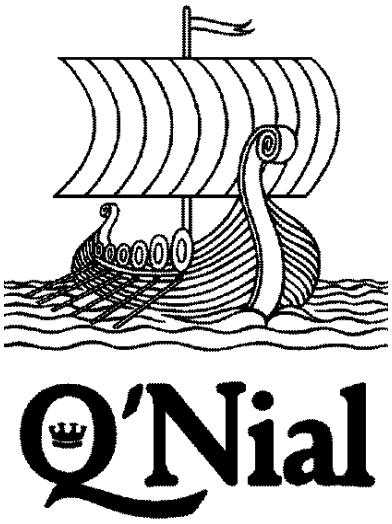


Using the Nial Data Engine

Version 6.3

August 2005

Preface

This Manual explains the use of the *Nial Data Engine* for Windows as a DLL from an application with Visual Basic capability. The approach is illustrated with in an application that has a Visual Basic User interface. The application is a simple financial planner for doing pension and investment computations.

Chapter 1 The DLL interface

The Nial Data Engine provides four main routines as its interface to the functionality of the Q'Nial Interpreter for the following tasks:

- initializing the interpreter
- executing a Nial action
- getting the output of the last Nial action
- stopping the interpreter

In addition to these main routines there are a number of other routines that can be used to initialize an environment that needs to control the interface to the Q'Nial interpreter and the execution of actions in more detail. The routines that implement these interfaces are described in the following sections.

The following table summarizes the complete interface available from the DLL. Only a few of these routines are needed for Visual Basic applications. The others are used in constructing the Nial Tools from one base of code.

Routine	**Purpose**
NC_CommandInterpret	requests the Interpreter to execute an action
NC_CopyBuffer	copies the output buffer to an given area
NC_CreateIOContext	set up interface I/O mode
NC_CreateSessionSettings	set up a structure for Session Settings
NC_CreateWindowSettings	set up a structure for Window Settings
NC_DestroyIOContext	clear a given I/O Context
NC_DestroySessionSettings	free a Session Settings structure
NC_DestroyWindowSettings	free a Window Settings structure
NC_GetBuffer	get the I/O Buffer
NC_GetPrompt	get the current prompt
NC_GetSessionSetting	get a specific Session Setting
NC_GetWindowSetting	get a specific Window Setting
NC_InitNial	initialize the interpreter
NC_LoadLibrary	load a DLL library
NC_ResetBuffer	reset the I/O buffer
NC_SetBufferSize	set the I/O buffer size
NC_SetCheckUserBreak	set the frequency of break checks
NC_SetDebugLevel	used to control internal Q'Nial debugging
NC_SetIOMode	sets the I/O mode
NC_SetNialRoot	sets the Nialroot string
NC_SetReadCharCommand	sets the routine for Readchar
NC_SetReadStringCommand	sets the routine for Readchars
NC_SetSessionSetting	set a specific Session Setting
NC_SetWindowSetting	set a specific Window Setting
NC_SetWriteCommand	sets the routine for writechars
NC_StopNial	stops the interpreter

Initializing the Q'Nial Interpreter

The behaviour of the interpreter depends on a number of parameters that have default settings, but can be set to values at startup and/or under program control. The parameters are divided into two groups: Session Settings and Window Settings. The former are parameters that are set for the Session and are initialized once; whereas the latter are reestablished at every return to the top level and can be different for different windows.

The Session Settings are:

Name	**Code**	**Default**	**Purpose**
Workspace_size	100	200000	initial size or workspace in words
Initial_defs	101	‘’	name of initial definition file
Quiet	102	False	switch to turn off banner
Expansion	103	True	allows expansion of workspace
Initial_workspace	104	‘clearws’	name of initial workspace
debugging_on	105	True	allows Nial level debugging
Session_version	106	0	which Session settings are in use
form_name	107	‘’	name of a Nial form file to process (CGI-Nial)

The Window Settings are:

Name	**Code**	**Default**	**Purpose**
triggered	400	True	allows faults to trigger an interrupt
nointerrupts	401	False	prevents interrupts
sketch	402	True	sets sketch/diagram mode
decor	403	False	sets decor/nodecor mode
messages	404	True	allows system messages to be displayed
debug_messages	405	False	allows internal debug messages to be displayed
trace	406	False	turns on expression tracing
box_chars	407	False	uses DOS line_drawing characters in pictures
log	408	False	allows logging of input/output for the window
logname	409	‘auto.nlg’	name of log file
format	410	‘%g’	real number format string
prompt	411	‘ ‘	prompt string at top level
screen_width	412	80	width of window in characters
Window_version	413	0	which Window settings are in use
screen_height	415	25	number of lines in window
use_history]var	416	True	whether the previous value can be captured using

The Sessions Settings and Windows Settings are stored in structures and the interpreter maintains an array of structures of each type. The first entry in each of the arrays is prebuilt and holds the default values for its settings. The routines *NC_CreateSessionSettings* and *NC_CreateWindowSettings* can be used to set up additional versions of the settings. The routines *NC_SetSessionSetting* and *NC_SetWindowSetting* are used to set a specific value in one of the arrays of structures. The routines *NC_GetSessionSetting* and *NC_GetWindowSetting* are used to retrieve a setting from the arrays of structures.

The interpreter has three different modes for handling buffers for input and output and two modes for controlling input/output. The buffer and I/O mode are stored in an I/O Context structure.

Mode	Code	Effect
Internal_buffer	1	output to an internal buffer (for NSL debugging use)
Buffer	2	output to a provided buffer
Output on	4	uses provided write function for output (console & GUI versions)
Output off	5	output thrown away

The routine *NC_Create_IOContext* is used to set up a structure that holds the Mode and routines to support readchar, readstring and writestring. An array of IOContexts is supported and the first one is set to default to using Buffer Mode.

The Visual Basic code to support initialization consists of the following pair of routines:

```
Rem --- Nial Initialization routine
 Declare Function XXInitNial Lib "nde32.dll" Alias "NC_INITNIAL" (ByVal
a As Long, ByVal b As Long, ByVal c As Long) As Long

Rem -- Cover function to set the third argument to 0,
Rem -- which means use BUFFER mode
Function InitNial(SS, WS) As Long
   InitNial = XXInitNial(SS, WS, 0)
End Function
```

The parameters to the function InitNial are integers representing the indices of the Settings structures to be used. The usual call is of the form

```
InitNial(0,0)
```

which starts the interpreter with the default settings. There are Visual Basic cover functions to allow access to the DLL routines for creating Settings structures and to set and modify the Settings. However, they are not needed for a simple application interface. See the file *nde32_ap.bas* for details.

The initialization routine defaults to using Buffer mode for output. This requires one additional function to set the Buffer size.

```
Declare Function SetBufferSize Lib "nde32.dll" Alias "NC_SETBUFFERSIZE"
(ByVal initsize As Long, ByVal incrementnn As Long) As Long
```

This routine sets an initial Buffer size and an incremental amount to increase it by if the Buffer needs to grow. The Buffer size can be reset using the function:

```
Declare Function ResetBuffer Lib "nde32.dll" Alias "NC_RESETBUFFER" ()
As Long
```

which resets the Buffer size to the initial size.

Using the Interpreter

The main interface to the Interpreter is through a command that executes a string of Nial program text. The Basic functions that support this are:

```
Declare Function XXCommandInterpret Lib "nde32.dll" Alias
"NC_COMMANDINTERPRET" (ByVal comm As String, ByVal b As Long, ByVal c
As Long) As Long

Rem -- Cover function to set the third argument to 0,
Rem -- which means use BUFFER output mode

Function CommandInterpret(ByVal inp As String, WS As Long) As Long
   CommandInterpret = XXCommandInterpret(inp, WS, 0)
End Function
```

The second argument to the function *CommandInterpret* is the WindowSetting index, which will be 0 if the default settings are in use. The effect of a call on the function is to execute the string of Nial program text in the context of the active workspace. This can be a *loaddefs* command to load program text or it can be an action that does a computation. If the result of the execution is an array value its picture is stored in the output Buffer.

Obtaining the Result

The following Basic function is provided to obtain the contents of the Buffer.

```
Declare Function CopyBuffer Lib "nde32.dll" Alias "NC_COPYBUFFER"
(ByVal buf As String, ByVal msize As Long) As Long
```

The contents of the internal Buffer are copied to the Basic String whose name is provided as the first argument. The argument msize gives the maximum amount of data that can be copied. The output produced by the interpreter includes the newline characters needed to produce a tabular display. Thus copying the output to the text field of a Visual Basic form results in the display of the same array picture that would be displayed by Q'Nial.

Exiting the Interpreter

When the application has finished its work that requires the Nial Interpreter, the Interpreter should be exited to clear up memory space. This is accomplished using the following Basic function.

```
Declare Function StopNial Lib "nde32.dll" Alias "NC_STOPNIAL" () As
Long
```

If the application terminates without calling *StopNial* then the DLL is closed and the space in use will be freed automatically.

Chapter 2 The Application

In this chapter we describe the interface used to build an application involving pension and investment computations. The application was initially built by Mike Jenkins as a pure Nial application for his own use. The Visual Basic interface was designed to make the application attractive to a financial planner who wished to use it with other clients.

The details of the Nial code will not be explained in detail. The basic idea is that four main computations are provided that do the following:

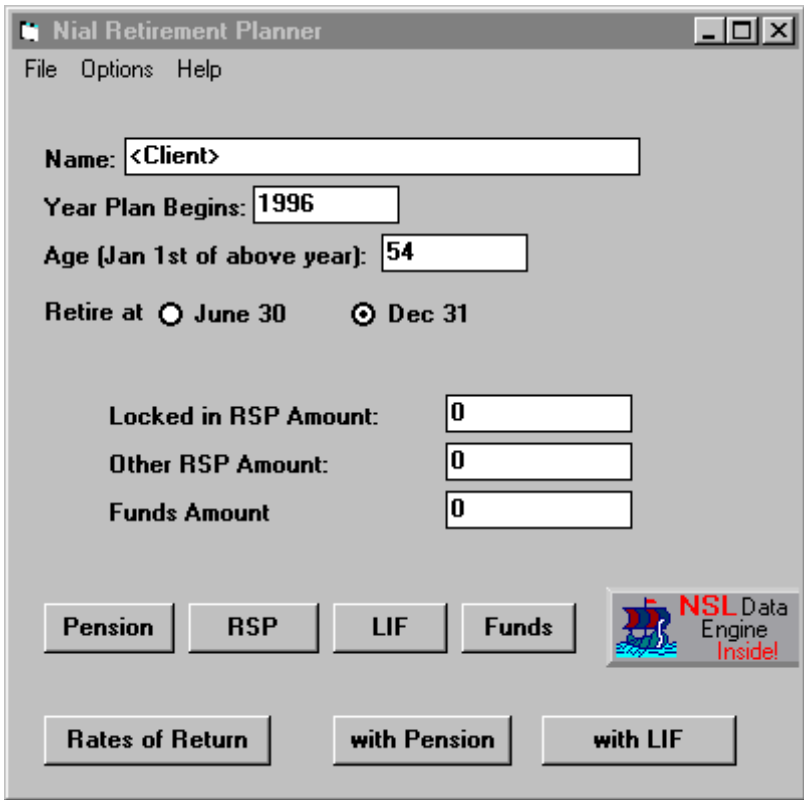
- estimate the revenue from the Queen's University Pension Plan (Pension)
- estimate the revenue and value from a Registered Retirement Savings Plan (RRSP)
- estimate the revenue and value from a Locked-in RRSP (LIF)
- estimate the revenue and value from a non-registered investment (Funds)

The application can report the results of each of the computations and can produce a summary using either the QPP or the LIF with the other computations.

The results of the various applications depend on parameters such as retirement age, number of years projected, assumed interest rates, etc. In the Nial code these are parameters to the operations and are provided explicitly when the routines are called. In the Visual Basic application they are provided by filling in fields on forms. The major work of the interface code written in Basic is to build the calls to the Nial operations using the data provided in the forms.

The application was developed with Visual Basic 3 and later converted to Visual Basic 4.

The Main Form



The above form provides the first level of interface to the application. The client fills in his/her personal information and the amounts in each of the categories. The code attached to this form does the initialization of the Data Engine. It is

```
Private Sub Form_Load()
   Dim rc1, rc2 As Long
   '//-- Start the interpreter with no particular option
   rc1 = InitNial(0, 0)
   '//-- Load in the code to do the calculations
   rc2 = CommandInterpret("loaddefs ""c:\planner\models", 0)
   '//-- Error Check the result codes

End Sub
```

This routine is executed when the Main Form is loaded. It calls *StartNial* to initialize the interpreter. It then issues the Nial expression

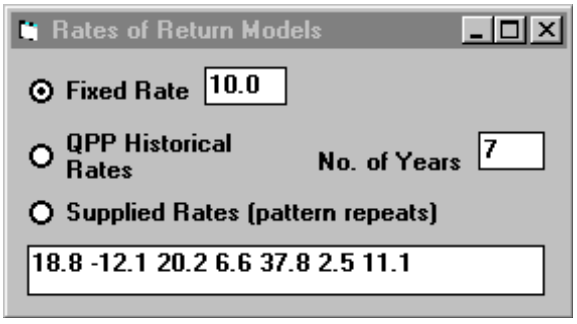
```
loaddefs "c:\planner\models
```

using *CommandInterpret* to load the application code written in Nial into the active workspace.

There are seven forms associated with the application:

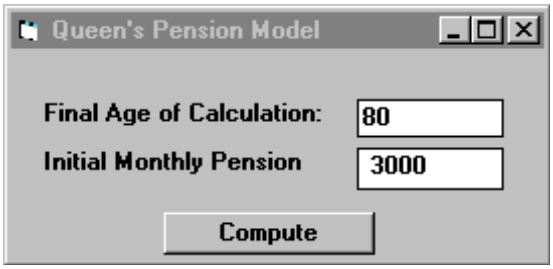
Form	**File name**	**Purpose**
Form 1	planner.frm	Main application form
Form 2	qnsmodel.frm	Sets up and computes the Pension model
Form 3	respmodel.frm	Sets up and computes the RRSP model
Form 4	lifmodel.frm	Sets up and computes the LIF model
Form 5	output.frm	Used for Output display of all of the compute forms
Form 6	funds.frm	Sets up and computes the Funds model
Form 7	rates.frm	Sets the interest rate assumptions for the computations

After filling in the fields in Form 1 the client usually clicks on the *Rate of Return* button. This pops up Form 2.



The user selects between one of the 3 forms of rates of return assumptions. The indicated data will then be used in all the computational models. There is no computation attached to this form.

The next step by the client is to click on either the Pension or LIF button on Form 1. Assume the Pension button is pressed. Then Form 2 pops up.



The client inserts the appropriate data and clicks on the *Compute* button. This triggers the following computation attached to this button. It is the following code:

```
Private Sub Command1_Click()
  Dim rc1 As Long
  Dim outbuf As String * 5000
  If Form7.Option1.Value Then
    Rates = Form7.Text1.Text
  ElseIf Form7.Option2.Value Then
    Rates = "(reverse (" + Form7.Text2.Text + " take Qupp))"
  Else
    Rates = "(" + Form7.Text3.Text + ")"
  End If
  rc3 = CommandInterpret("Client := '" + Form1.Text1 + "';", 0)
  com = "qpreport qpmodel "
  com = com + Form1.Text3.Text + " "
  com = com + Form1.Text2.Text + " "
  com = com + Form2.Text3.Text + " "
  com = com + Rates + " "
  com = com + Form2.Text2.Text + " "
  If Form1.Option1.Value Then
    com = com + "True "
  Else
    com = com + "False "
  End If
  rc1 = CommandInterpret(com, 0)
  rc1 = CopyBuffer(outbuf, 4999)
  Form5.Text1 = outbuf
  Form5.Visible = True
End Sub
```

The first step in the code is to choose between the rate of return models in order to set up a Basic string that will correspond to the *Rates* parameter for the call on the Nial operation *qpmodel*. This is either a single number in the first option or a list of numbers enclosed in parentheses in the second two options.

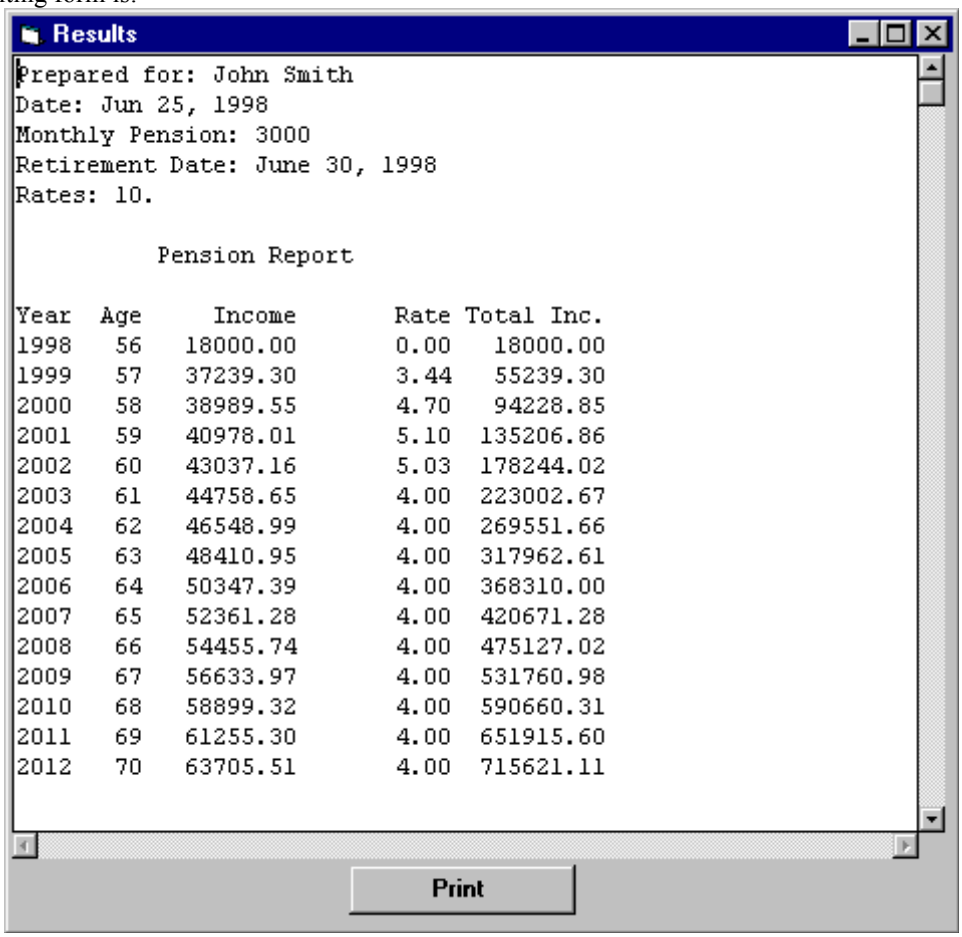
The next step is to call *CommandInterpret* to set up the *Client* variable in Nial to have the string given in Form1.Text1.

The third step is to build up the Nial expression that will do the computation and produce the output report. This involves concatenating the various fields from Forms 1 and 2 and preceding them with the operation to be applied. For example the expression might be

```
qpreport qpmodel 56 1998 3000 10.0 70 True
```

which would indicate Age 56 in 1998, Initial Monthly Pension $3,000, Rate 10.0%, compute to age 70, assuming June 30 retirement. The expression is computed using *CommandInterpret* and the result is

copied to the buffer *outbuf*. Then the output is assigned to the Text1 field in Form 5 and the latter is made visible.
The resulting form is:



The client continues with the computations for the RRSP form and the Funds form and then selects the Summary button *with Pension*. Each of the Compute and Summary buttons has as associated computation similar in form to the above code. To see the details run Visual Basic using the Planner.mak file and examine the code associated a click on each of the buttons.