL_BAUD_RATES, OL_SERIAL_DATA_BITS,
 OL_SERIAL_PARITY, & OL_SERIAL_STOP_BITS

Use the selections listed in the optilogic.h header file.

OL2602 continued

Read received Data OL_ReadSerialData()

Prototype : (defined in optilogic.h)

BOOL OL_ReadSerialData (NODEADDR addr, USHORT modno, BYTE type,
enum OL_SERIAL_PORTS, BYTE *numbytes, BYTE *flags,
BYTE *buffer)

where : modno = 0 or 1 for OL2602 or 81 for Base Comm Port
 type = 112 for OL2602 or 0 for Base Comm Port
 OL_SERIAL_PORTS = 0(base), 1(OL2602 - top), or 2(OL2602 - bottom)
 *numbytes = Pointer to the variable which will contain the # of receive
 bytes returned.
 *flags = Pointer to variable to store returned error flags associated with
 received bytes
 *buffer = Pointer to buffer to use to store received data

Write to Serial Port OL_WriteSerialData()

Prototype : (defined in optilogic.h)

BOOL OL_WriteSerialData(NODEADDR addr, USHORT modno, BYTE type,
enum OL_SERIAL_PORTS port, BYTE numbytes,
BYTE *buffer, BYTE *left)

where : modno = 0 or 1 for OL2602 or 81 for Base Comm Port
 type = 112 for OL2602 or 0 for Base Comm Port
 OL_SERIAL_PORTS = 0(base), 1(OL2602 - top), or 2(OL2602 - bottom)
 numbytes = Number of bytes being sent
 *buffer = Pointer to the buffer containing the data to send
 *left = Pointer to the 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 of 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.

```
BYTE      PortType;
NODEADDR  address;
USHORT    slot;

Slot      = 0;      /* slot 0 */
address   = 7;      /* node address = 7 */
PortType  = 0x70;   /* OL2602 = 0x70 */
OL_ConfigureSerialPort(address, slot, PortType, OL_SERIALPORT_1,
                        OL_SERIAL_BAUD_9600, OL_SERIAL_8_DATA_BITS,
                        OL_SERIAL_NO_PARITY, OL_SERIAL_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 clears from the buffer.

The interval should be set low enough to keep the data from overflowing. You should adjust the interval within the program until you get consistent results. An example of a routine used to read data from a serial port is shown below.

```
void ReceiveOL2602 (unsigned int *RxByteCount, unsigned char *RxBuffer,
                   NODEADDR address, USHORT modno, char PortNo)
{
  /* PortNo = serial port no (1or 2) */
  /* address = node address of RTU */
  /* modno = slot containing OL2602 */
  BYTE numbytes; /* number of bytes in response */
  BYTE flags;
  BYTE buffer[48]; /* return data buffer */
  int i; /* loop counter */
  type = 0x70; /* OL2602 */
  OL_ReadSerialData (address, slot, type, PortNo, &numbytes, &flags, &buffer);
  /*Buffer received data into Receive Buffer*/
  For (i = 0; i < numbytes - 1; i++)
  {
    RxBuffer[RxByteCount] = buffer[ i ];
    RxByteCount ++;
  }
}
```

Note: This routine does not check to see when a message is complete. You must do that depending on your own message format.

Writing Data to a Serial Port

The following routine is an example that can be used to transmit data out the top port of an OL2602. The node address, slot no, and Port are passed into this routine, as well as the buffered data and the number of bytes to transmit.

```
void TransmitData (unsigned int TransmitLength, unsigned char *TransmitBuffer,
NODEADDR address, USHORT modno, char PortNo)
{
    int i;
    unsigned int BytesTransmitted; /* total number of bytes that have been transmitted */
    unsigned char NumberToTransmit; /* number of bytes placed into tx buffer of OL2602*/
    unsigned char buffer[48];
    BytesTransmitted = 0; /* initialize tx byte count */
    BytesLeft = 48; /* start with Full tx buffer */
    if (TransmitLength > 48) /* 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 */
        {
            NumberToTransmit = Bytes Left / 2;
            /*Buffer data & increment counter*/
            For (i = 0; i < NumberToTransmit - 1; i ++)
            {
                buffer[ i ] = TransmitBuffer[BytesTransmitted];
                BytesTransmitted ++;
            }
            OL_WriteSerialData(address, modno, 0x70, PortNo, NumberToTransmit,
                &buffer, &BytesLeft);
        }
    }
    else /*Buffer entire message & send at same time*/
    {
        For (i = 0; i < TransmitLength - 1; i ++)
            buffer[ i ] = TransmitBuffer[ i ];

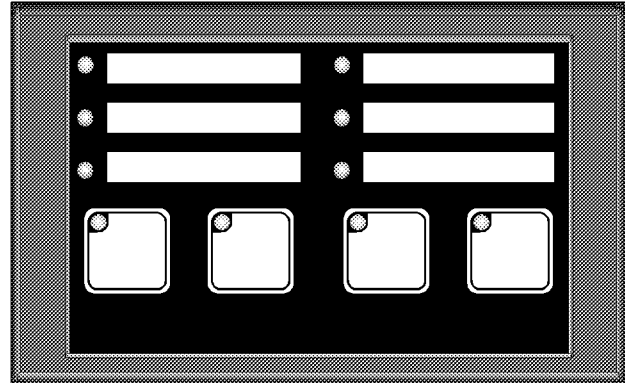
        OL_WriteSerialData(address, modno, 0x70, PortNo,
            TransmitLength, &buffer, &BytesLeft);
    }
}
```

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 status LED 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 optilogic.h)

BOOL OL_OL3406_StatusControl(NODEADDR addr, BYTE *data, BYTE *pbstatus)

where : data[4] is an array with the following definition

data[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.
data[1] =	on/off control of indicator light LEDs. Bits are in sequence starting at bit 0.
data[2] =	flash control of 4 inset button LEDs. LED must be on to flash. Bits are in sequence starting at bit 0.
data[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.

OL3406 continued

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. Conversely, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in optilogic.h)

```
BOOL OL_OL3406_ForceButtons(NODEADDR addr, enum OL34xx_BUTTON_FLAGS,  
                             BYTE ButtonData)
```

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

OL34xx_BUTTON_FLAGS defines the type of force as follows -

- = OL34xx_BUTTON_ALL ; force ALL buttons to a state matching ButtonData (force state)
- = OL34xx_BUTTON_OR ; logically OR current button state with ButtonData (force on)
- = OL34xx_BUTTON_AND ; logically AND current button state with the complement of ButtonData (force off)

Configure OL3406 Panel**OL_OL3406_ConfigurePanel()****Prototype : (defined in optilogic.h)**

```
BOOL OL_OL3406_ConfigurePanel(NODEADDR addr, BYTE Conflags)
```

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 optilogic.h)**

```
BOOL OL_OL3406_ReadPanelConfiguration(NODEADDR addr, BYTE *Conflags)
```

where : Conflags is defined above

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 light 3, flashes light 4 and performs an application process. At the end of the process, it forces the first button off.

```

BYTE  config ;           /* Configuration */
BYTE  req_config ;       /* Required configuration */
BYTE  litestat[4] ;      /* Light state array */
BYTE  butt_stat ;        /* Button status */
NODEADDR  address ;      /* Node address */

req_config = 0x07 ;      /* Required configuration = buttons 1 - 3 alternate action
                           button 4 momentary, no LED separation */

litestat[0] = 0 ;        /* Start with all lights out */
litestat[1] = 0 ;
litestat[2] = 0 ;
litestat[3] = 0 ;

OL_OL3406_ReadPanelConfiguration(address, &config) ;
if (config != req_config)
{
    OL_OL3406_ConfigurePanel(address, req_config) ;
}

while (1)
{
    /* Forever loop */
    /* Is the first button active ? */
    OL_OL3406_StatusControl(address, &litestat, &butt_stat) ;
    if (butt_stat & 0x01)
    {
        /* If so, turn lights 3 and 4 on & flash light 4 */
        litestat[1] |= 0x0C ;
        litestat[3] |= 0x08 ;
        OL_OL3406_StatusControl(address, &litestat, &butt_stat) ;

        /* Do function processing */

        /* Clear the first button */
        OL_OL3406_ForceButtons(address, OL34XX_BUTTON_AND, 0x01) ;
    }
}

```

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 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 optilogic.h)

BOOL OL_OL3420_StatusRequest(NODEADDR addr, BYTE LEDState, BYTE *Status)

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), must be on to flash
Status -	current status of panel buttons will be placed in this byte. Bits are in sequence starting at 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. Conversely, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in optilogic.h)

```
BOOL OL_OL3420_ForceButtons(NODEADDR addr, enum OL34xx_BUTTON_FLAGS,  
                             BYTE ButtonData)
```

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

OL34xx_BUTTON_FLAGS defines the type of force as follows -

- = OL34xx_BUTTON_ALL ; force ALL buttons to a state matching ButtonData (force state)
- = OL34xx_BUTTON_OR ; logically OR current button state with ButtonData (force on)
- = OL34xx_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 optilogic.h)**

```
BOOL OL_OL3420_SendTextDisplayMessage(NODEADDR addr, BYTE line, char *message)
```

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

OL3420 continued

Configure OL3420 Terminal
OL_OL3420_ConfigurePanel()

Prototype : (defined in optilogic.h)

BOOL OL_OL3420_ConfigurePanel(NODEADDR addr, BYTE buttonstate)

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 optilogic.h)

BOOL OL_OL3420_ReadConfiguration(NODEADDR addr, BYTE *buttonstate)

where : buttonstate is defined above

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.

```
#include <string.h> /* this file should be included in the program so the strcpy() function can
                    be used */

BYTE  config ;           /* Configuration */
BYTE  req_config ;       /* Required configuration */
BYTE  litestat ;         /* Inset LED state */
BYTE  lineno ;           /* Display line number */
BYTE  butt_stat ;        /* Button status */
NODEADDR  address ;      /* Node address */
char message[21] ;       /* array containing message string to send to panel */
int i ;

req_config = 0x07 ;      /* Required configuration = buttons 1 - 3 alternate action
                          button 4 momentary, no LED separation */
litestat = 0 ;           /* Start with all lights out (unused because no LED separation)*/

OL_OL3420_ReadConfiguration(address, &config) ;
if (config != req_config)
{
    OL_OL3420_ConfigurePanel(address, req_config) ;
}

while (1)
{
    /* Forever loop */
    /* Is the second button active ? */
    OL_OL3420_StatusRequest(address, litestat, &butt_stat) ;
    if (butt_stat & 0x02)
    {
        /* If so, send message to the display */
        for (i = 0; i < 21; i++)
        {
            message[i] = 0 x 20; /* fill with blank spaces */
        }
        strcpy(message, "Process Active".)
        lineno = 0 ;
        OL_OL3420_SendTextDisplayMessage(address, lineno, &message) ;

        /* Do function processing */

        /* Clear the second button */
        OL_OL3420_ForceButtons(address, 0x20, 0x02) ;
    }
}
```

OL3440 Display Panel

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



Send Text to OL3440 Display OL_OL3440_SendTextDisplayMessage()

Prototype : (defined in optilogic.h)

BOOL OL_OL3440_SendTextDisplayMessage(NODEADDR addr, BYTE line, char *message)

where : line is line number (0 - 3 from top to bottom)
message is 20 characters of text

Usage

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

```
#include <string.h> /* this file should be included in the program so the strcpy() function can  
be used */
```

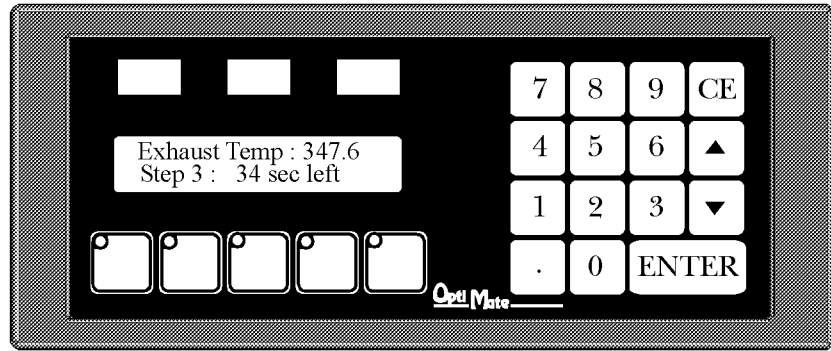
```
char message[21]; /* array containing message string to send to panel */  
int i; /* loop counter */
```

```
for (i = 0; i < 21; i ++)  
{  
    message[i] = 0 x 20 }  
    strcpy(message,"Text message..abcd") ;
```

```
OL_OL3440_SendTextDisplayMessage(6, 1, &message) ;
```

OL3850 Operator Terminal

The OL3850 Operator Terminal has a 2 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:

- OL_OL3850_StatusRequest() - Control lights, get button status and retrieve entered data
- OL_OL3850_SendTextDisplayMessage() - Send a text line to one of the two display lines
- OL_OL3850_ConfigurePanel() - Configure the operation of the five function keys
- OL_OL3850_ReadConfiguration() - Read the configuration of the five function keys
- OL_OL3850_ForceButtons() - Force the state of selected “alternate-action” buttons
- OL_OL3850_SendKeypadMessage() - Send a message with a field for the entry of a number input from the keypad
- OL_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 optilogic.h)

BOOL OL_OL3850_StatusRequest(NODEADDR addr, BYTE LEDState, BYTE LEDFlash, BYTE *Status, double *ReturnDATA)

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)
LEDFlash - for all LEDs and light bars, “1” = on , “0” - off
Status - lash controls for the same LEDs and light bars mentioned for LEDState. To flash, the light must be on.
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” indicated 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 stored in this variable.

Send a Text Message to the Display

OL_OL3850_SendTextDisplayMessage()

Prototype : (defined in optilogic.h)

BOOL OL_OL3850_SendTextDisplayMessage(NODEADDR addr, BYTE line, char *message)

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

OL3850 continued

Configure OL3850 Function Buttons

OL_OL3850_ConfigurePanel()

Prototype : (defined in optilogic.h)

BOOL OL_OL3850_ConfigurePanel(NODEADDR addr, BYTE buttonstate)

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 optilogic.h)

BOOL OL_OL3850_ReadConfiguration(NODEADDR addr, BYTE *buttonstate)

where : buttonstate is defined above

Force Alternate-Action Button Status

OL_OL3850_ForceButtons()

Prototype : (defined in optilogic.h)

BOOL OL_OL3850_ForceButtons(NODEADDR addr, enum OL34xx_BUTTON_FLAGS,
BYTE ButtonData)

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

OL34xx_BUTTON_FLAGS defines the type of force as follows -

= OL34xx_BUTTON_ALL ; force ALL buttons to a state matching ButtonData (force state)

= OL34xx_BUTTON_OR ; logically OR current button state with ButtonData (force on)

= OL34xx_BUTTON_AND ; 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 optilogic.h)**

```

BOOL OL_OL3850_SendKeypadMessage(NODEADDR addr, BYTE line, char *message,
                                   double StartDATA, BYTE decimalPoint)

```

where:	message-	Text message to display. Carets “^” should be used as place holders for the data entry field
	StartDATA-	Initial value
	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 StartDATA is passed as a double. The decimalPoint is the number of digits after the decimal. In other words, if StartDATA = 4567 and decimalPoint = 2, the starting value is 45.67.

Send Arrow Adjust Message
OL_OL3850_SendArrowMessage()**Prototype : (defined in optilogic.h)**

```

BOOL OL_OL3850_SendArrowMessage(NODEADDR addr, BYTE line, char *message,
                                  unsigned long *NumDATA, BYTE decimalPoint,
                                  unsigned long LowLimit, unsigned long HighLimit)

```

where:	message-	Text message to display. Carets “^” should be used as place holders for the data entry field
	NumDATA-	Initial value
	dp-	A beginning value must be passed, which the user can adjust using the arrow key. The numeric NumDATA is passed as a pointer to a long. The dp (decimal point) is the number of digits after the decimal. In other words, if NumDATA = 4567 and dp = 2, the starting value is 45.67.
	LowLimit -	The minimum value allowed for adjustment (the same dp value applies)
	HighLimit -	The maximum value allowed for adjustment (the same dp value applies)

For example, if NumDATA = 4567, dp = 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

This routine, Handle_OL3850(), is the body of the program. Several parameters are passed to/from this routine, making it more generic. When the routine is called, it performs several tasks.

- (1) It reads the configuration of the pushbuttons. If they are not as required by the program, the program will reconfigure them.
- (2) The program reads/updates the general status (lamps, LEDs, button status and current data entry value). Next, the program takes action depending upon certain parameters.
- (3) If button 3 is pressed, 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 arrowadjust message is not active, then it sends a text message to the top line and an arrow adjust message to the bottom line.
 - (2) Else 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.

```
#include <string.h> /* this file should be included in the program so the strcpy() function can
                    be used */
```

```
void Handle_OL3850(unsigned char LEDState, unsigned char LEDFlash, char KeypadActive,
                  char ArrowActive, char ForceActive, double KeypadData, unsigned long ArrowData)
{
```

```
/*+++++
   The node address for this example is 30!!
```

```
    LEDState - lamp and LED ON/OFF state
    LEDFlash - lamp and LED flash state
    KeypadActive - flag denoting that a keypad message is active on the LCD
    ArrowActive - flag denoting that an arrow adjust message is active on the LCD
    ForceActive - flag denoting that button force has occurred
    KeypadData - value of a keypad entry message as passed from the panel
    ArrowData - value of an arrow adjust message as passed from the panel
```

```
+++++*/
```

```
unsigned char config ;      /* configuration data read from panel */
unsigned char RequiredConfig ; /* Required configuration */
unsigned char ButtonStatus ; /* Button status read back from panel */
char message[21] ; /* array containing message string to send to the panel */
double ReturnValue ; /* current data entry value returned from the panel */
int i ; /* loop counter */
```

```
RequiredConfig = 0x19 ; /* buttons 1, 4, 5 alternate, buttons 2, 3 momentary */
```



```
/* read the configuration from the panel */
OL_OL3850_ReadConfiguration(30, config) ;
/* if the configuration read from the RTU doesn't match the required config, send the
   configuration message */
if (RequiredConfig != config)
    OL_OL3850_ConfigurePanel(30, RequiredConfig) ;

/* update and read the general status of the 3850 */
OL_OL3850_StatusRequest(30, LEDState, LEDFlash, &ButtonStatus, &ReturnValue) ;

/* if button 3 is pressed */
if (ButtonStatus & 0x04)

{
    /* send text message to top line */
    strcpy(message, "Key Data,Press ENTER") ;
    OL_OL3850_SendTextDisplayMessage(30, 0, &message) ;

    /* send keypad entry message to bottom line, initial data and decimal pt are both 0 */
    for (i = 0 ; i < 20 ; i++)

        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }
    strcpy(message, "Keypad Entry ^^^^^") ;
    OL_OL3850_SendKeypadMessage(30, 1, &message, 0, 0) ;
    KeypadActive = 1 ; /* set the keypad message active flag */
}

/* if a keypad message is active and the data available bit (the most significant bit of the button
   status) is active, keypad entry complete */
else if ((KeypadActive == 1) && (ButtonStatus & 0x80))
{
    KeypadData = ReturnValue ; /* store keypad entry data to pass to calling routine */

    /* send text message to blank top line of LCD */
    for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }
    OL_OL3850_SendTextDisplayMessage(30, 0, &message) ;

    /* send text message to bottom line of LCD */
    for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }
    strcpy(message, "Keypad Entry Done") ;
    OL_OL3850_SendTextDisplayMessage(30, 1, &message) ;
    KeypadActive = 0 ; /* reset keypad active flag */
}
```

```

/* if button 4 is active and force active flag inactive */
else if ((ButtonStatus & 0x08) && (ForceActive == 0))
{
    /* if arrow adjust message not active, send it */
    if (ArrowActive == 0)
    {
        /* send text message to top line */
        for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }
        strcpy(message, "Press F2 When Done") ;
        OL_OL3850_SendTextDisplayMessage(30, 0, &message) ;

        /* send arrow adjust message to bottom line, initial value is 3686.75, range
           is 3000.00 - 4000.00 */
        for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }

        strcpy(message, "Adjust Value ^^^^^^") ;
        OL_OL3850_SendArrowMessage(30, 1, &message, 368675, 2, 300000,
                                   400000);
        ArrowActive = 1 ; /* set the arrow adjust message active flag */
    }
    /* if arrow adjust message is active, read data */
    else if (ArrowActive == 1)
    {
        ArrowData = ReturnValue ; /* copy data to variable to return to calling routine so
                                   data can be updated as it changes */
        /* if button 2 is pressed, reset arrow message */
        if (ButtonStatus & 0x02)
        {
            /* send text message to blank top line of LCD */
            for (i = 0 ; i < 20 ; i++)
            {
                message[i] = 0x20 ; /* fill message buffer with spaces */
            }
            OL_OL3850_SendTextDisplayMessage(30, 0, &message) ;

            /* send text message to bottom line of LCD */
            for (i = 0 ; i < 20 ; i++)
            {
                message[i] = 0x20 ; /* fill message buffer with spaces */
            }

            strcpy(message, "Arrow Adjust Done") ;
            OL_OL3850_SendTextDisplayMessage(30, 1, &message) ;

            /* Force Button 4 OFF */
            OL_OL3850_ForceButtons(30, OL34XX_BUTTON_AND, 0x08) ;
        }
    }
}

```

```
        ArrowActive = 0 ; /* reset keypad active flag */
        ForceActive = 1 ; /* set force active flag to temporarily lock out the
                           arrow adjust message */
    }
}
/* else if button 4 cleared and force active flag set, disable force active flag */
else if (!(ButtonStatus & 0x08) && (ForceActive == 1))
{
    ForceActive = 0 ; /* reset force active flag */
}
}
```

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 TRUE returned indicates the function was successful. If a FALSE is returned, the error code can be retrieved by calling OL_GetLastError() - 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()

Prototype : (defined in optilogic.h)

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 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 to be updated (i.e. number of elements in the array) also has to be passed in to the function.

BOOL OL_Buffered_UpdateNodes (NODEADDR *NODEARRAY, DWORD arraysize)

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.

/* 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. */

```
int noslots, i;
BYTE mtype[9], stype[9], mver[9], minver, majver, basetype;
NODEADDR NodeList[9]; /* array containing list of active nodes */
DWORD NodeCount;      /* total number of nodes found */

NodeCount = 0;
for( i = 0; i < 99; i++)
{
    noslots = OL_QueryNode(i, basetype, mtype(0), stype(0), mver(0), minver, majver);
    if(noslots > 0)
    {
        NodeList[NodeCount] = i ; /* add node to array for buffered mode */
        NodeCount++; /* increment number of active nodes found */
    }
}
```

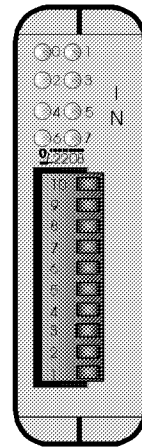
/* This code can be in another routine to read and update the status of each node at a fixed interval. */

```
status= OL_Buffered_UpdateNodes(NodeList, NodeCount);
```

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, 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 optilogic.h)

```
BOOL OL_Buffered_ReadDigitalInput(NODEADDR Addr, USHORT modno,
                                  BYTE type, ULONG *inputdata)
```

where:

Addr	-	node address
modno	-	RTU slot
type	-	input type number, 4 digital in, 8 digital in, etc.
inputdata-		input status returned from the RTU

Usage :

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

```
USHORT    modno ;
ULONG     digins ;
BYTE      modtype;
NODEADDR  nodeaddr ;
```

```
nodeaddr = 23 ;
modno = 2 ;
modtype = 1 ;           /* 8 channel digital inputs are type 1 */
```

```
if (!OL_Buffered_ReadDigitalInput(nodeaddr, modno, modtype, &digins))
{
    /* error processing */
}
```

```
/* Somewhere later in the loop */
if (digins & 0x08)
{
    /* application processing */
}
```

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 optilogic.h)

```
BOOL _stdcall OL_Buffered_ReadLatchedDigitalInput(NODEADDR Addr, USHORT modno,
    BYTE type, ULONG *inputdata);
```

where: Addr - node address
 modno - input slot
 type - input type number, 4 digital in, 8 digital in, etc.
 inputdata - input status returned from the RTU status of inputs
 (input 0 --> bit 0, input 1 --> bit 1 ...)

Usage :

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

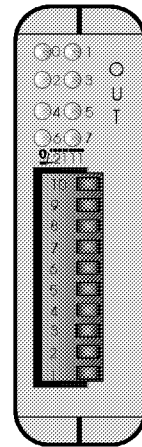
```
USHORT    slot;
ULONG     digins;
BYTE      modtype;
NODEADDR  nodeaddr;

nodeaddr = 23;
slot = 2;
modtype = 1;          /* 8 channel digital inputs are type 1 */

if (OL_Buffered_ReadLatchedDigitalInput(nodeaddr, slot, modtype, &digins))
{
    if (digins & 0x08)
    {
        /* application processing */
    }
}
```

Write Digital Outputs OL_Buffered_WriteDigitalOutput()

The OL_Buffered_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”, ie. 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 optilogic.h)

```
BOOL OL_Buffered_WriteDigitalOutput(NODEADDR addr, USHORT modno, BYTE type,
                                     ULONG data)
```

where:	addr	-	node address
	modno	-	RTU slot
	type	-	output type number, 4 digital out, 8 digital out, etc.
	data	-	output state

Usage :

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

```
USHORT    modno;
ULONG     digouts ;
BYTE      modtype;
NODEADDR  nodeaddr ;
```

```
/* Somewhere in the program loop */
nodeaddr = 23 ;
modno = 3 ;
modtype = 9 ;           /* 8 channel digital outputs are type 9 */
```

```
digouts = 0x68 ;
```

```
/* At least once each loop */
```

```
/* Send outputs, and use the boolean response */
if (!OL_Buffered_WriteDigitalOutput(nodeaddr, modno, modtype, digouts)
    {
        /* Error handling code */
    }
```


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 optilogic.h)

```
BOOL _stdcall OL_Buffered_ReadDigitalOutput(NODEADDR Addr, USHORT modno, BYTE
type, ULONG *inputdata);
```

where *inputdata = 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.

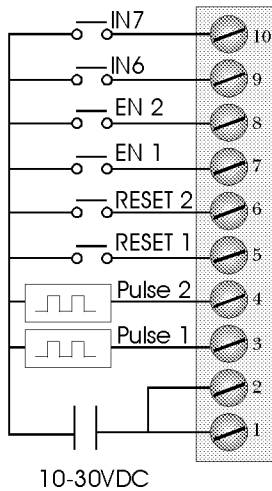
```
USHORT    slot;           /* slot output card resides in */
ULONG     digouts;        /* output status data */
BYTE      modtype;        /* output card type */
NODEADDR  nodeaddr;       /* node address */

nodeaddr = 23;            /* nodes address is 23 */
slot = 3;                 /* card resides in slot 3 */
modtype = 9;              /* 8 channel digital outputs are type 9 */

/* Send outputs, and use the boolean response */
if (OL_Buffered_ReadDigitalOutput(nodeaddr, slot, modtype, &digouts))
{
    /* if out 5 is on, process data */
    if (digouts & 0x20)
    {
        /* output 5 is ON, process data here */
    }
}
else
{
    /* Error handling code */
}
```

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 `optilogic.h`)

```
BOOL OL_Buffered_Configure_Counter(NODEADDR Addr, USHORT modno, BYTE ctlflags,
                                     BYTE dbSet1, BYTE dbSet2)
```

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

Send Counter Controls and Read Counts

OL_Buffered_Read_Counter()

BOOL OL_Buffered_Read_Counter (NODEADDR addr, USHORT modno, BYTE flags,
BYTE *countdata, ULONG *inData)

where : flags - control flags

bit 0 - channel 1 enable

bit 1 - channel 1 reset

bit 4 - channel 2 enable

bit 5 - channel 2 reset

*countdata - pointer to 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.

*inData- pointer to 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.

```

BYTE      mod_no, hw_control, sw_control;
NODEADDR  nodeaddr;
BYTE      debounce1, debounce2;
ULONG     digins;
struct count {
    ULONG   dat1;
    ULONG   dat2;
}
union count_info {
    BYTE  countdata[8];
    struct count count;
}
union count_info pulsar;

mod_no = 3;
nodeaddr = 7;
hw_control = 0x03;           // Enable ch 1 hardware control, disable ch 2 hw control
debounce1 = debounce2 = 2;   // 20KHz max rate
OL_Buffered_Configure_Counter (nodeaddr, mod_no, hw_control, debounce1, debounce2);
while (1)
{
    sw_control = 0x11;
    OL_Buffered_Read_Counter(nodeaddr, modno, sw_control, &pulsar, &digins);
    if (pulsar.count.dat1 > 400000)
    {
        // Send output signal
    }
    else
    {
        // Clear output signal
    }
    if (pulsar.count.dat2 > 1000000)
    {
        sw_control = 0x21;
        // Reset count
        OL_Buffered_Read_Counter(nodeaddr, modno, sw_control, &pulsar, &digins);
        // Perform required operation
    }
    OL_Buffered_UpdateNodes(NodeList, NodeCount);
}

```

OL2258 High Speed Pulse Counter

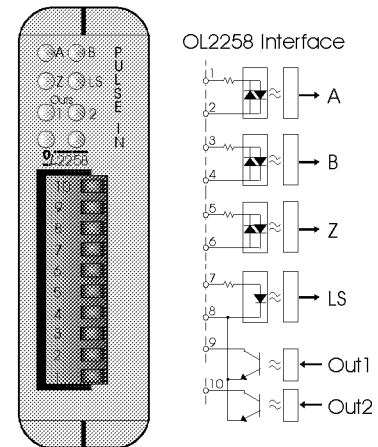
The OL2258 High Speed Pulse Counter is designed to interface to a variety of standard pulse encoder devices. The electrical interface is shown on the right. Differential, sourcing or sinking inputs can be interfaced to the OL2258.

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_Buffered_Configure_HS_Counter()** - Defines count type (pulse and direction, quadrature or up/down.), configures preset inputs, frequency period and preset value.
- **OL_Buffered_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	

Continued on next page.

Configure High Speed Counter OL_Buffered_Configure_HS_Counter()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_Configure_HS_Counter (NODEADDR Addr, USHORT modno, BYTE
ctlflags, BYTE cfgflags, BYTE *preSet);

where: modno = slot that card resides in (0 - 3)
 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)
 preSet = 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 optilogic.h)

BOOL OL_Buffered_HS_SendOutput_Range_Counter (NODEADDR Addr, USHORT modno,
BYTE output, BYTE *minValue, BYTE *maxValue);

where: modno = slot that card resides in (0 - 3)
 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_Buffered_Read_HS_Counter()

Prototype : (defined in optilogic.h)

```
BOOL OL_Buffered_Read_HS_Counter(NODEADDR Addr, USHORT modno, BYTE  
    *countdata, BYTE *freqdata, BYTE *inStat, BYTE *outStat);
```

where: modno = slot that card resides in (0 - 3)
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.

```
USHORT slot ;
NODEADDR node ;
BOOL status ;
BYTE configFlags, controlFlags, presetCntValue ;

slot = 3 ;                /* The OL2258 is in slot 3. */
node = 2 ;                /* The OL2258 is at node 2. */

controlFlags = 0x08 ;     /* Quadrature type counting */
configFlags = 0x11 ;     /* Output 1 range enabled and LS preset enabled */
presetCntValue = 12867 ;  /* set preset count value to 12867 */

status = OL_Buffered_Configure_HS_Counter(node, slot, controlFlags, configFlags,
&presetCntValue) ;
if (!status)
{
/* *****
process error
***** */
}
```

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.

```
USHORT slot ;
NODEADDR node ;
BOOL status ;
BYTE SendOutput, MinRangeValue, MaxRangeValue ;

slot = 3 ;                /* The OL2258 is in slot 3 */
node = 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(node, slot, SendOutput,
&MinRangeValue, &MaxRangeValue) ;
If (!status)
{
/* *****
process error
***** */
}
```


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 in Hertz. As in Examples A and B, the OL2258 resides at node 2 in slot 3.

```
USHORT slot ;
NODEADDR node ;
BOOL status ;
BYTE countValue[4], freqCount[2] ;
ULONG inputStatus, outputStatus, currentCount ;
unsigned int frequency, freq_in_Hz ;

slot =      3 ;           /* The OL2258 is in slot 3. */
node =      2 ;           /* The OL2258 is at node 2. */

status = OL_Buffered_Read_HS_Counter(node, slot, countValue, freqCount, &inputStatus,
&outputStatus) ;
if (!status)
{
    /* *****
    process error
    ***** */
}
else
{
    /* *****Convert the countValue array to a long ***** */

    currentCount = (countValue[0] << 24) + (countValue[1] << 16) + (countValue[2] << 8) +
countValue[3] ;

    /* *****Convert the frequency count to an integer ***** */
    frequency = (freqCount[0] << 8) + freqCount[1] ;

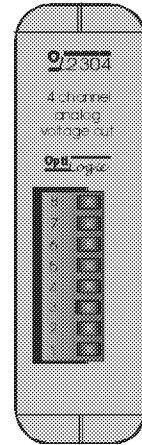
    /* *****Depending on time period, calculate frequency in Hz ***** */
    if (inputStatus & 0x80)
    {
        /* if time period = 1 second , f = (counts / 1 second) */
        freq_in_Hz = frequency ;
    }
    else
    {
        /* time period = .2 seconds, f = (counts / .2 seconds) */
        freq_in_Hz = frequency / 0.2 ;
    }
}
/* End of Example */
```

Note: You must call the OL_QueryNodes() routine to update the nodes. Calling the individual buffered mode routines only sends/retrieves data to/from the optilogicprotocol.dll file. The OL_QueryNodes() routine has to be called to update the nodes and retrieve data from them. This routine can be called via a timer or on some sort of fixed interval within the program.

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 (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 automatically output the value of itself minus 4095. (Ex. A value of 8121 yields $8121 - 4095 = 4026$. A value of 4026 will be outputted.)



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 `optilogic.h`)

```
BOOL OL_Buffered_ConfigAnalogOutput(NODEADDR addr, USHORT modno, BYTE type,
    BYTE range1, BYTE range2, BYTE range3, BYTE range4);
```

where:

- `addr` - node address
- `modno` - slot containing OL2304 (0 - 3)
- `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 optilogic.h)

BOOL OL_Buffered_WriteAnalogOutput(NODEADDR addr, USHORT modno, USHORT *buffer, BYTE type, BYTE numvalues);

where: type - 25 (4 analog outputs are type is 25)

buffer - pointer to address of first element of array (size = 4 elements) 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 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.

```
USHORT Analog_OutVal[4] ;      /* number between 0 and 4095 that is equivalent to
                                output voltage value */
USHORT Voltage_OutVal[4] ;     /* voltage value to output */

/* initialize output voltage values */
Voltage_OutVal[0] = 5 ;
Voltage_OutVal[1] = 2.5 ;
Voltage_OutVal[2] = 1.22 ;
Voltage_OutVal[3] = 3.3 ;

/* Convert to equivalent output value and place in an array */
for (i = 0 ; i < 4 ; i++)
{
    /* use the formula  $x = (y * 4095) / 5$  */
    Analog_OutVal[i] = (Voltage_OutVal[i] * 4095) / 5 ;
}

/* The results from the above calculations are as follows:
    Analog_OutVal[0] = 4095      5VDC output
    Analog_OutVal[1] = 2048      2.5VDC output
    Analog_OutVal[2] = 999       1.22VDC output
    Analog_OutVal[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 21 in slot 2.

```
BYTE range1, range2, range3, range4, modtype ;
USHORT slotno ;
NODEADDR node ;
BOOL status ;

node = 21 ;      /* node address */
slotno = 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(node, slotno, modtype, range1, range2, range3,
                                         range4) ;
```

```
if (!status)
{
    /* *****
    Place Error handling code here
    ***** */
}
/* End of Example */
```

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

```
USHORT analog_val[4], slot ;
BYTE modtype, numvalues ;
NODEADDR address ;
BOOL status ;

address = 21 ;          /* node address */
slot = 2 ;              /* OL2304 resides in slot 2 */
modtype = 25 ;          /* 4 channel analog outputs are type 25 */
numvalues = 4 ;         /* numvalues = # of outputs, since 4 channel, numvalues = 4 */

/* 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: */

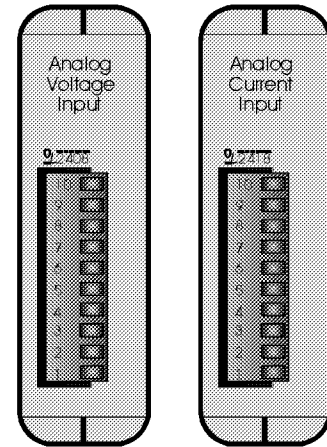
analog_val[0] = 4095 ;   /* 5VDC output */
analog_val[1] = 2048 ;   /* 2.5VDC output */
analog_val[2] = 999 ;    /* 1.22VDC output */
analog_val[3] = 2702 ;   /* 3.3VDC output */
status = OL_Buffered_WriteAnalogOutput(address, slot, analog_val, modtype, numvalues) ;
if (!status)
{
    /* *****
    Place Error handling code here
    ***** */
}

/* End of Example */
```

Note: You must call the OL_QueryNodes() routine to update the nodes. Calling the individual buffered mode routines only sends/retrieves data to/from the optilogicprotocol.dll file. The OL_QueryNodes() routine has to be called to update the nodes and retrieve data from them. This routine can be called via a timer or on some sort of fixed interval within the program.

Read Analog Inputs OL_Buffered_ReadAnalogInput()

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 optilogic.h)

```
BOOL OL_Buffered_ReadAnalogInput(NODEADDR addr, USHORT modno, BYTE type,
    USHORT *buffer)
```

where : buffer is an array. Analog input values are in order in the array. Therefore, Channel 0 is in array index 0.

Usage :

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

```
USHORT    modno, anaval[8] ;
BYTE      modtype ;
NODEADDR  nodeaddr ;

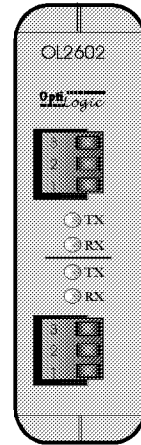
nodeaddr = 23 ;
modno = 0 ;
modtype = 18 ;          /* 8 channel analog inputs are type 18 */

if (OL_Buffered_ReadAnalogInput(nodeaddr, modno, modtype, &anaval))
{
}
else
{
    /* error processing */
}
```

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.

- `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 Serial Port

`OL_Buffered_ConfigureSerialPort()`

Prototype : (defined in `optilogic.h`)

```
BOOL OL_Buffered_ConfigureSerialPort(NODEADDR addr, USHORT modno, BYTE type,
    enum OL_SERIAL_PORTS port,
    enum OL_SERIAL_BAUD_RATES baudrate,
    enum OL_SERIAL_DATA_BITS databits,
    enum OL_SERIAL_PARITY parity,
    enum OL_SERIAL_STOP_BITS stopbits)
```

where :

- `modno` = 0 or 1 for OL2602 or 81 for Base Comm Port
- `type` = 112 for OL2602 or 0 for Base Comm Port
- `OL_SERIAL_PORTS` = 0 (base), 1 (OL2602 - top) or 2 (OL2602- bottom)
- `OL_SERIAL_BAUD_RATES`, `OL_SERIAL_DATA_BITS`,
`OL_SERIAL_PARITY`, & `OL_SERIAL_STOP_BITS`
Use the selections listed in the `optilogic.h` header file.

Read Serial Data

OL_Buffered_ReadSerialData()

Prototype : (defined in optilogic.h)

```
BOOL OL_Buffered_ReadSerialData(NODEADDR addr, USHORT modno, BYTE type,  
    enum OL_SERIAL_PORTS port, BYTE *numbytes, BYTE *flags, BYTE *buffer)
```

where :

- modno = 0 or 1 for OL2602 or 81 for Base Comm Port
- type = 112 for OL2602 or 0 for Base Comm Port
- OL_SERIAL_PORTS = 0(base), 1(OL2602 - top), or 2(OL2602 - bottom)
- *numbytes = Pointer to the variable which will contain the # of receive bytes returned.
- *flags = Pointer to variable to store returned error flags associated with received bytes
- *buffer = Pointer to buffer to use to store received data

Write Serial Data

OL_Buffered_WriteSerialData()

Prototype : (defined in optilogic.h)

```
BOOL OL_Buffered_WriteSerialData(NODEADDR addr, USHORT modno, BYTE type,  
    enum OL_SERIAL_PORTS port, BYTE numbytes, BYTE *buffer, BYTE *left)
```

where :

- modno = 0 or 1 for OL2602 or 81 for Base Comm Port
- type = 112 for OL2602 or 0 for Base Comm Port
- OL_SERIAL_PORTS = 0(base), 1(OL2602 - top), or 2(OL2602 - bottom)
- numbytes = Number of bytes being sent
- *buffer = Pointer to the buffer containing the data to send
- *left = Pointer to the 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 of 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.

```
BYTE      PortType;
NODEADDR  address;
USHORT    slot;

slot      = 0;      /* slot 0 */
address   = 7;      /* node address = 7 */
PortType  = 0x70; /* OL2602 = 0x70 */
OL_Buffered_ConfigureSerialPort(address, slot, PortType, OL_SERIALPORT_1,
                                OL_SERIAL_BAUD_9600, OL_SERIAL_8_DATA_BITS,
                                OL_SERIAL_NO_PARITY, OL_SERIAL_1_STOP_BIT);
OL_Buffered_UpdateNodes(NodeList, NodeCount);
```

OL2602 continued

Reading Data from a Serial Port

To read data from a serial port, the routines `OL_Buffered_UpdateNodes()` and `OL_Buffered_ReadSerialData()` must be called. Both routines have to be called often enough so that data loss does not occur. If the routines are 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 interval should be set low enough to keep the data from overflowing. You should adjust the interval within the program until you get consistent results. An example of a routine used to read data from a serial port is shown below.

```
void ReceiveOL2602 (unsigned int *RxByteCount, unsigned char *RxBuffer,
                   NODEADDR address, USHORT modno, char PortNo)
{
  /* PortNo = serial port no (1 or 2) */
  /* address = node address of RTU
  /* modno = slot containing OL2602 */
  BYTE numbytes; /* number of bytes in response */
  BYTE flags;
  BYTE buffer[48]; /* return data buffer */
  int i; /* loop counter */

  OL_Buffered_UpdateNodes(NodeList, NodeCount) ;
  type = 0x70; /* OL2602 */
  OL_Buffered_ReadSerialData (address, slot, type, PortNo, &numbytes, &flags, &buffer);
  /*Buffer received data into Receive Buffer*/
  For (i = 0; i < numbytes - 1; i++)
  {
    RxBuffer[RxByteCount] = buffer[i];
    RxByteCount++;
  }
}
```

Note: You must remember to call `OL_Buffered_UpdateNodes()` often. This routine does not check to see when a message is complete. You must do that depending on your own message format.

OL2602 continued

Writing Data to a Serial Port

The following routine is an example that can be used to transmit data out the top port of an OL2602. The node address, slot no, and Port are passed into this routine, as well as the buffered data and the number of bytes to transmit.

```
void TransmitData (unsigned int TransmitLength, unsigned char *TransmitBuffer,
    NODEADDR address, USHORT modno, char PortNo)
{
    int i;
    unsigned int BytesTransmitted; /* total number of bytes that have been transmitted */
    unsigned char NumberToTransmit; /* number of bytes placed into tx buffer of OL2602*/
    unsigned char buffer[48];

    BytesTransmitted = 0; /* initialize tx byte count */
    BytesLeft = 48; /* start with Full tx buffer */

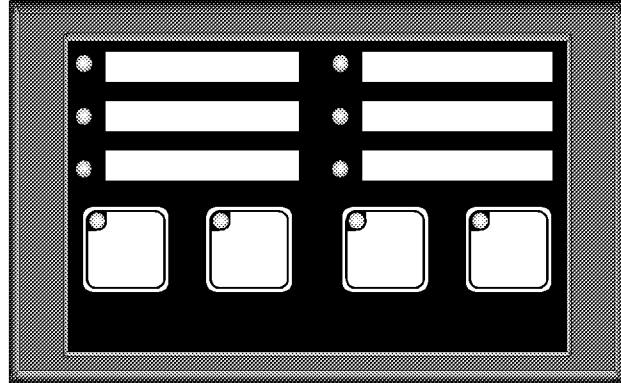
    if (TransmitLength > 48) /* if message longer than the OL2602 transmit buffer, send it
        in different pieces to prevent an overflow of the transmit buffer */
    {
        While (BytesTransmitted < TransmitLength) /* loop until entire message sent */
        {
            NumberToTransmit = Bytes Left / 2;
            /*Buffer data & increment counter*/
            For (i = 0; i < NumberToTransmit - 1; i ++)
            {
                buffer[ i ] = TransmitBuffer[BytesTransmitted];
                BytesTransmitted ++;
            }
            OL_Buffered_WriteSerialData(address, modno, 0x70, PortNo,
                NumberToTransmit, &buffer, &BytesLeft);
            OL_Buffered_UpdateNodes(NodeList, NodeCount);
        }
    }

    else /*Buffer entire message & send at same time*/
    {
        For (i = 0; i < TransmitLength - 1; i ++)
            buffer[ i ] = TransmitBuffer[ i ];
        OL_Buffered_WriteSerialData(address, modno, 0x70, PortNo,
            TransmitLength, &buffer, &BytesLeft);
        OL_Buffered_UpdateNodes(NodeList, NodeCount);
    }
}
```

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 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 optilogic.h)

```
BOOL OL_Buffered_OL3406_StatusControl(NODEADDR addr, BYTE *data,
                                       BYTE *pbstatus)
```

where : data[4] is an array with the following definition

data[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.
data[1] =	on/off control of indicator light LEDs. Bits are in sequence starting at bit 0.
data[2] =	flash control of 4 inset button LEDs. LED must be on to flash. Bits are in sequence starting at bit 0.
data[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.

OL3406 continued

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. Conversely, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in optilogic.h)

```
BOOL OL_Buffered_OL3406_ForceButtons(NODEADDR addr,
    enum OL34xx_BUTTON_FLAGS, BYTE ButtonData)
```

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

OL34xx_BUTTON_FLAGS defines the type of force as follows -

- = OL34xx_BUTTON_ALL ; force ALL buttons to a state matching ButtonData (force state)
- = OL34xx_BUTTON_OR ; logically OR current button state with ButtonData (force on)
- = OL34xx_BUTTON_AND ; logically AND current button state with the complement of ButtonData (force off)

Configure OL3406 Panel**OL_Buffered_OL3406_ConfigurePanel()****Prototype : (defined in optilogic.h)**

```
BOOL OL_Buffered_OL3406_ConfigurePanel(NODEADDR addr, BYTE ConFlags)
```

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 optilogic.h)**

```
BOOL OL_Buffered_OL3406_ReadPanelConfiguration(NODEADDR addr, BYTE *ConFlags)
```

where : ConFlags is defined above

OL3406 continued

Usage :

The following example sets the panel configuration. It then goes into a loop. Whenever the first button is sensed active, it lights indicator light 3, flashes light 4, and performs an application process. At the end of the process, it forces the first button off.

```

BYTE   req_config ;           /* Required configuration */
BYTE   litestat[4] ;          /* Light state array */
BYTE   butt_stat ;            /* Button status */
NODEADDR address ;            /* Node address */

req_config = 0x07 ;           /* Required configuration = buttons 1 - 3 alternate action
                                button 4 momentary, no LED separation */
litestat[0] = 0 ;              /* Start with all lights out */
litestat[1] = 0 ;
litestat[2] = 0 ;
litestat[3] = 0 ;
butt_stat = 0 ;

OL_Buffered_OL3406_ConfigurePanel(address, req_config) ; /* Configure Panel */

while (1)
{
    /* Forever loop */
    /* Is the first button active ? */
    if ((butt_stat & 0x01) && !fun_active)
    {
        /* If so, turn light 3 on & flash light 4 */
        litestat[1] | 0x0C ;
        litestat[3] | 0x08 ;
        fun_active = TRUE ;
        fun_done = FALSE ;
    }
    if (fun_active)
    {
        /* Do function processing */
        if (fun_done)
        {
            /* Clear the first button */
            OL_Buffered_OL3406_ForceButtons(address, OL34XX_BUTTON_AND, 0x01) ;
            /* when button clears, reset flag */
            if (!(butt_stat & 0x01))
            {
                fun_active = FALSE ;
            }
        }
    }

    /* Once in the loop Read & Update Status & Update the Node */
    OL_Buffered_OL3406_StatusControl(address, &litestat, &butt_stat) ;
    OL_Buffered_UpdateNodes(NodeList, NodeCount) ;
}

```

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 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 `optilogic.h`)

```
BOOL OL_Buffered_OL3420_StatusRequest(NODEADDR addr, BYTE LEDState,
                                       BYTE *Status)
```

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), must be on to flash
	Status -	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_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. Conversely, all alternate-action buttons can be forced to a state matching the pattern sent (force state).

Prototype : (defined in optilogic.h)

```
BOOL OL_Buffered_OL3420_ForceButtons(NODEADDR addr,  
    enum OL34xx_BUTTON_FLAGS, BYTE ButtonData)
```

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

OL34xx_BUTTON_FLAGS defines the type of force as follows -

- = OL34xx_BUTTON_ALL ; force ALL buttons to a state matching ButtonData (force state)
- = OL34xx_BUTTON_OR ; logically OR current button state with ButtonData (force on)
- = OL34xx_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 optilogic.h)**

```
BOOL OL_Buffered_OL3420_SendTextDisplayMessage(NODEADDR addr, BYTE line,  
    char *message)
```

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

OL3420 continued

Configure OL3420 Terminal

OL_Buffered_OL3420_ConfigurePanel()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3420_ConfigurePanel(NODEADDR addr, BYTE buttonstate)

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 optilogic.h)

BOOL OL_Buffered_OL3420_ReadConfiguration(NODEADDR addr, BYTE *buttonstate)

where : buttonstate is defined above

OL3420 continued

Usage :

The following example configures the panel, 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.

```
#include <string.h> /* this file should be included in the program so the strcpy() function can
                    be used */

BYTE req_config ;          /* Required configuration */
BYTE litestat ;            /* Inset LED state */
BYTE lineno ;              /* Display line number */
BYTE butt_stat ;           /* Button status */
NODEADDR address ;         /* Node address */
char msgstring[21] ;       /* message text buffer */

req_config = 0x07 ;        /* Required configuration = buttons 1 - 3 alternate action
                             button 4 momentary, no LED separation */
litestat = 0 ;             /* Start with all lights out (unused because LED separation disabled)*/
butt_stat = 0 ;

OL_Buffered_OL3420_ConfigurePanel(address, req_config) ;

while (1)
{
    /* Forever loop */
    /* Is the second button active ? */
    if ((butt_stat & 0x02) && !fun_active)
    {
        /* If so, send message to the display */
        lineno = 0 ;
        strcpy(msgstring, " Process Active ") ;
        OL_Buffered_OL3420_SendTextDisplayMessage(address, lineno, &msgstring) ;
        fun_active = TRUE ;
        fun_done = FALSE ;
    }
    if (fun_active)
    {
        /* Do function processing */

        if (fun_done)
        {
            /* Clear the second button */
            OL_Buffered_OL3420_ForceButtons(address, OL34XX_BUTTON_AND, 0x02) ;
            if (!(butt_stat & 0x02))
            {
                fun_active = FALSE ;
            }
        }
    }

    /* Once in the loop */
    OL_Buffered_OL3420_StatusControl(address, litestat, &butt_stat) ;
    OL_Buffered_UpdateNodes(NodeList, NodeCount) ;
}
```

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 optilogic.h)

```
BOOL OL_Buffered_OL3440_SendTextDisplayMessage(NODEADDR addr, BYTE line,  
                                                char *message)
```

where : line is line number (0 - 3 from top to bottom)
message is 20 characters of text

Usage:

```
#include <string.h> /* this file should be included in the program so the strcpy() function can  
                    be used */
```

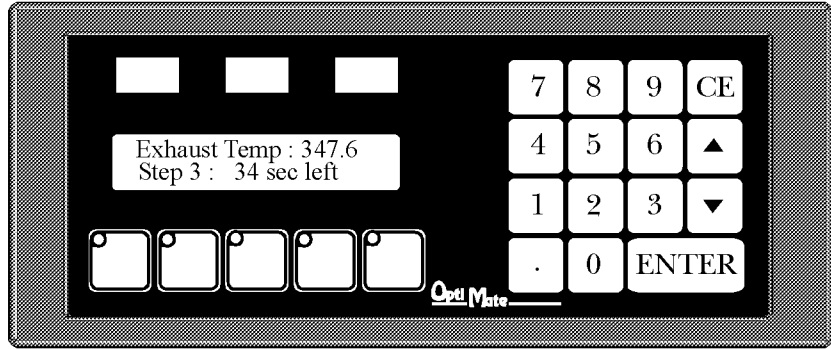
```
char msgstring[21]; /* message text buffer */
```

```
strcpy(msgstring, "Text message .. Abcd");  
OL_Buffered_OL3440_SendTextDisplayMessage(nodeaddr, lineno, &msgstring);
```

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 function calls are available for the OL3850 operator terminal

- OL_Buffered_OL3850_StatusRequest() - Control lights, get button status and retrieve entered data
- OL_Buffered_OL3850_SendTextDisplayMessage() - Send a text line 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

Send Light Controls and Read Status information

OL_Buffered_OL3850_StatusRequest()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3850_StatusRequest(NODEADDR addr, BYTE LEDState, BYTE LEDFlash, BYTE *Status, double *ReturnDATA)

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)
LEDFlash -	for all LEDs and light bars, “1” - on , “0” - off flash controls for the same LEDs and light bars mentioned for LEDState. To flash, the light must be on.
Status -	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” indicated 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 stored in this variable.

Send a Text Message to the Display

OL_Buffered_OL3850_SendTextDisplayMessage()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3850_SendTextDisplayMessage(NODEADDR addr, BYTE line, char *message)

where :

line	is the line number (0 = top, 1 = bottom) to send text to
message	is message text (20 character ASCII string)

OL3850 continued

Configure OL3850 Function Buttons

OL_Buffered_OL3850_ConfigurePanel()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3850_ConfigurePanel(NODEADDR addr, BYTE buttonstate)

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 optilogic.h)

BOOL OL_Buffered_OL3850_ReadConfiguration(NODEADDR addr, BYTE *buttonstate)

where : buttonstate is defined above

Force Alternate-Action Button Status

OL_Buffered_OL3850_ForceButtons()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3850_ForceButtons(NODEADDR addr,
 enumOL34xx_BUTTON_FLAGS, BYTE ButtonData)

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

OL34xx_BUTTON_FLAGS defines the type of force as follows -

- = OL34xx_BUTTON_ALL ; force ALL buttons to a state matching ButtonData (force state)
- = OL34xx_BUTTON_OR ; logically OR current button state with ButtonData (force on)
- = OL34xx_BUTTON_AND ; logically AND current button state with the complement
 of ButtonData (force off)

OL3850 continued

Send Keypad Data Entry Message

OL_Buffered_OL3850_SendKeypadMessage()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3850_SendKeypadMessage(NODEADDR addr, BYTE line,
char *message, double StartDATA, BYTE decimalPoint)

where : message- Text message to display. (20 characters of ASCII text) Carets “^” should be used as place holders for the data entry field.

 StartDATA- Initial 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 StartDATA is passed as a double. The decimalPoint is the number of digits after the decimal. In other words, if StartDATA = 4567 and decimalPoint = 2, the starting value is 45.67.

Send Arrow Adjust Message

OL_Buffered_OL3850_SendArrowMessage()

Prototype : (defined in optilogic.h)

BOOL OL_Buffered_OL3850_SendArrowMessage(NODEADDR addr, BYTE line,
char *message, unsigned long *NumDATA, BYTE decimalPoint,
unsigned long LowLimit, unsigned long HighLimit)

where : message- Text message to display. Carets “^” should be used as place holders for the data entry field

 NumDATA- Initial arrow adjust value

 decimalPoint- A beginning value must be passed, which the user can adjust using the arrow key. The numeric NumDATA is passed as a pointer to a long. The dp (decimal point) is the number of digits after the decimal. In other words, if NumDATA = 4567 and dp = 2, the starting value is 45.67.

 LowLimit - The minimum value allowed for adjustment (the same dp value applies)

 HighLimit - The maximum value allowed for adjustment (the same dp value applies)

For example, if NumDATA = 4567, dp = 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.

```
#Include <string.h> /* This file should be included in the program so the strcpy() function can be
                        used */

Void Handle_OL3850(unsigned char LEDState, unsigned char LEDFlash, char KeypadActive,
                  char ArrowActive, char ForceActive, double KeypadData, unsigned long ArrowData)
{
/*+++++
the node address for this example is 30!!

    LEDState - lamp and LED ON/OFF state
    LEDFlash - lamp and LED flash state
    KeypadActive - flag denoting that a keypad message is active on the LCD
    ArrowActive - flag denoting that an arrow adjust message is active on the LCD
    KeypadData - value of a keypad entry message as passed from the panel
    ArrowData - value of an arrow adjust message as passed from the panel
+++++*/

unsigned char config ; /* configuration data read from panel */
unsigned char RequiredConfig ; /* Required configuration*/
unsigned char ButtonStatus ; /* Button status read back from panel*/
char message[21] ; /* array containing message string to send to the panel */
double ReturnValue ; /* current data entry value returned from the panel */
int i ; /* loop counter */

RequiredConfig = 0x19 ; /* buttons 1, 4 , 5 alternate, 2 and 3 momentary */

/* read the configuration from the panel */
OL_Buffered_OL3850_ReadConfiguration(30, config) ;
/*if the configuration read from the RTU doesn't match the required config, send the
configuration message */
if (RequiredConfig != config)
    OL_Buffered_OL3850_ConfigurePanel(30, RequiredConfig) ;

/* if button 3 is pressed */
if (ButtonStatus & 0x04)
{
    /* send text message to top line */
    strcpy(message, "Key Data,Press ENTER") ;
    OL_Buffered_OL3850_SendTextDisplayMessage(30, 0, &message) ;
    /* send keypad entry message to bottom line, initial data and decimal pt are both 0 */
    for (i = 0; i < 20 ; i++)
    {
        message[i] = 0x20 ; /* fill message buffer with spaces */
    }
    strcpy(message, "Keypad Entry ^^^^^") ;
    OL_Buffered_OL3850_SendKeypadMessage(30, 1, &message, 0, 0) ;
    KeypadActive = 1 ; /* set the keypad message active flag */
}
}
```



```

/* if a keypad message is active and the data available bit (the most significant bit of the button
   status) is active, keypad entry complete */
else if ((KeypadActive == 1) && (ButtonStatus & 0x80))
{
    KeypadData = ReturnValue ; /* store keypad entry data to pass to calling routine */

    /* send text message to blank top line of LCD */
    for (i = 0 ; i < 20 ; i++)
    {
        message[i] = 0x20 ; /* fill message buffer with spaces */
    }
    OL_Buffered_OL3850_SendTextDisplayMessage(30, 0, &message) ;

    /* send text message to bottom line of LCD */
    for (i = 0 ; i < 20 ; i++)
    {
        message[i] = 0x20 ; /* fill message buffer with spaces */
    }
    strcpy(message, "Keypad Entry Done") ;
    OL_Buffered_OL3850_SendTextDisplayMessage(30, 1, &message) ;
    KeypadActive = 0 ; /* reset keypad active flag */
}

/* if button 4 is active and force active flag inactive */
else if ((ButtonStatus & 0x08) && (ForceActive == 0))
{
    /* if arrow adjust message not active, send it */
    if (ArrowActive == 0)
    {
        /* send text message to top line */
        for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }
        strcpy(message, "Press F2 When Done") ;
        OL_Buffered_OL3850_SendTextDisplayMessage(30, 0, &message) ;

        /* send arrow adjust message to bottom line, initial value is 3686.75, range
           is 3000.00 - 4000.00 */
        for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }

        strcpy(message, "Adjust Value ^^^^^^") ;
        OL_Buffered_OL3850_SendArrowMessage(30, 1, &message, 368675, 2,
                                             300000, 400000);
        ArrowActive = 1 ; /* set the arrow adjust message active flag */
    }
}

```

```

/* if arrow adjust message is active, read data */
else if (ArrowActive == 1)
{
    ArrowData = ReturnValue ; /* copy data to variable to return to calling routine so
                                data can be updated as it changes */
    /* if button 2 is pressed, reset arrow message */
    if (ButtonStatus & 0x02)
    {
        /* send text message to blank top line of LCD */
        for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }
        OL_Buffered_OL3850_SendTextDisplayMessage(30, 0,
            &message) ;

        /* send text message to bottom line of LCD */
        for (i = 0 ; i < 20 ; i++)
        {
            message[i] = 0x20 ; /* fill message buffer with spaces */
        }

        strcpy(message, "Arrow Adjust Done") ;
        OL_Buffered_OL3850_SendTextDisplayMessage(30, 1,
            &message) ;

        /* Force Button 4 OFF */
        OL_Buffered_OL3850_ForceButtons(30,
            OL_34XX_BUTTON_AND, 0x08) ;

        ArrowActive = 0 ; /* reset keypad active flag */
        ForceActive = 1 ; /* set force active flag to temporarily lock out the
                            arrow adjust message */
    }
}

/* else if button 4 cleared and force active flag set, disable force active flag */
else if ((!(ButtonStatus & 0x08)) && (ForceActive == 1))
{
    ForceActive = 0 ; /* reset force active flag */
}

/* update and read the general status of the 3850 */
OL_Buffered_OL3850_StatusRequest(30, LEDState, LEDFlash, &ButtonStatus, &ReturnValue);
OL_Buffered_UpdateNodes(NodeList, NodeCount) ;
}

```

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. `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 and 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 `optilogic.h`)

```
void    OL_NetworkRetryCount (int nRetryCount)
```

where : `nRetryCount` is the number of times to retry

Usage:

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

```
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 and works in conjunction with OL_NetworkRetryCount (see above).

Prototype : (defined in optilogic.h)

void OL_NetworkTimeout (LONG timeout)

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.

```
OL_NetworkTimeout(100) ;          /* sets the timeout value for a message to 100ms.*/
```

Check to see if 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 optilogic.h)

BOOL OL_IsNetworkInitialized (void)

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.

```
if (!OL_IsNetworkInitialized())  
{  
    /* Initialize Network */  
}
```

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 it was last reset. It can be useful in determining communication problems within a network.

Prototype: (defined in optilogic.h)

ULONG OL_GetNodeRetryCount (NODEADDR addr)

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 optilogic.h)

BOOL OL_ResetNodeRetryCount (NODEADDR addr)

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.

```
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 optilogic.h)

ULONG OL_GetLastErrorCode(void)

Usage:

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

unsigned long ErrorCode ;

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 optilogic.h)

const char* OL_GetLastErrorCodeString (void)

Usage:

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

char *ErrorCodeString ;

ErrorCodeString = 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 optilogic.h)

ULONG OL_Buffered_GetErrorCode (NODEADDR addr, BYTE modno)

Usage:

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

```
unsigned long  ErrorCode ;
int  status ;
int i ;
int NodeList[1] ;

NodeList[0] = 30 ;
status = OL_Buffered_UpdateNodes(NodeList, 1) ;
/* if call to update nodes returned an error, find error code */
if (status != 1)
{
    /* since using a four slot base, loop 4 times to check each card */
    for (i = 0; i < 4; i++)
    {
        ErrorCode = OL_Buffered_GetErrorCode (30, i) ;
        if (ErrorCode != 0)
        {
            /* handle error condition */
        }
    }
}
```

Get the Last Network Error Code String OL_Buffered_GetErrorCString()

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 optilogic.h)

```
const char* OL_Buffered_GetErrorCString (NODEADDR addr, BYTE modno)
```

Usage:

The following call will check for an error code string in each card plugged into node 30.

```
unsigned long ErrorCode ;
char *ErrorCodeString ;
int i, status ;
int NodeList[1] ;

NodeList[0] = 30 ;
status = OL_Buffered_UpdateNodes(NodeList, 1) ;
/* if call to update nodes returned an error, find error code string */
if (status != 1)
{
    /* since using a four slot base, loop 4 times to check each card */
    for (i = 0, i < 4; i++)
    {
        ErrorCode = OL_Buffered_GetErrorCode (30, i) ;
        if (ErrorCode != 0)
        {
            /* handle error condition */
            ErrorCodeString = OL_Buffered_GetErrorCString(30, i) ;
            /* make ErrorCodeString visible to operator */
        }
    }
}
```

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.”
40216	“Digital Input type does not exist.”
40217	“Digital Output type does not exist”
40218	“Analog Input type does not exist.”
40219	“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.”