# *NET-Integrator*

# User Guide

Rev. 1.7

Software Installation
Sample Programs
Object Definitions

LDS-Dactron

Phone. (408) 934-9160
Fax. (408) 934-9161
E-mail: technical.support@lds.spx.com
Web Site: www.lds-group.com

## Notice

# Table of Contents

# Introduction

## *This Manual*

This manual provides instructions for installing and using LDS-Dactron's NET™ software. It also describes the technical background of the technology and its application. For the NET-Integrator™ software, detailed function descriptions, naming conventions, and samples are provided.

## *Introducing Network Enabled Test™*

NET is LDS-Dactron's network-enabled data acquisition and real-time test product. NET consists of new hardware and software components that are designed for the Microsoft Windows 2000/XP environments. In contrast to traditional stand-alone instruments or software applications, NET allows the user to customize and integrate multiple devices together via Ethernet. The features of NET are:

- A highly scalable and customizable system
- The latest component software technology
- Enriched real time processing functions

LDS-Dactron data acquisition devices, dynamic signal analyzers, and vibration controllers are integrated and connected to other applications using Microsoft's ActiveX and COM technologies. The data is acquired and processed in real-time locally on the device, and then viewed and processed, and stored anywhere on the network. NET software is supported by LDS-Dactron's Photon, Focus, LASER, and COMET systems running either shaker control or dynamic signal analysis applications.

NET software consists of three components: ***NET-Remote***™, ***NET-View***™**,** and ***NET-Integrator***™.

- ***NET-Remote*** enables the LDS-Dactron Windows application, either Shaker Control software or RT Pro signal analysis software, to run remotely over an Ethernet network.
- ***NET-View*** allows the user to view testing results remotely without interrupting the processes on the computer controlling the testing.
- ***NET-Integrator*** utilizes COM and ActiveX technology to provide an integrated test environment in which multiple software applications can be operated and controlled concurrently.

For details about the network and DCOM configuration, please refer to following document:

Network and DCOM Configuration.doc

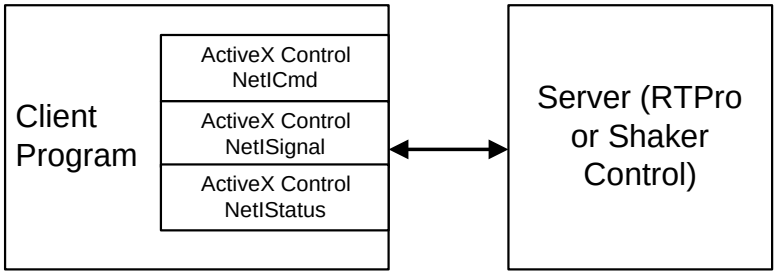
# NET-Integrator Overview

A basic knowledge of ActiveX controls and basic programming skills in Visual Basic or Visual C++ is very helpful in reading the sections of this manual describing NET-Integrator.

## *What is NET-Integrator?*

Based on ActiveX technology from Microsoft, NET-Integrator is a software tool that allows the user to control the operation and access the testing results of LDS-Dactron software applications. The user can create a customized user interface (*The Client*) in Visual Basic, Visual C++, LabView, MS-Excel, or MS-Word to control the LDS-Dactron applications (*The Server*) either locally or remotely through an *ActiveX control* provided by LDS-Dactron.

The relationship between the *Client*, the *Server*, and the *ActiveX control* is described in the following picture:

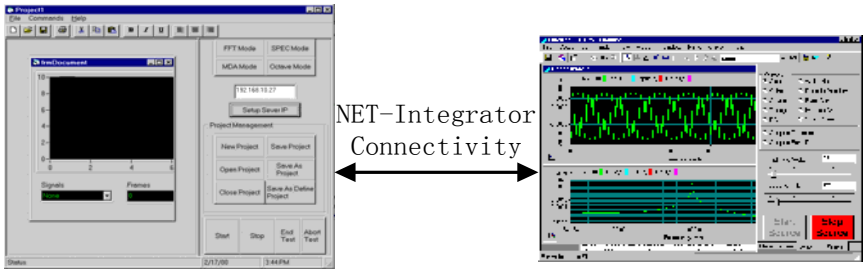



LDS-Dactron provides three ActiveX controls: **`NetICmd`**, **`NetISignal`** and **`NetIStatus`**. When in use, these ActiveX controls are embedded in the Client application and communicate with the Server. Microsoft's ActiveX controls are similar in concept to the Dynamic Link Library (DLL) or the shared library type of files. They provide additional benefits such as better version control and the ability to operate as abstract objects. In addition, ActiveX controls can also be used with other Windows applications such as Microsoft Word, Excel, or Internet Explorer. This provides access and control of the system via the Web.

With NET-Integrator, multiple LDS-Dactron applications can be operated simultaneously. Typically, in a complex measurement environment the measurement system is divided into small groups. Each group takes 1 to 16 measurement channels with its own test setup. The ActiveX container broadcasts the commands to each Windows applications and receives the status and measurement results from them. For example, an ActiveX Client can execute the following functions in the RTPro or LDS-Dactron Shaker Control applications:

- Open and start a specific project file (execute **Project Open/Project.prj** command)
- Receive messages posted from a server application
- Send commands to the application such as Start or Stop
- Retrieve and display the run status from the application
- Retrieve and display the data arrays from the application

The NET-Integrator client talks directly to the RTPro or Shaker Control applications (*The Server*). The client can be as simple as just two or three command buttons or as complicated as the RTPro application with a full-function user interface.




The NET-Integrator Client and the applications it controls can run either on the same computer or on different computers through a TCP/IP connection.

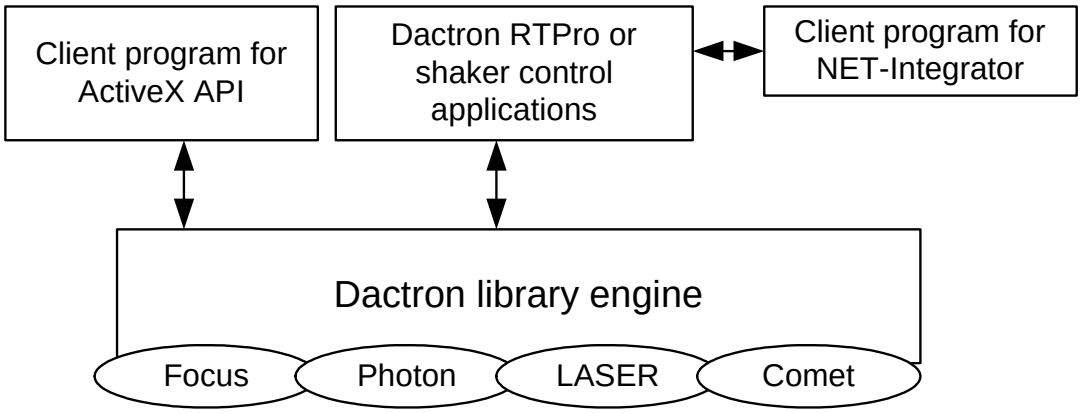
To execute the NET-Integrator functions, the RTPro or Shaker Control application must have NET-Integrator connectivity enabled. NET-Integrator includes a file called "`NetIntegrator.OCX`". This OCX file includes three ActiveX controls that are registered on the NET-Integrator Client computer.

The NET-Integrator product is also shipped with multiple samples built using Visual Basic and Visual C++. See Section 4, “NET-Integrator Sample Projects”, for details.

## *What’s the Difference Between NET-Integrator and ActiveX API?*

In addition to NET-Integrator, LDS-Dactron offers another line of product that is based on ActiveX technology, LDS-Dactron ActiveX API. API stands for Application Programming Interface. Both product lines are targeting at increasing the connectivity and flexibility of LDS-Dactron products. The differences are:

1. The client program of ActiveX API communicates to the library engine while that of NET-Integrator communicates to the user-interface level of RTPro or shaker controller. In the other words, the NET-Integrator simply replaces the operator to handle the project through LDS-Dactron application. The following picture shows the structure:




2. In ActiveX API, only a limited number of data acquisition and spectral analysis functions are available. On the other hand, NET-Integrator can utilize all the functions available in RTPro and shaker control.

## *OS Requirements for NET-Integrator*

NET-Integrator operates on any of the following operating systems.

- Windows 2000
- Windows XP

## *What is an ActiveX control?*

An ActiveX control is an object that supports a customizable, programmable interface. Using the ***Methods***, ***Events***, and ***Properties*** of a control, Web authors can automate their HTML pages. Examples of ActiveX controls include text boxes, command buttons, audio players, video players, stock tickers, and so on.

You can develop ActiveX controls using Microsoft Visual Basic 5.0 and later, Microsoft Visual C++, MatLab, VEE, LabView and Java etc..

### Properties

An ActiveX control fires events to communicate with its control container. The container, in return, uses methods and properties to communicate with the control. Methods and properties are similar in use and purpose, respectively, to member functions and member variables of a C++ class. Properties are data members of the ActiveX control that are exposed to any container. Properties provide an interface for applications that contain ActiveX controls, such as Automation clients and ActiveX control containers.

### Methods

An ActiveX control fires events to communicate between itself and its control container. A container can also communicate with a control by means of methods and properties. Methods and properties provide an exported interface for use by other applications, such as **Automation** clients and ActiveX control containers.

### Events

ActiveX controls use events to notify a container that something has happened to the control. Common examples of events include clicks on the control, data entered using the keyboard, and changes in the control's state. When these actions occur, the control fires an event to alert the container.

## *Implementation of the ActiveX controls*

Three types of the ActiveX controls are implemented in NET-Integrator. They are `NetICmd`, `NetISignal`, and `NetIStatus`

- `NetICmd`: Used for sending commands to a server application such as RTPro.
- `NetISignal`: Used for reading signal attributes and signal data from the server application. It also can fire the `VPU_ALLOC_SIGNAL_READY` and `VPU_UPDATE_SIGNAL` messages from the server to the client.
- `NetIStatus`: Used for reading the test status from the server application. It can fire the `VPU_UPDATE_STATUS` message from the server to the client.

A detailed description of the *Properties*, *Methods* and *Events* of these ActiveX controls is included in Section 3, "ACTIVEX Controls."

# NET-Integrator Installation

NET-Integrator consists of three fundamental components: A ***client program***, *a* ***LDS-Dactron OCX control***, and ***the server application*** that has NET-Integrator connectivity enabled.

## Client Program

The user creates his own client program in Visual Basic or Visual C++ or any ActiveX container.

## OCX Control

Install and register the `NetIntegrator.OCX` component, following the instructions below.

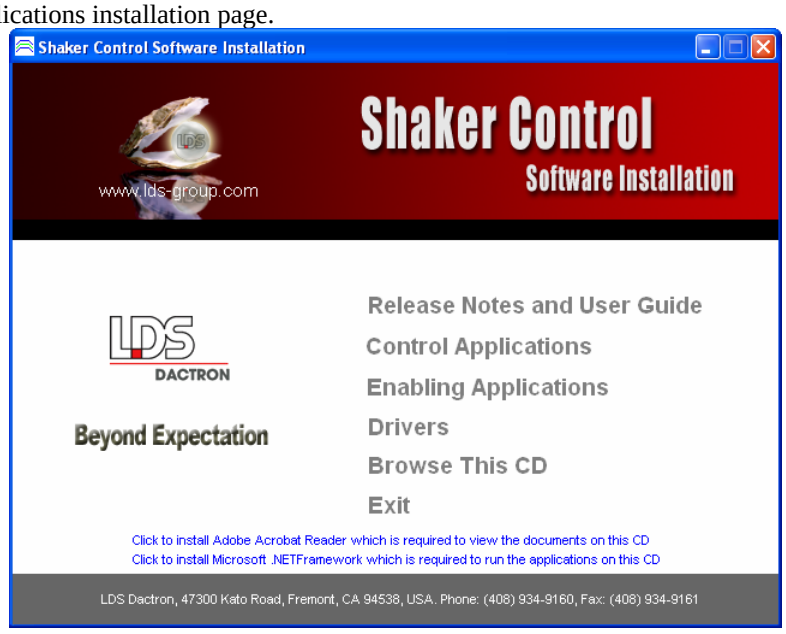
## Enable NET-Integrator on the Server Application

The application (RTPro or Shaker Control) that shipped with NET-Integrator from LDS-Dactron already has this function enabled. The client program will be able to communicate with it through an ActiveX control.

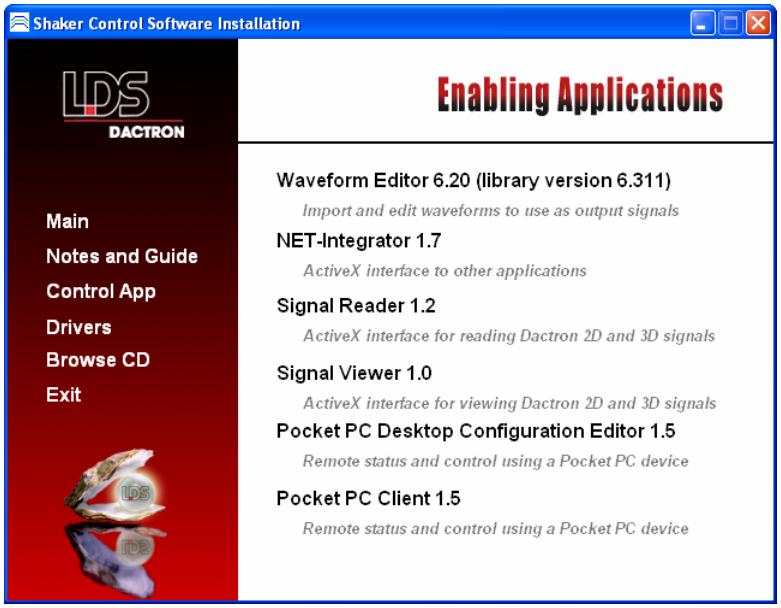
### *Install the OCX Control*

1. Insert the LDS-Dactron Shaker Control installation CD into the PC's CD-ROM drive. The Shaker Control Software Installation screen will automatically start.

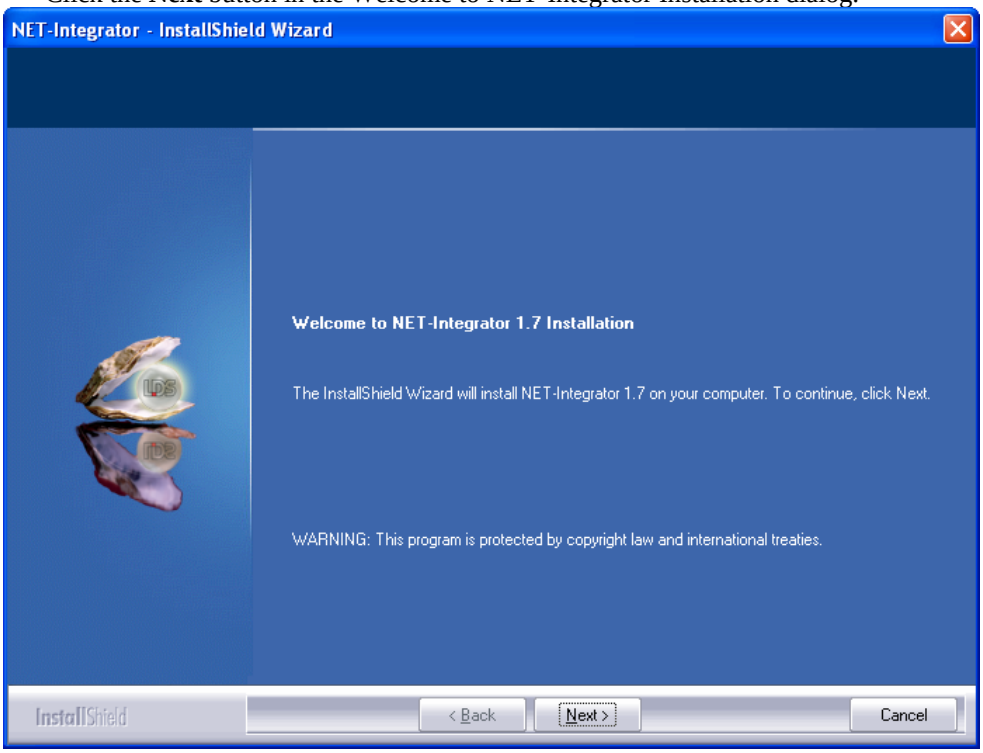
2. On the main page, click on the **Enabling Applications** link to continue to the Enabling Applications installation page.



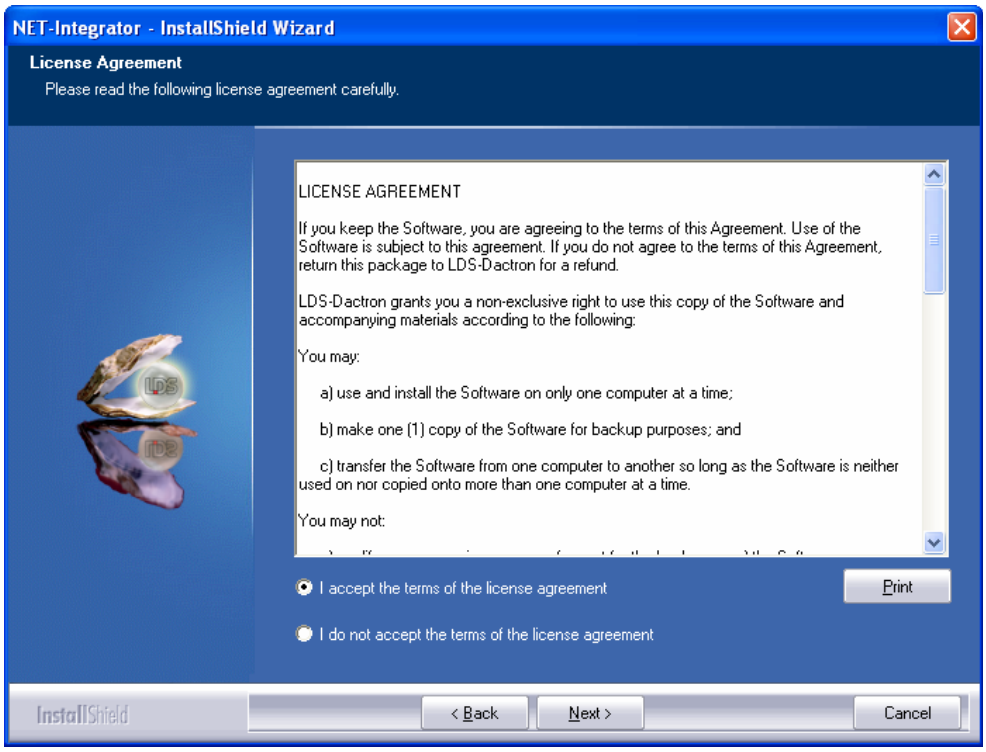

3. On the Enabling Applications page, click on the **NET-Integrator** link to install the NET-Integrator software.



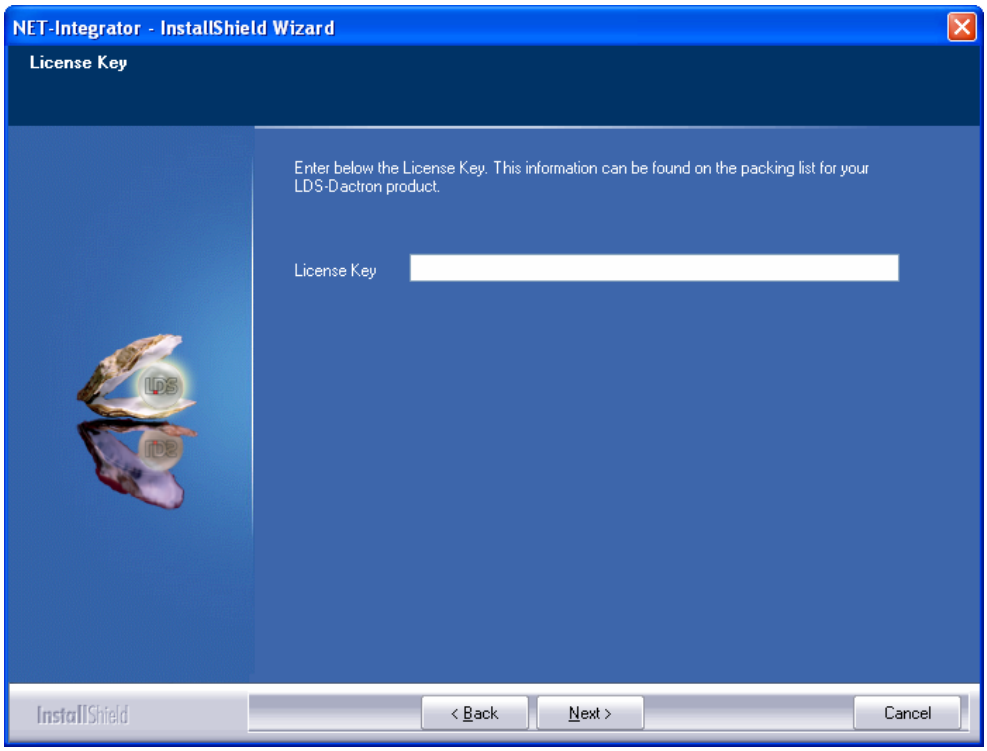

4. Click the **Next** button in the Welcome to NET-Integrator Installation dialog.



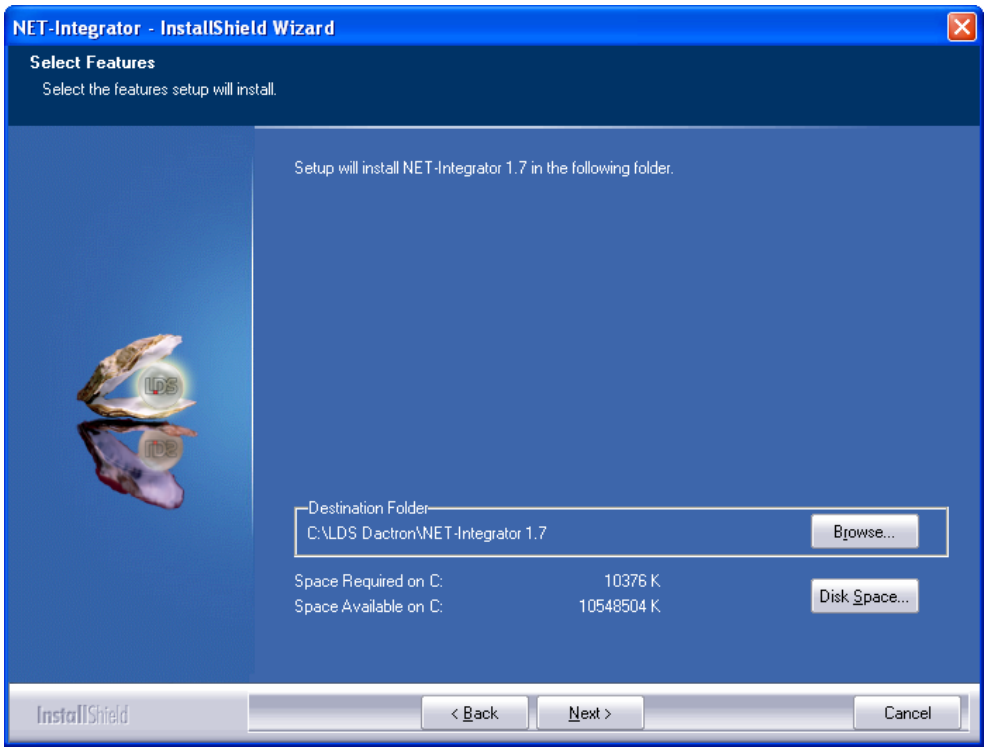
5. Select the **I accept the terms of the license agreement** button to accept the License Agreement then click the **Next** button.



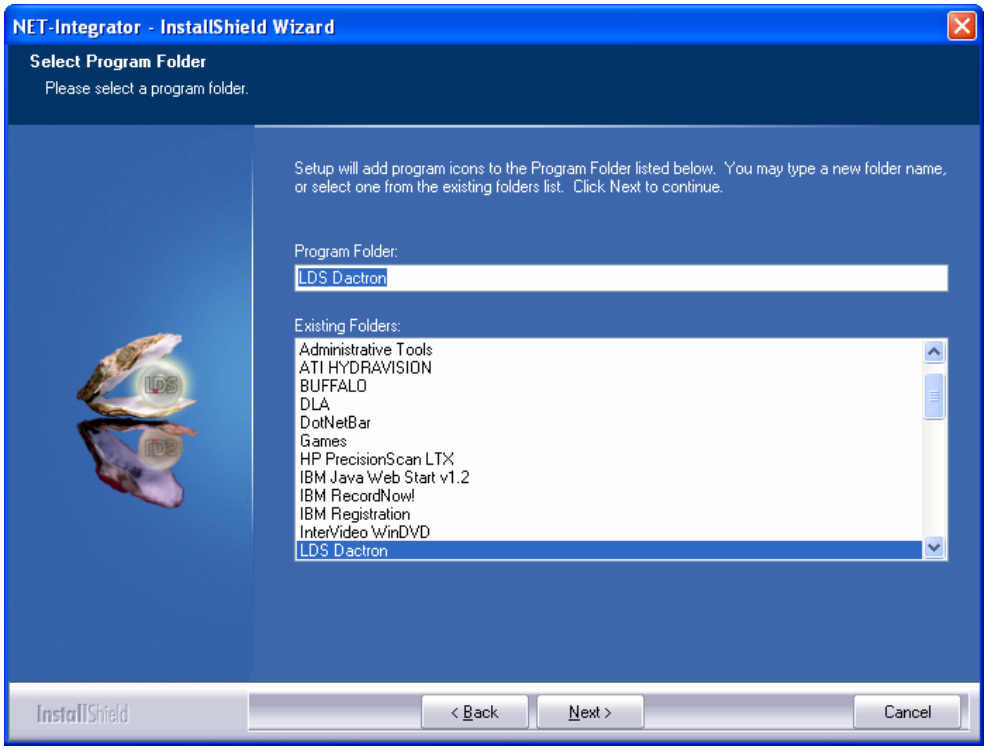

6. Type in your **NET-Integrator License Key** then click on the **Next** button.



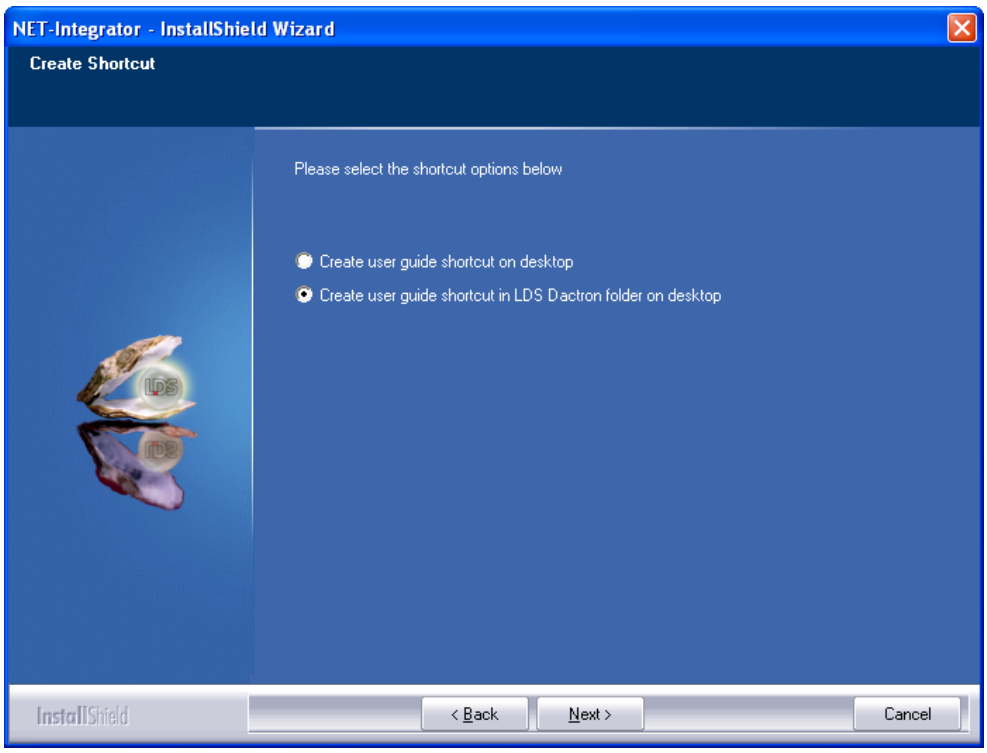
7. Assign the **Destination Folder** then click on the **Next** button.,



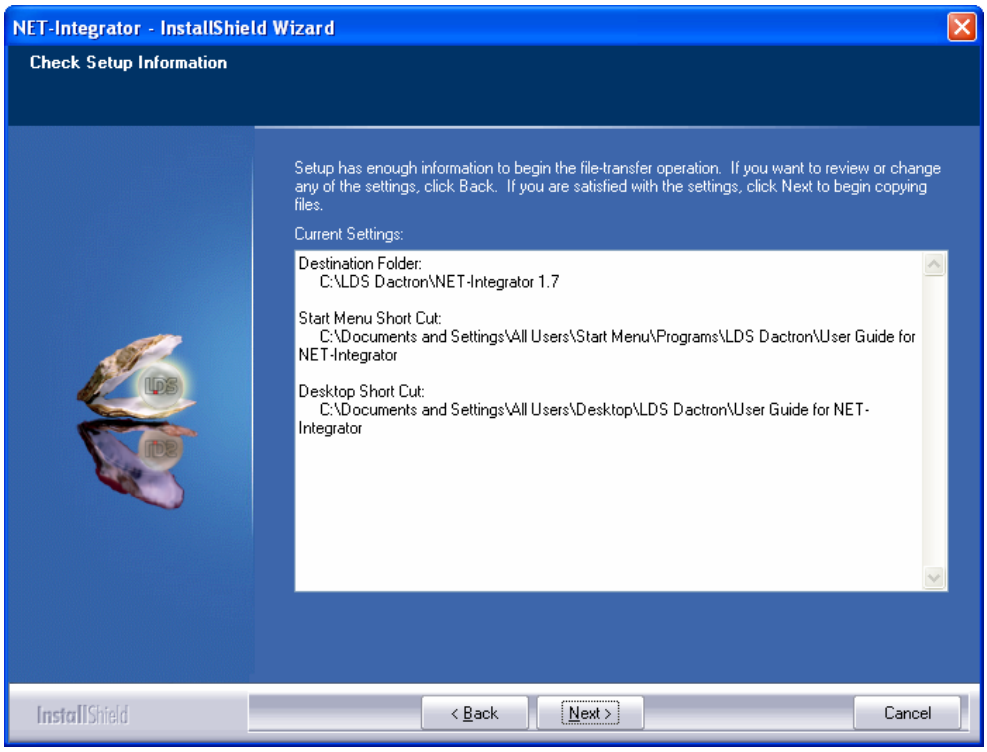

8. Select the **Program Folder** then click on the **Next** button.



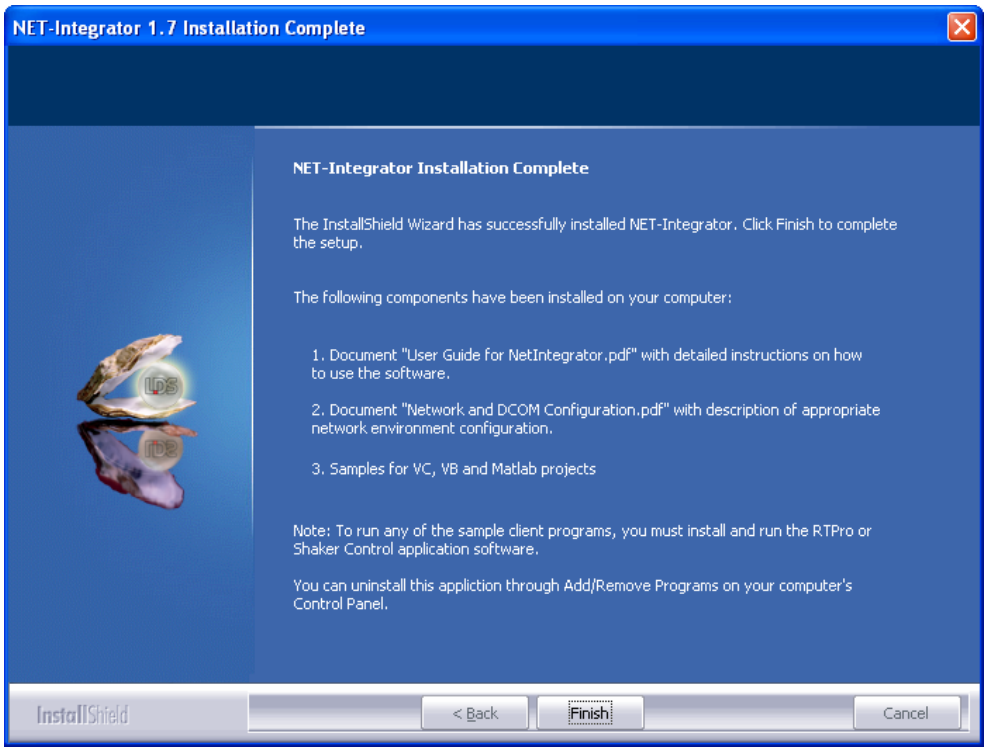
9. Select the shortcut preference for the NET-Integrator User Guide then click on **Next**.




10. Verify your NET-Integrator installation settings, if everything is OK click on **Next**.



11. Complete the NET-Integrator installation by clicking on **Finish**.



### *A Quick Example in Visual Basic*

Here is the general function sequence when the NET-Integrator ActiveX controls are used:

Step 1: Assign the value to property `ApplicationType` of `NetICmd`, `NetIStatus`, and `NetISignal`. The remote server can be RTPro or Shaker Control.

Step 2: Assign the value to property `AppComputerIP` of `NetICmd`, `NetIStatus`, and `NetISignal`.

Step 3: Call method `ConnectToServerApp` to connect the server application such as RTPro or Shaker Control for each of these ActiveX controls.

Step 4: Assign the value to property `ProjectType` for `NetICmd` control.

Step 5: With `NetICmd` control, send the command `CMD_NEWPRJ` to a new project based on the `ProjectType`, or send the command `CMD_OPENPRJ` to open an existing project.

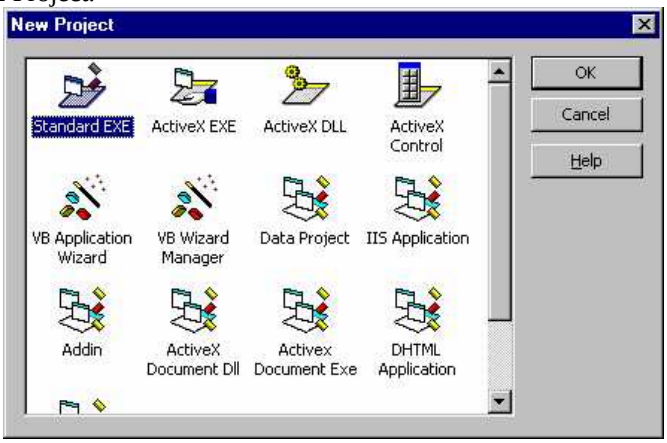
Step 6: With `NetICmd` control, send the command `CMD_STARTMEAS` to start the test.

Step 7: Retrieve the testing status or signals by using the events of `NetIStatus` or `NetISignals`.

The following section shows how to make a very simple program in Microsoft Visual Basic 6.0.

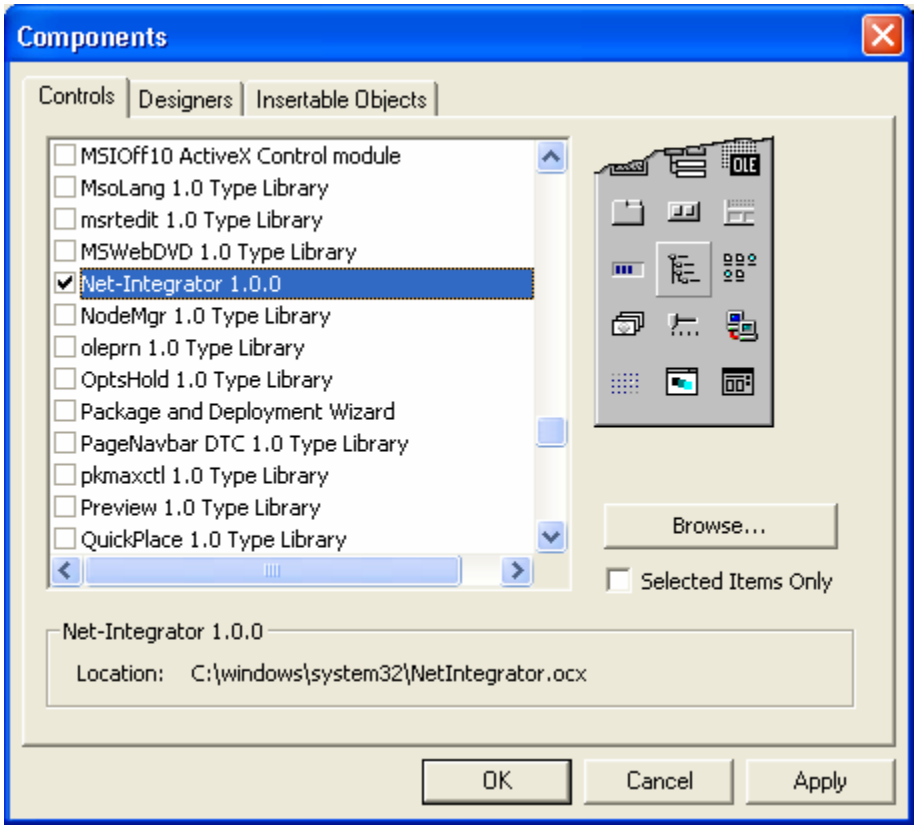
## Step 1. Open Microsoft Visual Basic V6.0

Create a **Standard.EXE** Project.




## Step 2: Project Settings

On the VB Menu Bar, Click Project.
Near the bottom of the menu, click Components...
Since you have already registered ***NetIntegrator.OCX***, the specific library is automatically inserted into Visual Basic Project's Component. On the Control tab, scroll down the list to find the Net-Integrator 1.0.0 check box and check it.



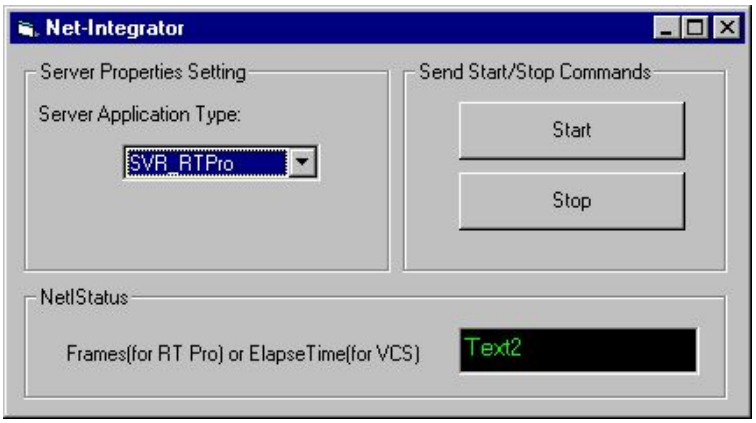
Click OK to close the Components dialog box.

Once the box is checked, three new icons 

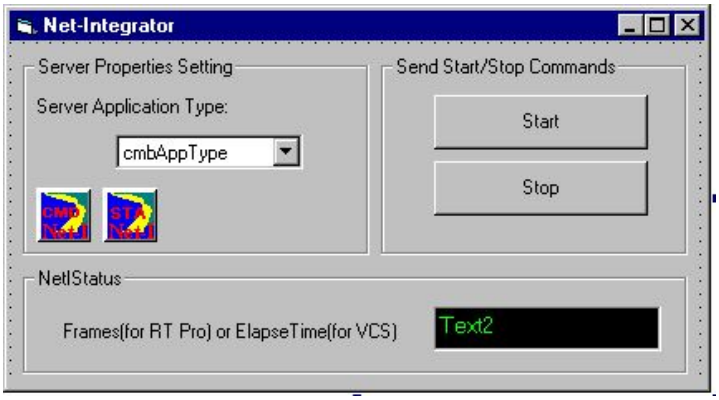
will come up on the Control Bar. They are the LDS-Dactron NET-Integrator ActiveX controls.

## Step 3. Design a Graphical User Interface

This example contains a Start button, a Stop button, a combo box allowing the user to select whether the RTPro or Shaker Control application is to be controlled, a text field to display the Frame Number of RTPro or the Elapse Time of Shaker Controller, and some text descriptions.
On the Visual Basic Form, insert all the controls as shown in the following dialog:




Change the combo box's Name to "`cmbAppType`", Style to "`2-Dropdown List`" and add the two list strings to List :"`SVR_RTPro`","`SVR_VCS`".
Add two command buttons and change Name to "`cmdStart`" and "`cmdStop`".
Change the Text2 Name to "`txtFrmNum`".
Drag and drop the NetICmd and NetIStatus components into the form.




Once the components are dragged to the form, this Visual Basic application becomes a container for these two ActiveX controls.

Displaying these ActiveX controls on the form is optional. You can either show or hide them by assigning the Visible property with a *True* or *False* value.

## Step 4. Edit the Source Code

Add the following line into the `cmdStart_Click()` sub-routine

```
NetIStatus1.ConnectToServerApp
```

Add the following line into the `cmdStop_Click()` sub-routine

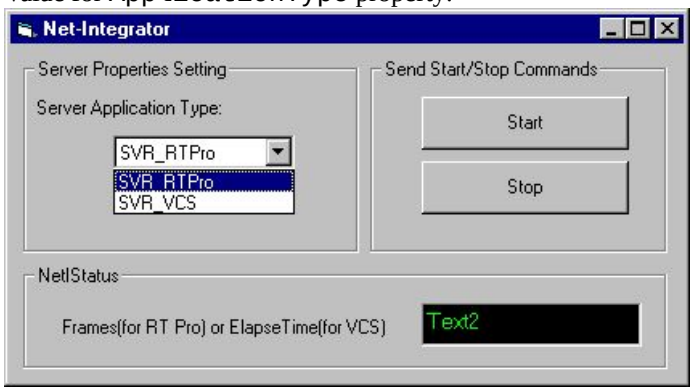
```
NetIStatus1.Disconnect
```

The following is the source code generated after you have completed the implementations above:

```
‘* Setup Server Application Type
Private Sub cmbAppType_Click()
    NetICmd1.ApplicationType = cmbAppType.ListIndex
    NetIStatus1.ApplicationType = cmbAppType.ListIndex
End Sub

 ‘* Start Command
Private Sub cmdStart_Click()
    ‘Calling the SendCommand method will connect the server
first and send start command.
    NetICmd1.SendCommand CMD_STARTMEAS
    'Connect the NetIStatus1 to Server
    NetIStatus1.ConnectToServerApp
End Sub

 ‘* Stop Command
Private Sub cmdStop_Click()
    NetICmd1.SendCommand CMD_STOPMEAS
    'Disconnect the NetIStatus1 from Server
    NetIStatus1.Disconnect
End Sub

‘* Handle the NetIStatus’s OnUpdateStatus Event
Private Sub NetIStatus1_OnUpdateStatus(ByVal nRunStatus As Long)
    NetIStatus1.ReadStatus
    Dim vdata As Variant, nPrjType As Long
    If NetIStatus1.ApplicationType = SVR_RTPRO Then
        nPrjType = NetIStatus1.GetIntStatusArray(vdata)
        txtFrmNum.Text = vdata(4)   'Index 4 is the Frames
Number of RTPro
    Else
        nPrjType = NetIStatus1.GetFloatStatusArray(vdata)
        txtFrmNum.Text = vdata(0)   'Index 0 is the ElapseTime
of VCS
    End If
End Sub
```

## Step 5. Execute the Application

Run LDS-Dactron Shaker Control or RTPro Software on the same computer that the VB sample is to be running. Manually open an existing project or create a new project.
On the same computer, start the Visual Basic application you have just created.
Choose the correct value for `ApplicationType` property:




Click the Start Command Button to start the measurement. The LDS-Dactron software will execute and the measurement will start. For RTPro, the frames number will be displayed; for Shaker Control, the elapse time will be displayed.

To stop the measurement, click the Stop Command Button.

**Note:** If the LDS-Dactron software and the Net-Integrator container application are running on different computers, make sure you assign the correct IP address to the `AppComputerIP` property of the `NetICmd` and `NetIStatus` controls before clicking the Start Command Button.

# ActiveX Controls

## *ActiveX Control: NetICmd*

NetICmd control is used to send the commands from the client to the server application. Some of the commands need parameters but most of them do not. The commands received by the server application will be processed logically. Commands may have no effect or may even damage the test if they are sent in an inappropriate situation. For example, before the command CMD_STARTMEAS is sent, any other operating command such as CMD_RESETAVG will have no effect.
The client can track the testing status by retrieving the status structure from the server using the NetIStatus control.

### NetICmd Properties

#### ApplicationType (ENUM_SVRMODE)

The ActiveX controls in Net-Integrator can connect to two mode applications: RTPro and Shaker Control. ApplicationType determines the mode, and the value can be SVR_RTPRO(equal to 0) or SVR_VCS(equal to 1). The default value is SVR_RTPRO which means the control will assume RTPro is the server application unless otherwise specified.

**Example in Visual Basic**

```
 'If the control is to connect to RT PRO.exe
NetICmd.ApplicationType = SVR_RTPRO

'If the control is to connect to Shaker Control
NetICmd.ApplicationType = SVR_VCS
```

#### AppComputerIP (BSTR)

AppComputerIP determines the target server application location. It can be an IP address or the host name that the server application is running. The default value is "127.0.0.1", an internationally recognized IP value known as the "loopback" address, which refers to the host computer.
Call this function before others are called. If the AppComputerIP isn't called, the ActiveX controls in this container will look for the server application running on the local computer.

**Example in Visual Basic**

```
NetICmd.AppComputerIP = "192.68.68.28"
```

#### ProjectType (Enum ENUM_PRJMODE)

This property decides which analysis software option is to be loaded in the RTPro or Shaker Control server application. When a project in RTPro or Shaker Control is created, the user must sets a ProjectType first. After this parameter is initialized, the **New** command will know what type of project to create.
The values for ProjectType are listed in the table below. Status numbers for each value are also included for reference.

| | |
|---|---|
| MPS_NONE= 0, | no option is loaded |
| MPS_FFT= 1, | RTPro, Basic Signal Analysis and Waveform Source |
| MPS_SPEC= 2, | RTPro, Shock Response Spectrum Analysis |
| MPS_ORDER= 3, | RTPro, Machinery Analysis (Order Tracking) |
| MPS_SWEPTSINE= 4, | RTPro, Not used |
| MPS_STEPSINE= 5, | RTPro, Not used |
| MPS_OCTAVE= 6, | RTPro, Basic Acoustic Analysis |
| MPS_LOGSPEC= 7, | RTPro, Not used |
| MPS_LONGCAPT= 8, | RTPro, Not used |
| MPS_DISKCAPT= 9, | RTPro, Not used |
| MPS_WAVELET= 10, | RTPro, Not used |
| MPS_MDA= 11, | RTPro, Basic Modal Data Acquisition mode |
| MPS_DISK_THROUGHPUT = 12, | RTPro, LWR(Long Waveform Recorder) Mode |
| MPS_MDS= 13, | RTPro, Mechanical Drop Shock mode |
| MPS_FFT_ADV= 21, | RTPro, Advanced Signal Analysis and Waveform Source* |
| MPS_SPEC_ADV= 22, | RTPro, Advanced Shock Response Spectrum Analysis* |
| MPS_OCTAVE_ADV= 26, | RTPro, Advanced Acoustic Analysis* |
| MPS_MDA_ADV= 31, | RTPro, Advanced Modal Data Acquisition mode* |
| SHOCK_SYSTEM = 50 | Shaker Control, Classical Shock control |
| RANDOM_SYSTEM= 51 | Shaker Control, Random control |
| SINE_SYSTEM= 52 | Shaker Control, Sine control |
| SPS_SYSTEM= 53 | Not used |
| NONE_SYSTEM= 54 | Not used |
| ROR_SYSTEM= 55 | Shaker Control, Random on random control |
| SOR_SYSTEM= 56 | Shaker Control, Sine on random control |
| SROR_SYSTEM= 57 | Shaker Control, Sine and Random on random |
| RSS_SYSTEM= 58 | Shaker Control, SRS Synthesis control |
| TTH_SYSTEM= 59 | Shaker Control, Transient control |
| RSTD_SYSTEM= 60 | Shaker Control, RSTD control |
| MPS_SYSTEM= 61 | Shaker Control, not used |
| LTH_SYSTEM= 62 | Shaker Control, LTH control |

The default value is MPS_NONE.

### Example in Visual Basic

```
NetICmd.ProjectType = MPS_FFT
…
NetICmd.ProjectType = RANDOM_SYSTEM
```

## NetICmd Methods

### VARIANT ConnectToServerApp()

This method connects the NetICmd control to a remote RTPro or Shaker Control server. Before this method is called, the property AppComputerIP must be assigned with the correct value. The type of return vale is a VT_BOOL. If it succeeds, it will return **true**, otherwise **false**. If a connection has been established between the control and the server, then a call to this method will disconnect the current connection and reconnect to the remote server.

### void Disconnect()

This method disconnects the current connection. Use it before the client program exits. If there is no current connection, this method will have no effect.

### void SetCmdParams(long nInteger1, long nInteger2, float fFloat1, float fFloat2)

This method sends parameter values for the LDS-Dactron Shaker Control commands that require them. Call this method to set the correct parameters before sending the commands.

### VARIANT SendCommand(ENUM_RTSVRCMD eCmdID)

This method sends a command to the application server. Unless specified, no other parameters are needed for the command. The command is defined by a type Enum ENUM_RTSVRCMD, and applies to both RTPro and LDS-Dactron Shaker Control as follows:

| | |
|---|---|
| CMD_NEWPRJ = 1 | create a new project |
| CMD_OPENPRJ = 2 | open an existing project. Parameter = path and filename for project |
| CMD_CLOSEPRJ = 3 | close the current project |
| CMD_SAVEPRJ = 4 | Save the project. Parameter = path and filename for project |
| CMD_SAVEPRJAS = 5 | Save the project. Parameter = path and filename for project |
| CMD_SAVEASDEF = 6 | Save the project as Default |
| CMD_STARTMEAS = 7 | Start the test |
| CMD_STOPMEAS = 8 | Stop the test |
| CMD_QUICKREPORT = 18 | Have the application server generate the quick report |

The following commands apply only to the RTPro application:

| | |
|---|---|
| CMD_ENDTEST = 9 | Not used |
| CMD_ABORTTEST = 10 | Abort the test |

| | |
|---|---|
| CMD_PAUSEMEAS = 11 | Pause the test |
| CMD_CONTINUEMEAS = 12 | Continue the test from where it is paused |
| CMD_STARTSOURCE = 13 | Start the signal source |
| CMD_STOPSOURCE = 14 | Stop the signal source |
| CMD_NEXTFRAME = 15 | Request the server to send another frame of display signals or status |
| CMD_RESETAVG = 16 | Reset the averaging |

The following commands apply only to the Shaker Control server application:

| | |
|---|---|
| CMD_HOLD = 19, | Hold sweep |
| CMD_RELEASE = 20, | Release sweep |
| CMD_STARTPREVIEW = 21, | Start the preview mode |
| CMD_STOPPREVIEW = 22, | End the preview mode |
| CMD_CONTROLS_CONTINUESCHEDULE = 23, | Continue the schedule |
| CMD_CONTROLS_PAUSESCHEDULE = 24, | Pause the schedule |
| CMD_CONTROLS_DISABLEABORT = 25, | Disable the abort checking |
| CMD_CONTROLS_ENABLEABORT = 26, | Enable the abort checking |
| CMD_CONTROLS_OPENLOOP = 27, | Not update the system transfer function (open the control loop) |
| CMD_CONTROLS_CLOSEDLOOP = 28, | Close the control loop |
| CMD_CONTROLS_SETLEVEL = 29, //need a param: fFloat1 | Set the level. The floating parameter is from 0.0 to 1.0, representing the level to set |
| CMD_CONTROLS_DECREASELEVEL = 30, | Decrease the level by the amount defined in the application server |
| CMD_CONTROLS_INCREASELEVEL = 31, | Increase the level by the amount defined in the application server |
| CMD_CONTROLS_NEXTEVENT = 32, | Jump to next event in the schedule |
| CMD_CONTROLS_NEXTPROFILE = 33, | Jump to the next schedule/profile |
| CMD_CONTROLS_SAVEHINVERSE = 34, | Save the system transfer function |
| CMD_CONTROLS_RESETAVERAGING = 37, | Reset the average number to 1 |
| CMD_CONTROLS_TOGGLESIGN = 38, | Reverse the pulse direction |
| CMD_CONTROLS_SWEEPDOWN = 39, | Frequency sweeps down |
| CMD_CONTROLS_SWEEPUP = 40, | Frequency sweeps up |
| CMD_CONTROLS_SETFREQUENCY = 41, //need a param: fFloat1 and fFloat2 = 0 | Jump to the frequency as specified as float parameter |
| CMD_CONTROLS_DECREASEFREQUENCY = 42, | Decrease the frequency by amount that is defined in the server application |
| CMD_CONTROLS_INCREASEFREQUENCY = 43, | Increase the frequency by amount that is defined in the server application |
| CMD_CONTROLS_SINGLEPULSE = 44, | Output single pulse or single waveform |
| CMD_CONTROLS_POLARITYPOSITIVE = 45, | Change the pulse polarity to positive |
| CMD_CONTROLS_POLARITYNAGATIVE = 46, | Change the pulse polarity to negative |
| CMD_CONTROLS_SOR_TURNONOFF = 47, //need a param: nInteger1, nInteger2 | Turn on or off the tones of sines in SOR |
| CMD_CONTROLS_ROR_TURNONOFF = 48, //need a param: nInteger1, nInteger2 | Turn on or off the random bands in ROR |
| CMD_CONTROLS_PRETESTTONOMALTEST = 49, | Not used |
| CMD_CONTROLS_RESTORELEVEL = 50, | Restore the testing level to that is defined by the level schedule |
| CMD_CONTROLS_SETH_RATIO = 51 //need a param: fFloat1 | For LTH, defined the update rate of system transfer function, 0.0 to 1.0 for float parameter |

The type of return value is a VT_BOOL. If the call succeeds, it will return **`true`**, otherwise **`false`**. If the connection has not been established when the method is called, the method will call `ConnectToServerApp` to establish a connection and then send the command to remote server.

The following commands require second or third parameters:

| Command | Parameter | Properties or Methods called before |
|---|---|---|
| CMD_NEWPRJ | Project mode | ProjectType |
| CMD_OPENPRJ | Project file pathname in server host | ProjectName |
| CMD_SAVEPRJ | | |
| CMD_SAVEPRJAS | | |
| CMD_CONTROLS_SETLEVEL | fFloat1 | SetCmdParams 0&,0&,fFloat1,0! |
| CMD_CONTROLS_SETREQUENCY | fFloat1, ffloat2=0 (must) | SetCmdParams 0&,0&,fFloat1,0! |
| CMD_CONTROLS_SOR_TURNONOFF | ninteger1, nInteger2 | SetCmdParams nInteger1, nInteger2, 0!, 0! |
| CMD_CONTROLS_ROR_TURNONOFF | | |
| CMD_CONTROLS_SETH_RATIO | fFloat1 | SetCmdParams 0&, 0&, fFloat1, 0! |

**Example in Visual Basic**

```
‘To make a new FFT project in RTPro.exe
‘The remote RTPro server has been connected…
NetICmd.ProjectType = MPS_FFT
If ( NetCmd.SendCommand( CMD_NEWPRJ) = False) Then
      ‘… error
End If

‘To open a Random project in LDS-Dactron Shaker Control.exe
‘The remote Shaker Control server has been connected…
NetICmd.ProjectName = “D:\Prj1(Random)\Prj1.prj”
If ( NetCmd.SendCommand( CMD_OPENPRJ) = False) Then
      ‘…open error
End If
```

**Example in Visual Basic**

```
‘Start and Stop the remote server testing
NetICmd.SendCommand CMD_STARTMEAS
…’do sth
NetICmd.SendCommand CMD_STOPMEAS
```

**Example in Visual Basic**

```
‘Send a few commands with parameters to LDS-Dactron Shaker
Control.exe
‘1 Set Level (need fFloat1)
Dim fFloat1 As Single
fFloat1 = 1.2345!
NetICmd.SetCmdParams 0&,0&,fFloat1,0!
NetICmd.SendCommand CMD_CONTROLS_SETLEVEL

‘2 Set Frequency (need fFloat1 and fFloat2=0)
NetICmd.SetCmdParams 0&,0&,fFloat1,0!
NetICmd.SendCommand CMD_CONTROLS_SETFREQUENCY

‘3 ROR turn onoff
Dim nInteger1 As Long, nInteger2 As Long
nInteger1 = 5&
nInteger2 = 6&
NetICmd.SetCmdParams nInteger1,nInteger2,0.0!,0.0!
NetICmd.SendCommand CMD_CONTROLS_ROR_TURNONOFF
```

### NetICmd Events

There is no *Event* for the NetICmd control.

## *ActiveX Control: NetISignal*

The NetISignal control is used for the client to retrieve the signals from the server application. A signal contains a collection of attributes and an array of data. By the time the OnAllocSignalReady event is received, the ReadAllSignalName and ReadSigAttrByName (or ReadSigMainAttrByName) events should be called to retrieve the attributes.
After the control receives the event OnUpdateSignal, it can read the content of the signal by going through the whole signal list.

### NetISignal Properties

#### ApplicationType (ENUM_SVRMODE)

The ActiveX controls in Net-Integrator can be connected to two types of LDS-Dactron applications:RTPro and Shaker Control. ApplicationType determines the mode. The value can be either SVR_RTPRO (equal to 0) or SVR_VCS(equal to 1). The default value is SVR_RTPRO which means the control will assume RTPro is the server application unless otherwise specified.

### Example in Visual Basic

```
 ‘If this control connects to RT PRO.exe
NetICmd.ApplicationType = SVR_RTPRO

 ‘If this control connects to LDS-Dactron Shaker Control.exe
NetICmd.ApplicationType = SVR_VCS
```

## AppComputerIP (BSTR)

`AppComputerIP` determines the target server application location. It can be an IP address or the host name that the server application is running. The default value is “127.0.0.1”, an internationally recognized IP value known as the “loopback” address, which refers to the host computer.
Call this function before others are called. If the `AppComputerIP` isn’t called, the ActiveX controls in this container will look for the server application running on the local computer.

**Example in Visual Basic**

```
NetISignal.AppComputerIP = “192.68.68.28”
```

## SignalCount (long, read only)

Returns the current number of signals on the RTPro server. The initial value is 0. After the method `ReadAllSignalName` has been called and succeeds, the value will be updated.

**Example in Visual Basic**

```
Dim nCount As Long
NCount = NetISignal.SignalCount
```

## Signals (return Signal object, param long Index, Read only)

Returns a signal object with the index value, which is based on 0. A signal object includes signal attributes like name, size, etc.

### Example in Visual Basic

```
Dim txt as String
txt = NetISiganl.Signals(1).name
```

This call will retrieve the name of the second signal in the list and assign it to the variable `txt`.

## Signal Object Properties

When the `ReadAllSigName()` method is called, a signal object collection will be established, then a user can access any signal name in the collection through `Signals(n).name`. After calling the `ReadAllSigAttrByName()` method, the signal attributes will assigned to the object. The object includes the following read-only properties according to the signal attributes:

| Property Name | Description |
|---|---|
| Size (long) | size of one dimension array<br>Size * Step = AllocSize |
| Step (long) | 1 or 2; 1 for one dimension signal; 2 for two dimension signal (such as a complex signal) |
| AllocSize (long) | total allocated size of array |
| Name (BSTR) | Name of the signal |
| OriginalName(BSTR) | Internal or Original used name of the signal |
| EULabel (BSTR) | Engineering Unit |
| XLabel (BSTR) | Horizontal Unit |
| YLabel (BSTR) | Vertical unit |
| MeasID (BSTR) | Measurement point identification |
| MeasDate (BSTR) | Measurement date |
| MeasTime (BSTR) | Measurement time |
| Analyzer (BSTR) | The name of the instrument that takes the measurement |
| Hbegin (double) | The beginning value of x axis |
| XIncrease (BOOL) | 0: Linear step; 1: Log step |
| HIncreaseStep (double) | the delta resolution of each step in horizontal. If Xincrease=0, the X value equals N* HincreaseStep; If Xincrease=1; the X value equals to Hbegin * (Xincrease)^(N) |
| ValidLowIndex (long) | Not used |
| ValidHighIndex (long) | Not used |
| Aliasing (double) | Not used |
| SampFreq (double) | The sampling frequency of data array |
| Scale (float) | Not used |
| Max (float) | Not used |
| Min (float) | Not used |
| Peak (float) | Not used |
| RMS (float) | Not used |
| PSDRMS (float) | |
| EnableOnLineSave (BOOL) | To save the signal on the server side, set this value to 1 and then call OnLineSaveSignal() method |

If `EnableOnLineSave` value is assigned to 1, then calling the `OnLineSaveSignal()` method will save this signal on the server side.

## NetISignal Methods

### VARIANT ConnectToServerApp()

This method connects the `NetISignal` control to a remote RT Pro or Shaker Control server. Before this method is called, the property `AppComputerIP` must be assigned with the correct value. The type of return value is a `VT_BOOL`. If it succeeds, it will return **`true`**, otherwise **`false`**. If a connection has been established between the client and server, then a call to this method will disconnect the current connection and reconnect to the remote server.

### void Disconnect()

Call this method to disconnect the current connection when the client program is about to exit. If there is no current connection, this method will have no effect.

### ReadAllSignalName()

Reads all current signal names from a remote server. This method will update the `SignalCount` property value, create the signal list in the ActiveX control and update the name field for each signal object in the list.

**Example in Visual Basic**

```
NetISignal.ReadAllSignalName
```

### BOOL ReadSigAttrByName(BSTR SigName)

This method reads a signal's attributes by the signal name `SigName`. It returns 1 if it reads successfully, otherwise 0. The signal attributes are defined in the table in the section on "Signal Object Properties."

### BOOL ReadSigMainAttrByName(BSTR SigName)

This method reads a signal's major attributes by the signal name `SigName`. It will return 1 if it reads successfully, otherwise `0`. The signal major attributes are defined as following:

| | |
|---|---|
| Size (long) | Size*Step <= m_nAllocSize; |
| Step (long) | 1 or 2; 1 for one dimension signal; 2 for two dimension signal (such as a complex signal) |
| AllocSize (long) | Size may change, but the AllocSize is fixed |
| HBegin (double) | the starting value of x-axis |
| XIncrease (BOOL) | 0: Linear step; 1: Log step |
| HIncreaseStep (double) | the delta resolution of each step in horizontal. If Xincrease=0, the X value equals N* HincreaseStep; If Xincrease=1; the X value equals to Hbegin * (Xincrease)^(N) |
| ValidLowIndex (long) | Not used |
| ValidHighIndex (long) | Not used |
| Aliasing (double) | Not used |
| SampFreq (double) | The sampling frequency of data array ,in Hz |
| Scale (float) | Not used |
| Max (float) | Not used |
| Min (float) | Not used |
| Peak (float); | Not used |
| RMS (float); | Not used |
| PSDRMS (float); | Not used |

### long FindSignalIndexByName(BSTR SigName)

Returns the signal index number in the signal object list by the signal name. When the user gets a signal's name and wants to access its attributes, the signal's index number is unknown in the signal object collection. Using the method returns the index number. An incorrectly entered or unknown signal name will return an error value of -1.

**Example in Visual Basic**

```
Dim n As Long
    n = NetISignal.FindSignalIndexByName(SignalName)
    With NetISignal.Signals(n)
        frmSigMainAttr.txtSize = .Size
        frmSigMainAttr.txtStep = .Step
        frmSigMainAttr.txtAllocSize = .AllocSize
        frmSigMainAttr.txtHBegin = .HBegin
        frmSigMainAttr.txtXInc = .XIncrease
        frmSigMainAttr.txtH = .HIncreaseStep
        frmSigMainAttr.txtLowIndex = .ValidLowIndex
        frmSigMainAttr.txtHighIndex = .ValidHighIndex
        frmSigMainAttr.txtAlias = .Aliasing
        frmSigMainAttr.txtSampFreq = .SampFreq
        frmSigMainAttr.txtScale = .Scale
        frmSigMainAttr.txtMax = .RealMax
        frmSigMainAttr.txtMin = .RealMin
        frmSigMainAttr.txtPeak = .Peak
        frmSigMainAttr.txtRMS = .RMS
    End With
```

### VARIANT ReadSignalData(BSTR SigName)

Returns a signal's data array by the signal name. The type of the return value is `VariantArray:VT_ARRAY|VT_R4`.

**Example in Visual Basic**

```
Dim vData As Variant, i As Long, fdata As Single
vData = NetISignal.ReadSignalData("input1(t)")
For i=0 to UBound(vData)
      fdata = vData(i)
      …do something
Next i
```

### OnLineSaveSignal()

Set the value of `EnableOnLineSave` property of signal objects to 1, then call this method to save the signal online.

#### Example in Visual Basic

```
NetISignal.Signal(0).EnableOnLineSave = 1
….
NetISignal.Signal(n).EnableOnLineSave = 1
NetISignal.OnLineSaveSignal
```

## NetISignal Events

### OnAllocSignalReady(long num)

The parameter num is reserved in this version.
This event will be fired from the server to the NetISignal control when all the signals are allocated before the test. Calling the ReadAllSignalName method to respond to the event can update the signal list in ActiveX client.

#### Example in Visual Basic

```
Private Sub NetISignal1_OnAllocSignalReady(ByVal nSigNum As
Long)
      NetISignal1.ReadAllSignalName()
      For i = 0 To NetISignal1.SignalCount - 1
      txtList(I).Text = NetISignal1.Signals(i).Name
      Next i
End Sub
```

### OnUpdateSignal(long num)

The parameter num is reserved in this version.
This event will be fired from the server application when its signals are updated. Calling the ReadSignalData(BSTR SigName) method to respond to the event returns the real time signal data.

#### Example in Visual Basic

```
Private Sub NetISignal1_OnUpdateSignal(ByVal nSigNum As Long)
Dim vData As Variant, i As Long
vData = NetISignal1.ReadSignalData(“input1(t)”)
For i =0 To UBound(vData)
      data = vData(i)
      …
Next I
End Sub
```

## *ActiveX Control: NetIStatus*

NetIStatus is an ActiveX control used for extracting testing status from the server to the client. The testing status is a collection of variables indicating the run-time situation of that application. For example, the user can retrieve a variable called “RunStatus” (always the first element in the array) in all the integer status properties in order to retrieve the current run mode. Or the user can retrieve the 12th element in the IMPSSTATUS array to retrieve the trigger status.

For purposes of internal implementation, LDS-Dactron separates the status data structure into two types, an integer type and a floating-point data type. The integer type of structure contains an array of 32-bit long integers. The floating point type of structure contains an array of 32-bit floating point numbers. Because there are a number of projects that can be loaded and run, their status definitions are all different. After the array is read, the user can interpret it according to this documentation.

The status data structure will be updated periodically. Any new status elements that are developed will be added to the end of the array (before the debug elements if there are any) to ensure compatibility. Therefore any existing programs the user has written will always be valid.

## `NetIStatus` Properties

### `ApplicationType (ENUM_SVRMODE)`

The ActiveX controls in Net-Integrator can connect to two mode applications: RTPro and Shaker Control. `ApplicationType` determines the mode. The `ApplicationType`'s value can be either SVR_RTPRO(value 0) or SVR_VCS(value 1). The default value is SVR_RTPRO which means the control will assume RTPro is the server application unless otherwise specified.

**Example in Visual Basic**

```
'If the control is connected to RT PRO.exe
NetIStatus.ApplicationType = SVR_RTPRO

'If the control is connected to Shaker Control application
NetIStatus.ApplicationType = SVR_VCS
```

### `AppComputerIP (BSTR)`

`AppComputerIP` determines the target server application location. It can be an IP address or the host name that the server application is running. The default value is "127.0.0.1", an internationally recognized IP value known as the "loopback" address, which refers to the host computer.

Call this function before others are called. If the `AppComputerIP` isn't called, the ActiveX controls in this container will look for the server application running on the local computer.

**Example in Visual Basic**

```
NetIStatus.AppComputerIP = "192.68.68.28"
```

### `ProjectType (Enum ENUM_PRJMODE,read only)`

See section ActiveX control: `NetICmd` Properties

## NetIStatus Methods

### VARIANT ConnectToServerApp()

This method connects the NetIStatus control to a remote RTPro or Shaker Control server. Before this method is called, the property AppComputerIP must be assigned with the correct value. The type of return vale is a VT_BOOL. If it succeeds, it will return **true**, otherwise **false**. If a connection has been established between the control and server, then a call to this method will disconnect the current connection and reconnect to the remote server.

### void Disconnect()

This method disconnects the current connection. Call it before the client program exits. If there is no current connection, this method will have no effect.

### BOOL ReadStatus()

Call this method to read real-time status from the server application and retrieve the status values to the control. To access the status value, you must call the GetIntStatusArray() or GetFloatStatusArray() methods.

In RTPro there are two status types: MPSstatus and OCTStatus. In Shaker Control there are four types: ShockStatus, RandomStatus, SineStatus, and LTHStatus. The status type can be retrieved from ProjectType property. The following table shows the relationships among ApplicationType, ProjectType, and status type.

| ApplicationType | ProjectType | status type | |
|---|---|---|---|
| | | Integer Status Array | Float Point Status Array |
| SVR_RTPRO | MPS_OCTAVE | iOCTSTATUS | fOCTSTATUS |
| | Other value | iMPSSTATUS | fMPSSTATUS |
| SVR_VCS | SHOCK_SYSTEM<br>RSS_SYSTEM<br>TTH_SYSTEM | iSHOCKSTATUS | fSHOCKSTATUS |
| | RANDOM_SYSTEM<br>ROR_SYSTEM<br>SOR_SYSTEM<br>SROR_SYSTEM | iRANDOMSTATUS | fRANDOMSTATUS |
| | SINE_SYSTEM<br>RSTD_SYSTEM | iSINESTATUS | fSINESTATUS |
| | LTH_SYSTEM | iLTHSTATUS | fLTHSTATUS |

"iXXXSTATUS" and "fXXXSTATUS" are variant arrays defined in the following sections.

### Long GetIntStatusArray(VARIANT)

Calling this method will retrieve all **integer** status values as a variant array (**VT_ARRAY|VT_I4**), with the return value as the application server's ProjectType.

After NETIStatus receives the OnUpdateStatus event, NETIStatus will call the ReadStatus() method. After this is called, GetIntStatusArray()retrieves the integer array. The interpretation of the integer array depends on the ProjectType. The following sections describe how the integer array is interpreted.

## iMPSSTATUS Array

This array is used for retrieving the integer status of general options in RTPro. The following table describes the index used for the integer status.

| Index | Content |
|---|---|
| 0 | version |
| 1 | RunStatus; /* input status */ |
| 2 | DACStatus; /* =1 if DAC is open, 0=not */ |
| 3 | RunType; |
| 4 | AvgNbr; |
| 5 | CurFrmNbr; /* Current frame number in this schedule, 1 based */ |
| 6 | CurDelayFrm; |
| 7 | TotalFrmNbr; /* Total Frm nbr in this schedule */ |
| 8 | TimeSize; /* whole frame Size */ |
| 9 | ValidDataSize; /* <= BlockSize, for scrolling display */ |
| 10 | RealOverlapSize; /* Overlap size in real implementation */ |
| 11 | IsSingleFrame; /* =1 if single frame sampling */ |
| 12 | TRGArmed; /* =1 if Trigger Armed */ |
| 13 | TriggerPoint; /* sample point at which trigger happened */ |
| 14 | WaveformType; /* Current output Waveform */ |

The variable `RunStatus` exists for all integer status values. The following table lists the definitions for possible values.

| RunStatus Value | Description |
|---|---|
| 0 | System is in idle mode, ready for new job |
| 1 to 6 | Used internally |
| 7 | The system is Ready for a normal (scheduled) test |
| 8 | The system is running a normal scheduled test |
| 9 | The system is paused in a scheduled test |
| 10 | End of all normal scheduled tests |
| 11 | System abnormally aborted |
| 12 | STOP command received, output ramp-down |
| 13, 14 | Used internally |
| 15, 16 | The system is in the process of being initialized |
| 17 and up | Used internally |

The Client program must track the `RunStatus` value in order to send the next command. Otherwise the server application will not respond correctly.

## iOCTSTATUS Array for RT Pro Octave Analysis

| Index | Content |
|---|---|
| 0 | version |
| 1 | RunStatus; /* see previous definition */ |
| 2 | DACStatus; /* =1 if DAC is open, 0=not */ |
| 3 | RunType; |
| 4 | AvgNbr; |
| 5 | CurFrmNbr; /* Current frame number in this schedule, 1 based */ |
| 6 | TimeSize; /* whole frame Size */ |
| 7 | OctSize; /* Octave(f) Size */ |
| 8 | nMulSpecCur; /* current spectrum position */ |
| 9 | TRGArmed; /* =1 if Trigger Armed */ |
| 10 | TriggerPoint; /* sample point at which trigger happened */ |

## iSHOCKSTATUS Array

This array is used for retrieving the integer status of the classical shock option in Shaker Control.

| Index | Content |
|---|---|
| 0 | version |
| 1 | RunStatus; /* see previous definition */ |
| 2 | ScheMode; /* SCHE_AUTO: Auto, SCHE_MANUAL: Manaul */ |
| 3 | LevelMode; /* 0: Auto, 1: Manaul */ |
| 4 | LoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED:Open */ |
| 5 | AbortMode; /* ABORT_ENABLED: Enabled, BORT_DISABLED:Disabled */ |
| 6 | AvgNbr; |
| 7 | ScheElapseFrmNbr /* Current frame number in this schedule, 1 based */ |
| 8 | TotalFrmNbr; /* Total Frm nbr in this schedule */ |
| 9 | CurScheNbr; /* Current schedule number in this test, 1 based */ |
| 10 | TotalScheNbr; /* Total Frm nbr in this test */ |
| 11 | BlockSize; |
| 12 | AlarmReason; /*bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low*/ |
| 13 | AbortReason; /* bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low bit 4 set: Timeout in pre-test bit 5 set: Drive reached max. limit*/ |
| 14 | abScheManualMode; |
| 15 | abLevelManualMode; |
| 16 | abLoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED:Open */ |
| 17 | abCurFrmNbr; |
| 18 | abCurScheNbr; |
| 19 | abAvgNbr; |
| 20 | abHiLinesAbort; |
| 21 | abLoLinesAbort; |
| 22 | ElapsedFrmNbrFullLevel; /* total elapsed frm nbr at full level*/ |
| 23 | iStopReason; /* Possibilities are: User, Abort, or Schedule */ |
| 24 | TotalElapseNbr; /* the total elapsed frm-nbr since START */ |
| 25 | AllScheduleFrmNbr; /*the total frame number of all schedules*/ |
| 26 | DriveMultiplier; /* e.g., Toggles between 1 and -1.0 */ |

**iRANDOMSTATUS array:**

This array is used for retrieving the integer status of the random option in Shaker Control.

| Index | Content |
|---|---|
| 0 | version |
| 1 | RunStatus; /* see previous definition */ |
| 2 | ScheMode; /* SCHE_AUTO: no pause, SCHE_MANUAL: pause schedule */ |
| 3 | LoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED:Open */ |
| 4 | AbortMode; /* ABORT_ENABLED: Enabled, ABORT_DISABLED:Disabled */ |
| 5 | AvgNbr; |
| 6 | ScheElapseFrmNbr; /* Current frame number in this schedule, 1 based */ |
| 7 | TotalFrmNbr; /* Total Frm nbr in this schedule */ |
| 8 | CurScheNbr; /* Current schedule number in this test, 1 based */ |
| 9 | TotalScheNbr; /* Total Frm nbr in this test */ |
| 10 | BlockSize; |
| 11 | AlarmReason; /* index=11, bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low*/ |
| 12 | RampStatus; |
| 13 | CurDelayFrm; /* until this value goes to 0, we will not avg the inputs*/ |
| 14 | AbortReason; /* index=14 bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low bit 4 set: Timeout in pre-test bit 5 set: Drive reached max. limit*/ |
| 15 | abScheManualMode; |
| 16 | abLevelManualMode; |
| 17 | abLoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED:Open */ |
| 18 | abCurFrmNbr; |
| 19 | abCurScheNbr; |
| 20 | abAvgNbr; |
| 21 | abHiLinesAbort; |
| 22 | abLoLinesAbort; |
| 23 | abHiLinesAlarm; |
| 24 | abLoLinesAlarm; |
| 25 | TargetLevelReached; /* 1: yes, 0: not yet */ |
| 26 | preSuccess; /* 0: not, 1: yes */ |
| 27 | MixMode; /* current mode */ |
| 28 | SOREnabled; /* bitwise flag */ |
| 29 | ROREnabled; /* bitwise flag */ |
| 30 | BroadBandOff; /* 1: yes, 0: not(default BroadBand ON) */ |
| 31 | FrmNbrFullLevel; |

### iSINESTATUS Array

This array is used for retrieving the integer status of the Sine control option in Shaker Control.

| Index | Content |
|---|---|
| 0 | Version |
| 1 | RunStatus; /* See previous definition */ |
| 2 | ResDwellStatus; /* 0: not resonant dwelling, else: need look at document */ |
| 3 | ScheMode; /* SCHE_AUTO: Auto, SCHE_MANUAL: Manaul */ |
| 4 | FreqTransMode; /* 0: freq not in transition mode 1: in trans. mode */ |
| 5 | AmplTransMode; /* 0: ampl. not in transition mode 1: in trans. mode */ |
| 6 | LevelMode; /* 0: Auto, 1: Manual */ |
| 7 | LoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED:Open */ |
| 8 | AbortMode; /* ABORT_ENABLED: Enabled, ABORT_DISABLED:Disabled */ |
| 9 | ScheElapseFrmNbr; /* Current frame number in this schedule, 1 based */ |
| 10 | TotalFrmNbr; /* Total Frm nbr in this schedule */ |
| 11 | CurScheNbr; /* Current schedule number in this test, 1 based */ |
| 12 | TotalScheNbr; /* Total Frm nbr in this test */ |
| 13 | LastFreqIndex; |
| 14 | FreqIndex; |
| 15 | IsSweepUp; /* 0: Up, 1:Down */ |
| 16 | AlarmReason; /* bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low*/ |
| 17 | AbortReason; /* bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low bit 4 set: Timeout in pre-test bit 5 set: Drive reached max. limit*/ |
| 18 | abLoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED:Open */ |
| 19 | abCurFrmNbr; |
| 20 | abCurScheNbr; |
| 21 | IsInitRampUpMode; /* when H is not available use this mode*/ |
| 22 | rstdStatus; /* not used in swept sine control */ |
| 23 | CurFrmNbrFullLevel; |
| 24 | iSweepFast; /* 0: slow sweep mode, 1: fast sweep mode */ |
| 25 | iElapsedSweeps; /* this parameter counts the sweeps of current sweep event*/ |
| 26 | FrmNbrThisEntry; /* Frm number at this entry*/ |

### iLTHSTATUS Array

This array is used for retrieving the integer status of the Long Time History option in Shaker Control.

| | |
|---|---|
| 0 | version |
| 1 | RunStatus; /* See previous definition */ |
| 2 | ScheMode; /* SCHE_AUTO: no pause, SCHE_MANUAL: Pause schedule */ |

| | |
|---|---|
| 3 | LoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED: Open */ |
| 4 | AbortMode; /*ABORT_ENABLED: Enabled, ABORT_DISABLED: Disabled */ |
| 5 | AvgNbr; |
| 6 | ScheElapseFrmNbr; /* Current frame number in this schedule, 1 based */ |
| 7 | TotalFrmNbr; /* Total Frm nbr in this schedule */ |
| 8 | CurScheNbr; /* Current schedule number in this test, 1 based */ |
| 9 | TotalScheNbr; /* Total Frm nbr in this test */ |
| 10 | BlockSize; |
| 11 | AlarmReason; /* index=11, bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low */ |
| 12 | RampStatus; |
| 13 | CurDelayFrm; /* until this value goes to 0, do not avg the inputs */ |
| 14 | AbortReason; /* index=14 bit 0 set: RMS high bit 1 set: RMS low bit 2 set: line high bit 3 set: line low bit 4 set: Timeout in pre-test bit 5 set: Drive reached max limit */ |
| 15 | abLoopMode; /* EQUALIZED: Closed, NOT_EQUALIZED: Open */ |
| 16 | abCurFrmNbr; |
| 17 | abCurScheNbr; |
| 18 | abAvgNbr; |
| 19 | abHiLinesAbort; |
| 20 | abLoLinesAbort; |
| 21 | abHiLinesAlarm; |
| 22 | abLoLinesAlarm; |
| 23 | TargetLevelReached; /* 1: yes, 0: not yet */ |
| 24 | preSuccess; /* 0: not, 1: yes */ |
| 25 | MixMode; /* current mode */ |
| 26 | SOREnabled; /* bitwise flag */ |
| 27 | ROREnabled; /* bitwise flag */ |
| 28 | BroadBandOff; /* 1: yes, 0: not(default BroadBand ON) */ |
| 29 | FrmNbrFullLevel; |
| 30 | TotalElapseNbr; /* the total elapsed frm-nbr since START */ |

### Long `GetFloatStatusArray(VARIANT)`

Calling this method will retrieve all **floating** status values as a safe array(**VT_ARRAY|VT_R4**), with return value as the application server's `ProjectType`.

After `NETIStatus` receives the `OnUpdateStatus` event, `NETIStatus` will call the `ReadStatus()` method. After this is called, call `GetFloatStatusArray()`to retrieve the floating point array. The interpretation of the floating point array depends on the `ProjectType`. The following sections describe how the floating array is interpreted.

## fMPSSTATUS Array

This array is used for retrieving the floating-point status of RTPro.

| | |
|---|---|
| 0 | version |
| 1 | ElapseTime; /* in seconds */ |
| 2 | DriveMultiplier; /* e.g., Toggle sign may toggle it between 1 and -1.0 */ |
| 3 | DrivePk; /* in Volts */ |
| 4 | WaveAmpl; /* Amplitude of the waveform */ |
| 5 | WaveFreq; /* Frequency of the waveform */ |
| 6 | WaveQuiet; /* Quiet duration*/ |
| 7 | WaveActive; /* Active duration */ |
| 8 | WaveLowFreq; /* Chirp: Low Frequency */ |
| 9 | WaveHighFreq; /* Chirp: High Frequency */ |

## fOCTSTATUS Array

This array is used for retrieving the float point status of octave analysis software option in RTPro.

| | |
|---|---|
| 0 | version |
| 1 | float ElapseTime; /* in seconds */ |
| 2 | TraceFreq; /* Hz */ |
| 3, 4, 5 | debug0; debug1; debug2; /* STATUS used as debugging tools */ |

## fSHOCKSTATUS Array

This array is used for retrieving the floating point status of the classical Shock control analysis software option in Shaker Control.

| | |
|---|---|
| 0 | version |
| 1 | ElapseTime; /* in seconds */ |
| 2 | TotalTime; /* Total scheduled test time in this schedule. in seconds */ |
| 3 | Level; /* current level, in amplitude ratio */ |
| 4 | RefPk; /* in EU */ |
| 5 | RefRms; /* in EU */ |
| 6 | DrivePk; /* in Volts */ |
| 7 | DriveRms; /* in Volts */ |
| 8 | CtlPk; /* in EU */ |
| 9 | CtlRms; /* in EU */ |
| 10 | NoisePk; /* in EU */ |
| 11 | NoiseRms; /* in EU */ |
| 12 | RampupRate; /* ratio */ |
| 13 | preDrivePk; /* Drive Peak at Pretest target level */ |
| 14 | preTargetLevel; /* pre-test target level */ |
| 15~22 | preNoise1~preNoise8; /* volt. Noise Peak. */ |
| 23~30 | prePeak1~prePeak8; /* volt. peak value of each channel at final-pretest level.*/ |
| 31 | RefDispPkPk; /* generated in Level 2 */ |
| 32 | RefVelPk; /* generated in Level 2 */ |
| 33 | FracFrmNbr; /* fractional frame number, (i-1, i] */ |

## fRANDOMSTATUS Array

This array is used for retrieving the floating point status of the Random control option in Shaker Control.

| | |
|---|---|
| 0 | version |
| 1 | ScheElapseTime; /* in seconds, the elapse time of either PRE-TEST or one of the Schedule */ |
| 2 | ScheTotalTime; /* Total scheduled test time in seconds */ |
| 3 | Level; /* current level, in amplitude ratio */ |
| 4 | DriveMultiplier; /* e.g., Toggles between 1 and -1.0 */ |
| 5 | SNR; /* ratio of (RMSSyy-RMSNoise)/RMSNoise */ |
| 6 | RefRms; /* in EU */ |
| 7 | DrivePk; /* in Volts */ |
| 8 | DriveRms; /* in Volts */ |
| 9 | CtlRms; /* in EU */ |
| 10 | NoiseRms; /* in EU */ |
| 11 | RampupRate; /* ratio */ |
| 12 | SyyRms; /* Instantaneous RMS of Syy, in EU */ |
| 13 | abElapseTime; /* in seconds */ |
| 14 | abTotalTime; /* Total scheduled test time in this schedule. in seconds */ |
| 15 | abLevel; /* current level, in amplitude ratio */ |
| 16 | abRefRms; /* in EU */ |
| 17 | abDrivePk; /* in Volts */ |
| 18 | abDriveRms; /* in Volts */ |
| 19 | abCtlRms; /* in EU */ |
| 20 | abNoiseRms; /* in EU */ |
| 21 | preDrivePk; /* Drive Peak at Pretest target level */ |
| 22 | preTargetLevel; /* pre-test target level */ |
| 23~30 | preNoise1~preNoise8; /* volt. Noise Peak. */ |
| 31~38 | prePeak1~prePeak8; /* volt. peak value of each channel at final-pretest level.*/ |
| 39 | RefDispPkPk; /* generated in Level 2 */ |
| 40 | RefVelPk; /* generated in Level 2 */ |
| 41 | ElapseTimeFullLevel; /* in seconds */ |
| 42 | TotalElapseTime; /* the elapse time since START */ |
| 43 | AllScheduleTime; /*the total time of all schedules*/ |

## fSINESTATUS Array

This array is used for retrieving the floating point status of the Sine control option in Shaker Control.

| | |
|---|---|
| 0 | version |
| 1 | ScheElapseTime; /* in seconds */ |
| 2 | ScheTotalTime; /* Total scheduled test time in seconds */ |
| 3 | Level; /* current level, in amplitude ratio */ |
| 4 | DriveMultiplier; /* e.g., Toggles between 1 and -1.0 */ |
| 5 | RefPk; /* in EU */ |
| 6 | DrivePk; /* in Volts */ |
| 7 | CtlPk; /* in EU */ |
| 8 | NoisePk; /* in EU */ |
| 9 | RampupRate; /* ratio */ |
| 10 | TargetFreq; |
| 11 | MeasFreq; /* the frequency attached to the current Meas. */ |
| 12 | NextFreq; /*the "next" frequency that will be used by FEP */ |
| 13 | NextDrive; /*the "next" sine amplitude that will be used by FEP */ |
| 14 | Speed; /* current sweeping speed, generally it is the same as fParams.Speed */ |
| 15 | dF; /* a parameter of Speed and timeInterval, it is the frequency change between two outputs */ |
| 16 | CprRate; /* current CprRate, Compression Rate */ |
| 17 | CprUpRate; |
| 18 | CprDownRate; |
| 19 | LoBound; /* the low frequency boundary for this profile */ |
| 20 | HiBound; /* the high frequency boundary for this profile */ |
| 21 | abElapseTime; /* in seconds */ |
| 22 | abLeftTime; /* in seconds */ |
| 23 | abLevel; /* in percentage */ |
| 24 | abDrivePk; /* in Volts */ |
| 25 | abCtlPk; /* in EU */ |
| 26 | RefDispPkPk; /* generated in Level 2 */ |
| 27 | RefVelPk; /* generated in Level 2 */ |
| 28 | ElapseTimeFullLevel; /* in seconds */ |
| 29 | TotalElapseTime; /* the elapse time since START */ |
| 30 | ElapseCycles; /* sweep cycles */ |
| 31 | ElapseCyclesFullLevel; /* sine cycles */ |
| 32 | ElapseTimeAtLevelThisEntry; /* time at level at this entry in seconds */ |

## fLTHSTATUS Array

This array is used for retrieving the floating point status of LTH control option in Shaker Control.

| | |
|---|---|
| 0 | version |
| 1 | ScheElapseTime; /* in seconds, the elapse time of either PRE-TEST or of the Schedule |
| 2 | ScheTotalTime; /* Total scheduled test time in seconds */ |
| 3 | Level; /* current level, in amplitude ratio */ |
| 4 | DriveMultiplier; /* e.g., Toggles between 1 and -1.0 */ |
| 5 | SNR; /* ratio of (RMSSyy-RMSNoise)/RMSNoise */ |
| 6 | RefRms; /* in EU */ |
| 7 | DrivePk; /* in Volts */ |
| 8 | DriveRms; /* in Volts */ |
| 9 | CtlRms; /* in EU */ |
| 10 | NoiseRms; /* in EU */ |
| 11 | RampupRate; /* ratio */ |
| 12 | SyyRms; /* Instantaneous RMS of Syy, in EU */ |
| 13 | preDrivePk; /* Drive Peak at Pretest target level */ |
| 14 | preTargetLevel; /* pre-test target level */ |
| 15~22 | preNoise1~preNoise8; /* volt. Noise Peak. */ |
| 23~30 | prePeak1~prePeak8; /* volt. peak value of each channel at final-pretest level. */ |
| 31 | RefDispPkPk; /* generated in Level 2 */ |
| 32 | RefVelPk; /* generated in Level 2 */ |

| | |
|---|---|
| 33 | `ElapseTimeFullLevel; /* in seconds */` |
| 34 | `TotalElapseTime; /* the elapsed time since START */` |
| 35 | `AllScheduleTime; /* the total time of all schedules */` |
| 36 | `TotalRefRMS; /* accumulated RMS */` |
| 37 | `TotalErrorRMS; /* accumulated RMS */` |
| 38 | `TotalEorRefRatio; /* accumulated Error/Ref RMS Ratio */` |
| 39 | `ErrorRMS; /* one frame RMS */` |
| 40 | `EorProfRatio; /* one frame Error/Ref RMS Ratio */` |

**Note:** To get the long or floating status array, it's important to call the `ReadStatus()` method to get the current status from the remote server and then restore client control.

## NetIStatus Events

### OnUpdateStatus()

The server will fire this event when the status changes. Call the ReadStatus() method to respond to this event

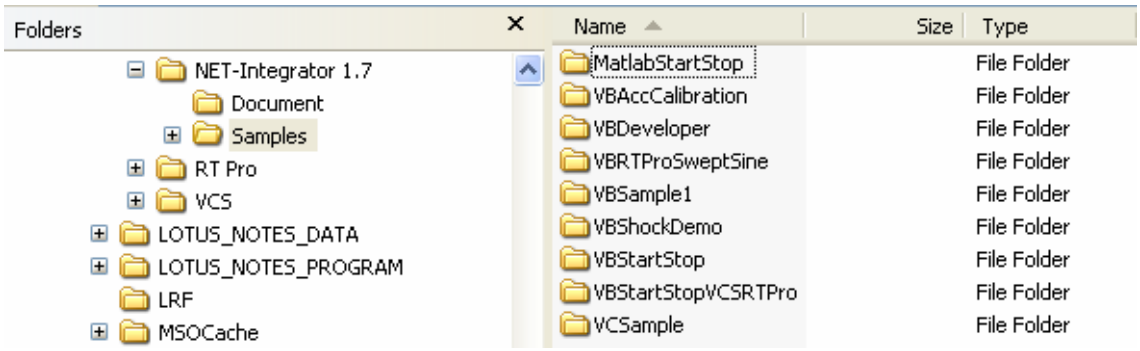
#### Example in Visual Basic

```
Private Sub NetIStatus_OnUpdateStatus()
      NetIStatus1.ReadStatus()
      Dim vStatus as Variant, nPrjType as Long
      nPrjType = GetIntStatusArray(vStatus)


      nPrjType = GetFloatStatusArray(vStatus)
End Sub
```

# NET-Integrator Sample Projects

Note: All the samples must work together with LDS-Dactron software Shaker Control or RTPro applications.

After the NET-Integrator software is installed, it will copy a few sample projects to the destination folder.




With the current releases of LDS-Dactron software, RTPro version 6.2 and Shaker Control version 6.2, the following samples are available:

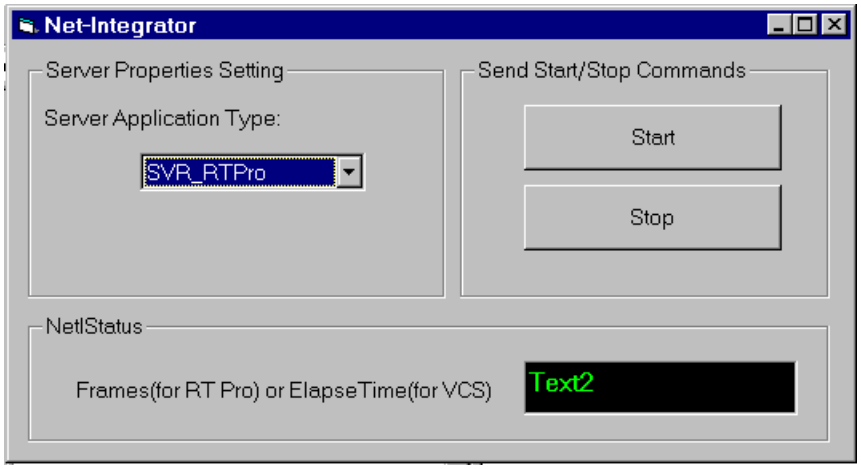
- MatlabStartStop
- VBAccCalibration
- VBDeveloper
- VBRTProSweptSine
- VBSample1
- VBShockDemo
- VBStartStop
- VBStartStopVCSRTPro
- VCSample

## *Visual BASIC Sample: VBStartStop*

Development Environment: Visual Basic 6.0
The following files are included in the **VBStartStop** folder:

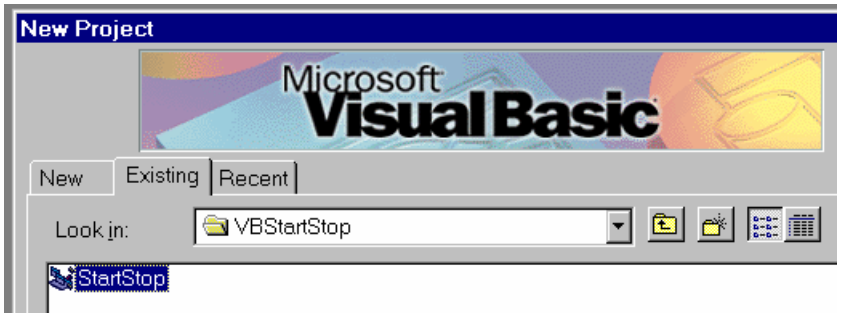
| Name | Size | Type | Modified |
|---|---|---|---|
| frmMain | 6KB | Visual Basic Form File | 3/21/00 10:10 AM |
| frmMain | 1KB | Visual Basic Form Binary File | 3/21/00 10:10 AM |
| mssccprj | 1KB | Microsoft SourceSafe Status | 3/27/00 4:16 PM |
| StartStop | 24... | Application | 3/27/00 4:21 PM |
| StartStop | 1KB | Visual Basic Project | 3/27/00 4:21 PM |
| StartStop | 1KB | Visual Basic Project Workspa... | 3/27/00 4:21 PM |
| vssver | 1KB | Microsoft SourceSafe Status | 3/21/00 4:25 AM |

**VBStartStop** is probably the simplest project that you can build for NET-Integrator. It contains Start/Stop buttons, a Server Type selection, and a status display. To activate the executable, run the **StartStop.EXE** application. This **Net-Integrator** dialog box will display:




This client application does not ask for the assigned IP address for the server application. Therefore, it can only control the applications running on the same machine.

To load the source code, first open Visual Basic 6.0, then open the existing project StartStop as shown below:




The source code is very simple. Following is the complete source listing for the file *form1(frmmain.frm)*:

```
Private Sub cmbAppType_Click()
    NetICmd1.ApplicationType = cmbAppType.ListIndex
    NetIStatus1.ApplicationType = cmbAppType.ListIndex
End Sub

Private Sub cmdStart_Click()
    NetICmd1.SendCommand CMD_STARTMEAS
    'Connect the NetIStatus1 to Server
    NetIStatus1.ConnectToServerApp
End Sub

Private Sub cmdStop_Click()
    NetICmd1.SendCommand CMD_STOPMEAS
    'Disconnect the NetIStatus1 from Server
    NetIStatus1.Disconnect
End Sub

Private Sub Form_Load()
```

```
    cmbAppType.ListIndex = 0
End Sub

Private Sub NetIStatus1_OnUpdateStatus(ByVal nRunStatus As Long)
    NetIStatus1.ReadStatus
    Dim vdata As Variant, l As Long
    If NetIStatus1.ApplicationType = SVR_RTPRO Then
        l = NetIStatus1.GetIntStatusArray(vdata)
        txtFrmNum.Text = vdata(4)   'Index 4 is the Frames
Number of Rt Pro
    Else
        l = NetIStatus1.GetFloatStatusArray(vdata)
        txtFrmNum.Text = vdata(0)   'Index 0 is the ElapseTime
of VCS
    End If
End Sub
```

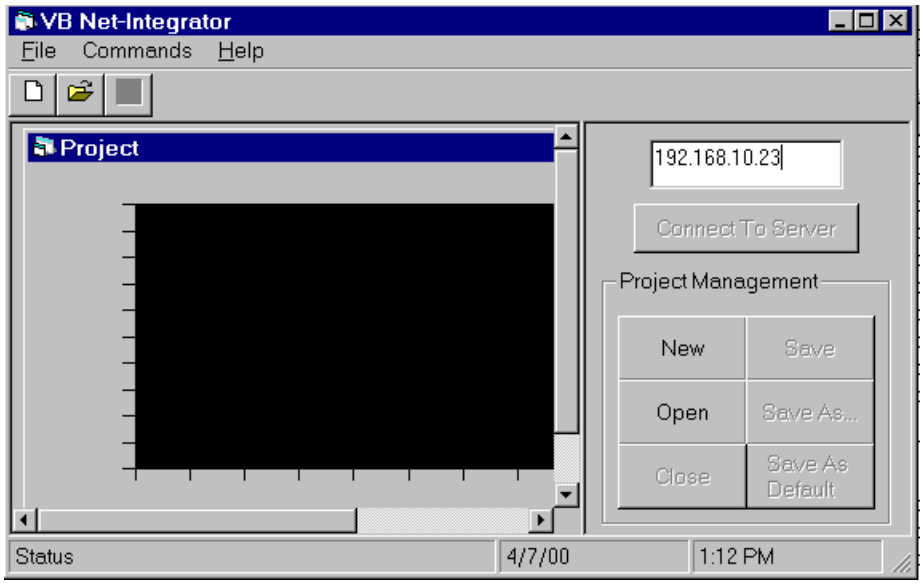
To execute the program within the Visual Basic environment, press the F5 key.

### *Visual BASIC Sample: VBSample1*

Development Environment: Visual Basic 6.0

This project is intended to serve as a Visual Basic project template. It can be modified to build other applications.

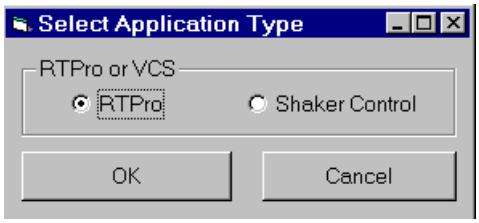
The application file name is **NetInt.EXE** and is meant to run concurrently with RTPro or Shaker Control. When run, this **VB Net-Integrator** window will display:




To make the connection from this client program to the server application, first assign the IP address of the computer that the server application is running and click the **Connect To Server** button.

**Note:** The IP address needs to be entered even if the server application is running on the same machine of this client program.

Select the server type in the Select Application Type dialog box:




Then click OK to continue.

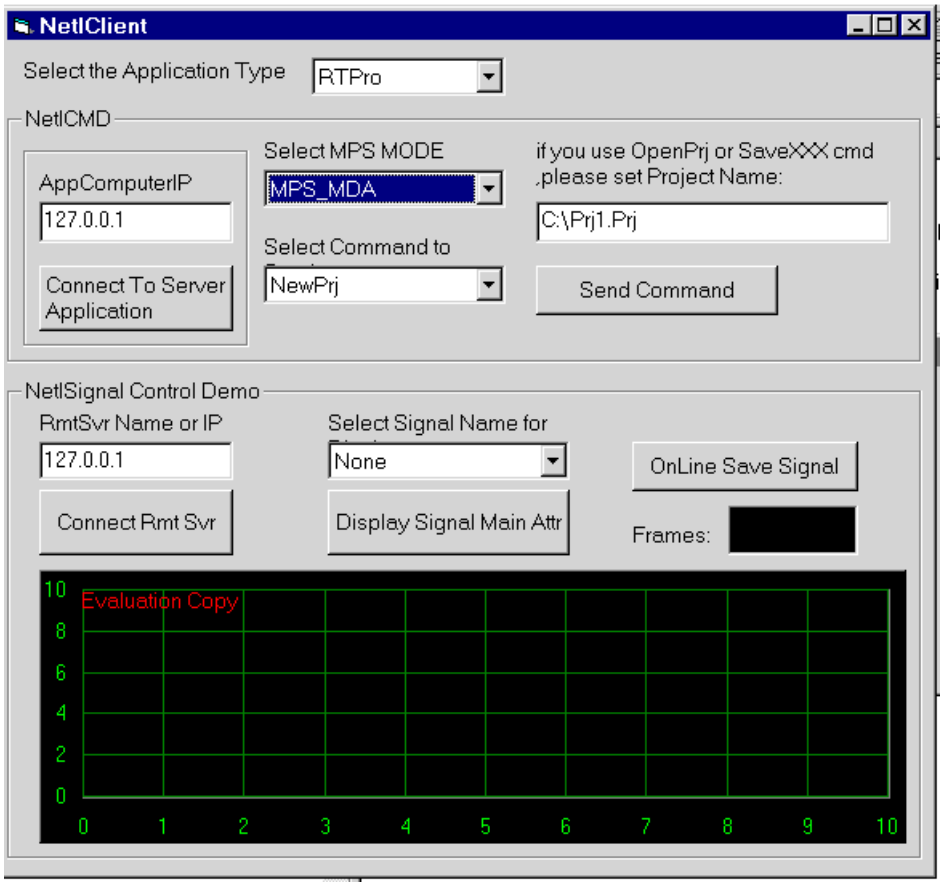
Click the New button to create a new project, or click the Open button to open an existing project. Once the project is loaded, push the Start button to run the application.

### *Visual BASIC Sample: VBDeveloper*

Development Environment: Visual Basic 6.0

This program is more complex than **StartStop** and is meant for Visual Basic developers. You can use this program to send various commands and test the program. It is also possible to use this program as a template to build your own program.

To run the program, click the **developer.exe** application in the **VBDeveloper** folder. The NetIClient window will display.




The **Developer** application allows more operations than the **StartStop** project. The user can select a software option to start, a command to send, a signal to display, etc. It also allows the client program to address a remote server application. The user must be very careful about the sequence of the operation or the Server application can run into illegal operation.

The files included with this project are:

| Name | Size | Type | Modified |
|---|---|---|---|
| Developer | 52KB | Application | 3/27/00 4 |
| Developer | 1KB | Visual Basic Project | 3/27/00 4 |
| Developer | 1KB | Visual Basic Project Workspa... | 3/27/00 4 |
| Form1 | 21KB | Visual Basic Form File | 3/22/00 7 |
| Form1 | 1KB | Visual Basic Form Binary File | 3/22/00 7 |
| Form1 | 1KB | Text Document | 2/24/00 5 |
| frmSaveSignals | 4KB | Visual Basic Form File | 3/22/00 6 |
| frmSaveSignals | 1KB | Visual Basic Form Binary File | 3/22/00 6 |
| frmSigMainAttr | 9KB | Visual Basic Form File | 3/22/00 6 |
| mssccprj | 1KB | Microsoft SourceSafe Status | 3/27/00 4 |
| Project1.PDM | 5KB | PDM File | 3/22/00 8 |

**Note:** This project includes an ActiveX component from National Instruments for displaying signals. A message will pop up indicating that this is an evaluation version of the component.

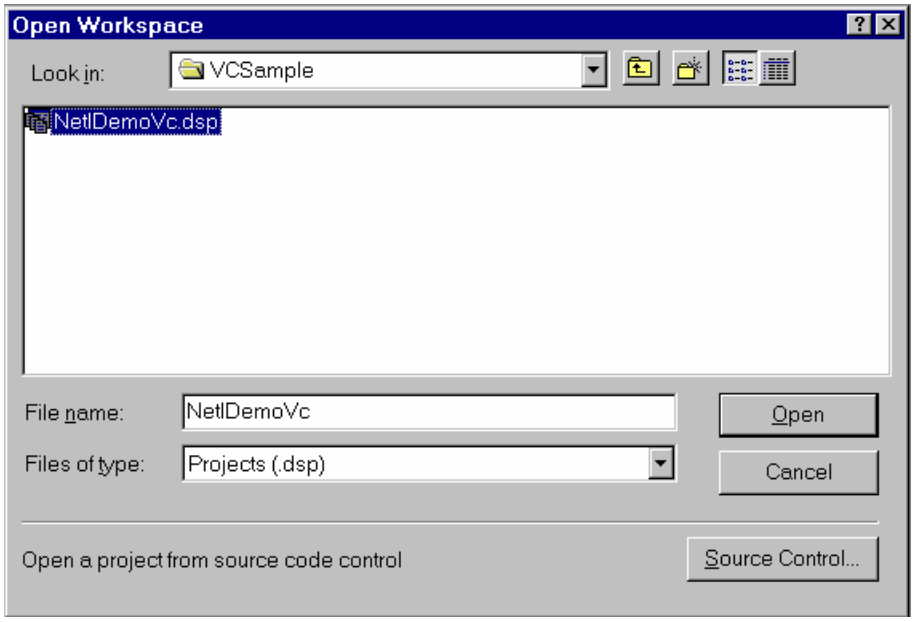
### *Visual C Sample: VCSample*

Development Environment: Visual C++ 6.0

This sample provides a basic skeleton program in Visual C++ showing how to program the NET-Integrator client.

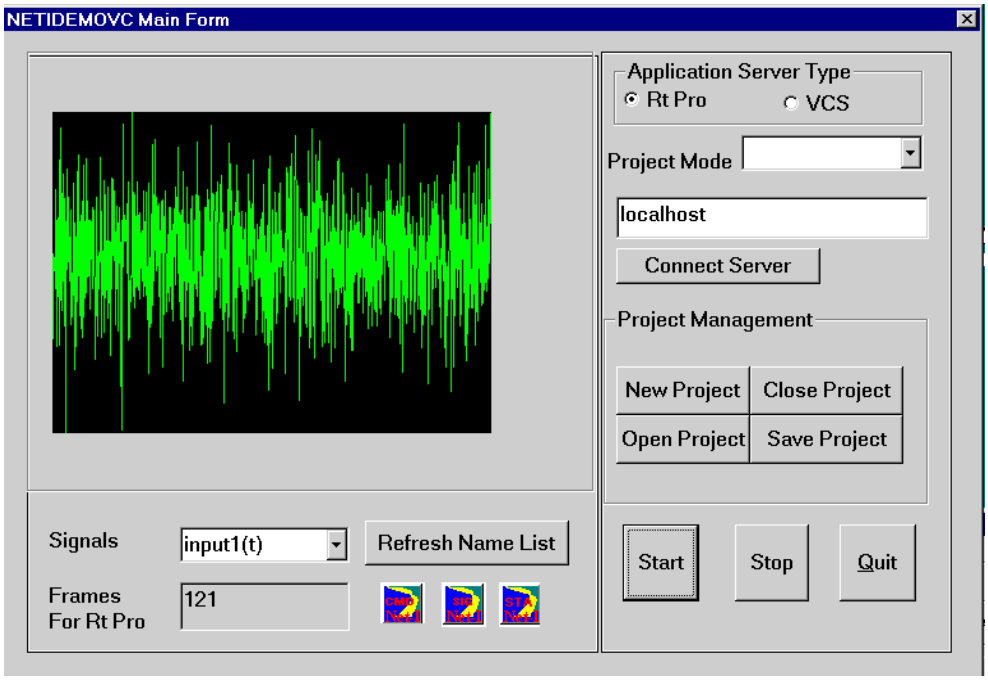
After the sample program is installed, manually create a folder called **\res** in the **\VCSample** folder. Then drag the file **NetIDemoVc.ico** and **NetIDemoVc.rc2** into the **\res** folder. (For future releases, this step will be omitted).

In Visual C++, select **Open Workspace** item, Change the **Files of type** to **Projects (.dsp)**. Then load **NetIDemoVc.dsp**.




Build the **NetIDemoVc** project.

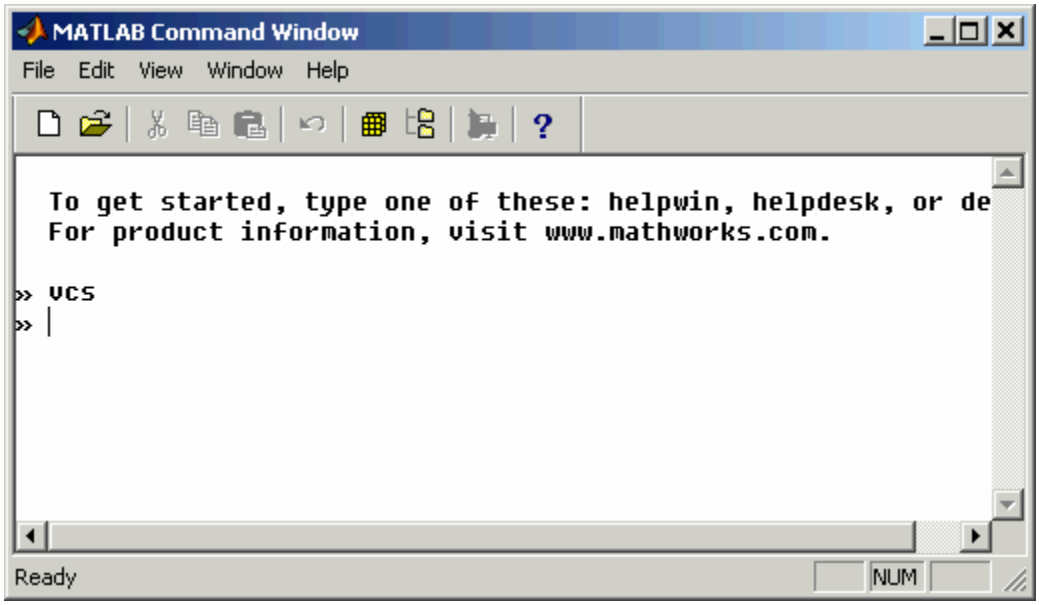
Start either the **RTPro** or **Shaker Control** application, then run the **NetIDemoVc** client program - shown below - which is very similar to the **VBSample1** project.

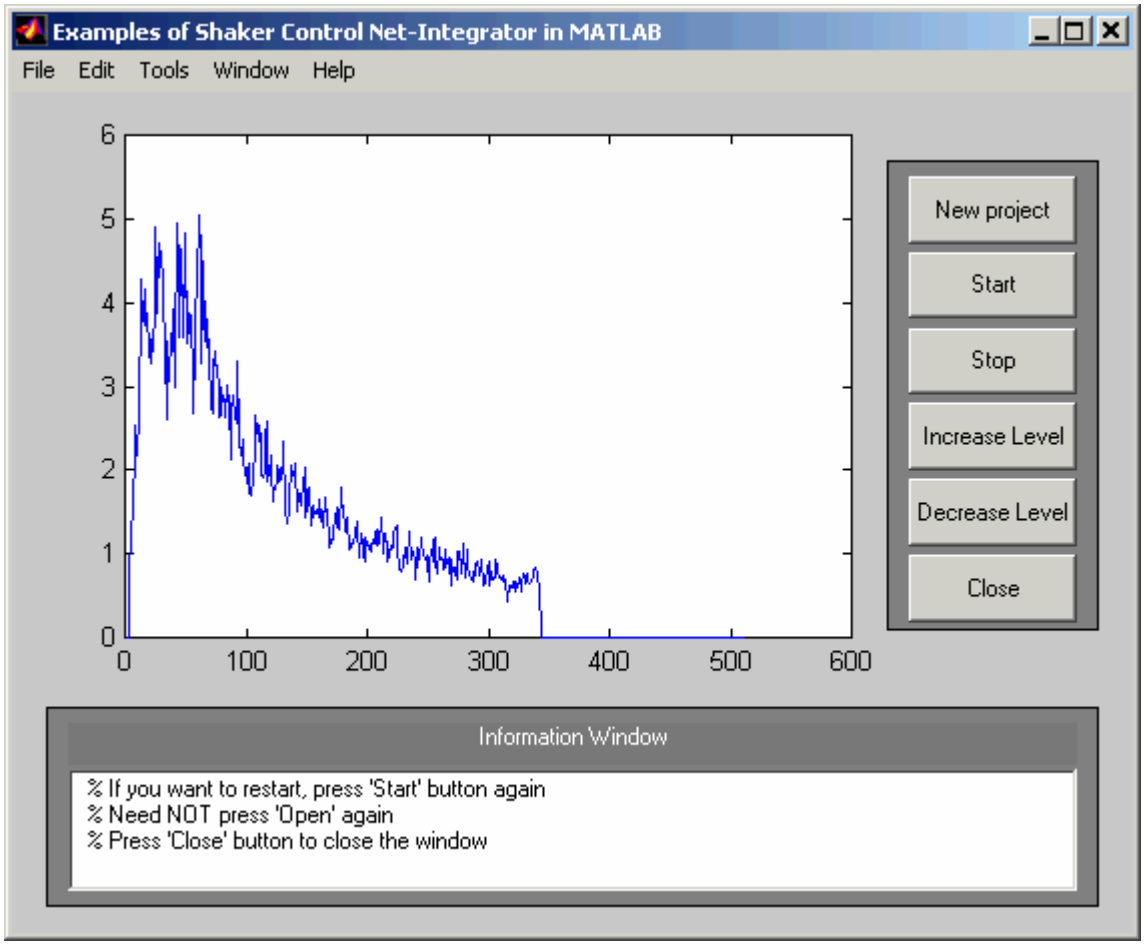



### *MatLab Samples: MatlabStartStop*

Development Environment: MatLab 5.3.1 or higher

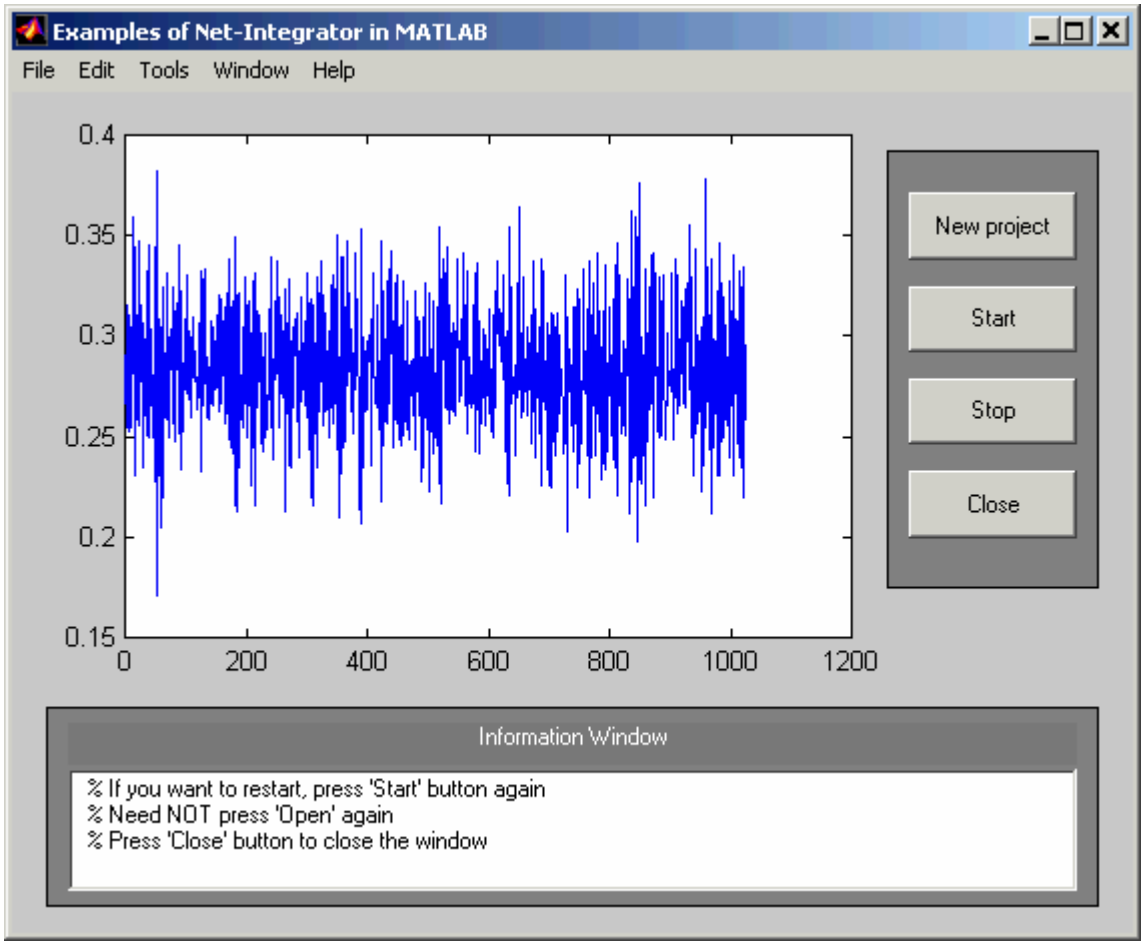
In the sample folder \MatLabStartStop, two examples are given, one for RTPro, one for Shaker Control. To run the Shaker Control sample, first initiate the LDS-Dactron Shaker Control application. Then in the MatLab command line type in command “vcs”. This command will call the MatLab rountine vcs.m.






After the screen shows the picture above, the user can click the “New Project” to initiate the new project within the Shaker Control application. Then click Start or Stop button to execute the test. The signal shown within the MatLab sample window is the control signal shown in Shaker Control. The user can certainly extract any of the other signals from the test.

Similarly, the RTPro project can be executed. Below is the picture showing the RTPro sample after the user enters the “rtpro” from the command line.




In the sample, the signal "input1(t)", is the time domain signal from channel 1 of RTPro, is displayed.

# General Questions and Answers

## *Can ActiveX controls Access a Remote Application? How?*

Yes. The ActiveX controls provided with LDS-Dactron NET-Integrator allow the user to access the server application remotely. To make the connection, each of the following controls have to be connected to the server application by setting property `ApplicationType` and `AppComputerIP`, and calling the method `ConnectToServerApp`:

**`NetICmd`**, **`NetISignal`**, and **`NetIStatus`**

### Example in Visual Basic:

```
NetICmd.ApplicationType = SVR_RTPRO
NetICmd.AppComputerIP = “192.68.68.28”
NetICmd.ConnectToServerApp
……
NetISignal.ApplicationType = SVR_RTPRO
NetISignal.AppComputerIP = “192.68.68.28”
NetISignal.ConnectToServerApp
……
NetIStatus.ApplicationType = SVR_RTPRO
NetIStatus.AppComputerIP = “192.68.68.28”
NetIStatus.ConnectToServerApp
……
```

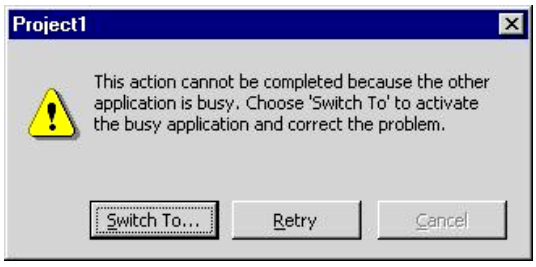
After these functions are called, all three ActiveX controls are connected to the RTPro application running on the computer with IP address “192.68.68.28”.

## *Can the Client Connect to a Local Server Application?*

Yes. In many cases, the client application, i.e., the ActiveX container, is running on the same machine as the server application. In this case, simply do not call the `ConnectToServerApp` method. The ActiveX controls will automatically connect to the applications running on the same computer that the client is running.

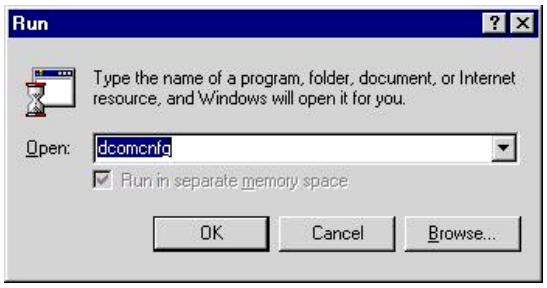
## *What if ActiveX Tries to Connect a Server that is not Running?*

The server application must be manually launched before ActiveX can connect, otherwise the following dialog box will display on the client side.

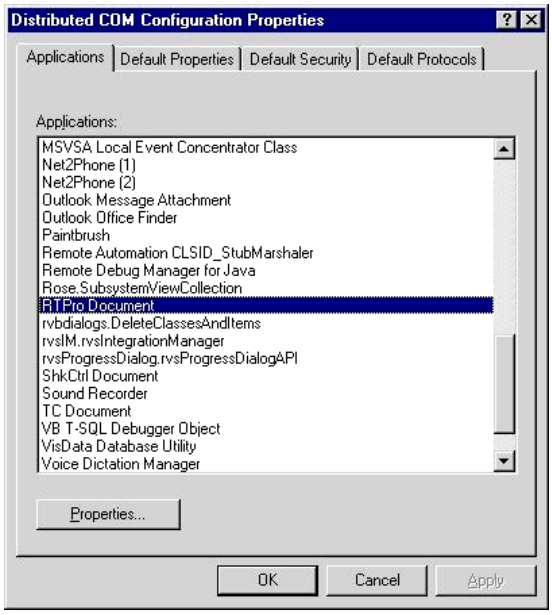


Some computers may wait for a few minutes to get this task resolved. To avoid the long waiting time, use dcomcnfg.exe to edit the Launching Permission of RTPro Document and ShkCtrl Document. Follow these steps:

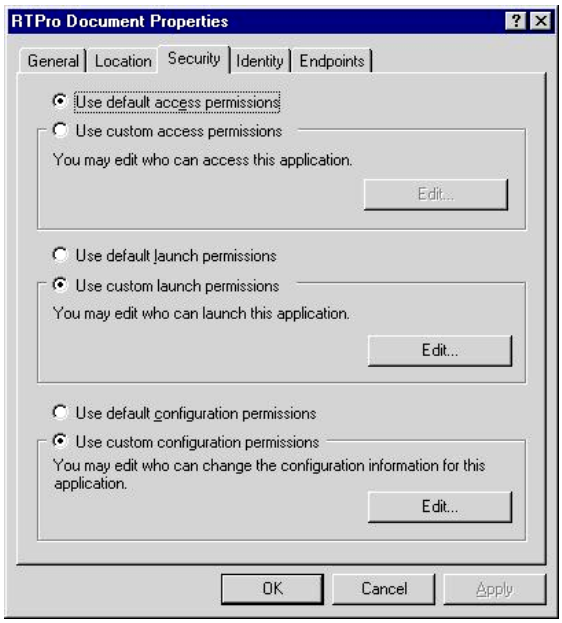
Step 1: Run **`dcomcnfg.exe`** from Start->Run... menu.



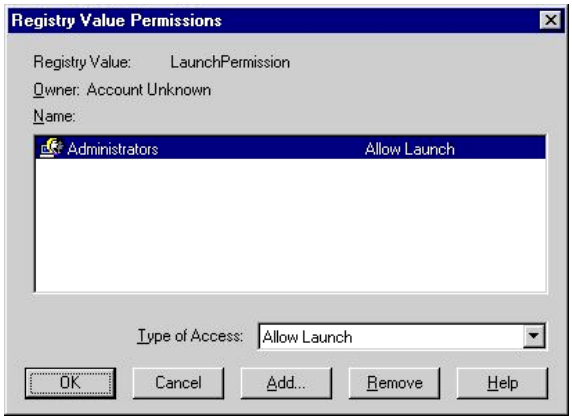

Step 2: At the Applications tab, choose the RTPro Document or ShkCtrl Document item and click Properties.




Step 3: At the Security tab, choose Use custom launch permissions and click Edit**.**




Step 4: Remove all the items under Name: except for the Administrator, and select Allow Launch in the Type of Access field.




After this operation, the error code will be returned immediately if the server application is not running.

### *How Can I Create a New Project on the Server Application?*

To create a new project on the Server Application, follow these steps:

Step 1: Assign the appropriate value to the `ApplicationType` property of the `NetICmd` ActiveX control. If you want to create the new project within the RTPro application, assign the value SVR_RTPRO; otherwise SVR_VCS for the Shaker Control application. This operation will connect the ActiveX control to the appropriate type of application. For example:

```
NetICmd.ApplicationType = SVR_RTPRO
```

Step 2: Assign the appropriate value to the `ProjectType` property of the `NetICmd` ActiveX control.

```
NetICmd.ProjectType = MPS_FFT
```

Step 3: Send the command: `CMD_NEWPRJ` (its value is `1`)

```
NetICmd.SendCommand CMD_NEWPRJ
```

This implementation will create a new project on a server application that has been launched.

### *How Do I Open an Existing Project on the Server Application?*

To open an existing project on the Server Application, follow these steps:

Step 1: Assign the appropriate value to the `ApplicationType` property of the `NetICmd` ActiveX control. If you want to specify a project within the RTPro application, assign the value SVR_RTPRO; otherwise SVR_VCS for the Shaker Control application. This operation will connect the ActiveX control to the appropriate type of application. For example:

```
NetICmd.ApplicationType = SVR_RTPRO
```

Step 2: Assign the appropriate value to the `ProjectName` property of `NetICmd` ActiveX control

```
NetICmd.ProjectName = "C:\LDS-Dactron\RTPro\FFT
Test.prj"
```

Step 3: Send the command `CMD_OPENPRJ`

```
NetICmd.SendCommand CMD_OPENPRJ
```

This implementation will tell the server application to open an existing project called "`FFT Test.prj`" that is located in the "`C:\LDS-Dactron\RTPro`" folder.

### *How Do I Send a Command to the Server Application?*

Sending commands from the Client ActiveX control to the server is easy. First, connect the `NetICmd` control to the server. Then call the `SendCommand` method to send the command to the server application. If the command needs to go with parameters, assign the appropriate value(s) to the `NetICmd` property before the `SendCommand` is called. Here is a VB example:

```
NetICmd1.SendCommand CMD_STARTMEAS
```

### *How Do I Read the Status from the Server Application?*

The server application returns the testing status to the ActiveX client. The testing status is categorized into two groups: an integer group and a floating point data group. All of them have a data type in 32-bit wide. The user can interpret the array on the client side after the status array is received. For example, the following list shows the first five elements of the integer array that is used in the RTPro.

| Index | Content |
|---|---|
| 0 | RunStatus; |
| 1 | DACStatus; /* =1 if DAC is open, 0=not */ |
| 2 | RunType; |
| 3 | AvgNbr; |
| 4 | CurFrmNbr; |

Here is an example in Visual Basic:

```
Private Sub NetIStatus1_OnUpdateStatus(ByVal nRunStatus As Long)
    NetIStatus1.ReadStatus
    Dim vdata As Variant, l As Long
    If NetIStatus1.ApplicationType = SVR_RTPRO Then
        l = NetIStatus1.GetIntStatusArray(vdata)
        txtFrmNum.Text = vdata(4)   'Index 4 is the Frames Number of Rt Pro
    Else
        l = NetIStatus1.GetFloatStatusArray(vdata)
        txtFrmNum.Text = vdata(0)   'Index 0 is the ElapseTime of VCS
    End If
End Sub
```

This example shows that the edit box content `txtFrmNum.Text` is assigned based on the server type. If the server is running RTPro, the frame number is retrieved. If it is a LDS-Dactron Shaker Control system, the Elapsed Time is retrieved.

The status can be retrieved at any time, but it is retrieved more frequently when the NetIStatus ActiveX control receives the `OnUpdateStatus` event, as shown in this example.

### *How Can I Identify All of the Signals in the Server Application?*

A signal is an array of data with a number of attributes. The array and attributes can be read by using signal names. That is, in order to read the signal, the client ActiveX control must know the name of the signal first.

Call Method `ReadAllSignalName` for the `NetISignal` ActiveX control to obtain all the signal names. After this method is called, the property `SignalCount` indicates the total number of signals. Following is a Visual Basic sample describing how to read all the signal names from the server application and assign all the names to a combo box represented by `cmbSignalNames.`

```
Private Sub AddSignal()
    'Get all signals
    Dim i As Long
'The follow call will read all the signal names into client
    NetISignal1.ReadAllSignalName

'Assign all the names to the combo box
    With cmbSignalNames
        While .ListCount > 1
            .RemoveItem .ListCount - 1
        Wend
    End With
    With NetISignal1
        For i = 0 To .SignalCount - 1
            cmbSignalNames.AddItem .Signals(i).Name, i + 1
        Next i
        If .SignalCount > 0 Then
            cmbSignalNames.ListIndex = 1
        End If
    End With
```

```
End Sub
```

### *How Do I Read the Signal Attributes from the Server Application?*

Call the method `ReadSigAttrByName` or `ReadSigMainAttrByName` to read the attributes for all the signals. Then go through the signal list by index to retrieve the signal attributes of an individual signal. Here is a Visual Basic sample:

```
NetISignal1.ReadSigMainAttrByName cmbSignalNames.Text
Dim n As Long
Dim nAllocSize As Long
Dim nDH As Single

'Find the index
n = NetISignal1.FindSignalIndexByName(cmbSignalNames.Text)
With NetISignal1.Signals(n)
    nAllocSize = .AllocSize
    nDH = .HIncreaseStep
End With
```

### *How Do I Read the Signal Array from the Server Application?*

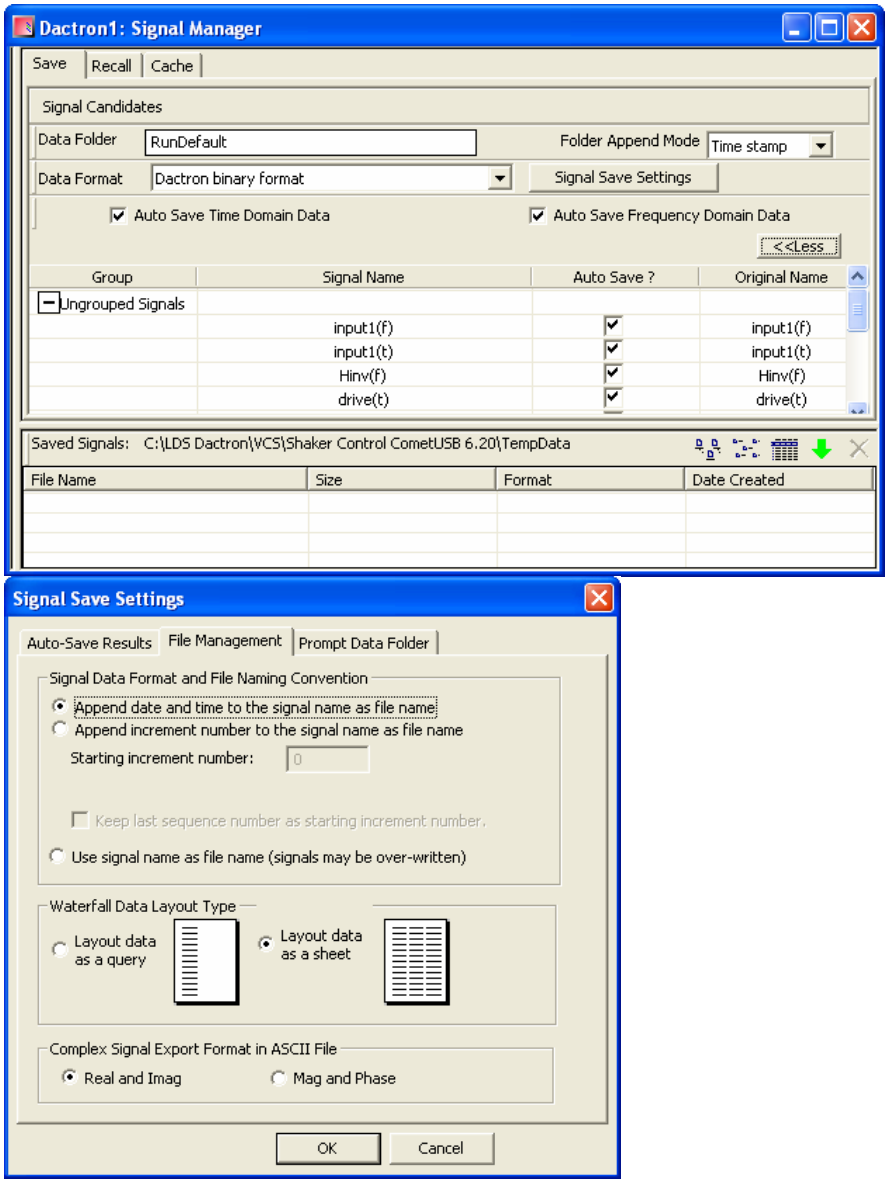
Usually the user wants to read the signal when the `NetISignal` ActiveX receives the `OnUpdateSignal` Event.

Simply call the `ReadSignalData` method to read the content of a data array. Then use the index to retrieve each element of the array. Here is an VB sample showing how to read the signal with name "`input1(t).sig`":

```
Private Sub NetISignal1_OnUpdateSignal(ByVal nSigNum As Long)
      Dim v As Variant
      Dim firstPoint as Single
      v = NetISignal1.ReadSignalData("input1(t).sig")
      ' obtain the first element of the array v
      firstPoint = v(0)
End Sub
```

## *How Can I Save the Signals?*

The client ActiveX can instruct the server application to save the signals, but the server application needs to know which signal to save. Before the server application saves the signals, the user must assign a destination folder and instruct how to name the signal files. Here are the dialog boxes in the Shaker Control application for setting up these parameters:




In the above setup, the signal names will be appended with date and time stamps and the signals will be saved in LDS-Dactron binary format to the current Data Folder folder. .

On the ActiveX client side, the user needs to set an `EnableOnLineSave` property then call the method `OnLineSaveSignal` to invoke the actions of saving signals on the server side. Here is a Visual Basic sample:

```
Private Sub cmdOK_Click()
    Dim n As Long, i As Integer
    With frmNIClient.NetISignal1
        For i = 0 To lstSaveSig.ListCount - 1
            lstSaveSig.ListIndex = i
            n = .FindSignalIndexByName(lstSaveSig.Text)
            .Signals(n).EnableOnLineSave = 1
        Next i
        .OnLineSaveSignal
    End With
End Sub
```

## Limited Warranty Statement

LDS warrants to you, the Buyer, that LDS hardware, accessories and supplies will be free from defects in material and workmanship, for a period of one year from date of shipment. If LDS receives notice of such defects during the warranty period, LDS will, at its option, either repair or replace products which prove to be defective. Replacement products may be either new or equivalent in performance to new.

LDS warrants to you that LDS software will not fail to execute its programming instructions, for a period of one year from date of shipment, due to defects in material and workmanship when properly installed. If LDS receives notice of such defects during the warranty period, LDS will, at its option, either repair or replace software media which does not execute its programming instructions due to such defects

LDS does not warrant that the operation of LDS products will be uninterrupted or error free.

LDS products may contain remanufactured parts equivalent to new in performance or may have been subject to incidental use.

Warranty does not apply to defects resulting from (a) improper or inadequate maintenance or calibration, (b) software, interfacing, parts or supplies not supplied by LDS, (c) unauthorized modification or misuse, (d) operation outside of the published environmental specifications for the product, or (e) improper site preparation or maintenance.

NO OTHER WARRANTY OR CONDITION, WHETHER WRITTEN OR ORAL, IS EXPRESSED OR IMPLIED AND LDS SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE.

THE REMEDIES IN THIS WARRANTY STATEMENT ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. EXCEPT AS INDICATED ABOVE, IN NO EVENT WILL LDS OR ITS SUPPLIERS BE LIABLE FOR LOSS OF DATA OR FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL (INCLUDING LOST PROFIT OR DATA), OR OTHER DAMAGE, WHETHER BASED IN CONTRACT, TORT, OR OTHER LEGAL THEORY.

**ASSISTANCE**

If you are unable to solve a problem with your LDS-Dactron product, contact your LDS Representative. To locate the LDS Representative in your area refer to LDS web site www.lds-group.com. You can also contact LDS-Dactron directly at using the contact information on the cover page.

**TO RECEIVE WARRANTY REPAIR SERVICE**

To obtain warranty service or repair, LDS-Dactron products must be returned to a service facility designated by LDS. Buyer shall prepay shipping charges to LDS and LDS shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to LDS from countries and locations outside of the United States.

## Manual Revision History

| Date | Manual Version | Software Version | Contents added into the Manual in this version |
|---|---|---|---|
| Apr. 2000 | 1.0 | RT Pro 2.1<br>Shaker Control 3.6 | Initial product Release for NET-Remote, NET-View; 109 pages. |
| June 2002 | 1.1 | RTPro 3.2<br>Shaker control 4.7 | Second release, separate the NET-Integrator and network/DCOM configuration into two manuals |
| March 2003 | 1.3 | RT Pro 4.0<br>Shaker Control 5.0 | Revise index table for NetIStatus |
| June 2007 | 1.7 | RT Pro 6.2<br>Shaker Control 6.2<br>NET-Integrator 1.7 | Updated Manual Version and Technical Support email address. Removed references to SpectraBook and Win95/98/NT Operating Systems. Revised NET-Integrator installation instructions, list of available NET-Integrator Sample Projects, and “How Can I Save the Signals” section, |