Programmable Industrial Control
Using The NCL Programming Language
Telemetry And Control System Engineering Series

Version 2.06
03 March 2013

Navionics Research, Inc.
Saint Louis, Missouri USA
wireless-telemetry.com

TABLE OF CONTENTS

1 INTRODUCTION

In 1995, Navionics Research introduced the **WiSTAR** Network, an acronym derived from **Wi**reless **S**ystem **T**elemetry **A**nd **R**emote-Control. This product was designed to solve the problems posed by the complex distributed control and monitoring requirements of the rural water and wastewater industries. Early in the development stages, it became apparent that the WiSTAR RTU should support a control language which offered the flexibility of field programming and interactive debugging. This meant that, in addition to its wireless communication and telemetry functions, the WiSTAR RTU should offer the power and flexibility of programmable industrial control logic. Furthermore, because the control decisions of rural water systems are optimally made across the wireless link, the control language would be most effective if it contained a library of functions which specifically address inter-site control, data-sharing, and radio-link status evaluation. As a result of these demanding requirements, **NCL**, an acronym derived from **N**etwork **C**ontrol **L**anguage, was developed. It is offered as Navionics Research's open control language with a focus on solving difficult distributed wireless control problems.

This reference and tutorial is designed for electricians, technicians, and engineers who wish to learn the art and science of NCL programming. It is assumed that the reader already has a modest amount of programming experience in a language such as C++, BASIC, FORTRAN, Assembly Language, or PLC (Programmable Logic Controller) Relay Ladder Logic.

Although NCL is a very simple programming language, it can be used to develop sophisticated applications. In structure, the language contains many familiar features, which have been borrowed from other high-level programming languages. At the same time, Navionics' implementation of the integrated compiler/interpreter/debugger, which is a component of Navionics Research's RTU firmware, is lean and compact.

The design of the NCL programming language has been focused on programming simplicity in every possible instance. In the C++ programming language, variables and registers are must be predefined by the programmer as belonging to a certain data type. These data types can be either characters (1 byte), short-integers (2 bytes), long-integers (4 bytes), floating-points (4 bytes), or double-precision floating-points (8 bytes). NCL, on the other hand, uses a data stack whose members are all defined as the same data type – 8-byte double-precision floating-point numbers. With this generalized, "over-precision" data stack, the programmer conveniently does not need to define variable data types. And as long as floating-point data types do not exceed 8-byte precision and integer data types do not exceed 4-byte precision, conversion and rounding errors are eliminated.

The inter-site control functionality of NCL is accomplished through "status-sharing". Every RTU maintains its own status-file, which contains a description of its status. It is up to the NCL programmer to decide what to put in this status-file. The wireless networking protocol ensures that each site contains updated copies of the status-files of the other network RTU's. Therefore, an RTU can incorporate the status of another RTU site (or sites) into its control decisions. For example, a pump station may primarily turn its pumps ON and OFF based upon what a remote water tower desires (as conveyed through the sharing of its status-file). However, if the wireless link fails, then the pump station will not be able to receive an up-to-date copy of the water tower's status-file. In this case, the programmer may wish to turn the pumps OFF, or he may wish to fail over the pump control into another mode of operation, such as ON/OFF control based upon local-pressure readings, timer, or an external control device.

In addition to the status-file, each site also holds a setpoints-file, which contains a description of its setpoints. Again, it is up to the NCL programmer to decide what to put into this setpoints-file. The wireless networking protocol contains provisions that enable a system operator to request, view, and modify the setpoints of any remote RTU from any RTU in the system. This functionality is at the very core of a WiSTAR system's powerful capabilities. In the example case of this manual, it will be demonstrated how setpoint modifications can (and should) be used to implement remote-control capabilities. As the NCL programmer, you will be tasked with deciding which parameters will be "hard-wired" into the program, and which will be designated as customer-modifiable setpoints. However, keep in mind that Navionics Research's NCL programming philosophy strongly suggests that you should provide the customer with an abundance of setpoints. The philosophy is simple: ***When in doubt as to whether a number should be "hard-wired", make it a setpoint, and let the customer decide where to set it.***

Navionics Research has created a single, multitasking executable; and this program is embedded within an IBM-compatible CPU. This program is named: ***WINCOM.EXE***. *WINCOM* services three (3) concurrent processes: communication, control, and the user-interface.

With the single-executable approach, the overall number of hardware components within the system is reduced; and greater reliability is achieved. However, with this approach, it is also imperative that adequate safeguards be constructed around the NCL control process to protect the operations of the communication and user-interface functions. Toward achieving this purpose, a software "firewall" has been built around the NCL interpreter to isolate the effects of any stray NCL programming errors. Although such errors are not acceptable, the "firewall" traps NCL coding errors, and prevents them from corrupting the concurrent communication and user-interface operations. At the same time, the firewall provides assistance to the programmer in locating certain NCL errors.

As with any language, the best way to learn programming is to program. And therefore, the techniques taught in this document are based upon an actual program, which is in operation in the "Village Of Walnut Hill (IL) Public Water District". Although the example is a relatively short program, it exercises many of the most important aspects of NCL. This program contains logic to control a pump station to the following specifications:

1. Control and monitor one (1) pump based upon one of the following modes of operation:

- **a.** Water tower control (customer-modifiable setpoints at the water tower)
- **b.** Local pressure (customer-modifiable pressure setpoints)
- **c.** Local timer (customer-modifiable time period setpoints)
- **d.** Manual override (customer selectable ON or OFF)

2. Enable the pump station to automatically fail over to a local-pressure-mode operation in the case of: (1.) a communication failure with the water tower, or (2.) a pressure transducer failure at the water tower.

3. Monitor 3-phase power status, and force pump OFF in the case of a phase fault.

4. Monitor suction and discharge pressures.

5. Automatically turn OFF the pump in case of low suction pressure.

6. Automatically turn ON the pump in case of low discharge pressure.

7. Monitor the discharge and suction pressure transducers for failures.

8. Monitor station for flood condition, and force pump OFF in the case of a flood.

9. Monitor room temperature and pump motor bearing temperature differential.

10. Accumulate pump runtime.

2 THE “LOGIC.NPP” FILE

NCL programs are contained in a single file, and they are always given the same name: **LOGIC.NPP** (the NCL program file). The extension **".NPP"** stands for: **N**CL **P**lus **P**lus. The file is stored as a simple ASCII text file, and therefore it can be produced using a text editor. Windows “Notepad” and Windows “Wordpad” are both examples of appropriate text editors.

The “LOGIC.NPP” file is logically divided into two parts: The “Header” and the “Control Logic”.

The “Header” contains setup information, network definitions, and the aliases of the memory registers and I/O modules. For example, the alias of "Solid-State-Relay #1" may be defined as "Pump Starter Relay". The use of aliases, rather than register addresses and I/O module-numbers makes programs more readable and easier to debug. Also, the use of aliases makes programs easier to port to different clients. Although strictly optional, their use is highly encouraged. Also, the first part of the “Header” section is strictly formatted. In other words, blank lines cannot be inserted where they are not expected; and all fields must be filled in according to the specifications set forth in this document. The second part of the “Header” allows an unformatted structure, and also allows for programmer comments.

The “Control Logic”, which is typically much larger than the “Header”, contains sequences of commands which describe the control decision-making process. The “Control Logic” must contain at least one main subroutine (always named "main"); and "main" may call other subroutines (which may also call subroutines, and so on…).

Throughout the example program, there are numerous embedded comments. It is good programming practice to document your program with comments for several reasons. First, it makes your program easier for others to understand; and second, it will make your job easier when you are trying to modify one of your old programs months (or years) later. Comments are delimited with the '#' character. Any text after and including the "#' character is ignored by the NCL compiler. Also, to make your programs more readable, blank lines may be freely used to separate functional blocks of code in your "LOGIC.NPP" file.

LOGIC.NPP File Structure:

```
$HEADER

<All Formatted “Header” Information Here>

<All Unformatted “Header” Information Here>

$NCL

<All Unformatted “Control Logic” Here>
```

3 "LOGIC.NPP" FILE HEADER STRUCTURE

The header of the "LOGIC.NPP" file contains two sections. The first section (blocked off in RED) contains setup information and network parameters. This first section must be written to a strict format as defined in this tutorial. The second section (blocked off in BLUE) contains alias definitions, and the programmer may create this section to a more relaxed format. Let's analyze the header file of our example program:

```
# Header (Setup) Info: UPS Station at Walnut Hill, IL
   1     # Number of Digital Setpoints
  16     # Number of Analog  Setpoints
   2     # Number of Integer Setpoints
   4     # Number of Digital Input  Modules
   4     # Number of Analog  Input  Modules
   0     # Number of Integer Input  Modules
  16     # Number of Digital Flag   States
   4     # Number of Analog  Flag   States
   2     # Number of Integer Flag   States
   1     # Number of Relay   Output Modules
   0     # Number of Analog  Output Modules
# Remote Setup Information ... (No Blank Lines Allowed...)
   1     # Number of Dependent Sites (Dependent Sites Follow)
   001      # Index 0 (Zero): Walnut Hill Water Tower
```

```
# Variable Name Definitions ... (Blank Lines Allowed...)

ALIAS   FAILOVER_TO_PRESSURE_MODE            LDS    0

ALIAS   LOW_SUCTION_CUTOUT_PSI               LAS    0
ALIAS   LOW_SUCTION_RELEASE_PSI              LAS    1
ALIAS   HIGH_SUCTION_CUTIN_PSI               LAS    2
ALIAS   HIGH_SUCTION_RELEASE_PSI             LAS    3
ALIAS   HIGH_DISCHARGE_CUTOUT_PSI            LAS    4
ALIAS   HIGH_DISCHARGE_RELEASE_PSI           LAS    5
ALIAS   LOW_DISCHARGE_CUTIN_PSI              LAS    6
ALIAS   LOW_DISCHARGE_RELEASE_PSI            LAS    7

ALIAS   PRESSURE_MODE_PUMP_OFF_PSI           LAS    8
ALIAS   PRESSURE_MODE_PUMP_ON_PSI            LAS    9

ALIAS   TIMER_1_START_HOUR                   LAS    10
ALIAS   TIMER_1_STOP_HOUR                    LAS    11
ALIAS   TIMER_2_START_HOUR                   LAS    12
ALIAS   TIMER_2_STOP_HOUR                    LAS    13
ALIAS   TIMER_3_START_HOUR                   LAS    14
ALIAS   TIMER_3_STOP_HOUR                    LAS    15

ALIAS   PUMP_MODE{AUTO-ON-OFF}               LIS    0
ALIAS   STATION_MODE{RADIO-PRESSURE-TIMER}   LIS    1

ALIAS   POWER_MODULE                         LDM    0
ALIAS   PUMP_POSITIVE_INDICATOR_MODULE       LDM    1
ALIAS   PHASE_FAULT_DETECT_MODULE            LDM    2
ALIAS   FLOOD_DETECT_MODULE                  LDM    3

ALIAS   DISCHARGE_PSI_MODULE                 LAM    0
ALIAS   SUCTION_PSI_MODULE                   LAM    1
ALIAS   PUMP_TEMP_DEGF_MODULE                LAM    2
ALIAS   PUMP_ROOM_TEMP_DEGF_MODULE           LAM    3
```

```
ALIAS   DISCHARGE_WORKING                    LAMV   0
ALIAS   SUCTION_WORKING                      LAMV   1

DISPL   POWER_ON                             LDF    0
DISPL   PUMP_RELAY                           LDF    1
DISPL   PUMP_ON                              LDF    2
DISPL   PHASE_FAULT_DETECT                   LDF    3
DISPL   FLOOD_DETECT                         LDF    4
DISPL   PUMP_FAIL                            LDF    5
DISPL   COMM_FAILURE                         LDF    6
DISPL   RADIO_MODE                           LDF    7
DISPL   PRESSURE_MODE                        LDF    8
DISPL   TIMER_MODE                           LDF    9
DISPL   LOW_SUCTION_CUTOUT                   LDF   10
DISPL   HIGH_SUCTION_CUTIN                   LDF   11
DISPL   HIGH_DISCHARGE_CUTOUT                LDF   12
DISPL   LOW_DISCHARGE_CUTIN                  LDF   13
DISPL   DISCHARGE_TRANSDUCER_FAIL            LDF   14
DISPL   SUCTION_TRANSDUCER_FAIL              LDF   15

DISPL   DISCHARGE_PSI                        LAF    0
DISPL   SUCTION_PSI                          LAF    1
DISPL   PUMP_TEMP_DEGF                       LAF    2
DISPL   PUMP_ROOM_TEMP_DEGF                  LAF    3

DISPL   UP_TIME_MIN                          LIF    0
DISPL   PUMP_RUNTIME_MIN                     LIF    1

ALIAS   PUMP_SSR                             LDR    0

DISPL   COMM_TO_WATER_TOWER                  VLD    0
DISPL   TOWER_LEVEL_FT                       RAF    0   0
DISPL   TOWER_CALL_PUMP                      RDF    0   2
DISPL   TOWER_TRANSDUCER_FAIL                RDF    0   6

ALIAS   LOW_SUCTION_TIMER                    TMR    0
ALIAS   LOW_SUCTION_OK_TIMER                 TMR    1
ALIAS   HIGH_DISCHARGE_TIMER                 TMR    2
ALIAS   HIGH_DISCHARGE_OK_TIMER              TMR    3
ALIAS   HIGH_SUCTION_TIMER                   TMR    4
ALIAS   HIGH_SUCTION_OK_TIMER                TMR    5
ALIAS   LOW_DISCHARGE_TIMER                  TMR    6
ALIAS   LOW_DISCHARGE_OK_TIMER               TMR    7
ALIAS   PHASES_OK_TIMER                      TMR    8
ALIAS   POWER_OK_TIMER                       TMR    9
ALIAS   FLOOD_OK_TIMER                       TMR    10
ALIAS   PUMP_FAIL_TIMER                      TMR    11

ALIAS   PUMP_RUNTIME_SECS                    USR    0
ALIAS   LASTCALL_TIME                        USR    1
ALIAS   DELTA_TIME                           USR    2
ALIAS   AOK                                  USR    3
ALIAS   EMERGENCY_CUTIN                      USR    4
ALIAS   PRESSURE_PUMP                        USR    5
ALIAS   TIMER_PUMP                           USR    6
ALIAS   TOWER_PUMP                           USR    7
ALIAS   LOCAL_PUMP                           USR    8
```

```
$NCL

(CONTROL LOGIC FOLLOWS...)
```

Header Deconstruction:

Line Number	Content and Purpose
1	Comment Line
2	Define the number of **Digital Setpoints** (Setpoints which are defined as either a one (1) or a zero (0)). Stored in LDS registers.
3	Define the number of **Analog Setpoints** (Setpoints which are defined as a 4-byte floating point.) Stored in LAS registers.
4	Define the number of **Integer Setpoints** (Setpoints which are defined as one of a small selection of choices - also referred to as **Radiobutton Setpoints**). Stored in LAS registers.
5	Define the number of **Digital Input Modules** (Modules which read either a TRUE or FALSE state, such as power-detect or flood-detect). Stored in LDM registers.
6	Define the number of **Analog Input Modules** (Modules which read an analog level such as pressure, temperature, or chlorination). Stored in LAM registers.
7	Define the number of **Integer Input Modules** (Modules which count events, such as those generated by the outputs of high-speed pickup meters. These modules automatically calculate totalization and rate information. The maximum integer reading is 999,999,999 after which the integer "rolls-over" to zero). Stored in LIM registers.
8	Define the number of **Digital Flag States** (Digital states which are to be placed in this RTU's status-file). Stored in LDF registers.
9	Define the number of **Analog Flag States** (Analog states which are to be placed in this RTU's status-file). Stored in LAF registers.
10	Define the number of **Integer Flag States** (Integer states which are to be placed in this RTU's status-file). Stored in LIF registers.
11	Define the number of **Relay Output Modules** (SSR's or Solid-State Relays. Writing a zero (0) to a SSR opens the contact, and writing a one (1) to a SSR closes the contact. Stored in LDR registers.
12	Define the number of **Analog Output Modules**. Stored in LAOM registers.
13	Comment line. You may put anything on this line that you wish.
14	Define the **Number of Dependent Sites** (whose status-files will be integrated into the control decisions of this RTU.
Next Lines...	**Network Addresses** of Dependent Sites are placed on the subsequent lines if the Number of Dependent Sites is greater than zero (0). The first site will be known as Dependent Site #0, the second site will be Dependent Site #1, etc. …
Next Line	Comment line. You may put anything on this line that you wish.
Next Line(s)...	Alias definitions of the registers, input modules, and relays.

Alias Variable Declarations:

When creating an alias, the following syntax is used:

ALIAS *AliasName* *RegisterName*

The AliasName must be different from all of the acceptable RegisterNames, and the AliasName must be between 1 and 48 characters long. An AliasName may contain alphabetic characters, numbers, brackets, parentheses, underscores, and dashes. However, the first character of an AliasName cannot be a number. The summary of available Register Names is given in the next chapter.

Legal Characters For AliasNames: **A–Z** , **a–z** , **0–9** , **_** , **–** , **{** , **}** , **[** , **]** , **(** , **)**

Radiobutton Setpoint – Standard Naming Conventions:

When creating a Radiobutton Setpoint (A setpoint which represents an integer selector), it is highly-recommended that the NCL programmer use the following naming convention:

ALIAS SETTINGNAME{SELECT_1–SELECT_2–SELECT_3} LIS n

An example:

ALIAS PUMP_1{AUTO – ON – OFF} LIS 0

In this example, each of the three possible numeric settings corresponds to one of the selectors. The selectors are contained within curly brackets, and are separated by dashes.

Numeric Setting	Action
1	AUTO
2	ON
3	OFF

By following these naming convention guidelines, the programmer will benefit from the automatic detection and configuration capabilities built into the Navionics Research GUI (Graphical User Interface) software; and therefore simplify the setup of the GUI.

Realtime Display Customized Configuration:

Note that certain Aliases are declared using the “DISPL” identifier, rather than the “ALIAS” identifier. Aliases that are declared in this manner will be placed in the “Realtime Display” of the *WINCOM* program. The compiler will automatically decide whether the a DISPL register should be displayed as a digital, analog, or counter register. If you wish to override the compiler defaults, then replace DISPL explicitly with DISPL_D (digital), DISPL_A (analog), or DISPL_C (counter).

The following syntax describes the use of the DISPL identifier:

DISPL	*AliasName*	*RegisterName*
DISPL_D	*AliasName*	*RegisterName*
DISPL_A	*AliasName*	*RegisterName*
DISPL_C	*AliasName*	*RegisterName*

Advanced Serial Display Customized Configuration:

The WiSTAR RTU supports a Serial Display (eg VFD420 by SEETRON) on its Terminal Port. Variables whose DISPL parameter are preceded with an ‘S’ will be displayed on the Serial Display:

SDISPL	*AliasName*	*RegisterName*
SDISPL_D	*AliasName*	*RegisterName*
SDISPL_A	*AliasName*	*RegisterName*
SDISPL_C	*AliasName*	*RegisterName*

Advanced Alarm Display Customized Configuration:

For Color Terminals attached to the Terminal Port, the WiSTAR RTU can display discrete states in color to denote an alarm state:

(S)DISPL_D1	ON=GREEN	OFF=BLANK
(S)DISPL_D1	ON=RED/BLINKING	OFF=BLANK
(S)DISPL_D1	ON=GREEN	OFF=RED/BLINKING
(S)DISPL_D1	ON=GREEN/BLINKING	OFF=BLANK

In order to provide for customized spacings and page designs within the REALTIME DISPLAY pages, the following compiler directives are allowed in the NCL file:

$BLANK	Places a blank line after the previously-declared variable.
$BLANK n	Places ‘n’ blank lines after the previously-declared variable.
$PAGE	Inserts a page break after the previously-declared variable.

Realtime Display Default Configuration:

In order to minimize the work of the NCL programmer, the Realtime Display is automatically configured by default. In the absence of any specified DISPL identifiers, the compiler will automatically assign DISPL identifiers to the following register variables:

All LDF’s	(Local Digital Flags)
All LAF’s	(Local Analog Flags)
All LIF’s	(Local Integer Flags)
All RDF’s	(Remote Digital Flags)
All RAF’s	(Remote Analog Flags)
All RIF’s	(Remote Integer Flags)
All VLD’s	(Remote Communication Status Flags)

4 NCL REGISTER NAMES

The NCL registers which are used as memory and I/O locations within NCL programs, are listed below. (Note: n denotes the index number; and j denotes the dependent site index. All indices are referenced from zero, as in the C++ programming language.)

LDS n	Local Digital Setpoint (Read-only from program)
LAS n	Local Analog Setpoint (Read-only from program)
LIS n	Local Integer Setpoint (Read-only from program)
LDM n	Local Digital Input Module (Read-only from program)
LAM n	Local Analog Input Module (Read-only from program)
LAMV n	Local Analog Module Validity (1 if module/transducer working, 0 if failure detected) (Read-only from program)
LIM n	Local Integer Input Module (Read/write from program)
LARM n	Local Analog Rate Module (Fixed Delta-T Method) (1st derivative with respect to time of LAM n) (Units = LAM/minute) (Read-only from program)
LIRM n	Local Integer Rate Module (1st derivative with respect to time of LIM n) (Read-only from program)
LDF n	Local Digital Flag State (Read/write from program)
LAF n	Local Analog Flag State (Read/write from program)
LIF n	Local Integer Flag State (Read/write from program)
LDR n	Local Digital Output Relay (Read/write from program)
LAOM n	Local Analog Output Module (Read/write from program)
RDF j n	Remote Digital Flag State (Read-only from program)
RAF j n	Remote Analog Flag State (Read-only from program)
RIF j n	Remote Integer Flag State (Read-only from program)
VLD j	Valid Status File (1 if valid, 0 if stale) (Read-only from program)
MIS j	Number Of Comm Misses To j Since Retrieving Status File (Read-only from program)
TMR n	Time-Delay Register (Read/write from program)
USR n	User Memory Register (Read/write from program)
LECRM n	Local Sensus Total Module (Read-only from program)
LECRRM n	Local Sensus Flow Module (Read-only from program)
LECRMV n	Local Sensus Meter Module Validity (1 if module working, 0 if failure detected) (Read-only from program)
M_SIU j n	Modbus Input: Short Integer Unsigned (j=devid, n=zero-based reg) (Read-only from program)
M_SIS j n	Modbus Input: Short Integer Signed (j=devid, n=zero-based reg) (Read-only from program)
M_LIU j n	Modbus Input: Long Integer Unsigned (j=devid, n=zero-based reg) (Read-only from program)
M_LIS j n	Modbus Input: Long Integer Signed (j=devid, n=zero-based reg) (Read-only from program)

M_FI	**j n**	Modbus Input: 32-bit Floating Point (j=devid, n=zero-based reg) (Read-only from program)
M_DI	**j n**	Modbus Input: Discrete Input (j=devid, n=zero-based reg) (Read-only from program)
M_SOU	**j n**	Modbus Output: Short Integer Unsigned (j=devid, n=zero-based reg) (Read-only from program)
M_SOS	**j n**	Modbus Output: Short Integer Signed (j=devid, n=zero-based reg) (Read-only from program)
M_DO	**j n**	Modbus Output: Discrete Output (Read-only from program)
V_SIU	**j n**	Toshiba VFD Input: Short Integer Unsigned (j=devid, n=zero-based reg) (Read-only from program)
V_SOU	**j n**	Toshiba VFD Output: Short Integer Unsigned (j=devid, n=zero-based reg) (Read-only from program)
ADM_DI	**j n**	ADAM4000 Discrete Input (j=devid, n=zero-based reg) (Read-only from program)
ADM_DO	**j n**	ADAM4000 Discrete Output (j=devid, n=zero-based reg) (Read/Write from program)
ADM_AI	**j n**	ADAM4000 Analog Input (j=devid, n=zero-based reg) (Read-only from program)
ADM_AO	**j n**	ADAM4000 Analog Output (j=devid, n=zero-based reg) (Read/Write from program)
ADM_II	**j n**	ADAM4000 Integer Input (j=devid, n=zero-based reg) (Read-only from program)

5 BASIC PROGRAMMING TECHNIQUES

Before proceeding further, it is necessary to become acquainted with the elementary techniques of NCL programming.

A NCL program is made up of a "main" routine, which is capable of calling other subroutines (and which are also capable of calling subroutines, and so on …). The deepest level of subroutine calling or recursion allowed is 16 (including the "main"). This should be more than sufficient for even the most demanding applications.

A NCL "main" or subroutine consists of a sequence of commands, each of which is typed on its own separate line in the **"LOGIC.NPP"** file. There are basically four (4) types of commands:

Memory Management - These commands are used to read/write numbers from/to:

a. the memory registers
b. the data stack
c. the address stack
d. the industrial I/O modules

Data Stack Arithmetic - These commands are used to perform arithmetic operations on the members of the data stack.

Execution Branching - These commands are used to control the instruction pointer of the program.

Macros - These commands are used to perform complex calculations using data on the data stack, or using data referenced by the addresses on the address stack. The Macros are provided as a library of pre-defined subroutines for the convenience of the NCL programmer.

The Data Stack and The Address Stack.

There are two (2) stacks available for use by the NCL programmer. The first stack is called the “Data Stack”. The Data Stack is used as a workspace for holding numbers which are needed for control logic calculations. The second stack is called the “Address Stack”. The Address Stack is used as a workspace for holding addresses of registers (or constants) which are needed for control logic calculations. It was decided to create separate stacks: one for numbers and one for addresses, so that programming and debugging would be simplified.

A “stack” is a data buffer that has been created for the convenience of the programmer. It is called a stack because it behaves as if the programmer is stacking numbers on top of each other. Numbers can be "loaded" on top of the stack, or they can be "popped" off the stack. Arithmetic operations can be performed on members of the Data Stack (usually the top number, or the top pair of numbers). In the Data Stack, the top of the stack is called the "X-Register" and the second from the top is called the "Y-Register".

Here is a simple example of how a NCL programmer may utilize the Data Stack to perform the addition of two numbers (3.0 + 2.0) …

NCL Command	**Data Stack**	**Description**
	{empty}	The stack contains no numbers initially.
LOAD 3.0	{ X=3.0 }	"3" has been loaded onto the data stack.
LOAD 2.0	{ X=2.0 , Y=3.0 }	"2" has been loaded on top of the data stack. The "3" is *underneath* the "2".
+	{ X=5.0 }	The top two numbers (2.0 and 3.0) have been popped off of the data stack and added together. The result, "5.0", is loaded onto the top of the data stack.

6 NCL COMMAND SUMMARY

In the NCL programming language, there are approximately 80 available commands. The complete list of commands is shown below, and each is grouped according to its functionality. You do not need to memorize all of the commands at this time (or really at any time); but you should become familiar with the available functionality of the command set.

Memory Management

LOAD	register_id or f	Load the contents of a register or input module onto the data stack. Or Load a number onto the data stack.
POP	n	Pop (discard) n numbers from the data stack. (Default: n=1)
STORE	register_id	Store a copy of the X-Register (data stack) to a register or output module.
PSTORE	register_id	Same as STORE, except followed by POP
COPY	n	Load a duplicate copy of a data stack register onto the top of the data stack. “n” denotes stack position relative to the top, with zero (0) indexing the top stack element (the X-Register). If “n” is not specified, then n defaults to zero (0).
SWAP		Swap the contents of the X-Register and the Y-Register (data stack).
SDELAY	delay_register_id	Set The Delay For "delay_register_id" To the value held in the X-Register.
PSDELAY	delay_register_id	Same as SDELAY, except followed by a POP.
TIMEOUT	delay_register_id	Force the output of “delay_register_id” to one (1).
LOADA	register_id	Load the address of register_id onto the address stack.
LOADV	n	Load the value of the variable, whose address is “element n” on the address stack (the top address has an index of zero), onto the top of the data stack.
POPA	n	Pop (discard) n addresses from the address stack. (Default: n=1)
STOREV	n	Store a copy of the X-Register (data stack) to the register pointed to by element n on the address stack.
PSTOREV	n	Same as STOREV, except followed by a POP.

Data Stack Arithmetic

+	Pop X and Y, Load (Y+X)
-	Pop X and Y, Load (Y-X)
*****	Pop X and Y, Load (Y*X)
/	Pop X and Y, Load (Y/X)
MOD **%**	Pop X and Y, Load (Remainder of Y/X)
OR **|**	Pop X and Y, If X *OR* Y is non-ZERO, Load 1.0. Otherwise Load 0.0.
XOR	Pop X and Y, If X#0 and Y=0, Load 1.0; If X=0 and Y<>0, Load 1.0. Otherwise Load 0.0.
AND **&**	Pop X and Y, If X *AND* Y are both non-ZERO, Load 1.0. Otherwise Load 0.0.
NOT **!** **X=0?**	Pop X, If X is equal to ZERO, Load 1.0. Otherwise, Load 0.0.
X<>0? **X><0?**	Pop X. If X is not equal to ZERO, Load 1.0. Otherwise Load 0.0.
Y>X? **X<Y?**	Pop X and Y. If Y is greater than X, Load 1.0. Otherwise Load 0.0.
Y>=X? **X<=Y?**	Pop X and Y. If Y is greater than or equal to X, Load 1.0. Otherwise Load 0.0.
Y=X? **X=Y?**	Pop X and Y. If Y is equal to X, Load 1.0. Otherwise Load 0.0.
Y<>X? **Y><X?** **X<>Y?** **X><Y?**	Pop X and Y. If Y is not equal to X, Load 1.0. Otherwise Load 0.0.
Y<X? **X>Y?**	Pop X and Y. If Y is less than X, Load 1.0. Otherwise Load 0.0.
Y<=X? **X>=Y?**	Pop X and Y. If Y is less than or equal to X, Load 1.0. Otherwise Load 0.0.
MIN	Pop X and Y. Load MIN(X , Y)
MAX	Pop X and Y. Load MAX(X , Y)
BETWEEN_SECS	Pop X and Y. If Y<DAYTIME_SECS<X, Load 1.0. Otherwise Load 0.0.
BETWEEN_HOURS	Pop X and Y. If Y<DAYTIME_HOURS<X, Load 1.0. Otherwise Load 0.0.
INCR **++**	Pop X, Load (X+1)
DECR **--**	Pop X, Load (X-1)
ABS	Pop X, Load (|X|)
INT	Pop X, Load (INT(X))
SIN	Pop X, Load (SIN(X)), where X is in radians.

COS	Pop X, Load (COS(X)) , where X is in radians.
TAN	Pop X, Load (TAN(X)) , where X is in radians.
Y^X	Pop X and Y, Load (Y^X)
X^2	Pop X, Load (X*X)
SQRT	Pop X, Load (SQRT(|X|))
CHS	Pop X, Load (-X)
LOG	Pop X, Load (LOG10(|X|))
LN	Pop X, Load (LN(|X|))
10^X	Pop X, Load (10^(X))
E^X	Pop X, Load (E^(X))
ASIN	Pop X, Load (ASIN(X)) in radians
ACOS	Pop X, Load (ACOS(X)) in radians
ATAN	Pop X, Load (ATAN(X)) in radians
1/X	Pop X, Load (1/X)

Utility Functions

UPTIME	Load {Seconds Since RTU Execution Started}
SYSTIME	Load {Seconds Since 01 Jan 1970 GMT}
DAYTIME_SECS	Load {Seconds Since Midnight (Takes Into Account Daylight Savings Time)}
DAYTIME_HOURS	Load {Hours Since Midnight (Takes Into Account Daylight Savings Time)}
DAY_OF_WEEK	Load {Day Of Week (1=Sunday … 7=Saturday)}
FIRSTRUN?	If First Call Of NCL Program, Load 1.0. Otherwise, Load 0.0.
NEW_SETPOINTS?	If Setpoints Modified Since The Last Call, Load 1.0. Otherwise, Load 0.0.
ANNOUNCE	Generate Status Announcements To All Sites On Announce-List. Stacks Unaffected.
FLUSH	Store Status and USR Variables To Disk. (Win32/64 Only: Flush History Point To Disk.)
W32_ONLINE	Win32/64 Only: If Desktop/Notebook Has A Compatible UPS And 120VAC Power, Load 1.0. Otherwise, Load 0.0.
CHG%_nn	If one or more local analog flags (LAFs) has changed by more than nn%, then signal to the program that a STATUS_ANNOUNCEMENT should be made all remote sites defined within the announcement list.
W32_BATT_PERCENT	Win32/64 Only: If Desktop/Notebook Has A Compatible UPS, Load Battery Strength (0-100%). Otherwise, Load: –1.0%.
MA_VLD	If last Modbus read was successful or if last Modbus read is a valid cached value, then a ‘1’ is

	placed on the stack. Otherwise, a ‘0’.
MA_CACHED	If last Modbus read was unsuccessful, but the last Modbus read is a valid cached value, then a ‘1’ is placed on the stack. Otherwise, a ‘0’.
MA_TIMEOUT	If last Modbus read was unsuccessful AND the Modbus channel has timed out (a valid cached value is not available), then a ‘1’ is placed on the stack. Otherwise, a ‘0’.
LOADM j n	Load ‘n’ words (16-bit) from Modbus device_id ‘j’ into the Modbus data stack.
CAST_INT n	Copies a 16-bit word from the Modbus data stack onto the data stack, source location based upon index ‘n’. Casts the value as a 16-bit signed integer.
CAST_UINT n	Copies a 16-bit word from the Modbus data stack onto the data stack, source location based upon index ‘n’. Casts the value as a 16-bit unsigned integer.
CAST_LONG n	Copies a 32-bit word from the Modbus data stack onto the data stack, source location based upon index ‘n’. Casts the value as a 32-bit signed integer.
CAST_ULONG n	Copies a 32-bit word from the Modbus data stack onto the data stack, source location based upon index ‘n’. Casts the value as a 32-bit unsigned integer.
CAST_FLOAT n	Copies a 32-bit word from the Modbus data stack onto the data stack, source location based upon index ‘n’. Casts the value as a 32-bit floating point.
BITMASK n	Tests the ‘nth’ bit of the element on the top of the stack. Replaces top stack element with the result (1 or 0).

Execution Branching

LBL	**RoutineName**	Defines Subroutine Name And Labels The Beginning Of The Subroutine.
GOSUB	**RoutineName**	Sends Execution To Top Of "RoutineName"
GOTO	**LineNumber**	Sends Execution To "LineNumber"
MACRO	**MacroName**	Sends Execution To A Pre-Defined Macro
CONTINUE		Does Nothing. Useful As The Target Of A GOTO Statement.
RTN		Returns Execution To The Line Below The Calling GOSUB Statement
END		Ends NCL Program Execution. Returns Execution To Top Of "Main"
IF_TRUE		If X is non-zero, continue execution; otherwise skip over the next command
IF_FALSE		If X is zero, continue execution; otherwise skip over the next command
CHG%_nn		If one or more local analog flags (LAFs) has changed by more than nn%, then signal to the program that a STATUS_ANNOUNCEMENT should be made all remote sites defined within the announcement list.

Example Program Analysis

At this point, you have the necessary background to examine the example program listed in Appendix B. Notice that a large number of comments are interspersed throughout the program. This will assist in debugging, future modifications, and code re-use. Also, notice that both the data stack and address stacks are kept “clean” throughout the program. In other words, when a set of calculations has been completed, all remaining data on the data stack is removed, and all remaining data on the address stack is removed. Again, this optional programming practice simplifies the debugging process, if debugging is required.

7 USING MACROS TO SIMPLIFY NCL PROGRAMS

In addition to the core NCL commands, a group of 12 predefined “Macros” is provided with the compiler. The Macros are analogous to subroutines in C++, and each has been tailored to solve a common control logic problem. A typical NCL program will consist of both Macros and core commands. Each Macro has been optimized and field-tested, so the NCL programmer may use them with confidence.

Built-In Macros

HYSTERESIS_HI	**A Simple Hi-Level Cutoff Switch**
HYSTERESIS_LO	A Simple Lo-Level Cutoff Switch
HYSTERESIS_HI_W_TIMER	A Hi-Level Cutoff Switch with time delay
HYSTERESIS_LO_W_TIMER	A Lo-Level Cutoff Switch with time delay
HYBRID_PRESSURE_HI	A Hybrid-Level/Timer Hi-Level Cutoff Switch
HYBRID_PRESSURE_LO	A Hybrid-Level/Timer Lo-Level Cutoff Switch
SYMMETRIC_DEADBAND	An ON/OFF Analog Switch With A Symmetric Deadband Around The Setpoint.
BOUNDS_CHECK	An “Upper and Lower Bounds Checker” for radiobutton setpoints.
VFD_SPEED	A Proportional-Feedback Analog Controller. Error function a combination of a Hi-Level Boundary and a Lo-Level Boundary.
BPS_MODE_CALC	Calculates The Appropriate Pump Station Mode From The Following Choices: Radio, Pressure, Timer, External.
PUMP_SEQUENCE_SETUP2	Sets Up Pump Alternation / No Alternation For A 2-Pump Station.
PUMP_SEQUENCE_SETUP3	Sets Up Pump Alternation / No Alternation For A 3-Pump Station.
SPHEROID_STATS	Approximate Tank Level In Spheroid Tank
CUBIC_SOLVER	Solve for X: AX^3 + BX^2 + CX + D = 0

MACRO: HYSTERESIS_HI

Example Usage:

```
LOADA       LINE_PRESSURE
LOADA       HI_THRESHOLD_SETTING_PSI
LOADA       HI_THRESHOLD_RELEASE_SETTING_PSI
LOADA       HI_CUTOFF_STATUS
MACRO       HYSTERESIS_HI
PSTORE      HI_CUTOFF_STATUS
```

Internal Compiler Implementation:

```
LBL         HYSTERESIS_HI

LOADV       3
LOADV       2
Y>=X?

LOADV       3
LOADV       1
Y>=X?
LOADV       0
AND

OR
POPA        4
RTN
```

MACRO: HYSTERESIS_LO

Example Usage:

```
LOADA     LINE_PRESSURE
LOADA     LO_THRESHOLD_SETTING_PSI
LOADA     LO_THRESHOLD_RELEASE_SETTING_PSI
LOADA     LO_CUTOFF_STATUS
MACRO     HYSTERESIS_LO
PSTORE    LO_CUTOFF_STATUS
```

Internal Compiler Implementation:

```
LBL       HYSTERESIS_LO

LOADV     3
LOADV     2
Y<=X?

LOADV     3
LOADV     1
Y<=X?
LOADV     0
AND

OR
POPA      4
RTN
```

MACRO: HYSTERESIS_HI_W_TIMER

Example Usage:

```
LOADA     LINE_PRESSURE
LOADA     HI_THRESHOLD_SETTING_PSI
LOADA     HI_THRESHOLD_RELEASE_SETTING_PSI
LOADA     HI_CUTOFF_TIMER
LOADA     HI_RELEASE_TIMER
LOADA     HI_CUTOFF_STATUS
MACRO     HYSTERESIS_HI_W_TIMER
PSTORE    HI_CUTOFF_STATUS
```

Internal Compiler Implementation:

```
LBL       HYSTERESIS_HI_W_TIMER

LOADV     5
LOADV     4
Y>=X?
PSTOREV   2

LOADV     5
LOADV     3
Y<=X?
PSTOREV   1

LOADV     2
LOADV     1
NOT
LOADV     0
AND
OR

POPA      6
RTN
```

MACRO: HYSTERESIS_LO_W_TIMER

Example Usage:

```
LOADA     LINE_PRESSURE
LOADA     LO_THRESHOLD_SETTING_PSI
LOADA     LO_THRESHOLD_RELEASE_SETTING_PSI
LOADA     LO_CUTOFF_TIMER
LOADA     LO_RELEASE_TIMER
LOADA     LO_CUTOFF_STATUS
MACRO     HYSTERESIS_LO_W_TIMER
PSTORE    LO_CUTOFF_STATUS
```

Internal Compiler Implementation:

```
LBL       HYSTERESIS_LO_W_TIMER

LOADV     5
LOADV     4
Y<=X?
PSTOREV   2

LOADV     5
LOADV     3
Y>=X?
PSTOREV   1

LOADV     2
LOADV     1
NOT
LOADV     0
AND
OR

POPA      6
RTN
```

MACRO: HYBRID_PRESSURE_HI

Example Usage:

```
LOADA     DISCHARGE_PRESSURE
LOADA     HI_DISCHARGE_THRESHOLD_PSI
LOADA     HI_DISCHARGE_TIMER
LOADA     HI_DISCHARGE_RELEASE_TIMER
MACRO     HYBRID_PRESSURE_HI
PSTORE    HI_DISCHARGE_CUTOUT
```

Internal Compiler Implementation:

```
LBL       HYBRID_PRESSURE_HI

LOADV     3
LOADV     2
Y>=X?
PSTOREV   1
LOADV     1
NOT
PSTOREV   0
LOADV     0
NOT
POPA      4
RTN
```

MACRO: HYBRID_PRESSURE_LO

Example Usage:

```
LOADA      SUCTION_PRESSURE
LOADA      LO_SUCTION_THRESHOLD_PSI
LOADA      LO_SUCTION_TIMER
LOADA      LO_SUCTION_RELEASE_TIMER
MACRO      HYBRID_PRESSURE_LO
PSTORE     LO_SUCTION_CUTOUT
```

Internal Compiler Implementation:

```
LBL        HYBRID_PRESSURE_LO

LOADV      3
LOADV      2
Y<=X?
PSTOREV    1
LOADV      1
NOT
PSTOREV    0
LOADV      0
NOT
POPA       4
RTN
```

MACRO: SYMMETRIC_DEADBAND

Example Usage:

```
LOADA     HEATER_ON
LOADA     RTU_TEMPERATURE
LOADA     RTU_THERMOSTAT
LOAD      5.0
MACRO     SYMMETRIC_DEADBAND
STORE     HEATER_ON
PSTORE    HEATER_RELAY
```

Internal Compiler Implementation:

```
LBL       SYMMETRIC_DEADBAND
COPY
LOADV     0
+
LOADV     1
Y>X?
LOADV     2
AND
SWAP
CHS
LOADV     0
+
LOADV     1
Y>X?
OR
POPA      3
RTN
```

MACRO: BOUNDS_CHECK

Example Usage:

```
LOADA     VALVE{AUTO-OPEN-CLOSED}
LOAD      3
LOAD      1
MACRO     BOUNDS_CHECK
```

Internal Compiler Implementation:

```
LBL       BOUNDS_CHECK
LOADV     0
MAX
MIN
PSTOREV   0
POPA
RTN
```

MACRO: FEEDBACK_CONTROL

Example Usage:

```
LOAD        SPEED_PERCENT
LOAD        FEEDBACK_GAIN
LOAD        SPEED_MAXSTEP_PERCENT
LOAD        SUCTION_PRESSURE
LOAD        DISCHARGE_PRESSURE
LOAD        SUCTION_LIMIT_PSI
LOAD        DISCHARGE_LIMIT_PSI
MACRO       FEEDBACK_CONTROL
STORE       SPEED_PERCENT
LOAD        100.0
/
PSTORE      SPEED_CONTROL_MODULE
```

Internal Compiler Implementation (C++):

```
ERR1    =       FEEDBACK_GAIN * ( DISCHARGE_LIMIT_PSI – DISCHARGE_PRESSURE )
ERR2    =       FEEDBACK_GAIN * ( SUCTION_PRESSURE –  SUCTION_LIMIT_PSI )
IF ( ERR1 < 0 )
        {
        IF ( ERR2 < 0 )
                {
                // both negative…
                ERR = ERR1 + ERR2
                }
        ELSE
                {
                // only one negative…
                ERR = ERR1
                }
        }
ELSE
        {
        IF ( ERR2 < 0 )
                {
                // one negative…
                ERR = ERR2
                }
        ELSE
                {
                // both positive…
                ERR = ERR1 + ERR2
                }
        }
ERR = MIN ( ERR , SPEED_MAXSTEP_PERCENT )
ERR = MAX ( ERR ,  –SPEED_MAXSTEP_PERCENT )
SPEED = SPEED + ERR
```

[POP ALL SEVEN (7) VALUES OFF THE DATA STACK; AND ADD ONE (1) ELEMENT TO THE DATA STACK: VFD_SPEED_PERCENT.]

MACRO: BPS_MODE_CALC

Example Usage:

```
LOADA    MODE{RADIO-PRESS-TIMER-EXT}
LOADA    FAILOVER{PRESS-TIMER-EXT}
LOADA    COMM_TO_TOWER
LOADA    TOWER_TRANSDUCER_FAIL
MACRO    BPS_MODE_CALC
PSTORE   EXTERNAL_MODE
PSTORE   TIMER_MODE
PSTORE   PRESSURE_MODE
PSTORE   RADIO_MODE
```

Internal Compiler Implementation:

```
LBL      BPS_MODE_CALC

# TOWER_CONTROL_FAIL ...
LOADV    1
NOT
LOADV    0
OR
LOADV    3
LOAD     1.0
X=Y?
AND
PSTOREV  4

# RADIO_MODE ...
LOADV    4
NOT
LOADV    3
LOAD     1.0
X=Y?
AND

# PRESSURE_MODE ...
LOADV    4
LOADV    2
LOAD     1.0
X=Y?
AND
LOADV    3
LOAD     2.0
X=Y?
OR

# TIMER_MODE ...
LOADV4
LOADV2
LOAD     2.0
X=Y?
AND
LOADV3
LOAD     3.0
X=Y?
OR

# EXT_MODE CALC ...
LOADV4
LOADV2
LOAD     3.0
X=Y?
AND
LOADV3
LOAD     4.0
X=Y?
OR

POPA     5
RTN
```

MACRO: PUMP_SEQUENCE_SETUP2

Example Usage:

```
LOADA     ALTERNATE_PUMPS
LOADA     SEQUENCE_POINTER
LOADA     LEAD_PUMP{P1-P2}
LOADA     LAG_PUMP{P1-P2}
LOADA     LEAD_PUMP_DEF
LOADA     LAG_PUMP_DEF
MACRO     PUMP_SEQUENCE_SETUP2
```

Internal Compiler Implementation (C++):

```
IF ( ! ALTERNATE_PUMPS )
        {
        SEQUENCE_POINTER    =    1
        }
IF ( SEQUENCE_POINTER = 1 )
        {
        LEAD_PUMP_DEF       =    LEAD_PUMP{P1-P2}
        LAG_PUMP_DEF        =    LAG_PUMP{P1-P2}
        }
ELSE IF ( SEQUENCE_POINTER = 2 )
        {
        LEAD_PUMP_DEF       =    LAG_PUMP{P1-P2}
        LAG_PUMP_DEF        =    LEAD_PUMP{P1-P2}
        }
```

[Data Stack Pointer: Unchanged. Pop 6 Addresses Off The Address Stack.]

MACRO: PUMP_SEQUENCE_SETUP3

Example Usage:

```
LOADA       ALTERNATE_PUMPS
LOADA       SEQUENCE_POINTER
LOADA       LEAD_PUMP{P1-P2-P3}
LOADA       LAG_PUMP{P1-P2-P3}
LOADA       TRAIL_PUMP{P1-P2-P3}
LOADA       LEAD_PUMP_DEF
LOADA       LAG_PUMP_DEF
LOADA       TRAIL_PUMP_DEF
MACRO       PUMP_SEQUENCE_SETUP3
```

Internal Compiler Implementation (C++):

```
IF ( ! ALTERNATE_PUMPS )
        {
        SEQUENCE_POINTER      =       1
        }
IF ( SEQUENCE_POINTER = 1 )
        {
        LEAD_PUMP_DEF         =       LEAD_PUMP{P1-P2-P3}
        LAG_PUMP_DEF          =       LAG_PUMP{P1-P2-P3}
        LAG_PUMP_DEF          =       TRAIL_PUMP{P1-P2-P3}
        }
ELSE IF ( SEQUENCE_POINTER = 2 )
        {
        LEAD_PUMP_DEF         =       LAG_PUMP{P1-P2-P3}
        LAG_PUMP_DEF          =       TRAIL_PUMP{P1-P2-P3}
        TRAIL_PUMP_DEF        =       LEAD_PUMP{P1-P2-P3}
        }
ELSE IF ( SEQUENCE_POINTER = 3 )
        {
        LEAD_PUMP_DEF         =       TRAIL_PUMP{P1-P2-P3}
        LAG_PUMP_DEF          =       LEAD_PUMP{P1-P2-P3}
        TRAIL_PUMP_DEF        =       LAG_PUMP{P1-P2-P3}
        }
```

[Data Stack Pointer: Unchanged. Pop 6 Addresses Off The Address Stack.]

MACRO: SPHEROID_STATS

Example Usage:

```
LOAD       TANK_LEVEL_FT
LOAD       TANK_FLOW_FT_PER_MINUTE
LOAD       TOP_OF_BOWL_HEIGHT
LOAD       BOTTOM_OF_BOWL_HEIGHT
LOAD       TANK_VOLUME_GAL
MACRO      SPHEROID_STATS
PSTORE     CURRENT_VOLUME_GAL
PSTORE     CURRENT_FLOW_GPM
PSTORE     CURRENT_FLOW_FEET_PER_HOUR
PSTORE     CURRENT_DIAMETER
```

Internal Compiler Implementation (C++):

```
// Volume (spheroid)   = 7.48 x PI x DIAM**2 x HEIGHT / 6
// Diameter (spheroid) = SQRT{ 6 x VOLUME_GAL / 7.48 / PI / HEIGHT }
tank_level_ft            = dStack[dSP-4] ;
current_flow_fpm         = dStack[dSP-3] ;
top_of_bowl_ft           = dStack[dSP-2] ;
bottom_of_bowl_ft        = dStack[dSP-1] ;
tank_capacity_gal= dStack[dSP] ;
delta_height_ft          = top_of_bowl_ft - bottom_of_bowl_ft ;
delta_height_ft          = MAX(delta_height_ft,(double)0.001) ;
rz                       = (double)0.5 * delta_height_ft ;
rz2                      = rz * rz ;
tank_capacity_ft3        = tank_capacity_gal / (double)7.48 ;
tank_max_diameter_ft     = (double)6.0 * tank_capacity_ft3 / DPI / delta_height_ft ;
tank_max_diameter_ft     = sqrt( fabs(tank_max_diameter_ft) ) ;
rx                       = (double)0.5 * tank_max_diameter_ft ;
rx2                      = pow(rx,2) ;

// Tank Level Limiter ...
// (Do not allow tank level to be above/below bowl)...
tank_level_limited = min(tank_level_ft,top_of_bowl_ft) ;
tank_level_limited = max(tank_level_limited,bottom_of_bowl_ft) ;

// Current Diameter ...
// 2 * sqrt( fabs( rx2 - pow( rx*(tank_lev_lim-bot_ht-rz)/rz , 2 ) ) )
current_diameter_ft = (double)2 * sqrt(fabs(rx2-pow(rx*(tank_level_limited-bottom_of_bowl_ft-rz)/rz,2))) ;

// Flow Rate ...
current_flow_gpm= 7.48 * DPI * pow(current_diameter_ft,2) * 0.25 * current_flow_fpm ;
current_flow_fph         = current_flow_fpm * 60 ;

// Current Tank Volume ...
current_volume_gal       = 7.48 * DPI * rx2 ;
current_volume_gal       *= (tank_level_limited-bottom_of_bowl_ft-rz) + (double)0.6666666667 * rz - 0.3333333 *
                              pow(tank_level_limited - bottom_of_bowl_ft - rz , 3 )/rz2 ;

dSP = ( (dSP-1) & DATA_STACK_MASK ) ;
ElementsOnStack-- ;
dStack[dSP]      = current_volume_gal ;
dStack[dSP-1]    = current_flow_gpm ;
dStack[dSP-2]    = current_flow_fph ;
dStack[dSP-3]    = current_diameter_ft ;
```

MACRO: CUBIC_SOLVER

Example Usage:

```
LOAD        A_PARAMETER
LOAD        B_PARAMETER
LOAD        C_PARAMETER
LOAD        D_PARAMETER
LOAD        LOWER_BOUND
LOAD        UPPER_BOUND
LOAD        ACCEPTABLE_ERROR
MACRO       CUBIC_SOLVER
PSTORE      X_ROOT
```

Internal Compiler Implementation (C++):

Solves for the first located root of the polynomial equation:

Y = 0 = A * X^3 + B * X^2 + C * X + D

First, the program searches 20 evenly-spaced values for X between LOWER_BOUND and UPPER_BOUND for an approximate zero-crossing point Y.

Second, the program refines the search using the SECANT method to derive the root within precision determined by ACCEPTABLE_ERROR.

The program returns the root to the stack.

8 CONVERTING RELAY-LADDER-LOGIC TO NCL

RLL (Relay-Ladder-Logic) diagrams can play a useful part in the creation of NCL programs. In a situation where a WiSTAR RTU is installed as a replacement to an obsolete control panel (which may contain complex groups of relays, timers, and transducers), the panel diagram can be implemented as the control logic for the WiSTAR RTU. This provides the benefit of familiar operating characteristics for the system.

NCL offers all of the capabilities of RLL as well as the capabilities of complex logic blocks in one easy-to-use language. This single-language solution provides the benefits of simplified programming and troubleshooting combined with a compact, high-performance interpreter and debugger.

To get acquainted with the basic methods of RLL conversion to NCL, let's start with a simple example of regulating a heater:

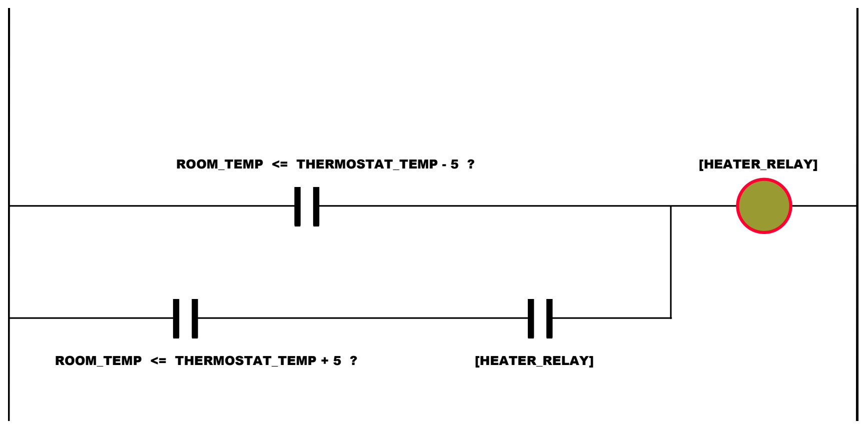


Figure 1. Temperature Regulation Relay Diagram

For this example, let us assume that the thermostat temperature is set at 50degF. When the temperature drops to 45degF or below, then the heater relay will be energized by both the upper and lower rungs of the ladder. After the heater relay is activated, it is assumed that the temperature will begin to climb. When the temperature rises above 45degF, the upper rung will not pass energy to the relay, but the lower rung will continue to pass energy until the temperature climbs above 55degF. This type of 10 degF hysteresis (deadband) is common in level control, as it reduces the wear on the switch and the equipment which it controls.

This heater control example can be implemented in NCL with the following program:

Core Command Implementation:

```
LBL     MAIN

# Logic For Upper Rung ...

LOAD    ROOM_TEMP
LOAD    THERMOSTAT_TEMP
LOAD    5.0
-
Y<=X?

# Logic For Lower Rung ...

LOAD    ROOM_TEMP
LOAD    THERMOSTAT_TEMP
LOAD    5.0
+
Y<=X?
LOAD    HEATER_RELAY
AND

# Add Them Together And Energize
# (or De-Energize) The Relay ...

OR
STORE   HEATER_ON
PSTORE  HEATER_RELAY

END
```

Macro Implemtation:

```
LBL      MAIN
LOADA    HEATER_ON
LOADA    ROOM_TEMP
LOADA    THERMOSTAT_TEMP
LOAD     5.0
MACRO    SYMMETRIC_DEADBAND
STORE    HEATER_ON
PSTORE   HEATER_RELAY
END
```

9 LOADING THE “LOGIC.NPP” FILE INTO THE RTU

After the **"LOGIC.NPP"** program file has been written using the text editor of your choice, the next step is to download the program to the target RTU:

A. Copy your **"LOGIC.NPP"** file onto your palmtop (or notebook) computer, if it is not already on it.

B. Make a serial port connection between your palmtop computer and the RS232-DTE port of the RTU using a null modem cable. The palmtop (or notebook) terminal emulation program should be configured with the following communication settings :

* 19200 bps
* NoParity
* 8 DataBits
* 1 StopBit
* “ANSI” Terminal Emulation

C. Press <ENTER> on the palmtop to activate the "Login" screen, and log in as **"FACTORY”**. Type the **"<X>"** hotkey to "Exit To DOS".

D. Download LOGIC.NPP: At the DOS prompt, type the following command:

D: <ENTER>
CD RT <ENTER>
TRANSFER /R LOGIC.NPP <ENTER>

E. Escape back to the terminal emulation program and send the **"LOGIC.NPP"** file using the XMODEM protocol. When the transfer is complete, you will see the DOS prompt on the terminal display again.

F. Execute *WINCOM*: At the DOS prompt, type the following command:

GO <ENTER>

(This will restart the *WINCOM* software and compile the new NCL program. If there are any compilation errors, *WINCOM* will tell you which line contains the error, what the error is, and halt execution and exit to DOS. If there are any errors, then you will need to edit and fix the flawed **"LOGIC.NPP"** file on your palmtop. After the fix, download the new file to the RTU (Repeat steps D-F.).

G. After successfully compiling the program, you are now ready to evaluate and debug your new program using **“NDB”**, the NCL Debugger, which is described in detail in the next chapter.

10 DEBUGGING WITH *"NDB"*: THE NCL DEBUGGER

The *NDB* debugger may be invoked at any time from the "Login" screen of the *WINCOM* program when the user logs in using the "control"-level password:

`Enter Password :` **CONTROL <ENTER>**

After a few seconds, the following message should appear on your screen:

```
Begin NCL Debugger At Line 1.

   1:   LBL   MAIN
NDB>
```

As a programmer, you may have defined separate logic branches, depending upon certain conditions. For example, during program startup (the "first run"), it is customary to initialize registers and timers. Also, after a setpoint change has occurred, certain registers and timers must be re-initialized. In order to debug these logic branches, the following commands are available to simulate these conditions:

NDB> FIRSTRUN <ENTER> *Toggle "First Run" Simulation ON/OFF*

NDB> NEWSETPOINTS <ENTER> *Toggle "New Setpoints" Simulation ON/OFF*

Note: When the *NDB* debugger is invoked, all communication functions are disabled. For this reason, you can be assured that the status-file(s) retrieved from a remote site(s) will be stable for the duration of your debugging session.

LISTING THE SOURCE CODE

At this point, the first program line has not yet been executed. *NDB* always displays the next line of the program that will be executed. Therefore, upon debugger startup, *NDB* displays the first line of the control logic. Also notice that *NDB* numbers the lines (ignoring any blank lines). This will prove useful should you locate a "bug" (programming error) during the debugging session, as it will help to pinpoint the line(s) that need fixing when you return to your text editor.

Sometimes while debugging, it is useful to "look ahead" several lines so as to anticipate the next commands. This capability is provided by using the "LIST" command at the NDB> prompt. Here is an example of its usage:

```
NDB> LIST <ENTER>     "Lists the next 10 (default) lines in the program"
   1:    LBL          MAIN
   2:    FIRSTRUN?
   3:    IF_FALSE
   4:    GOTO         10
   5:    LOAD         0.0
   6:    STORE        DELTA_TIME              (USR  16)
   7:    STORE        PUMP_START_TIME         (USR  12)
   8:    PSTORE       PUMP_STOP_TIME          (USR  13)
   9:    SYSTIME
   10:   PSTORE       LOW_SUCTION_BEG_TIME    (USR  22)
```

If you type: **LIST *n* <ENTER>**, then the next *n* lines of the program will be displayed.

Notice that when a command uses an "alias" for a register name, *NDB* also displays the actual register name as well. For example, on line 6, DELTA_TIME is the alias name for the "USR 16" register. To provide you with the most possible information during your debugging session, *NDB* displays both actual and alias names of all registers.

“STEPPING THROUGH” THE CONTROL LOGIC

Stepping through the control logic is accomplished by using the "STEP" command at the NDB> prompt. Since this is the most used command, you can simply hit the <ENTER> key, as well, to accomplish a "STEP":

```
NDB> STEP <ENTER>        "Executes one (1) line of NCL code."
     Or …
NDB>      <ENTER>        "Executes one (1) line of NCL code."
```

For many *NDB* commands, you can simply type the first letter as an abbreviated form of the command, if this first letter identifies the command without ambiguity. However, the "STEP" command is the exception, as the "S" is reserved as an abbreviation for the "STACK" command defined below. The "<ENTER>" key should be used as the abbreviated version of "STEP".

If you wish to execute “n” lines of the NCL program, then use the “RUN” command at the prompt:

```
NDB> RUN 20 <ENTER>      "Execute the next 20 lines of code."
```

If you wish to execute the entire program, non-stop, until the END is reached, then use the "RUN" command at the prompt without specifying the number of lines of code:

`NDB>` **RUN <ENTER>** *"Execute code from present location until the end."*

When you reach the END of the program (by using either the "RUN" command or by using "STEP" commands), *NDB* will pause and display the new values of the digital, analog, and integer FLAG registers.

DISPLAYING THE DATA STACK ELEMENTS

By default, the elements on the data stack are printed to the screen, but none will be displayed if the data stack is empty. If you wish for *NDB* to re-print the data stack elements to the screen, use the "STACK" command at the prompt:

`NDB>` **STACK <ENTER>** *"Display the elements of the data stack."*

NDB will also interpret "S" as the "STACK" command.

It is considered good programming practice to pop numbers from the data stack after they are no longer needed by the program. However, this is not strictly required, as the data stack may be overflowed without causing an error condition. When the data stack length exceeds 32, the excess numbers are simply discarded. However, if you choose not to follow this stern recommendation, then your debugging sessions will be much more difficult, as you will be constantly trying to determine which data stack members are of active interest, and which are not.

DISPLAYING THE ADDRESS STACK ELEMENTS

By default, the elements on the address stack are printed to the screen, but none will be displayed if the address stack is empty. If you wish for *NDB* to re-print the stack elements to the screen, use the "ASTACK" command at the prompt:

`NDB>` **ASTACK <ENTER>** *"Display the elements of the address stack."*

It is considered good programming practice to pop numbers from the address stack after they are no longer needed by the program. However, this is not strictly required, as the address stack may be overflowed without causing an error condition. When the address stack length exceeds 32, the excess numbers are simply discarded. However, if you choose not to follow this stern recommendation, then your debugging sessions will be much more difficult, as you will be constantly trying to determine which address stack members are of active interest, and which are not.

DISPLAYING THE CONTENTS OF REGISTERS AND INPUT MODULES

NDB provides the capability to view the contents of all registers and input modules with the "PRINT" command. "PRINT" is invoked from the command line in the following manner:

`NDB>` **PRINT "RegisterName"** *"Display contents of RegisterName."*

Some possible examples would be:

`NDB>` **PRINT POWER_MODULE** *"Display status of the power module."*
Or …
`NDB>` **PRINT TOWER_CALL_PUMP** *"Display one of the water tower's flags."*

MODIFYING THE CONTENTS OF REGISTERS AND OUTPUT MODULES

NDB provides the capability to modify the contents of certain registers and all output modules with the "ASSIGN" command. "ASSIGN" is invoked from the command line in the following manner:

`NDB>` **ASSIGN "RegisterName" Value** *"Assign Value To RegisterName."*

A possible example would be …

`NDB>` **ASSIGN DISCHARGE_PSI 60.0** *"Change Discharge_Psi flag to 60.0"*

The following example is not acceptable, and *NDB* will display an error if you attempt to perform this command:

`NDB>` **ASSIGN DISCHARGE_PSI_MODULE 60.0** *"Illegal! – Read Only Register"*

The DISCHARGE_PSI_MODULE cannot be modified because it is an "input module", and therefore it is "read-only". All registers that are "read-only" cannot be modified with the "ASSIGN" command. When in doubt, refer back to the table in **Chapter 4**, which specifies whether a register or module is "read-only".

MODIFYING THE TIMEOUT OF A TIME-DELAY REGISTER

NDB provides the capability to modify the timeout delay of a time-delay register with the "SETDELAY" command. "SETDELAY" is invoked from the command line in the following manner:

NDB> **SETDELAY "Time-Delay-RegisterName" Value**
... "Change The Timeout Of "Time-Delay-RegisterName" To "Value" seconds.

A possible example would be …

NDB> **SETDELAY LOW_SUCTION_TIMER 25.0**
... "Change Low Suction Timeout To 25 seconds."

EDITING THE NCL HEADER FILE OR LOGIC FILE DURING EXECUTION

NDB provides the capability to edit the header file and/or the logic file from within the debugger environment. The editor is invoked from the command line in the following manner:

NDB> **EDITNPP** *"Edit the NCL Program File."*

It is important to note that any changes made to the NCL Program File do not take effect immediately. After the editing changes are made, a recompilation of the LOGIC.NPP file must follow. The recompilation is invoked from the command line in the following manner:

NDB> **REC** *"Recompile the .NPP Program File."*

EXITING THE DEBUGGER AND CONTINUING *WINCOM* EXECUTION

In order to exit from *NDB* but continue with *WINCOM* execution, use the "EXIT" (or "X") command at the prompt. The "Login" screen will appear, and *WINCOM* execution will continue normally.

NDB> **X** *"Exit The Debugger, And Resume Normal Execution."*

EXITING BOTH THE DEBUGGER AND *WINCOM*

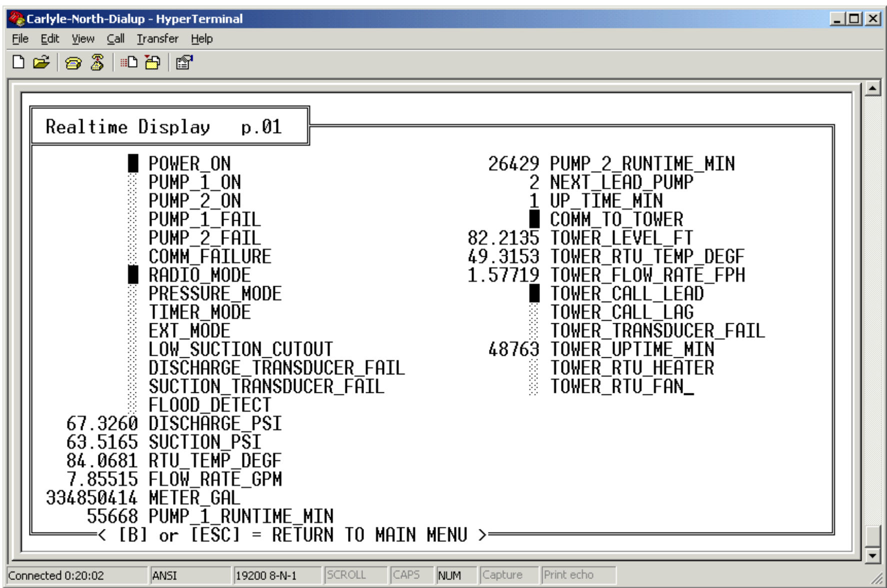
In order to exit from both *NDB* and *WINCOM*, use the "QUIT" (or "Q") command at the prompt. Warming: The program will halt execution and exit to a DOS prompt.

NDB> **Q** *"Exit The Debugger and WINCOM, Return To A DOS system prompt."*

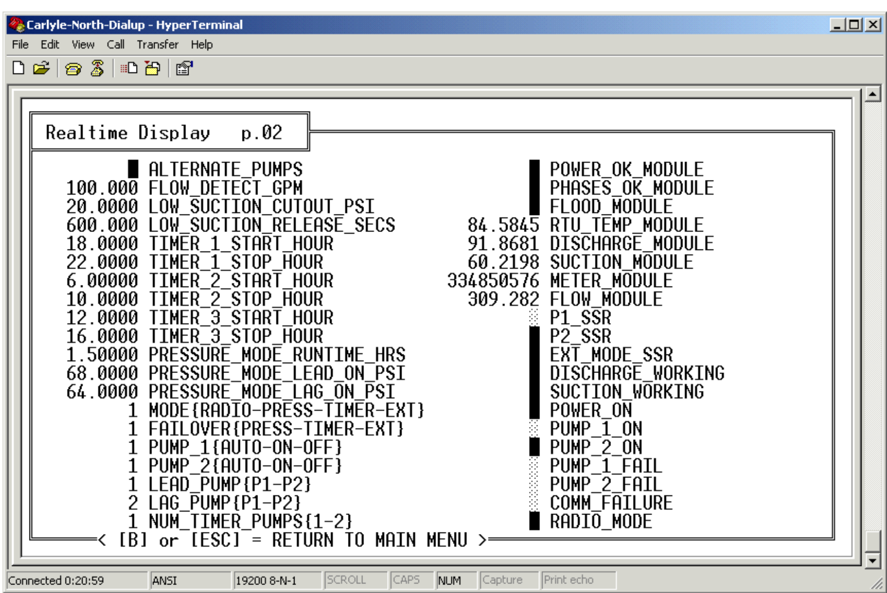
11 DEBUGGING WITH THE ADVANCED REALTIME DISPLAY

1. In addition to *NDB*, the NCL debugger that is documented in the previous chapter, there exists another technique for debugging the control logic of an RTU. This alternative method uses an extended feature of the “Realtime Display”, and is available to personnel who are logged into the RTU with the “factory”-level password.

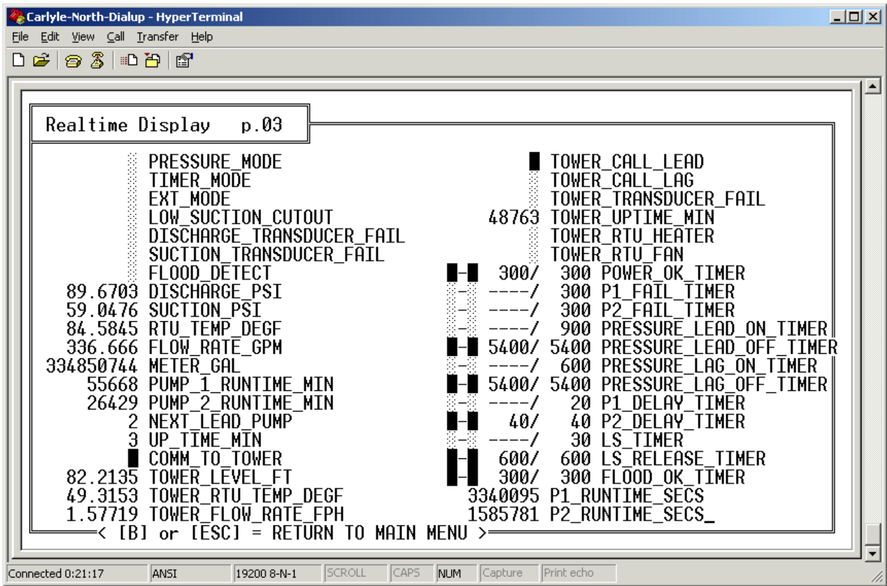
2. While logged in at the “factory”-level, and from the “Main Menu”, press the <8> hotkey to enter the “Realtime Display”. The standard “Realtime Display” is shown:



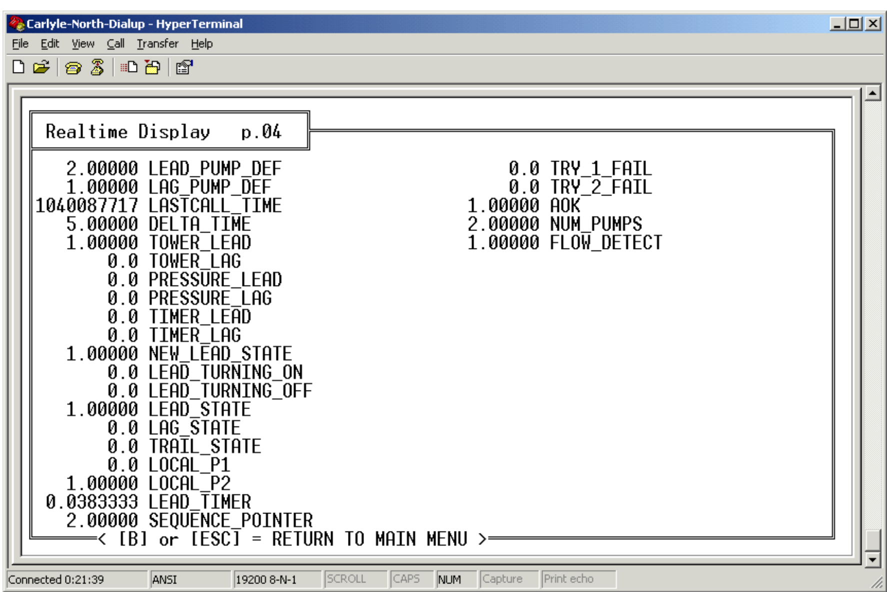
3. When the user presses the <TAB> key, a “Realtime Display” of internal control logic registers is shown. Notice that the page number is incremented to page #2:



4. When the user presses the <TAB> key again, the more internal control logic registers are shown. For user reference, the page number of each screen is shown. Notice the “POWER_OK_TIMER” register, and those of similar type. These are “Time-Delay” Registers. The left block denotes the input signal; and the right block denotes the output signal. The left number denotes the “time since energized”; and the left number denotes the “timeout” of the timer.



5. When the user presses the <TAB> key again, the final page of internal control logic registers is shown. Note that the number of displayed registers is application-specific, and therefore the number of display pages will vary between applications.



6. If the user presses the <TAB> key again, the display will return to the standard “Realtime Display” page.

12 CONCLUDING REMARKS

Over the past 18 years, the rural water and wastewater industries have been moving toward wireless telemetry networks for their reduced costs and increased effectiveness. In an almost parallel time period, the move toward the use of industrial PC's for high-performance industrial control has been equally rapid and consistent. The development of the NCL-programmable WiSTAR RTU represents the natural marriage of these two technologies in a single, integrated product.

The goal of this tutorial has been to provide you with the knowledge and the tools to build innovative and useful distributed control programs for the rural water and wastewater industries. After the first reading, the techniques may seem difficult to master; but this should not deter you – You will get it eventually.

NCL empowers the programmer to tap into the expertise of the operators and engineers of water and wastewater systems; and in turn provide them with control and information networks that perform exactly as desired, and in ways in which no other RTU or PLC could compare.

You now have what you need to build the great NCL applications of tomorrow!

APPENDIX A: DIMENSIONAL LIMITS

The following list documents the present dimensional limits of the most recent NCL firmware (Navionics Research RTU firmware as of 03 December 2002). In a few instances, the limits are based upon computer memory provisions. However, in most cases, it is simply a matter of supporting the industrial I/O hardware of the standard RTU. For example, the standard unit allows 48 digital input modules, 48 relay output modules, 24 analog input modules, 24 analog output modules, and 3 integer input modules (event counters). The dimensional limitations thereby prevent the programmer from attempting to address more modules than the standard hardware configuration will support. However, for special hardware configurations with additional I/O capacity, upgraded software versions can be easily created upon request.

Maximum Usable Data Stack Length	32 Numbers
Maximum Usable Address Stack Length	32 Numbers
Maximum Number Of Aliases	300 Aliases
Maximum Alias Name Length	48 Characters
Maximum "In-Core" NCL Program Length	1000 Lines
Maximum "Out-Of-Core" NCL Program Length	5000+ Lines
Maximum Number Of Digital Setpoints	48
Maximum Number Of Analog Setpoints	40
Maximum Number Of Integer (Radiobutton) Setpoints	40
Maximum Number Of Time-Delay Registers	64
Maximum Number Of Digital Flags	48
Maximum Number Of Analog Flags	40
Maximum Number Of Integer Flags	40
Maximum Number Of Digital Input Modules	48
Maximum Number Of Relay Output Modules	48
Maximum Number Of Analog Input Modules	24
Maximum Number Of Analog Output Modules	24
Maximum Number Of Integer Input (Counter) Modules	3
Maximum Modbus/ADAM/Toshiba Variable Names	32
Maximum Number Of Remote Dependent Sites	10
Maximum Number Of USR Registers	128
Maximum Number of Subroutines	32 Subroutines, Including Main
Maximum Recursion Or Subroutine Calling Depth	16 Subroutines, Including Main
Maximum Subroutine Name Length	36 characters

APPENDIX B: EXAMPLE PROGRAM SOURCE

```
$NCH - Header Info: UPS Station at Walnut Hill, IL
   1     # Number of Digital Setpoints
  16     # Number of Analog  Setpoints
   2     # Number of Integer Setpoints
   4     # Number of Digital Input  Modules
   4     # Number of Analog  Input  Modules
   0     # Number of Integer Input  Modules
  16     # Number of Digital Flag   States
   4     # Number of Analog  Flag   States
   2     # Number of Integer Flag   States
   1     # Number of Relay   Output Modules
   0     # Number of Analog  Output Modules
# Remote Setup Information ... (No Blank Lines Allowed...)
   1     # Number of Dependent Sites (Dependent Sites Follow)
   001      # Index 0 (Zero): Walnut Hill Water Tower
# Variable Name Definitions ... (Blank Lines Allowed...)

ALIAS   FAILOVER_TO_PRESSURE_MODE           LDS    0

ALIAS   LOW_SUCTION_CUTOUT_PSI              LAS    0
ALIAS   LOW_SUCTION_RELEASE_PSI             LAS    1
ALIAS   HIGH_SUCTION_CUTIN_PSI              LAS    2
ALIAS   HIGH_SUCTION_RELEASE_PSI            LAS    3
ALIAS   HIGH_DISCHARGE_CUTOUT_PSI           LAS    4
ALIAS   HIGH_DISCHARGE_RELEASE_PSI          LAS    5
ALIAS   LOW_DISCHARGE_CUTIN_PSI             LAS    6
ALIAS   LOW_DISCHARGE_RELEASE_PSI           LAS    7

ALIAS   PRESSURE_MODE_PUMP_OFF_PSI          LAS    8
ALIAS   PRESSURE_MODE_PUMP_ON_PSI           LAS    9

ALIAS   TIMER_1_START_HOUR                  LAS    10
ALIAS   TIMER_1_STOP_HOUR                   LAS    11
ALIAS   TIMER_2_START_HOUR                  LAS    12
ALIAS   TIMER_2_STOP_HOUR                   LAS    13
ALIAS   TIMER_3_START_HOUR                  LAS    14
ALIAS   TIMER_3_STOP_HOUR                   LAS    15

ALIAS   PUMP_MODE{AUTO-ON-OFF}              LIS    0
ALIAS   STATION_MODE{RADIO-PRESSURE-TIMER}  LIS    1

ALIAS   POWER_MODULE                        LDM    0
ALIAS   PUMP_POSITIVE_INDICATOR_MODULE      LDM    1
ALIAS   PHASE_FAULT_DETECT_MODULE           LDM    2
ALIAS   FLOOD_DETECT_MODULE                 LDM    3

ALIAS   DISCHARGE_PSI_MODULE                LAM    0
ALIAS   SUCTION_PSI_MODULE                  LAM    1
ALIAS   PUMP_TEMP_DEGF_MODULE               LAM    2
ALIAS   PUMP_ROOM_TEMP_DEGF_MODULE          LAM    3

ALIAS   DISCHARGE_WORKING                   LAMV   0
ALIAS   SUCTION_WORKING                     LAMV   1

DISPL   POWER_ON                            LDF    0
DISPL   PUMP_RELAY                          LDF    1
DISPL   PUMP_ON                             LDF    2
DISPL   PHASE_FAULT_DETECT                  LDF    3
DISPL   FLOOD_DETECT                        LDF    4
DISPL   PUMP_FAIL                           LDF    5
```

```
DISPL   COMM_FAILURE                    LDF    6
DISPL   RADIO_MODE                      LDF    7
DISPL   PRESSURE_MODE                   LDF    8
DISPL   TIMER_MODE                      LDF    9
DISPL   LOW_SUCTION_CUTOUT              LDF   10
DISPL   HIGH_SUCTION_CUTIN              LDF   11
DISPL   HIGH_DISCHARGE_CUTOUT           LDF   12
DISPL   LOW_DISCHARGE_CUTIN             LDF   13
DISPL   DISCHARGE_TRANSDUCER_FAIL       LDF   14
DISPL   SUCTION_TRANSDUCER_FAIL         LDF   15

DISPL   DISCHARGE_PSI                   LAF    0
DISPL   SUCTION_PSI                     LAF    1
DISPL   PUMP_TEMP_DEGF                  LAF    2
DISPL   PUMP_ROOM_TEMP_DEGF             LAF    3

DISPL   UP_TIME_MIN                     LIF    0
DISPL   PUMP_RUNTIME_MIN                LIF    1

ALIAS   PUMP_SSR                        LDR    0

DISPL   COMM_TO_WATER_TOWER             VLD    0
DISPL   TOWER_LEVEL_FT                  RAF    0  0
DISPL   TOWER_CALL_PUMP                 RDF    0  2
DISPL   TOWER_TRANSDUCER_FAIL           RDF    0  6

ALIAS   LOW_SUCTION_TIMER               TMR    0
ALIAS   LOW_SUCTION_OK_TIMER            TMR    1
ALIAS   HIGH_DISCHARGE_TIMER            TMR    2
ALIAS   HIGH_DISCHARGE_OK_TIMER         TMR    3
ALIAS   HIGH_SUCTION_TIMER              TMR    4
ALIAS   HIGH_SUCTION_OK_TIMER           TMR    5
ALIAS   LOW_DISCHARGE_TIMER             TMR    6
ALIAS   LOW_DISCHARGE_OK_TIMER          TMR    7
ALIAS   PHASES_OK_TIMER                 TMR    8
ALIAS   POWER_OK_TIMER                  TMR    9
ALIAS   FLOOD_OK_TIMER                  TMR    10
ALIAS   PUMP_FAIL_TIMER                 TMR    11

ALIAS   PUMP_RUNTIME_SECS               USR    0
ALIAS   LASTCALL_TIME                   USR    1
ALIAS   DELTA_TIME                      USR    2
ALIAS   AOK                             USR    3
ALIAS   EMERGENCY_CUTIN                 USR    4
ALIAS   PRESSURE_PUMP                   USR    5
ALIAS   TIMER_PUMP                      USR    6
ALIAS   TOWER_PUMP                      USR    7
ALIAS   LOCAL_PUMP                      USR    8

$NCL

# NCL Program
#
# Station : UPS Pump Station ( Village of Walnut Hill PWD )
# Author  : Jim Mimlitz, Navionics Research Inc.
# Date    : 30 September  1997
#

# TRANSFER MODULE INPUTS TO FLAG INPUTS ...

      LBL      MAIN

      LOAD     PUMP_POSITIVE_INDICATOR_MODULE
```

```
        PSTORE   PUMP_ON

        LOAD     DISCHARGE_PSI_MODULE
        PSTORE   DISCHARGE_PSI

        LOAD     SUCTION_PSI_MODULE
        PSTORE   SUCTION_PSI

        LOAD     DISCHARGE_WORKING
        NOT
        PSTORE   DISCHARGE_TRANSDUCER_FAIL

        LOAD     SUCTION_WORKING
        NOT
        PSTORE   SUCTION_TRANSDUCER_FAIL

        LOAD     PUMP_ROOM_TEMP_DEGF_MODULE
        PSTORE   PUMP_ROOM_TEMP_DEGF

        LOAD     PUMP_TEMP_DEGF_MODULE
        LOAD     PUMP_ROOM_TEMP_DEGF
        -
        PSTORE   PUMP_TEMP_DEGF


# FIRSTRUN HANDLER & DELTA-TIME HANDLER ...

        FIRSTRUN?
        IF_FALSE
        GOTO     10

        SYSTIME
        PSTORE   LASTCALL_TIME

        LOAD     5.0
        SDELAY   LOW_SUCTION_TIMER
        SDELAY   HIGH_SUCTION_TIMER
        PSDELAY  LOW_DISCHARGE_TIMER
        LOAD     12.0
        PSDELAY  HIGH_DISCHARGE_TIMER
        LOAD     300.0
        SDELAY   LOW_SUCTION_OK_TIMER
        SDELAY   HIGH_SUCTION_OK_TIMER
        SDELAY   LOW_DISCHARGE_OK_TIMER
        SDELAY   HIGH_DISCHARGE_OK_TIMER
        SDELAY   PHASES_OK_TIMER
        SDELAY   POWER_OK_TIMER
        PSDELAY  FLOOD_OK_TIMER
        LOAD     180.0
        PSDELAY  PUMP_FAIL_TIMER

        GOSUB    SANITY_CHECKS

10      POP

        SYSTIME
        LOAD     LASTCALL_TIME
        -
        PSTORE   DELTA_TIME
        SYSTIME
        PSTORE   LASTCALL_TIME
```

```
# IF NEW SETPOINTS, SANITY CHECK THE SETPOINTS ...

      NEW_SETPOINTS?
      IF_FALSE
      GOTO      20
      GOSUB     SANITY_CHECKS
20    POP


# CALCULATE SYSTEM UPTIME ...

      UPTIME
      LOAD      60.0
      /
      PSTORE    UP_TIME_MIN


# CHECK COMMUNICATION STATUS & PRESSURE_MODE CALC ...

      LOAD      COMM_TO_WATER_TOWER
      NOT
      STORE     COMM_FAILURE
      LOAD      TOWER_TRANSDUCER_FAIL
      OR
      LOAD      STATION_MODE{RADIO-PRESSURE-TIMER}
      LOAD      1.0
      X=Y?
      AND
      LOAD      FAILOVER_TO_PRESSURE_MODE
      AND
      LOAD      STATION_MODE{RADIO-PRESSURE-TIMER}
      LOAD      2.0
      X=Y?
      OR
      PSTORE    PRESSURE_MODE


# TIMER_MODE CALC ...

      LOAD      STATION_MODE{RADIO-PRESSURE-TIMER}
      LOAD      3.0
      X=Y?
      PSTORE    TIMER_MODE


# RADIO_MODE CALC ...

      LOAD      COMM_FAILURE
      NOT
      LOAD      STATION_MODE{RADIO-PRESSURE-TIMER}
      LOAD      1.0
      X=Y?
      AND
      PSTORE    RADIO_MODE


# LOW SUCTION CALC (W/ DELAY TIMER) ...

      LOAD      SUCTION_PSI
      LOAD      LOW_SUCTION_CUTOUT_PSI
      Y<=X?
      PSTORE    LOW_SUCTION_TIMER
```

```
      LOAD     LOW_SUCTION_TIMER
      NOT
      PSTORE   LOW_SUCTION_OK_TIMER

      LOAD     LOW_SUCTION_OK_TIMER
      NOT
      LOAD     SUCTION_PSI
      LOAD     LOW_SUCTION_RELEASE_PSI
      Y<X?
      LOAD     LOW_SUCTION_CUTOUT
      AND
      OR
      PSTORE   LOW_SUCTION_CUTOUT


# HIGH_DISCHARGE CALC (W/ DELAY TIMER) ...

      LOAD     DISCHARGE_PSI
      LOAD     HIGH_DISCHARGE_CUTOUT_PSI
      Y>=X?
      PSTORE   HIGH_DISCHARGE_TIMER

      LOAD     HIGH_DISCHARGE_TIMER
      NOT
      PSTORE   HIGH_DISCHARGE_OK_TIMER

      LOAD     HIGH_DISCHARGE_OK_TIMER
      NOT
      LOAD     DISCHARGE_PSI
      LOAD     HIGH_DISCHARGE_RELEASE_PSI
      Y>X?
      LOAD     HIGH_DISCHARGE_CUTOUT
      AND
      OR
      LOAD     DISCHARGE_TRANSDUCER_FAIL
      NOT
      AND
      PSTORE   HIGH_DISCHARGE_CUTOUT


# HIGH SUCTION CALC (W/ DELAY TIMER) ...

      LOAD     SUCTION_PSI
      LOAD     HIGH_SUCTION_CUTIN_PSI
      Y>=X?
      PSTORE   HIGH_SUCTION_TIMER

      LOAD     HIGH_SUCTION_TIMER
      NOT
      PSTORE   HIGH_SUCTION_OK_TIMER

      LOAD     HIGH_SUCTION_OK_TIMER
      NOT
      LOAD     SUCTION_PSI
      LOAD     HIGH_SUCTION_RELEASE_PSI
      Y>X?
      LOAD     HIGH_SUCTION_CUTIN
      AND
      OR
      LOAD     SUCTION_TRANSDUCER_FAIL
      NOT
      AND
      PSTORE   HIGH_SUCTION_CUTIN
```

```
# LOW DISCHARGE CALC (W/ DELAY TIMER) ...

      LOAD      DISCHARGE_PSI
      LOAD      LOW_DISCHARGE_CUTIN_PSI
      Y<=X?
      PSTORE    LOW_DISCHARGE_TIMER

      LOAD      LOW_DISCHARGE_TIMER
      NOT
      PSTORE    LOW_DISCHARGE_OK_TIMER

      LOAD      LOW_DISCHARGE_OK_TIMER
      NOT
      LOAD      DISCHARGE_PSI
      LOAD      LOW_DISCHARGE_RELEASE_PSI
      Y<X?
      LOAD      LOW_DISCHARGE_CUTIN
      AND
      OR
      LOAD      DISCHARGE_TRANSDUCER_FAIL
      NOT
      AND
      PSTORE    LOW_DISCHARGE_CUTIN

      LOAD      PHASE_FAULT_DETECT_MODULE
      PSTORE    PHASES_OK_TIMER
      LOAD      PHASES_OK_TIMER
      NOT
      PSTORE    PHASE_FAULT_DETECT

      LOAD      POWER_MODULE
      PSTORE    POWER_OK_TIMER
      LOAD      POWER_OK_TIMER
      PSTORE    POWER_ON

      LOAD      FLOOD_DETECT_MODULE
      PSTORE    FLOOD_OK_TIMER
      LOAD      FLOOD_OK_TIMER
      NOT
      PSTORE    FLOOD_DETECT


# EQUIPMENT FAILURE CALC (W/ RELEASE TIMER) ...

      LOAD      LOW_SUCTION_CUTOUT
      NOT
      LOAD      HIGH_DISCHARGE_CUTOUT
      NOT
      AND
      LOAD      PHASE_FAULT_DETECT
      NOT
      AND
      LOAD      FLOOD_DETECT
      NOT
      AND
      LOAD      POWER_ON
      AND
      PSTORE    AOK


# EMERGENCY_CUTIN CALC ...
```

```
     LOAD      LOW_DISCHARGE_CUTIN
     LOAD      HIGH_SUCTION_CUTIN
     OR
     PSTORE    EMERGENCY_CUTIN


# TIMER-MODE HANDLER ...

     LOAD      TIMER_1_START_HOUR
     LOAD      TIMER_1_STOP_HOUR
     BETWEEN_HOURS
     LOAD      TIMER_2_START_HOUR
     LOAD      TIMER_2_STOP_HOUR
     BETWEEN_HOURS
     LOAD      TIMER_3_START_HOUR
     LOAD      TIMER_3_STOP_HOUR
     BETWEEN_HOURS
     OR
     OR
     LOAD      TIMER_MODE
     AND
     PSTORE    TIMER_PUMP


# PRESSURE-MODE HANDLER ...

     LOAD      DISCHARGE_PSI
     LOAD      PRESSURE_MODE_PUMP_ON_PSI
     Y<=X?
     LOAD      DISCHARGE_PSI
     LOAD      PRESSURE_MODE_PUMP_OFF_PSI
     Y<X?
     LOAD      PRESSURE_PUMP
     AND
     OR
     LOAD      PRESSURE_MODE
     AND
     PSTORE    PRESSURE_PUMP


# TOWER-MODE HANDLER ...

     LOAD      TOWER_CALL_PUMP
     LOAD      RADIO_MODE
     AND
     PSTORE    TOWER_PUMP


# LOCAL-MODE PUMP HANDLER ...

     LOAD      PRESSURE_PUMP
     LOAD      TIMER_PUMP
     LOAD      EMERGENCY_CUTIN
     OR
     OR
     PSTORE    LOCAL_PUMP


# FINAL PUMP ON-OFF CALC ...

     LOAD      TOWER_PUMP
     LOAD      LOCAL_PUMP
```

```
        OR
        LOAD      PUMP_MODE{AUTO-ON-OFF}
        LOAD      2.0
        X=Y?
        OR
        LOAD      AOK
        AND
        LOAD      PUMP_MODE{AUTO-ON-OFF}
        LOAD      3.0
        X<>Y?
        AND
        STORE     PUMP_RELAY
        PSTORE    PUMP_SSR


# PUMP FAIL CALC (WITH DELAY TIMER) ...

        LOAD      PUMP_RELAY
        LOAD      PUMP_ON
        XOR
        PSTORE    PUMP_FAIL_TIMER
        LOAD      PUMP_FAIL_TIMER
        PSTORE    PUMP_FAIL


# PUMP RUNTIME CALC ...

        LOAD      PUMP_ON
        LOAD      DELTA_TIME
        *
        LOAD      PUMP_RUNTIME_SECS
        +
        ABS
        STORE     PUMP_RUNTIME_SECS
        LOAD      60.0
        /
        LOAD      1000000000.0
        MOD
        PSTORE    PUMP_RUNTIME_MIN

        END


        LBL       SANITY_CHECKS


# CHECK PUMP_1_MODE RANGE ...

        LOAD      3.0
        LOAD      1.0
        LOAD      PUMP_MODE{AUTO-ON-OFF}
        MAX
        MIN
        PSTORE    PUMP_MODE{AUTO-ON-OFF}


# CHECK STATION_MODE RANGE ...

        LOAD      3.0
        LOAD      1.0
        LOAD      STATION_MODE{RADIO-PRESSURE-TIMER}
        MAX
        MIN
```

```
PSTORE   STATION_MODE{RADIO-PRESSURE-TIMER}

RTN
```

APPENDIX C: ADVANCED EXAMPLE I – WATER TOWER

```
$NCH Control Logic Setup Info: RE Water - Angle Road Elevated Tank
   0     # Number of Discrete Setpoints
   9     # Number of Analog   Setpoints
   0     # Number of Integer  Setpoints
   1     # Number of Discrete Input  Modules
   2     # Number of Analog   Input  Modules
   0     # Number of Integer  Input  Modules
   8     # Number of Discrete Flag   States
   3     # Number of Analog   Flag   States
   1     # Number of Integer  Flag   States
   1     # Number of Relay    Output Modules
   0     # Number of Analog   Output Modules
# Remote RTU Setup Information ...
   1     # Number of Dependent Sites (Dependent Sites Follow)
 002     # Berryville BPS

# Variable Name Definitions ...

ALIAS   LEAD_OFF_LEVEL               LAS    0
ALIAS   LEAD_ON_LEVEL                LAS    1
ALIAS   LAG_OFF_LEVEL                LAS    2
ALIAS   LAG_ON_LEVEL                 LAS    3
ALIAS   HEATER_THERMO_DEGF           LAS    4
ALIAS   AUX_HIGH_FT                  LAS    5
ALIAS   AUX_HIGH_RELEASE_FT          LAS    6
ALIAS   AUX_LOW_FT                   LAS    7
ALIAS   AUX_LOW_RELEASE_FT           LAS    8

ALIAS   POWER_MODULE                 LDM    0

ALIAS   TANK_LEVEL_MODULE            LAM    0
ALIAS   RTU_TEMP_MODULE              LAM    1

ALIAS   LEVEL_WORKING                LAMV   0

ALIAS   TANK_LEVEL_RATE              LARM   0

ALIAS   POWER_ON                     LDF    0
ALIAS   CALL_FOR_LEAD                LDF    1
ALIAS   CALL_FOR_LAG                 LDF    2
ALIAS   RTU_HEATER_ON                LDF    3
ALIAS   COMM_FAILURE                 LDF    4
ALIAS   AUX_HIGH                     LDF    5
ALIAS   AUX_LOW                      LDF    6
ALIAS   TRANSDUCER_FAIL              LDF    7

ALIAS   TANK_LEVEL_FT                LAF    0
ALIAS   RTU_TEMP_DEGF                LAF    1
ALIAS   TANK_FLOW_RATE_FPH           LAF    2

ALIAS   UP_TIME_MIN                  LIF    0

ALIAS   RTU_HEATER_SSR               LDR    0

ALIAS   LEAD_ON_TIMER                TMR    0
ALIAS   LEAD_OFF_TIMER               TMR    1
ALIAS   LAG_ON_TIMER                 TMR    2
ALIAS   LAG_OFF_TIMER                TMR    3
ALIAS   LOW_ON_TIMER                 TMR    4
ALIAS   LOW_OFF_TIMER                TMR    5
```

```
ALIAS   HIGH_ON_TIMER                 TMR    6
ALIAS   HIGH_OFF_TIMER                TMR    7


ALIAS   LASTCALL_TIME                 USR    0
ALIAS   DELTA_TIME                    USR    1

ALIAS   COMM_TO_PUMP_STATION          VLD    0
ALIAS   COMM_MISSES_TO_PUMP           MIS    0

$NCL

# NCL Program
#
# Client  : RE Water Corporation
# Station : Angle Road Elevated Tank
# Author  : Tatyana Mimlitz, Navionics Research Inc.
# Date    : 19 November 2002
#

# Transfer Module Inputs To Flag Inputs ...
# ----------------------------------------
      LBL      MAIN

      LOAD     POWER_MODULE
      PSTORE   POWER_ON

      LOAD     TANK_LEVEL_MODULE
      PSTORE   TANK_LEVEL_FT

      LOAD     RTU_TEMP_MODULE
      PSTORE   RTU_TEMP_DEGF

      LOAD     TANK_LEVEL_RATE
      LOAD     60.0
      *
      PSTORE   TANK_FLOW_RATE_FPH

      LOAD     COMM_TO_PUMP_STATION
      NOT
      PSTORE   COMM_FAILURE

      LOAD     LEVEL_WORKING
      NOT
      PSTORE   TRANSDUCER_FAIL

# CALCULATE DELTA-TIME SINCE LAST CALL ...

      SYSTIME
      LOAD     LASTCALL_TIME
      -
      PSTORE   DELTA_TIME
      SYSTIME
      PSTORE   LASTCALL_TIME

# CALCULATE SYSTEM UPTIME ...

      UPTIME
      LOAD     60.0
      /
      PSTORE   UP_TIME_MIN

# DEFINE TIMEOUTS
```

```
      FIRSTRUN?
      IF_FALSE
      GOTO     110
      LOAD     0
      STORE    LOW_ON_TIMER
      STORE    LOW_OFF_TIMER
      STORE    HIGH_ON_TIMER
      STORE    HIGH_OFF_TIMER
      STORE    LEAD_ON_TIMER
      STORE    LEAD_OFF_TIMER
      STORE    LAG_ON_TIMER
      PSTORE   LAG_OFF_TIMER

      LOAD     60
      SDELAY   LOW_ON_TIMER
      SDELAY   LOW_OFF_TIMER
      SDELAY   HIGH_ON_TIMER
      SDELAY   HIGH_OFF_TIMER
      SDELAY   LEAD_ON_TIMER
      SDELAY   LEAD_OFF_TIMER
      SDELAY   LAG_ON_TIMER
      PSDELAY  LAG_OFF_TIMER
110   POP


# LEAD PUMP CALC ...

      LOADA    TANK_LEVEL_FT
      LOADA    LEAD_ON_LEVEL
      LOADA    LEAD_OFF_LEVEL
      LOADA    LEAD_ON_TIMER
      LOADA    LEAD_OFF_TIMER
      LOADA    CALL_FOR_LEAD
      MACRO    HYSTERESIS_LO_W_TIMER
      LOAD     TRANSDUCER_FAIL
      NOT
      AND
      PSTORE   CALL_FOR_LEAD


# LAG PUMP CALC ...

      LOADA    TANK_LEVEL_FT
      LOADA    LAG_ON_LEVEL
      LOADA    LAG_OFF_LEVEL
      LOADA    LAG_ON_TIMER
      LOADA    LAG_OFF_TIMER
      LOADA    CALL_FOR_LAG
      MACRO    HYSTERESIS_LO_W_TIMER
      LOAD     TRANSDUCER_FAIL
      NOT
      AND
      PSTORE   CALL_FOR_LAG


# HEATER CALC ...

      LOADA    RTU_HEATER_ON
      LOADA    RTU_TEMP_DEGF
      LOADA    HEATER_THERMO_DEGF
      LOAD     5.0
      MACRO    SYMMETRIC_DEADBAND
      STORE    RTU_HEATER_ON
```

```
      PSTORE   RTU_HEATER_SSR


# AUX_HIGH_LEVEL CALC ...

      LOADA    TANK_LEVEL_FT
      LOADA    AUX_HIGH_FT
      LOADA    AUX_HIGH_RELEASE_FT
      LOADA    HIGH_ON_TIMER
      LOADA    HIGH_OFF_TIMER
      LOADA    AUX_HIGH
      MACRO    HYSTERESIS_HI_W_TIMER
      LOAD     TRANSDUCER_FAIL
      NOT
      AND
      PSTORE   AUX_HIGH


# AUX_LOW_LEVEL CALC ...

      LOADA    TANK_LEVEL_FT
      LOADA    AUX_LOW_FT
      LOADA    AUX_LOW_RELEASE_FT
      LOADA    LOW_ON_TIMER
      LOADA    LOW_OFF_TIMER
      LOADA    AUX_LOW
      MACRO    HYSTERESIS_LO_W_TIMER
      LOAD     TRANSDUCER_FAIL
      NOT
      AND
      PSTORE   AUX_LOW


      END
```

APPENDIX D: ADVANCED EXAMPLE II – PUMP STATION

```
$NCH Control Logic Setup Info: RE Water - Berryville BPS
   1     # Number of Discrete Setpoints
  21     # Number of Analog   Setpoints
   6     # Number of Integer  Setpoints
   5     # Number of Discrete Input  Modules
   4     # Number of Analog   Input  Modules
   2     # Number of Integer  Input  Modules
  16     # Number of Discrete Flag   States
   8     # Number of Analog   Flag   States
   6     # Number of Integer  Flag   States
   8     # Number of Relay    Output Modules
   2     # Number of Analog   Output Modules
# Remote RTU Setup Information ...
   1     # Number of Dependent Sites (Dependent Sites Follow)
 001     # New Elevated Tank At Angle Road

# Variable Name Definitions ...

ALIAS   ALTERNATE_PUMPS                        LDS    0

ALIAS   GST_VALVE_CLOSE_FT                     LAS    0
ALIAS   GST_VALVE_OPEN_FT                      LAS    1
ALIAS   VALVE_FEED_LIMIT_PSI                   LAS    2
ALIAS   VALVE_GAIN                             LAS    3
ALIAS   VALVE_MAXSTEP                          LAS    4
ALIAS   VALVE_XDUCER_FAIL_OPEN_PERCENT         LAS    5
ALIAS   LOW_GST_CUTOUT_FT                      LAS    6
ALIAS   LOW_GST_RELEASE_SECS                   LAS    7

ALIAS   VFD_DISCHARGE_LIMIT_PSI                LAS    8
ALIAS   VFD_GAIN                               LAS    9
ALIAS   VFD_MAXSTEP                            LAS   10
ALIAS   VFD_XDUCER_FAIL_SPEED_PERCENT          LAS   11
ALIAS   FLOW_DETECT_GPM                        LAS   12

ALIAS   PRESSURE_MODE_RUNTIME_HRS              LAS   13
ALIAS   PRESSURE_MODE_LEAD_ON_PSI              LAS   14
ALIAS   TIMER_1_START_HOUR                     LAS   15
ALIAS   TIMER_1_STOP_HOUR                      LAS   16
ALIAS   TIMER_2_START_HOUR                     LAS   17
ALIAS   TIMER_2_STOP_HOUR                      LAS   18
ALIAS   TIMER_3_START_HOUR                     LAS   19
ALIAS   TIMER_3_STOP_HOUR                      LAS   20

ALIAS   MODE{RADIO-PRESS-TIMER-EXT}            LIS    0
ALIAS   FAILOVER{PRESS-TIMER-EXT}              LIS    1
ALIAS   PUMP_1{AUTO-ON-OFF}                    LIS    2
ALIAS   PUMP_2{AUTO-ON-OFF}                    LIS    3
ALIAS   LEAD_PUMP{P1-P2}                       LIS    4
ALIAS   LAG_PUMP{P1-P2}                        LIS    5

ALIAS   POWER_OK_MODULE                        LDM    0
ALIAS   PUMP_FEEDBACK_MODULE                   LDM    1
ALIAS   GST_VALVE_OPEN_MODULE                  LDM    3
ALIAS   GST_VALVE_CLOSED_MODULE                LDM    4

ALIAS   DISCHARGE_MODULE                       LAM    0
ALIAS   GST_MODULE                             LAM    1
ALIAS   ROOM_TEMP_MODULE                       LAM    2
ALIAS   FEED_MODULE                            LAM    3
```

```
ALIAS   DISCHARGE_WORKING                    LAMV    0
ALIAS   GST_WORKING                          LAMV    1
ALIAS   FEED_WORKING                         LAMV    3

ALIAS   METER_OUT_MODULE                     LIM     0
ALIAS   METER_IN_MODULE                      LIM     1

ALIAS   P1_SSR                               LDR     0
ALIAS   P2_SSR                               LDR     1
ALIAS   EXT_MODE_SSR                         LDR     7

ALIAS   VALVE_POSITION_MODULE                LAOM    4
ALIAS   VFD_SPEED_MODULE                     LAOM    5

DISPL   POWER_ON                             LDF     0
DISPL   PUMP_1_ON                            LDF     1
DISPL   PUMP_2_ON                            LDF     2
DISPL   PUMP_1_FAIL                          LDF     3
DISPL   PUMP_2_FAIL                          LDF     4
DISPL   GST_VALVE_OPEN                       LDF     5
DISPL   GST_VALVE_FAIL                       LDF     6
DISPL   COMM_FAILURE                         LDF     7
DISPL   RADIO_MODE                           LDF     8
DISPL   PRESSURE_MODE                        LDF     9
DISPL   TIMER_MODE                           LDF    10
ALIAS   EXT_MODE                             LDF    11
DISPL   LOW_GST_CUTOUT                       LDF    12
DISPL   DISCHARGE_TRANSDUCER_FAIL            LDF    13
DISPL   GST_TRANSDUCER_FAIL                  LDF    14
DISPL   FEED_TRANSDUCER_FAIL                 LDF    15

DISPL   DISCHARGE_PSI                        LAF     0
DISPL   GST_LEVEL_FT                         LAF     1
DISPL   FEED_PSI                             LAF     2
DISPL   FLOW_RATE_IN_GPM                     LAF     3
DISPL   FLOW_RATE_OUT_GPM                    LAF     4
DISPL   VFD_SPEED_PERCENT                    LAF     5
DISPL   GST_VALVE_OPEN_PERCENT               LAF     6
DISPL   ROOM_TEMP_DEGF                       LAF     7

DISPL   METER_IN_GAL                         LIF     0
DISPL   METER_OUT_GAL                        LIF     1
DISPL   PUMP_1_RUNTIME_MIN                   LIF     2
DISPL   PUMP_2_RUNTIME_MIN                   LIF     3
ALIAS   CURRENT_LEAD_PUMP                    LIF     4
ALIAS   UP_TIME_MIN                          LIF     5

DISPL   COMM_TO_TOWER                        VLD     0
DISPL   TOWER_LEVEL_FT                       RAF     0  0
DISPL   TOWER_CALL_PUMP                      RDF     0  1
DISPL   TOWER_TRANSDUCER_FAIL                RDF     0  7

ALIAS   POWER_OK_TIMER                       TMR     0
ALIAS   P1_FAIL_TIMER                        TMR     1
ALIAS   P2_FAIL_TIMER                        TMR     2
ALIAS   PRESSURE_LEAD_ON_TIMER               TMR     3
ALIAS   PRESSURE_LEAD_OFF_TIMER              TMR     4
ALIAS   P1_DELAY_TIMER                       TMR     5
ALIAS   P2_DELAY_TIMER                       TMR     6
ALIAS   P1_OFF_TIMER                         TMR     7
ALIAS   P2_OFF_TIMER                         TMR     8
ALIAS   LGST_TIMER                           TMR     9
```

```
ALIAS   LGST_RELEASE_TIMER                    TMR   10
ALIAS   HSP_SAMPLE_TIMER                      TMR   11
ALIAS   GST_VALVE_FAIL_TIMER                  TMR   12

ALIAS   P1_RUNTIME_SECS                       USR    0
ALIAS   P2_RUNTIME_SECS                       USR    1
ALIAS   LEAD_PUMP_DEF                         USR    2
ALIAS   LAG_PUMP_DEF                          USR    3
ALIAS   LASTCALL_TIME                         USR    4
ALIAS   DELTA_TIME                            USR    5
ALIAS   TOWER_LEAD                            USR    6
ALIAS   PRESSURE_LEAD                         USR    7
ALIAS   TIMER_LEAD                            USR    8
ALIAS   NEW_LEAD_STATE                        USR    9
ALIAS   LEAD_TURNING_ON                       USR   10
ALIAS   LEAD_TURNING_OFF                      USR   11
ALIAS   LEAD_STATE                            USR   12
ALIAS   LAG_STATE                             USR   13
ALIAS   LOCAL_P1                              USR   14
ALIAS   LOCAL_P2                              USR   15
ALIAS   TOWER_CONTROL_FAIL                    USR   16
ALIAS   LEAD_TIMER                            USR   17
ALIAS   SEQUENCE_POINTER                      USR   18
ALIAS   TRY_1_FAIL                            USR   19
ALIAS   TRY_2_FAIL                            USR   20
ALIAS   AOK                                   USR   21
ALIAS   P1_FINAL                              USR   22
ALIAS   P2_FINAL                              USR   23
ALIAS   LAST_PULSE_TIME                       USR   24
ALIAS   LAST_METER_TIME                       USR   25
ALIAS   LAST_METER_OUT                        USR   26
ALIAS   FLOWING_OUT_USR                       USR   27
ALIAS   GST_VALVE_OPEN_USR                    USR   28


$NCL


# NCL Program
#
# Client  : RE Water Corporation
# Station : Berryville Booster Pump Station
# Author  : Tatyana Mimlitz, Navionics Research Inc.
# Date    : 18 November 2002
#

# TRANSFER MODULE INPUTS TO FLAG INPUTS ...

      LBL      MAIN


# IF FIRSTRUN, INITIALIZE VARIABLES AND TIMERS ...

      FIRSTRUN?
      IF_FALSE
      GOTO     10

      SYSTIME
      STORE    LAST_PULSE_TIME
      PSTORE   LASTCALL_TIME

      LOAD     METER_OUT_MODULE
      STORE    LAST_METER_OUT
```

```
      PSTORE   METER_OUT_GAL

      LOAD     0.0
      STORE    FLOW_RATE_IN_GPM
      PSTORE   FLOW_RATE_OUT_GPM

      LOAD     0.0
      STORE    TRY_1_FAIL
      STORE    TRY_2_FAIL
      STORE    PUMP_1_FAIL
      STORE    PUMP_2_FAIL
      STORE    LGST_TIMER
      STORE    P1_FAIL_TIMER
      STORE    P2_FAIL_TIMER
      STORE    P1_DELAY_TIMER
      STORE    P2_DELAY_TIMER
      STORE    HSP_SAMPLE_TIMER
      PSTORE   PRESSURE_LEAD_ON_TIMER

      LOAD     1.0
      STORE    P1_OFF_TIMER
      STORE    P2_OFF_TIMER
      STORE    LGST_RELEASE_TIMER
      PSTORE   POWER_OK_TIMER

      LOAD     25.0
      PSDELAY  HSP_SAMPLE_TIMER

      LOAD     100.0
      SDELAY   P1_OFF_TIMER
      SDELAY   P2_OFF_TIMER
      SDELAY   P1_DELAY_TIMER
      PSDELAY  P2_DELAY_TIMER

      LOAD     120.0
      PSTORE   LGST_TIMER

      LOAD     LOW_GST_RELEASE_SECS
      PSDELAY  LGST_RELEASE_TIMER

      LOAD     900.0
      PSDELAY  PRESSURE_LEAD_ON_TIMER

      LOAD     PRESSURE_MODE_RUNTIME_HRS
      LOAD     3600.0
      *
      PSDELAY  PRESSURE_LEAD_OFF_TIMER

      LOAD     600.0
      SDELAY   P1_FAIL_TIMER
      PSDELAY  P2_FAIL_TIMER

10    POP


# IF NEW-SETPOINTS OR FIRSTRUN, SANITY CHECK THE SETPOINTS ...

      NEW_SETPOINTS?
      FIRSTRUN?
      OR
      IF_FALSE
      GOTO     20
      LOAD     PRESSURE_MODE_RUNTIME_HRS
```

```
      LOAD     3600.0
      *
      PSDELAY  PRESSURE_LEAD_OFF_TIMER

      LOAD     LOW_GST_RELEASE_SECS
      PSDELAY  LGST_RELEASE_TIMER

      GOSUB    SANITY_CHECKS
      GOSUB    MY_PUMP_SEQUENCE_SETUP

      LOAD     LEAD_PUMP_DEF
      PSTORE   CURRENT_LEAD_PUMP

20    POP


# TIME CALCULATOR ...

      SYSTIME
      LOAD     LASTCALL_TIME
      -
      PSTORE   DELTA_TIME
      SYSTIME
      PSTORE   LASTCALL_TIME


# SYSTEM UPTIME CALCULATOR ...

      UPTIME
      LOAD     60.0
      /
      PSTORE   UP_TIME_MIN


# TRANSFER MODULE STATES INTO FLAG STATES ...

      LOAD     POWER_OK_MODULE
      PSTORE   POWER_OK_TIMER
      LOAD     POWER_OK_TIMER
      PSTORE   POWER_ON

      LOAD     PUMP_FEEDBACK_MODULE
      LOAD     FLOWING_OUT_USR
      AND
      COPY
      LOAD     P1_SSR
      AND
      PSTORE   PUMP_1_ON
      LOAD     P2_SSR
      AND
      PSTORE   PUMP_2_ON

      LOAD     DISCHARGE_MODULE
      PSTORE   DISCHARGE_PSI

      LOAD     GST_MODULE
      PSTORE   GST_LEVEL_FT

      LOAD     FEED_MODULE
      PSTORE   FEED_PSI

      LOAD     ROOM_TEMP_MODULE
      PSTORE   ROOM_TEMP_DEGF
```

```
      LOAD      DISCHARGE_WORKING
      NOT
      PSTORE    DISCHARGE_TRANSDUCER_FAIL

      LOAD      GST_WORKING
      NOT
      PSTORE    GST_TRANSDUCER_FAIL

      LOAD      FEED_WORKING
      NOT
      PSTORE    FEED_TRANSDUCER_FAIL


# "FLOW RATE IN" CALCULATION (LOW-SPEED PULSE METER)...

      LOAD      METER_IN_MODULE
      LOAD      METER_IN_GAL
      -
      IF_FALSE
      GOTO      692
      LOAD      60.0
      *
      SYSTIME
      LOAD      LAST_PULSE_TIME
      -
      LOAD      1.0
      MAX
      /
      PSTORE    FLOW_RATE_IN_GPM
      SYSTIME
      PSTORE    LAST_PULSE_TIME
      LOAD      METER_IN_MODULE
      PSTORE    METER_IN_GAL
      LOAD      0.0
692   POP
      LOAD      GST_VALVE_OPEN
      LOAD      FLOW_RATE_IN_GPM
      *
      PSTORE    FLOW_RATE_IN_GPM


# "FLOW RATE OUT" CALCULATION (HIGH-SPEED PULSE METER)...

      LOAD      METER_OUT_MODULE
      PSTORE    METER_OUT_GAL

      LOAD      1.0
      PSTORE    HSP_SAMPLE_TIMER
      LOAD      HSP_SAMPLE_TIMER
      IF_FALSE
      GOTO      693
      LOAD      0.0
      PSTORE    HSP_SAMPLE_TIMER
      LOAD      1.0
      PSTORE    HSP_SAMPLE_TIMER

      LOAD      LAST_METER_OUT
      LOAD      METER_OUT_GAL
      STORE     LAST_METER_OUT
      -
      ABS
      LOAD      LAST_METER_TIME
```

```
      SYSTIME
      STORE    LAST_METER_TIME
      -
      ABS
      LOAD     1.0
      MAX
      /
      LOAD     60.0
      *
      PSTORE   FLOW_RATE_OUT_GPM
      SYSTIME
      PSTORE   LAST_METER_TIME
693   POP

      LOAD     FLOW_RATE_OUT_GPM
      LOAD     FLOW_DETECT_GPM
      Y>X?
      PSTORE   FLOWING_OUT_USR


# GST FILL VALVE HANDLER ...

      LOADA    GST_LEVEL_FT
      LOADA    GST_VALVE_OPEN_FT
      LOADA    GST_VALVE_CLOSE_FT
      LOADA    GST_VALVE_OPEN
      MACRO    HYSTERESIS_LO
      PSTORE   GST_VALVE_OPEN_USR


# GST FILL VALVE "PERCENT OPEN" CALCULATION ...

      LOAD     GST_VALVE_OPEN_PERCENT
      LOAD     VALVE_GAIN
      LOAD     VALVE_MAXSTEP
      LOAD     FEED_PSI
      LOAD     0.0
      LOAD     VALVE_FEED_LIMIT_PSI
      LOAD     0.0
      MACRO    FEEDBACK_CONTROL
      LOAD     FEED_TRANSDUCER_FAIL
      NOT
      *
      LOAD     VALVE_XDUCER_FAIL_OPEN_PERCENT
      LOAD     FEED_TRANSDUCER_FAIL
      *
      +
      LOAD     GST_VALVE_OPEN_USR
      *
      STORE    GST_VALVE_OPEN_PERCENT
      LOAD     100.0
      /
      PSTORE   VALVE_POSITION_MODULE


# VALVE FAIL CALCUALTION ...

      LOAD     GST_VALVE_OPEN_MODULE
      LOAD     GST_VALVE_OPEN_PERCENT
      LOAD     100.0
      X=Y?
      XOR
      LOAD     GST_VALVE_CLOSED_MODULE
```

```
        LOAD      GST_VALVE_OPEN_PERCENT
        LOAD      0.0
        X=Y?
        XOR
        OR
        PSTORE    GST_VALVE_FAIL_TIMER
        LOAD      GST_VALVE_FAIL_TIMER
        PSTORE    GST_VALVE_FAIL


# "VALVE OPEN" CALCULATION ...

        LOAD      GST_VALVE_CLOSED_MODULE
        NOT
        PSTORE    GST_VALVE_OPEN


# LOW GST CUTOUT CALC ...

        LOADA     GST_LEVEL_FT
        LOADA     LOW_GST_CUTOUT_FT
        LOADA     LGST_TIMER
        LOADA     LGST_RELEASE_TIMER
        MACRO     HYBRID_PRESSURE_LO
        LOAD      GST_TRANSDUCER_FAIL
        NOT
        AND
        PSTORE    LOW_GST_CUTOUT


# CHECK COMMUNICATION STATUS ...

        LOAD      COMM_TO_TOWER
        NOT
        PSTORE    COMM_FAILURE


# BPS_MODE_CALC ...

        LOADA     MODE{RADIO-PRESS-TIMER-EXT}
        LOADA     FAILOVER{PRESS-TIMER-EXT}
        LOADA     COMM_TO_TOWER
        LOADA     TOWER_TRANSDUCER_FAIL
        MACRO     BPS_MODE_CALC
        STORE     EXT_MODE
        NOT
        PSTORE    EXT_MODE_SSR
        PSTORE    TIMER_MODE
        LOAD      DISCHARGE_TRANSDUCER_FAIL
        NOT
        AND
        PSTORE    PRESSURE_MODE
        PSTORE    RADIO_MODE


# TIMER HANDLER ...

        LOAD      TIMER_1_START_HOUR
        LOAD      TIMER_1_STOP_HOUR
        BETWEEN_HOURS
        LOAD      TIMER_2_START_HOUR
        LOAD      TIMER_2_STOP_HOUR
        BETWEEN_HOURS
```

```
      LOAD      TIMER_3_START_HOUR
      LOAD      TIMER_3_STOP_HOUR
      BETWEEN_HOURS
      OR
      OR
      LOAD      TIMER_MODE
      AND
      PSTORE    TIMER_LEAD


# TOWER HANDLER ...

      LOAD      TOWER_CALL_PUMP
      LOAD      RADIO_MODE
      AND
      PSTORE    TOWER_LEAD


# PRESSURE-LEAD HANDLER ...

      LOADA     DISCHARGE_PSI
      LOADA     PRESSURE_MODE_LEAD_ON_PSI
      LOADA     PRESSURE_LEAD_ON_TIMER
      LOADA     PRESSURE_LEAD_OFF_TIMER
      MACRO     HYBRID_PRESSURE_LO
      LOAD      PRESSURE_MODE
      AND
      PSTORE    PRESSURE_LEAD


# LEAD_STATE CALC ...

      LOAD      TOWER_LEAD
      LOAD      PRESSURE_LEAD
      LOAD      TIMER_LEAD
      OR
      OR
      STORE     NEW_LEAD_STATE
      LOAD      LEAD_STATE
      NOT
      AND
      STORE     LEAD_TURNING_ON
      LOAD      NEW_LEAD_STATE
      NOT
      LOAD      LEAD_STATE
      AND
      STORE     LEAD_TURNING_OFF
      OR
      IF_FALSE
      GOTO      40
      LOAD      0
      PSTORE    LEAD_TIMER
40    POP

      LOAD      NEW_LEAD_STATE
      PSTORE    LEAD_STATE


# LOCAL_P1 & LOCAL_P2 CALC ...

      LOAD      LEAD_STATE
      LOAD      LEAD_PUMP_DEF
      LOAD      1.0
```

```
        X=Y?
        AND
        PSTORE    LOCAL_P1

        LOAD      LEAD_STATE
        LOAD      LEAD_PUMP_DEF
        LOAD      2.0
        X=Y?
        AND
        PSTORE    LOCAL_P2


# AOK CALC ...

        LOAD      POWER_ON
        LOAD      LOW_GST_CUTOUT
        NOT
        AND
        PSTORE    AOK


# FINAL P1 CALC ...

        LOAD      LOCAL_P1
        LOAD      PUMP_1{AUTO-ON-OFF}
        LOAD      2.0
        X=Y?
        OR
        LOAD      AOK
        AND
        LOAD      PUMP_1{AUTO-ON-OFF}
        LOAD      3.0
        X=Y?
        NOT
        AND
        LOAD      P2_SSR
        NOT
        AND
        PSTORE    P1_DELAY_TIMER
        LOAD      P1_DELAY_TIMER
        STORE     P1_FINAL
        NOT
        PSTORE    P1_OFF_TIMER
        LOAD      P1_OFF_TIMER
        NOT
        LOAD      P1_FINAL
        OR
        PSTORE    P1_SSR


# P1 FAIL CALC ...

        LOAD      P1_SSR
        LOAD      PUMP_1_ON
        XOR
        PSTORE    P1_FAIL_TIMER
        LOAD      P1_FAIL_TIMER
        STORE     TRY_1_FAIL
        LOAD      PUMP_1_FAIL
        LOAD      PUMP_1_ON
        NOT
        AND
        OR
```

```
      PSTORE   PUMP_1_FAIL


# FINAL P2 CALC ...

      LOAD     LOCAL_P2
      LOAD     PUMP_2{AUTO-ON-OFF}
      LOAD     2.0
      X=Y?
      OR
      LOAD     AOK
      AND
      LOAD     PUMP_2{AUTO-ON-OFF}
      LOAD     3.0
      X=Y?
      NOT
      AND
      LOAD     P1_SSR
      NOT
      AND
      PSTORE   P2_DELAY_TIMER
      LOAD     P2_DELAY_TIMER
      STORE    P2_FINAL
      NOT
      PSTORE   P2_OFF_TIMER
      LOAD     P2_OFF_TIMER
      NOT
      LOAD     P2_FINAL
      OR
      PSTORE   P2_SSR


# P2 FAIL CALC ...

      LOAD     P2_SSR
      LOAD     PUMP_2_ON
      XOR
      PSTORE   P2_FAIL_TIMER
      LOAD     P2_FAIL_TIMER
      STORE    TRY_2_FAIL
      LOAD     PUMP_2_FAIL
      LOAD     PUMP_2_ON
      NOT
      AND
      OR
      PSTORE   PUMP_2_FAIL


# VFD SPEED CALCULATION...
# Note that when the pumps are to be shut down
#  (p1_final=0 AND p2_final=0), the discharge pressure
#  limit is artificially set to zero.  This ensures that
#  the speed is tapered down to zero before pump shutdown.
#
      LOAD     VFD_SPEED_PERCENT
      LOAD     VFD_GAIN
      LOAD     VFD_MAXSTEP
      LOAD     0.0
      LOAD     DISCHARGE_PSI
      LOAD     0.0
      LOAD     VFD_DISCHARGE_LIMIT_PSI
      LOAD     P1_FINAL
      LOAD     P2_FINAL
```

```
     OR
     *
     MACRO    FEEDBACK_CONTROL
     LOAD     DISCHARGE_TRANSDUCER_FAIL
     NOT
     *
     LOAD     VFD_XDUCER_FAIL_SPEED_PERCENT
     LOAD     DISCHARGE_TRANSDUCER_FAIL
     *
     +
     LOAD     P1_SSR
     LOAD     P2_SSR
     OR
     *
     STORE    VFD_SPEED_PERCENT
     LOAD     100.0
     /
     PSTORE   VFD_SPEED_MODULE


# PUMP-1 RUNTIME ...
# (WILL ROLLOVER AFTER ~20 YEARS OF RUNTIME)

     LOAD     PUMP_1_ON
     LOAD     DELTA_TIME
     *
     LOAD     P1_RUNTIME_SECS
     +
     ABS
     LOAD     600000000.0
     MOD
     STORE    P1_RUNTIME_SECS
     LOAD     60.0
     /
     PSTORE   PUMP_1_RUNTIME_MIN


# PUMP-2 RUNTIME ...
# (WILL ROLLOVER AFTER ~20 YEARS OF RUNTIME)

     LOAD     PUMP_2_ON
     LOAD     DELTA_TIME
     *
     LOAD     P2_RUNTIME_SECS
     +
     ABS
     LOAD     600000000.0
     MOD
     STORE    P2_RUNTIME_SECS
     LOAD     60.0
     /
     PSTORE   PUMP_2_RUNTIME_MIN


# IF THE LEAD PUMP FAILS, THE LEAD HAS RUN FOR 12 HOURS,
# OR PUMP-A JUST TURNED OFF, INCREMENT ALTERNATOR ...

     LOAD     LEAD_TIMER
     LOAD     12
     Y>X?

     LOAD     LEAD_PUMP_DEF
     LOAD     1
```

```
      X=Y?
      LOAD     TRY_1_FAIL
      LOAD     PUMP_1{AUTO-ON-OFF}
      LOAD     3.0
      X=Y?
      OR
      AND

      LOAD     LEAD_PUMP_DEF
      LOAD     2
      X=Y?
      LOAD     TRY_2_FAIL
      LOAD     PUMP_2{AUTO-ON-OFF}
      LOAD     3.0
      X=Y?
      OR
      AND

      OR
      LOAD     LEAD_STATE
      AND
      LOAD     LEAD_TURNING_OFF
      OR
      OR
      LOAD     ALTERNATE_PUMPS
      AND
      IF_FALSE
      GOTO     110
      LOAD     SEQUENCE_POINTER
      LOAD     2.0
      MOD
      INCR
      PSTORE   SEQUENCE_POINTER

      LOAD     0.0
      PSTORE   LEAD_TIMER

      GOSUB    MY_PUMP_SEQUENCE_SETUP

      LOAD     LEAD_PUMP_DEF
      PSTORE   CURRENT_LEAD_PUMP
110   POP


# INCREMENT LEAD TIMER ...

      LOAD     LEAD_STATE
      IF_FALSE
      GOTO     555
      LOAD     LEAD_TIMER
      LOAD     DELTA_TIME
      LOAD     3600
      /
      +
      PSTORE   LEAD_TIMER
555   POP

      END


# ==========================================
#
# ADDITIONAL SUBROUTINES...
```

```
#
# ==========================================

     LBL       SANITY_CHECKS

     LOADA     LEAD_PUMP{P1-P2}
     LOAD      2.0
     LOAD      1.0
     MACRO     BOUNDS_CHECK

     LOADA     LAG_PUMP{P1-P2}
     LOAD      2.0
     LOAD      1.0
     MACRO     BOUNDS_CHECK

     LOADA     PUMP_1{AUTO-ON-OFF}
     LOAD      3.0
     LOAD      1.0
     MACRO     BOUNDS_CHECK

     LOADA     PUMP_2{AUTO-ON-OFF}
     LOAD      3.0
     LOAD      1.0
     MACRO     BOUNDS_CHECK

     LOADA     MODE{RADIO-PRESS-TIMER-EXT}
     LOAD      4.0
     LOAD      1.0
     MACRO     BOUNDS_CHECK

     LOADA     FAILOVER{PRESS-TIMER-EXT}
     LOAD      3.0
     LOAD      1.0
     MACRO     BOUNDS_CHECK

     RTN

# ==========================================

     LBL       MY_PUMP_SEQUENCE_SETUP

     LOADA     ALTERNATE_PUMPS
     LOADA     SEQUENCE_POINTER
     LOADA     LEAD_PUMP{P1-P2}
     LOADA     LAG_PUMP{P1-P2}
     LOADA     LEAD_PUMP_DEF
     LOADA     LAG_PUMP_DEF
     MACRO     PUMP_SEQUENCE_SETUP2
     RTN

# ==========================================
```