










## A High Performance Data Exchange Control for Microsoft Windows


### Introduction

-  License Agreement
  -  Overview of App-Link
  -  Whats New in Version 2.0?
  -  Installation
  -  Getting Started



### Control Reference

-  Properties
  -  Methods
  -  Events
  -  Functions
  -  Trappable Errors

### Operational Components

-  Using the APPLINK.INI File
  -  Communications Servers
  -  Network Setup
  -  Distribution of App-Link Applications

### Troubleshooting

-  Common Questions and Answers
  -  Technical Support

For Help on Help, Press F1

## License Agreement

### SYNERGY SOFTWARE TECHNOLOGIES INC.

#### APP-LINK SOFTWARE LICENSE AGREEMENT IMPORTANT: PLEASE READ CAREFULLY

**THIS IS A LEGAL AGREEMENT BETWEEN YOU AND SYNERGY SOFTWARE TECHNOLOGIES INC. ALSO KNOWN AS SSTI. CAREFULLY READ ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT PRIOR TO USING THE APP-LINK CONTROL. OPENING THE ENVELOPE CONTAINING THE SOFTWARE, OR USING THE SOFTWARE INDICATES THAT YOU ACCEPT THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THESE TERMS, DO NOT USE THE APP-LINK SOFTWARE.**

*The App-Link control is licensed to you as an individual, and may not be shared between users unless each user sharing the product has purchased an App-Link license. This restriction also extends to installation on a network, if more than one workstation will be accessing the product. The purchaser is hereby granted the right to distribute the RUN-TIME version of the App-Link control set and the NETBIOS server, so long as the purpose of the distribution is to augment software being distributed, and is not in any way competitive with App-Link itself as a programmer tool. At no time may the DESIGN-TIME version of the App-Link control be distributed.*

You may not sublicense, assign or transfer this License or the Software except as permitted by SSTI (call SSTI for transfer authorization and re-registration). Any attempt to sublicense, assign or transfer any of the rights, duties or obligations under this License is void. Solely for your own backup purposes, you may make a single copy of the Software in the same form as provided to you. You may not remove, obscure, or alter any notice of patent, copyright, trademarks, trade secret or other proprietary rights in the Software. You may not disassemble the software, or attempt to reverse engineer it in any way.

SSTI has made reasonable checks of the Software to confirm that it will perform in normal use on compatible equipment substantially as described in the then current specifications for the Software. However, due to the inherently complex nature of computer software, SSTI does not warrant that the Software or the Documentation is completely error free, will operate without interruption, is compatible with all equipment and software configurations, or will otherwise meet your needs.

ACCORDINGLY, THE SOFTWARE AND DOCUMENTATION ARE PROVIDED AS-IS, AND YOU ASSUME ALL RISKS ASSOCIATED WITH THEIR USE. SSTI MAKES NO WARRANTIES EXPRESSED OR IMPLIED, WITH RESPECT TO THE SOFTWARE OR THE DOCUMENTATION, THEIR MERCHANTABILITY, OR THEIR FITNESS FOR ANY PARTICULAR PURPOSE OTHER THAN THOSE WARRANTIES WHICH ARE IMPLIED BY OR INCAPABLE OF EXCLUSION, RESTRICTION OR MODIFICATION BY APPLICABLE LAW. TO THAT EXTENT, ALL WARRANTIES, EXPRESS OR IMPLIED, WILL TERMINATE UPON THE EXPIRATION OF 30 DAYS FOLLOWING DELIVERY OF THE SOFTWARE TO YOU. IN NO EVENT WILL SSTI BE LIABLE FOR INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS OF INCOME, USE, OR INFORMATION. AS YOUR SOLE REMEDY FOR ANY BREACH OF WARRANTY, you may return to SSTI the original copies of the Software and Documentation, along with proof of purchase (if you purchased the Software alone or in combination with any other software) and any backup copies, for replacement or (at SSTI's choice) for a refund of the amount you paid, provided the return is completed within 30 days following delivery to you.

**Manual Copyright**      ©1996 Synergy Software Technologies Inc. All rights reserved.  
First printing, September 1996

Synergy Software Technologies Inc.  
159 Pearl Street  
Essex Junction, Vermont 05452

**Trademarks**            App-Link, RADX and Rapid Application Data eXchange are trademarks of Synergy Software Technologies Inc. Microsoft, MS and Visual Basic are registered trademarks of Microsoft Corporation. Other brands or products are trademarks or registered trademarks of their respective holders and should be treated as such.

**Copy and Use Restrictions**        Although you are encouraged to make backup copies of the Software for your own use, you are not allowed to make unlimited copies. The Software is protected by

the copyright laws that pertain to computer software. It is illegal to make copies of the software except for backups. It is illegal to give copies to another person, or to duplicate the Software by any other means, including electronic transmission. The Software contains trade secrets, and in order to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to human-perceivable form. You may not modify, adapt, translate, lease, loan, resell for profit, distribute, network or create derivative works based upon the Software or any part thereof, except as allowed in the license agreement.

**Media Warranty**

Synergy Software Technologies Inc. warrants that the original media this Software is distributed on is free from defects, assuming normal use, for a period of ninety (90) days from the date of purchase. If a defect occurs during this period, simply contact Synergy and return the faulty media. Upon receipt of the defective items, Synergy will immediately supply you with replacement media.

IN ANY CASE, THE LIABILITY OF SYNERGY SOFTWARE TECHNOLOGIES UNDER THE WARRANTY SET FORTH ABOVE SHALL BE LIMITED TO THE AMOUNT PAID BY THE CUSTOMER FOR THE PRODUCT. IN NO EVENT SHALL SYNERGY SOFTWARE TECHNOLOGIES INC. BE LIABLE FOR ANY SPECIAL, CONSEQUENTIAL, OR OTHER DAMAGES FOR BREACH OF WARRANTY.

# Thanks!

*The folks at Synergy would like to thank the beta testers and existing customers that provided input for App-Link RADX. Your input is greatly appreciated!*

## Overview of App-Link

Thank you for purchasing the App-Link Rapid Application Data Exchange (RADX) Control Set. We are sure that you will find the control set to be easy to use, yet powerful and flexible. We designed App-Link to provide the most intuitive, reliable and efficient method for data exchange available. Consider App-Link your own personal data-exchange superhero -- App-Link does all of the dirty work so that you can stick to the fun stuff! After all, that is what visual programming is all about.

As you probably already know, most visual programming environments present information to application users on what it calls a "form". A form is essentially a window that contains controls or "objects", program code, and data. When you create an application visually, you define forms, their contents, and how they are presented to the user and logically interact during execution. Forms are a kind of "building block" for your application. Each form is in essence its own encapsulated "mini-application", or "mini-app" (we like that word, "app" -- it's easier to say than "application").

Now, to quote Rod Serling, imagine if you will... that your mini-apps (forms) are rooms. Further, imagine that your application is a house containing these rooms. Up one more level, imagine that a single computer is a neighborhood containing houses. (Are you still with us?) Lastly, imagine that groups of neighborhoods are like PCs on a Local Area Network. If you can imagine this metaphor, then throw in one more detail. Forms are like the rooms in this hierarchy, but unfortunately, the rooms do not have any phone jacks, which makes it very difficult to communicate from one room to another. In essence, App-Link allows you to place virtual phone jacks in your rooms. In fact, the App-Link custom control icon looks just like a phone jack when you place it on a form!



This is the graphic of the App-Link control. We call it a socket. For those of you familiar with TCP/IP sockets, the concept is essentially the same. We do not pretend to have invented the term socket, but we think that it is a great conceptual representation for a delivery/receipt address.

Now, before we go too far, please understand that App-Link is *not* a telephony communication control. The phone socket metaphor is used simply to illustrate the intent of the App-Link control set. App-Link is, however, a high performance data exchange control, and allows applications to transmit and receive data messages between the App-Link sockets that they contain. Best of all, App-Link is very easy to use, allowing you to rapidly enable your applications to exchange data on many different levels. Rapid Application Data eXchange means App-Link is fast, reliable and easy to use in a large variety of Windows visual programming environments.

### Three Dimensional Data eXchange

App-Link allows for data exchange at three different levels within your application. The methodology used at each level is the same, however, making usage consistent within your code. The three levels at which data exchange may occur within App-Link are:

#### *Between sockets within a single application.*

Data exchange at this level allows for the loose coupling of forms within an application. All application activity can be driven by messages between forms. This allows you to create your own events that are fired by the receipt of App-Link based messages.

#### *Between sockets in separate apps running on a single computer.*

A more powerful application of App-Link's abilities, this level of data exchange allows data to be easily

transferred between applications without the use of disk files. Also, App-Link can load an application into memory automatically so that it may receive a message from a socket on demand. Using App-Link in this context allows for the creation of encapsulated program objects that can be driven entirely from App-Link messages. This level of exchange is similar to using DDE for local messaging, except that the App-Link control eases the requirements on the programmer by providing a highly intuitive interface, and performing much of the underlying dirty work normally required.

*Between sockets in separate applications running on different computers, and connected via a physical transport, such as a Local Area Network.*

Perhaps the most exciting use of App-Link, this level provides high-performance data exchange between sockets residing on separate physical computers. Using App-Link at this level allows the programmer to instantly LAN-enable an application. Applications that use App-Link can become network aware, and dynamically exchange data without the need for shared disk files. A much faster alternative to NETDDE, App-Links network data transport provides high performance, reliability, and best of all, extreme ease of use.

The control is designed such that support for a variety of transports is modular and is selected via a property. The base App-Link control set is shipped with a NetBIOS transport that may be used without royalties or restrictions. Synergy Software Technologies Inc. is developing other transport drivers that will be released over time, making App-Link a highly versatile tool for data exchange on a variety of physical and logical platforms.

## Lets talk Sockets

Think of a socket as a means of addressing a source or destination within a computer for routing an App-Link message. Note that a socket address is unique within a physical computer. The nice part is that you have the ability to control the address by giving each socket a name. Of course, App-Link will create a unique name for you if you wish. The important part to remember is that each socket that is in memory on the same computer must have a unique name.

Sockets that reside on different computers may have the same name. What makes them unique to each other is the name of the computer on which they reside. Considering this, lets look at the following examples:

**Socket1**                      A local socket called Socket1, assumed to be on the same computer on which it is referenced.

**\\Server\\Socket1**            A remote socket called Socket1 which resides on the computer named Server.

Remember, the socket name is assigned directly by the programmer using the *SocketName* property, or App-Link will generate it automatically. The computer name is taken from the environment, and is usually the name assigned to the computer during network setup. If you are unsure what the name of a particular computer is, then you can use the [GetComputerName method](#) to retrieve the name. For example:

```
Text1.Text = skt1.GetComputerName()
```

This statement will set the contents of the text box Text1 to the name of the local computer that the program is running on.

## How App-Link is used to send a message

If you are like most programmers, you are just dying to get this thing loaded in your application and start winging around messages. You'll be pleased to know that App-Link's ease of use is its strong point, and in no time at all you will be an App-Link expert.

Once you install the App-Link control set, you are ready to use the App-Link control objects within in your development environment. The control set includes a variety of control types, each intended for different development environments. The most versatile of the controls are the ActiveX types (AL16.OCX and AL32.OCX), as they can be used in a number of different containers. For the purpose of this illustration, we will assume that you are using the OCX version of the control in the Microsoft Visual Basic Integrated Development Environment (IDE). If you are using a different development facility, then refer to its user manual for information on using custom controls. App-Link adheres to custom control standards, thus, it should be usable in any environment that supports these control types.

When you add the App-Link OCX to your application, the familiar App-Link socket icon appears in your Visual Basic toolbox. Selecting the App-Link icon or dragging it to a form in your application places a socket on the form. It appears on the form as a graphic icon of a phone socket, which is a good way of thinking of it. Incidentally, you can place as many sockets on a form as you like. We've seen some forms that look like a gigantic pbx phone hub!

Like other controls, an App-Link socket has its own properties, events, and methods (pseudo methods are used with VBX because custom methods are not supported by the Microsoft VBX architecture). Once on your form, each individual App-Link socket can be assigned its own properties.

One of these properties is `SocketName`. As mentioned earlier, the [SocketName property](#) essentially provides an App-Link socket with an address. Once a socket has a name, other sockets may specify it as a destination. For example, to send the string Hello there from socket `skt1` to socket `skt2` simply use the `Send` method as follows:

```
skt1.Send(skt2,Hello there)
```

Pretty easy, huh?

The `Send` method also allows you to specify a message length, and a `flags` parameter that allows for various sending options. You may ask why the message length would ever need to be passed? (good question!) Well, App-Link includes functions that allow you to convert Visual Basic user defined types into strings so you can send structures within App-Link messages. Because these structures sometimes contain embedded nulls, there is really no way to accurately determine their length by convention, thus the need for the length parameter. If you are simply sending a string, then there is no need to specify the length. App-Link will calculate it internally.

In the following example, a message is sent from `skt1` to `skt2` when the user clicks on the command button `cmdSend`. Note that the length of the message is explicitly specified, and the send flag `START_DEST_APP` is used to indicate that the application containing `skt2` is to be loaded if it is not already in memory.

```
Sub cmdSend_Click()  
    skt1.Send(skt2,Hello there,11,START_DEST_APP)  
End Sub
```

It almost seems too easy, doesn't it? Well, it's just as easy on the receiving side too. Each socket has its own event that fires when a message arrives for it to receive. The event is called [ReceiveMessage](#) and provides everything necessary for the processing code to do its thing.

Lets take a look at the `ReceiveMessage` event for `skt2` that will be fired when the message sent by `skt1` arrives:

```

Sub skt2_ReceiveMessage(Orig As String, MsgLen As Long, Msg As String, RetCode As Long)
    txtFrom.Text = Orig
    txtMessage.Text = Msg
End Sub

```

When the ReceiveMessage event fires, the text box txtFrom will contain whom the message came from (the originator) and txtMessage will contain the message text. The MsgLen parameter provides the length of the message, and the RetCode parameter can be used to provide a return code to the sending socket when a message is sent synchronously.

As you can see, App-Link makes it very easy to exchange data between sockets. The previous example demonstrated sending data between sockets that reside on the same computer. Sending data between physically different computers is only a matter of adding the computer's address to the socket name in the destination parameter.

The address of a computer is typically some kind of name that is assigned by a network administrator. This name is added to the socket name to fully qualify the target socket. For example:

```

skt1.Send(\\SERVER\skt1,Hello there)

```

This statement sends the message Hello there to the destination socket *skt1* on computer *SERVER*. That's all there is to sending data over a LAN with App-Link. App-Link does all of the internal nitty-gritty work associated with network data transmission. All you need to do is tell App-Link where to send the message!

You can also broadcast a message to a specific socket name on the LAN by using the Broadcast method. Receiving sockets can reply back to the broadcaster, resulting in a registration of the socket and the establishment of a manageable network connection.

The following example broadcasts the message Please check in to all sockets named *initsock* on the LAN:

```

skt1.Broadcast("initsock",Please check in)

```

As you can see, App-Link makes it very easy to exchange data between your applications, or even within the same application. By handling all of the low level protocol expectations, error recovery, retries, etc. App-Link allows you to focus on the design and development of your application.

## Ideas for App-Link development

We have illustrated a very basic example of how to use App-Link to exchange data between your applications in this introduction. App-Link has a variety of other capabilities that are described in detail in this manual.

The best way to learn about App-Link is to dive in and start sending messages. We have provided some example projects that are designed to illustrate App-Link functionality. Playing around with these samples is an excellent way of getting your feet wet with the control.

As you use App-Link, you will begin to discover that the power associated with the control is exciting. It opens up new avenues for the development of creative applications, especially in the area of network-based development. Some examples of how you could use App-Link are:

### Groupware Development

App-Link is the ticket to turning a stand-alone application into a Groupware product. Using App-Link, a programmer can easily establish real-time connections between applications over a network. Since receipt of App-Link messages is event-driven, the program designer can easily integrate Groupware functionality into an existing stand-alone app. Groupware is seen as being one of the highest growth areas in software for the '90s. What Visual Basic brings to application development in terms of productivity App-Link offers in the area of application messaging, which is fundamental to Groupware. The combination of a visual development tool and App-Link is a powerful force in the Groupware application development arena.

### E-Mail and Chat Facilities

Have you ever thought of developing your own form of E-Mail or Chat facilities? Perhaps you need the ability to exchange E-Mail from within an existing application, but you do not want to have to learn the API's associated with large monolithic E-Mail products. App-Link offers you an easy solution, making development and integration of E-Mail functionality with applications simple and affordable.

### Networked Telephony

Telephony has become one of the hottest application development growth areas. With the advent of custom controls that allow Visual Basic to interface with PBX cards, telephony application development has skyrocketed. App-Link takes telephony development one step further, and provides the developer with a means of networking the application, thus providing LAN access to the information received and transmitted by the PBX adapter through the telephony interface.

### Networked Data Acquisition and Control Systems

Many DACS hardware manufacturers now provide custom controls as the interface to their hardware. Using App-Link, the DACs developer can easily develop a distributed DACS application. DACS hardware residing on distributed PCs can transfer and receive data over the LAN using App-Link. Since App-Link is event driven, it lends itself easily to the DACS environment, providing the ability to report and control the DACS information on a real-time basis.

### Database Polling Elimination

Many developers use database polling to control their applications on a network. What this amounts to is having a timer control on a form that fires every so often. When the timer fires, application code is run that queries a shared database or file for information on state changes within the application. While this method can work, it is not optimal. Polling is slow, and depending upon the frequency, does not report state changes on a real-time basis. It also chews up disk space unnecessarily, and can present joint database access problems. App-Link eliminates the need for database polling, and offers a quick and easy solution for the monitoring of application state changes between networked computers.

### Client-Server Development

The fundamental building block for Client-Server development is messaging. App-Link offers a powerful messaging infrastructure to the developer, making Client-Server development easy. With App-Link's ability to automatically load an application into memory, a programmer can write server applications that can be invoked on demand from clients. Once the server has done its work for the client, it can be unloaded from memory, freeing up resources for other tasks. Also, server objects can be shared between programs, thus reducing the overhead and redundancy of tasks common to multiple applications.

### Creative Events

How often have you wished for a special event to fire within your application whenever an event that YOU



defined occurs? With App-Link, you can in essence create your own events that fire at your discretion. By placing a socket on a form, you create a means of sending a message to another socket within the same application. When the message is sent, it will cause a ReceiveMessage event to fire at the other socket. The message content could then be examined in order for the application to take the appropriate action. Thus, by using the SENDMSG ability of App-Link within code, you can fire an event whenever you feel it appropriate within your application.

The potential uses of App-Link go on and on. We feel that App-Link opens the doors to new and exciting applications that were difficult or impossible without App-Link's ease of use and versatility. We hope that you enjoy using App-Link, and wish you the best of luck in creating that next killer-app!

## What's New in Version 2.0

The following items run down the major differences between App-Link 2.0 and version 1.1:

### ✓ **New 32-bit and 16-bit ActiveX (OCX) Controls**

Now App-Link can let you communicate between any development environment (container) that supports a VBX or OCX.

### ✓ **32-bit High Performance NetBIOS Server**

The best just got better!

### ✓ **New Property Page Dialog Available for OCX**

Easily configure your socket properties.

### ✓ **New Methods**

You asked for it, you got it!

*Broadcast*  
*GetComputerName*  
*GetNextQueuedMessage*  
*ReceiveNextQueuedMessage*  
*Send*

### ✓ **New Events**

More of what you asked for...

*Error*  
*NotifyMessage*

### ✓ **New Manual**

The App-Link manual has been updated with new information and corrections to prior material.

### ✓ **New Example Programs**

New examples are included that further illustrate the use of the control.

### ✓ **New World Wide Web Site**

We now have a World Wide Web site with loads of information. Check it out at:

**[www.synergysw.com](http://www.synergysw.com)**

Note that App-Link 2.0 is upward compatible with version 1.1.

## Installation

To install the App-Link RADX Control Set, simply place the installation diskette in your floppy drive, and execute the SETUP.EXE program found on the diskette from the Microsoft Windows Program Manager. The setup program will lead you through the entire installation process.

Upon completion of the setup procedure, one or more of the following files are copied to your system depending on the setup options chosen:

### Common Files:

<u>Target Directory \ File Name</u>	<u>Description</u>
WINDOWS\SYSTEM\APPLINK.HLP	App-Link help file
WINDOWS\SYSTEM\APPLINK.LIC	Design-time license file
WINDOWS\APPLINK.INI	Configuration settings
AL20\README.WRI	General Readme file in Write format
AL20\UNINSTAL.EXE	Uninstaller program

### 16-bit Environment:

<u>Target Directory \ File Name</u>	<u>Description</u>
WINDOWS\SYSTEM\APPLINK.VBX	Design/run-time custom control VBX
WINDOWS\SYSTEM\AL16.OCX	16-bit ActiveX control
WINDOWS\SYSTEM\APLKAPI.DLL	16-bit message transport interface
WINDOWS\SYSTEM\APLKDAEM.EXE	Message transport daemon process
WINDOWS\SYSTEM\APLKNB.EXE	16-bit NetBIOS server
WINDOWS\SYSTEM\APLKNBP.DLL	16-bit NetBIOS server library
AL20\SAMPLES\OCX\	16-bit OCX sample applications
AL20\SAMPLES\VBX\	16-bit VBX sample applications
AL20\REDISTWIN16\	16-bit run-time redistribution files

### 32-bit Environment:

<u>Target Directory \ File Name</u>	<u>Description</u>
WINDOWS\SYSTEM\AL32.OCX	32-bit ActiveX control
WINDOWS\SYSTEM\APLKAPI.DLL	Thunk layer for 16-bit support under Win32
WINDOWS\SYSTEM\ALAPI32.DLL	32-bit message transport interface
WINDOWS\SYSTEM\ALNB32.EXE	32-bit NetBIOS server
AL20\SAMPLES\OCX\	32-bit OCX sample applications
AL20\REDISTWIN32\	32-bit run-time redistribution files


The README.WRI file should be printed and read prior to using App-Link. It will contain last-minute updates to the manual and other tid-bits of information that we would really like you to know about before charging forward.

## Getting Started

As with many programming tools, the best way to get started with App-Link is to dive right in and start using it. There are two routes that we recommend you choose from for your initial exposure to the control. If you are an avid and experienced Visual Basic programmer, then we recommend that you take a look at the sample programs that were shipped with the App-Link package. They provide a good overview of the basic operation of the control, and can serve as a spring-board for getting you going with App-Link in your application.

The other route is to "follow the bouncing ball", if you will, and take the steps outlined in the following script. The intent of the script is to guide you through the creation of a very simple App-Link application using Visual Basic. From there we encourage you to experiment with the control, and to decide for yourself how to best use it in your application. If you are using another environment such as Delphi or C, then the environment procedures (i.e. loading the control into the environment, etc.) will be slightly different, but the concepts are the same. Have fun!

Follow these steps to create your first App-Link application:

1. Start Visual Basic (Version 4 assumed) with the default project.
2. From the Tools menu, choose the Custom Controls... option. Select **Synergy App-Link Control** in the Available Controls list and click on the OK button to add App-Link to your project.
3. Great! You should now see the App-Link socket icon (  ) in your toolbox.
4. Click on the socket icon in the toolbox, and then go to the blank form to the right and drag the mouse on the form. Releasing the left mouse button should result in an App-Link socket being placed on the form. The socket cannot be sized (you won't need to size it).
5. With the socket still selected, press F4 to bring up the properties window for the socket. In the SocketName property, type the name **Socket1**. You have now given the socket a name that can be used as an address. Note that this is different from the Visual Basic standard property called *Name*, which is the name for the control. It is a good practice, however, to keep these two "names" the same.
6. Resize the form so that it is about 2" by 2", and then add a command button to the form. Now update the caption property for the command button to "SEND".
7. Add two small text boxes to the form. Their names should default to **Text1** and **Text2**. Blank out their *Text* properties. Put a caption next to the text box Text1 that says "MSG OUT", and one next to the text box Text2 that says "MSG IN".
8. Add another blank form to the project.
9. Add an App-Link socket control to the new form. Using the properties window, update the *SocketName* property for the socket to be **Socket2**.
10. Add a command button and two text boxes (blank out their text properties) to the form, making it look identical to the first form (Form1). Add the captions to the text boxes as in step 7.
11. Go back to Form1 and double-click on the command button to bring up its click event procedure. Type in the following code:

```
Socket1.Dest = "socket2"  
Socket1.Msg = Text1.Text  
Socket1.Msglen = Len(Text1.Text)  
Command1.Enabled = False  
Socket1.Command = 1  
Command1.Enabled = True
```

12. Double click on the socket Socket1 to bring up its [ReceiveMessage event](#) procedure. Type in the following code:

```
Text2.Text = Msg
```

13. Go over to Form2 and double click on the command button to bring up its click event procedure. Type in the following code:

```
Socket2.Dest = "socket1"  
Socket2.Msg = Text1.Text  
Socket2.Msglen = Len(Text1.Text)  
Command1.Enabled = False  
Socket2.Command = 1  
Command1.Enabled = True
```

14. Double click on the socket Socket2 to bring up its ReceiveMessage event procedure. Type in the following code:

```
Text2.Text = Msg
```

15. Okay. There is one more thing to do before we run this example. Go back to Form1 and double-click on the form itself to bring up its Form\_Load procedure. Type in the following code:

```
Form2.Show
```

Since Form1 is the startup form, when it loads this line will cause Form2 to also load. Go ahead and click on the **Run** button.

If you get an error, then make sure that you typed in the code exactly as displayed.

16. When the application starts, type in some text into either of the MSG OUT text boxes and click on the SEND button. The result should be that the content of the text box is transferred to the MSG IN box of the other window. Congratulations, you have just written a very simple CHAT facility using App-Link!

The application that you have just produced is using App-Link at its most simplistic level. That is, you are sending messages between windows that are within the same application.

For your next step, try breaking up the application such that the two forms are in two *separate* applications. App-Link only needs the socket name for an address, so the actual code does not have to change even if the forms reside in two separate executables.

Once you have succeeded in getting the application to work as separate executables, the next step is to get them to work across a Local Area Network. To do this, simply add the computer name to the [Dest](#) properties. For example, if the Dest property of the sending socket is now **Socket1**, but the application containing **Socket1** is now being moved to another computer with the name of **COMP1**, then the new Dest property on the sending machine should be:

"\\COMP1\\Socket1"

If you are unsure what your computer names are, then refer to the section on the Dest property for more detailed information.

Once you have finished this exercise, we highly recommend that you take a look at the sample programs. They offer examples of using App-Link at all levels, and are a great kick-start for your adventures with the control.

## Properties

The properties used by the App-Link control set are described in this section. Some of the properties are standard Microsoft Visual Basic properties, and are not described in this manual. For information on standard Visual Basic properties, please refer to the *Microsoft Visual Basic Language Reference*.

The number of properties implemented in the control set has been purposefully kept at a minimum in order to minimize the complexity of App-Link. Also, note that many of the properties are maintained solely for the purpose of upward compatibility between the VBX and ActiveX/OCX control versions. The methods available with the ActiveX version of the control make some of the properties obsolete (for example, the Command property is no longer needed). However, in order to ease the transition from VBX to ActiveX implementations, we continue to support the VBX-oriented properties. However, when using the ActiveX control, we strongly encourage the use of methods because they make the code more compact.

The properties used by the control set are as follows.

- (About)
- (Custom)
- [Command](#)
- [Dest](#)
- Enabled
- [Error](#)
- [ErrorText](#)
- [Flags](#)
- Index
- Left
- [Msg](#)
- [MsgLen](#)
- Name
- [Priority](#)
- [Protocol](#)
- [QueueAccess](#)
- [QueueCount](#)
- [QueueStorage](#)
- [SocketName](#)
- [Timeout](#)
- Tag
- Top

VBX	OCX16	OCX32
-----	-------	-------

## Command Property

### Example

#### Description

Visual Basic VBX (Visual Basic Extension) custom controls cannot have custom methods associated with them (unlike the newer ActiveX/OCX controls). The App-Link VBX uses a common technique for the implementation of custom methods that we call pseudo-methods. The technique employed treats the assignment of a property as a request to execute a method. The App-Link *Command* property is that special property.

Setting this property equal to one of the methods or command values tells App-Link to perform a particular action.

*This property may also be used by the ActiveX control for upward compatibility, however, we encourage the use of the App-Link ActiveX methods as a more compact and readable alternative to setting properties.*

---

#### Usage

`[form.]Socket.Command = method%`

#### Remarks

The method is selected by specifying one of the following values for *method%*. Constants for these values may be found in the Applink.Bas file.

Value	Symbol	Description
1	SENDMSG	Sends a message to the destination socket.
2	RCVMSG	Receives the next message from the App-Link message queue.

App-Link uses the Microsoft DDE Management Library (DDEML) as the messaging infrastructure within a single computer. The DDE protocol is based on the exchange of pre-defined Windows messages between participating applications. Some messages are sent directly, while others are posted to the recipients message queue. Since the Windows default message queue size can only accommodate 8 posted messages, it may become necessary to change the size of the message queue, particularly if you intend to send a bunch of asynchronous messages all at once (e.g. within a loop of some kind). The message queue size may be changed by updating the DefaultQueueSize option under the [Windows] section in the WIN.INI file found in the Windows subdirectory. For most applications, this is not necessary, however we recommend increasing the default queue size for message-intensive processing.

#### Data Type

Integer





## Command property example

This example invokes the custom *Send* method to deliver the message Hello to a destination socket named *Receiver* within the same computer.

```
Sub SendButton_Click ()  
    skt.Msg = "Hello"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "Receiver"  
    skt.Command = SENDMSG  
End Sub
```



## Dest Property

### Example

#### Description

This property specifies the name of the App-Link socket to receive the message being sent.

*This property may also be used by the ActiveX control for upward compatibility, however, we encourage the use of the App-Link ActiveX methods as a more compact and readable alternative to setting properties. This property is now included as a parameter to the Send method.*

---

#### Usage

[form.]Socket.Dest[ = destsocketaddr\$ ]

#### Remarks

The *destsocketaddr* value consists of at most two parts: an optional computer name, followed by the destination socket name. If the specification includes the computer name component, a pair of backslash characters (\\) must precede the computer name and a single backslash character (\) must separate the computer name and socket name. For example, the following assignment specifies a destination socket named *Receiver* on node *NODEX*.

```
MySocket.Dest = "\\NODEX\\Receiver"
```

The computer name, otherwise known as the "machine name" is a mnemonic assigned to the subject computer to allow it to be addressed on a LAN. This name is typically assigned by a network administrator during installation of the network software on the user's machine. In some cases, however, the name may not be apparent. For example, the Novell Netware NETBIOS emulator does not provide a means for naming a computer. In such a case, App-Link provides the *MachineName* option in the App-Link .INI file, which allows a name to be assigned to the computer. Optionally, the *MachineName* option may be expressed as a parameter upon invocation of the App-Link communications server.

For a local socket, only the socket name needs to be specified, as shown:

```
MySocket.Dest = "Receiver"
```

If you are unsure of the name of your local computer, then the App-Link ActiveX method *GetComputerName* may be used to extract the computer name from the system. Refer to the Methods section for more information.

#### Broadcast Communications

App-Link broadcast communications provides the programmer with the capability of sending a message to a given socket on every computer that has an App-Link communications server loaded. The destination application containing the receiving socket must also be running for the message to be delivered. Auto-application loading is not supported for broadcast sends.

Broadcast support is a connectionless service. That is to say, there is no guarantee that messages will be delivered to the destination socket, or that they will arrive in the same sequence in which they were sent. This is a network restriction, *not* an App-Link restriction. Regardless of whether a message arrives safely, no acknowledgement of reception is provided. Broadcast messages are always sent

asynchronously using [NO\\_WAIT](#) mode. If you require assurance that a message is delivered to its destination, then you should explicitly specify the destination computer name and send the message synchronously.

Broadcast messages are typically very small. App-Link limits the maximum size of a message sent using broadcast communications to 256 bytes. An error will occur if you attempt to broadcast a message larger than 256 bytes.

Many system designers want to use broadcast from the server side to implement some sort of automated user sign-in or registration. We recommend originating user sign-in on the client side by having the client send the server a message that, in effect, says I'm here, and here is my name. The server can then maintain a connection table to communicate with its clients. Using this method, synchronous transactions may be used, allowing the client to determine if the server is available. Also, once a connection has been established the server can monitor the connection by sending small, periodic messages to clients to test the connection. While this is not necessary, the amount of regulation and monitoring of connections depends upon the needs of the individual application.

To specify that a message should be broadcast, simply specify an asterisk (\*) for the computer name within the *Dest* property and include the destination socket name as usual. For example, assigning the *Dest* property a value of "\\\*\\INBASKET" specifies that a message should be broadcast to a socket named INBASKET on every computer having an App-Link communications server loaded.

*The ActiveX version of App-Link now includes a Broadcast method. Refer to the Methods section for more information on using Broadcast.*

---

## **Data Type**

String <51 bytes>



## Dest property example

### Example 1

This example sends a message to a socket on the same computer as the sender. The destination socket is named *LocalSocket*.

```
Sub SendButton_Click ()  
    skt.Msg = "This message is for a local socket"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "LocalSocket"  
    skt.Command = SENDMSG  
End Sub
```

### Example 2

This example sends a message to the same socket as in Example 1, but located on another computer. The destination computer name is *NODEX*.

```
Sub SendButton_Click ()  
    skt.Msg = "This message is for a remote socket"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "\\NODEX\\LocalSocket"  
    skt.Command = SENDMSG  
End Sub
```

### Example 3

This example broadcasts a message to a socket called REGISTER on all remote computers that have the App-Link communications server loaded.

```
Sub GetRegistration_Click ()  
    skt.Msg = "Sign in please!"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "\\*\\REGISTER"  
    skt.Command = SENDMSG  
End Sub
```



## Error Property

### Example

### Description

The Error property holds the last return code recorded by the socket. It may be used by the sending program to retrieve the user-defined value stored in the *RetCode* parameter of the [ReceiveMessage event](#) handler. This value is available to the sender during a synchronous send. It can also contain any number of return codes from normal App-Link operation. The Error property is read-only at run time, and not available at design time.

### Usage

```
error& = [form.]Socket.Error
```

### Remarks

When a program executes a synchronous send to another socket, the receiving socket has the option of setting the value of the *RetCode* parameter of the ReceiveMessage event. Upon exit from the ReceiveMessage event, the value contained in the *RetCode* parameter will be placed in the *Error* property of the sending socket. When the synchronous send is complete, the sending program may query this property to obtain any type of user-defined error code from the receiving socket. Note that the error may also be an internal App-Link-generated error. In such a case, the error code will be equal to one of the ones listed in the back of the manual.

### Data Type

Long



## Error property example

This example sends a synchronous message to a socket named *DBQUERY*. The routine then queries the *Error* property upon return from the send to determine if an error occurred on the receiving end.

```
Sub SendButton_Click ()  
    On Error Resume Next  
    skt.Command = SENDMSG  
    If skt.Error = 0 Then  
        txtMsg.text = "The server replied all is OK!"  
    Else  
        ErrorHandler(skt.Error)  
    End If  
End Sub
```



## ErrorText Property

### Example

### Description

The ErrorText property is used to obtain the textual description of the [Error property](#) value. This property is read-only at run time, and not available at design time.

### Usage

```
errortext$ = [form.]Socket.ErrorText
```

### Remarks

If an App-Link internal error occurs during a synchronous send, then this property may be used to display a textual description of the error encountered. If the *Error* property is set to a user-defined error code as a result of the receiving socket setting the *RetCode* property of the *ReceiveMessage* event, then the *ErrorText* property will contain the text "User defined error". Also, if the error code is zero the error text returned will be "No errors".

A complete list of App-Link error codes and their descriptions may be found in a later section of this manual.

### Data Type

String



## ErrorText property example

This example displays a message box containing the textual error description when a non-zero error is encountered following a synchronous send.

```
Sub SendButton_Click ()
    skt.Msg = "POST 00345998735 Smith, James"
    skt.MsgLen = Len(skt.Msg)
    skt.Dest = "\\MAILMAN\INBOX"
    On Error Resume Next
    skt.Command = SENDMSG
    If skt.Error <> 0 Then
        Use the Windows MessageBox API because VBs MsgBox
        implementation will eat App-Link messages while displayed
        x = MessageBox(0,skt.ErrorText,Send Error,0)
    End If
End Sub
```





## Flags Property

### [Example](#)

### Description

This property is used to specify options that control the behavior of App-Link. The Flags property is not available during design time.

*This property may also be used by the ActiveX control for upward compatibility, however, we encourage the use of the App-Link ActiveX methods as a more compact and readable alternative to setting properties. This property is now included as a parameter to the Send method.*

---

### Usage

*[form.]Socket.Flags[ = setting& ]*

### Remarks

One or more flag options may be combined together using the Visual Basic OR operator. Combining flag values turns more than one option on at the same time. Constants for these flags can be found in the Applink.Bas file.

The Flags property settings are:

Value	Symbol/Description
&H0	<b>DEFAULT_FLAGS</b>  The default value is 0 (all flags set to OFF). This setting defines a synchronous, non-queued send without auto-start capability.
&H1	<b>QUEUE_MSG</b>  This flag causes the Send method to place the message in the message queue associated with the destination socket. If this option is not set, the message will be sent directly, the receiving sockets <i>ReceiveMessage</i> event will fire immediately upon receipt of the message.  If a message is queued, the receiving socket must use the RECVMSG pseudo-method (or ReceiveNextQueuedMessage or GetNextQueuedMessage methods) to receive the message off the queue. See the <a href="#">Command property</a> description or <a href="#">Methods section</a> of this manual for more information.
&H2	<b>START_SINGLE_INSTANCE or START_DEST_APP</b>  When set, this flag instructs the Send method to load the destination application <i>if it is not in memory</i> . Once loaded, the message that caused the load process to occur is delivered to the applications socket. <b><u>The target socket must initially have the same name as the .EXE file being loaded.</u></b> This is how App-Link determines the application to load. (The application must also be located somewhere along the PATH in order to auto-load it). If there is a need to load multiple application instances, use the START_MULTI_INSTANCE flag instead.

#### **&H4 NO\_WAIT**

This flag is used when you want to send a message to another socket asynchronously (i.e. you don't want to wait around for the other socket to tell you that it got your message). Sending a message in this manner causes control to return to the caller almost immediately, without waiting for an acknowledgement from the receiving socket (a return from the `ReceiveMessage` event procedure).

It is important to point out that the sender will not receive an acknowledgement from a synchronous send **until the receiving sockets [ReceiveMessage event procedure](#) is exited**. At that time the sending socket may query the App-Link [Error property](#) to obtain the results of the send.

#### **&H10 START\_MULTI\_INSTANCE**

This flag will cause another instance of an App-Link enabled application to be loaded.

#### **Data Type**

Long



## Flags property example

### Example 1

This example sends a queued message to a socket named *INBASKET* on computer *MAIL*. The message will be placed in the destination socket's message queue and must be explicitly retrieved using any of the following techniques: *RCVMSG* pseudo-method, the *ReceiveNextQueuedMessage* method or the *GetNextQueuedMessage* method.

```
Sub SendButton_Click ()
    skt.Msg = "This message will be placed in the queue"
    skt.MsgLen = Len(skt.Msg)
    skt.Dest = "\\MAIL\INBASKET"
    skt.Flags = QUEUE_MSG
    skt.Command = SENDMSG
End Sub
```

### Example 2

This example sends a message to a socket named *FIRE* on computer *ALARM*. The message will be sent such that the application named *FIRE* (remember, by convention App-Link looks for an .EXE name with the same name as the destination socket) will be loaded into memory automatically if it is not already loaded.

```
Sub SendButton_Click ()
    skt.Msg = "0010 FIRE ALARM 33"
    skt.MsgLen = Len(skt.Msg)
    skt.Dest = "\\ALARM\FIRE"
    skt.Flags = START_SINGLE_INSTANCE
    skt.Command = SENDMSG
End Sub
```

Note that the socket is located on a remote computer named *ALARM*. It is important to point out that in order for the application to be automatically loaded on the remote computer, the App-Link [messaging server](#) must be loaded on the receiving machine at the time of the send. App-Link will automatically load the communications server on the sending end.

### Example 3

This example sends a message to a socket named *PING* on computer *MONITOR*. The message will be sent asynchronously, meaning that the sender will not receive a formal acknowledgement from the receiving socket. Also, the *START\_SINGLE\_INSTANCE* flag is set to load the target application if it is not already in memory.

```
Sub SendButton_Click ()
    skt.Msg = "NODE 0010 ONLINE"
    skt.MsgLen = Len(skt.Msg)
    skt.Dest = "\\MONITOR\PING"
    skt.Flags = NO_WAIT OR START_SINGLE_INSTANCE
    skt.Command = SENDMSG
End Sub
```

#### Example 4

This example sends a message to a socket named *TICKER* on computer *CENTRAL*. The message will be sent synchronously, meaning that the sender *will* receive a formal acknowledgement from the receiving socket (this is the default value of the flags setting, as opposed to specifying the *NO\_WAIT* flag). The *START\_MULTI\_INSTANCE* flag is also set which causes another instance of the application to load if necessary.

```
Sub SendButton_Click ()  
    skt.Msg = "AALA 50.60 51.50"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "\\CENTRAL\\TICKER"  
    skt.Flags = START_MULTI_INSTANCE  
    skt.Command = SENDMSG  
End Sub
```



## Msg Property

### Description

This property contains the actual message data to send to the destination socket. This property is not available during design time.

*This property may also be used by the ActiveX control for upward compatibility, however we encourage the use of the App-Link ActiveX methods as a more compact and readable alternative to setting properties. This property is now included as a parameter to the Send method.*

---

### Usage

```
[form.]Socket.Msg[ = messagetext$ ]
```

### Remarks

The content of the message is user-defined, and may be up to 64,000 bytes long for non-broadcast messages. Broadcast messages are limited to 256 bytes in length. App-Link messages may contain embedded nulls.

User defined types are supported through the use of two conversion APIs that are exported from the control module: *AplkVBTypeToString* and *AplkStringToVBType*. [\*AplkVBTypeToString\*](#) converts a user type to a string and must be called prior to assigning a user type to the *Msg* property. [\*AplkStringToVBType\*](#) takes a string (user type converted) and converts it back to the original user type. See the Functions section for more details.

### Data Type

String <64,000 bytes for non-broadcasts, 256 bytes for broadcasts>



## MsgLen Property

### Example

### Description

The MsgLen property is used to specify the length of the message to be sent. Because App-Link supports the use of embedded nulls in a message, it is not possible to accurately determine the true length of a message string, thus the need for the MsgLen property. This property is not available during design time.

### Usage

```
[form.]Socket.MsgLen[ = length& ]
```

### Remarks

The value is specified in *number of bytes* and must fall between 1 and 64,000 inclusive for non-broadcast messages, and 1 and 256 for broadcast messages.

### Data Type

Long



## MsgLen property example

This example sends a message to a socket named *XLATE*. The message length is determined by the Visual Basic Len function.

```
Sub SendButton_Click ()  
    skt.Msg = "Mes forms parlez avec des App-Link."  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "XLATE"  
    skt.Command = SENDMSG  
End Sub
```



## Priority Property

### Example

### Description

This property assigns a priority number to messages sent through the `subject` socket. It is useful for controlling the order of message processing when handling queued messages.

### Usage

`[form.]Socket.Priority[ = number% ]`

### Remarks

The message priority number determines the placement of messages in the destination socket's message queue when the *QueueAccess* property is set to `PRIORITY-BASED`.

The lower the number, the higher the priority. Therefore priority 100 messages will appear before priority 200 messages in the socket queue. A message with a priority equal to one or more existing messages in the queue is placed after the last message containing that same priority.

This property is ignored if the *QueueAccess* property is set to `LIFO` (Last-In-First-Out) or `FIFO` (First-In-First-Out). Refer to the [QueueAccess property](#) description for more information.

Priority numbers may range from 100 - 32767 inclusive.

### Data Type

Integer





## Priority property example

This example sends a queued, priority *100* message to the socket named *INBASKET* on computer *MAIL*. The message will be placed in the queue before messages with a priority greater than 100 and after any other priority 100 messages that may exist.

```
Sub SendButton_Click ()  
    skt.Msg = "This is a queued, priority 100 message"  
    skt.MsgLen = Len( skt.Msg )  
    skt.Dest = "\\MAIL\INBASKET"  
    skt.Priority = 100  
    skt.Flags = QUEUE_MSG  
    skt.Command = SENDMSG  
End Sub
```



## Protocol Property

### Description

This property establishes the communications server to use for delivering messages to sockets located on remote computers. The App-Link RADX control set ships with a communications server for the NetBIOS protocol. Other plug-in servers are under development for LAN transports such as TCP/IP and IPX/SPX. Contact Synergy Software Technologies Inc. for more information regarding the licensing of other transports.

### Usage

[*form.*]Socket.Protocol[ = *setting%* ]

### Remarks

The Protocol property settings for the base product are:

<u>Value</u>	<u>Symbol</u>	<u>Description</u>
0%	NETBIOS	NetBIOS protocol stack (default)

In addition, other settings will be allowed if the appropriate optional server has been installed. The value for each optional server is provided with the server license package.

### Data Type

Integer



## QueueAccess Property

### Description

The QueueAccess property allows the programmer to specify the ordering of queued messages in the socket's queue. This property may only be set at design time and is read-only at run time.

### Remarks

The QueueAccess property settings are:

0	PRIORITY-BASED	Based upon message priority
1	FIFO	First In, First Out
2	LIFO	Last In, First Out

When the QUEUE\_MSG option is used during an App-Link send, the subject message is queued at the receiving end. This property may be used to indicate how messages are to be processed from the queue.

### Data Type

Integer



## QueueCount Property

### Example

### **Description**

The QueueCount property allows the application program to query the number of items in the socket's queue at any given time. This property is read-only at runtime and not available at design time.

### **Usage**

i& = [*form.*]Socket.QueueCount

### **Remarks**

When the QUEUE\_MSG option is set during an App-Link send, the subject message is queued at the receiving end. This property may be used to determine how many messages are in the queue at any given time.

### **Data Type**

Long



## QueueCount property example

This example displays the number of messages pending in skt1s queue.

```
Sub Query_Queue ()  
    Dim nMessages As Long  
    nMessages = skt1.QueueCount  
    If nMessages = 0 Then  
        txtMsg.Text = "No messages are waiting in the queue"  
    Else  
        txtMsg.Text = "There are " & LTrim$(Str$(nMessages)) & "messages"  
    End If  
End Sub
```



## QueueStorage Property

### Description

The QueueStorage property defines the internal storage representation of messages that are queued at a destination socket. This property is only available at design time.

### Remarks

The QueueStorage property settings are:

<u>Value</u>	<u>Description</u>
0%	Memory-based storage (default)

This property only applies to messages that are sent using the [QUEUE\\_MSG option](#) .

### Data Type

Integer



## SocketName Property

### Example

### Description

The SocketName property identifies the App-Link communications socket to the system. It is in effect a local address for the socket.

### Usage

```
[form.]Socket.SocketName[ = socketname$ ]
```

### Remarks

Messages are routed to a socket using the socket's name. Names must be unique within a given computer. An error occurs during the assignment of this property if the name is already used by another socket on the same computer.

A socket's name may be dynamically changed at run-time. If the socket name is changed during run-time, all queued messages for the old socket name are discarded. **Be sure to process all queued messages for a socket prior to changing its name during run-time.**

#### *Automatic SocketName Generation*

If the *SocketName* property is set to a Null, Blank or "\_#", App-Link will automatically generate a unique socket name for the subject socket. The application program may query the name generated by examining the *SocketName* property following the assignment.

App-Link reserved names begin with "\_#" followed by a sequence number. You should not use this convention when naming your sockets.

### Data Type

String <40 bytes>



## SocketName property example

This example names four sockets on a form during the form's `Form_Load` event. Socket *skt4* will be assigned an App-Link generated name because its *SocketName* property is empty.

```
Sub Form_Load ()  
    skt1.SocketName = "InBasket"  
    skt2.SocketName = "OutBasket"  
    skt3.SocketName = "CircularFile"  
    Ask App-Link to generate a name for me  
    skt4.SocketName = ""  
    Display the generated name. Use the Windows MessageBox API because VB's MsgBox  
    implementation will eat App-Link messages while displayed  
    x = MessageBox(0,Generated name:  & skt4.SocketName,Information,0)  
End Sub
```





## Timeout Property

### Example

### Description

The *Timeout* property specifies the length of time the *Send* method will wait for an acknowledgement from the system on a synchronous transmission.

### Usage

```
[form.]Socket.Timeout[ = number& ]
```

### Remarks

Timeout values are expressed in milliseconds. A value of -1 instructs the Send method to wait **indefinitely** for the acknowledgement. This value is defined in Applink.Bas as INFINITE\_WAIT.

The *Timeout* property is ignored for messages delivered asynchronously, that is, if the NO\_WAIT flag has been set in the [Flags property](#).

Note that other processing continues while waiting for a synchronous request to complete. Control returns to the statement following the send after the recipients ReceiveMessage event is exited or a timeout occurs.

### Data Type

Long



## Timeout property example

This routine will send a synchronous message and wait 10 seconds for an acknowledgement from the recipient prior to timing out.

```
Sub SendButton_Click ()  
    skt.Msg = "I'm only waiting 10 seconds for you to reply!"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "\\MAIL\INBASKET"  
    skt.Timeout = 10000&  
    On Error Resume Next  
    skt.Command = SENDMSG  
    If skt.Error <> 0 Then  
        Process timeout or other error condition  
    End If  
End Sub
```

## Methods

Unlike the original Microsoft VBX control specification, the new ActiveX/OCX technology supports the use of methods. App-Link RADX 2.0 now includes method counterparts for the pseudo-methods previously supported via the *Command* property. Although the *Command* property is still available for you to use, we encourage you to use the method implementations instead. They are more intuitive and easier to use than setting properties.

Of course, these methods are not available in the VBX version of App-Link.

The following methods are supported in the ActiveX version:

[Broadcast](#)

[GetComputerName](#)

[GetNextQueuedMessage](#)

[ReceiveNextQueuedMessage](#)

[Send](#)

## Broadcast Method

### Example

#### Description

App-Link broadcast communications provides the programmer with the capability of sending a message to a given socket on every computer that has an App-Link communications server loaded. The destination application containing the receiving socket must also be running for the message to be delivered. Auto-application loading is not supported for broadcast sends.

Broadcast support is a connectionless service. That is to say, there is no guarantee that messages will be delivered to the destination socket, or that they will arrive in the same sequence in which they were sent. This is a network restriction, *not* an App-Link restriction. Regardless of whether a message arrives safely, no acknowledgement of reception is provided. Broadcast messages are always sent asynchronously using NO\_WAIT mode. If you require assurance that a message is delivered to its destination, then you should not use broadcast support. You must explicitly specify the destination computer name and send the message synchronously.

Broadcast messages are typically very small. App-Link limits the maximum size of a message sent using broadcast communications to 256 bytes. An error will occur if you attempt to broadcast a message larger 256 bytes.

#### Usage

```
RetCode = skt.Broadcast(Dest, Msg [,MsgLen])
```

#### Return Value

*RetCode* Long Contains the return code from the broadcast.

#### Parameters

<i>Dest</i>	String	The name of the destination socket. Note that only the socket name is required. Refer to the description of the <a href="#">Dest property</a> for more information regarding destination formats.
<i>Msg</i>	String	The text of the message to send. See the <a href="#">Msg property</a> for more information.
<i>MsgLen</i>	Long	An optional parameter that specifies the length of the message. If omitted, App-Link determines the length of the text by looking for the first null character in the string. If your message contains binary data, specify its length using this parameter.

#### Remarks

Many system designers want to use broadcast from the server side to implement some sort of automated user sign-in or registration. We recommend originating user sign in on the client side by having the client send the server a message that, in effect, says Im here, and here is my name. The server can then maintain a connection table to communicate with its clients. Using this method, synchronous transactions may be used, allowing the client to determine if the server is available. Also, once a connection has been established the server can monitor the connection by sending small, periodic

messages to clients to test the connection. While this is not necessary, the amount of regulation and monitoring of connections depends upon the needs of the individual application.



## Broadcast method example

### Example 1

This example broadcasts a message to a socket called REGISTER on all remote computers.

```
Sub GetRegistration_Click ()
    Dim RetCode as Long
    RetCode = skt.Broadcast(REGISTER, "Sign in please!")
    If RetCode <> 0 Then
        Process error condition
        x = MessageBox(0, skt.ErrorText, Broadcast Error, 0)
    End If
End Sub
```

In comparison, the following code performs the same operation using the VBX properties.

```
Sub GetRegistration_Click ()
    skt.Msg = "Sign in please!"
    skt.MsgLen = Len(skt.Msg)
    skt.Dest = "\\*\REGISTER"
    On Error Resume Next
    skt.Command = SENDMSG
    If skt.Error <> 0 Then
        Process error condition
        x = MessageBox(0, skt.ErrorText, Broadcast Error, 0)
    End If
End Sub
```

### Example 2

This example broadcasts a message to a socket called PLUGIN on all remote computers. Assume that the string parameter *BinaryData* is a user defined type that has been converted to a string using the App-Link `AplkVBTypeToString` function. A message length is explicitly specified because the message may contain embedded nulls.

```
Sub SendPlugInData (BinaryData as String)
    Dim RetCode as Long
    RetCode = skt.Broadcast(PlugIn, BinaryData, Len(BinaryData))
    If RetCode = 0 Then
        x = MessageBox(0, Broadcast successful!, Broadcast, 0)
    Else
        Process error condition
        x = MessageBox(0, skt.ErrorText, Broadcast Error, 0)
    End If
End Sub
```

OCX16	OCX32
-------	-------

## GetComputerName Method

### Example

### Description

This method allows you to programmatically determine the name of the local computer.

### Usage

*Computer* = *skt*.**GetComputerName**()

### Return Value

*Computer* String Contains the name of the local computer if the call is successful. An empty string is returned if the call fails. You can query the [Error property](#) on failures to determine the cause of failure.

### Parameters

None



## GetComputerName method example

This example sets a label control to the name of the local computer.

```
Sub DisplayComputerName()  
    Dim ComputerName As String  
    ComputerName = skt.GetComputerName()  
    If skt.Error <> 0 Then  
        Set label to local computer name  
        label = ComputerName  
    Else  
        Process error condition  
        label = <unknown>  
        x = MessageBox(0,skt.ErrorText,GetComputerName Error,0)  
    End If  
End Sub
```





## GetNextQueuedMessage Method

### Example

### Description

When a message is sent using the queuing option, it is stored in an internally managed App-Link message queue. This is one of two methods that may be used to extract a message from the queue. In this case, the message content and additional message information is returned in parameter placeholders in the call.

### Usage

*Msg* = *skt*.**GetNextQueuedMessage**(*Orig*, *MsgLen*, *Priority*, *Flags*)

### Return Value

<i>Msg</i>	String	Contains the message string if the call is successful. An empty string is returned if the call fails.
------------	--------	---

### Parameters

<i>Orig</i>	String	Originating computer and socket name. If the message was sent from a local socket, <i>Orig</i> contains the socket name only. Refer to the <a href="#">Dest property</a> for more information on the format of this string.
-------------	--------	---

<i>MsgLen</i>	Long	Length of the message data.
---------------	------	-----------------------------

<i>Priority</i>	Integer	Priority assigned to the message.
-----------------	---------	-----------------------------------

<i>Flags</i>	Long	Send flags in effect when the message was sent. See the <a href="#">Flags property</a> description for more details.
--------------	------	--

### Remarks

The GetNextQueuedMessage method will trigger a run-time error if the call is unsuccessful. Be sure to activate an On Error handler before you call this method. If an On Error handler is not present, and an error is generated by the GetNextQueuedMessage method, your application will end.



## GetNextQueuedMessage method example

### Example 1

This example retrieves all messages in the queue and places them in a listbox control.

```
Sub EmptyQueue_Click()  
    Declare method parameters  
    Dim m_Msg As String  
    Dim m_Orig As String  
    Dim m_MsgLen As Long  
    Dim m_Priority As Integer  
    Dim m_Flags As Long  
    Activate error handler  
    On Error Resume Next  
    Loop while messages are still left in queue  
    Do Until skt.QueueCount = 0  
        Get next message from queue  
        m_Msg = GetNextQueuedMessage(m_Orig,m_MsgLen,m_Priority,m_Flags)  
        If skt.Error = 0 Then  
            lst.AddItem ( & m_Orig & ) & m_Msg  
        End If  
    Loop  
End Sub
```

### Example 2

This example process priority 100 messages and discards all others.

```
Sub ProcessPriority100_Click()  
    Declare method parameters  
    Dim m_Msg As String  
    Dim m_Orig As String  
    Dim m_MsgLen As Long  
    Dim m_Priority As Integer  
    Dim m_Flags As Long  
    Activate error handler  
    On Error Resume Next  
    Loop while messages are still left in queue  
    Do Until skt.QueueCount = 0  
        Get next message from queue  
        m_Msg = GetNextQueuedMessage(m_Orig,m_MsgLen,m_Priority,m_Flags)  
        Only process priority 100 messages, discard all others  
        If skt.Error = 0 And m_Priority = 100 Then  
            lst.AddItem ( & m_Orig & ) & m_Msg  
        End If  
    Loop  
End Sub
```

### Example 3

This example sends a confirmation message back to the originator of a queued message. In this case,

the first 10 bytes of the received message contains a message identifier that is sent back to the originator on reply.

```
Sub ConfirmQueuedMessage_Click()
    Declare method parameters
    Dim m_Msg As String
    Dim m_Orig As String
    Dim m_MsgLen As Long
    Dim m_Priority As Integer
    Dim m_Flags As Long
    Loop while messages are still left in queue
    Do Until skt.QueueCount = 0
        Dim RetCode As Long
        Get next message from queue
        m_Msg = GetNextQueuedMessage(m_Orig,m_MsgLen,m_Priority,m_Flags)
        RetCode = skt.Send(m_Orig,ACK msgid= & Left$(m_Msg,10))
        If RetCode <> 0 Then
            Process error condition
            x = MessageBox(0,skt.ErrorText,Confirm Error,0)
        End If
    Loop
End Sub
```



## ReceiveNextQueuedMessage Method

### Example

### **Description**

When a message is sent using the queuing option, it is stored in an internally managed App-Link message queue. This is one of two methods that may be used to extract a message from the queue. In this case, the [ReceiveMessage event](#) is fired with the next message in the queue. For more information regarding how to process the receipt of a message, refer to the *ReceiveMessage* event description.

### **Usage**

```
RetCode = skt.ReceiveNextQueuedMessage()
```

### **Return Value**

*RetCode*   Long   Contains the return code from the method call.

### **Parameters**

None



## ReceiveNextQueuedMessage method example

### Example

This example retrieves all messages in the queue and fires the ReceiveMessage event handler.

```
Sub EmptyQueue_Click()  
    Dim RetCode As Long  
    Loop while messages are left in queue  
    Do Until skt.QueueCount = 0  
        Receive next message from queue  
        RetCode = ReceiveNextQueuedMessage()  
        If RetCode <> 0 Then  
            Process error condition  
            x = MessageBox(0,skt.ErrorText,Receive Error,0)  
        End If  
    Loop  
End Sub
```



## Send Method

### Example

### Description

This method is used to send a message to a destination socket.

### Usage

```
RetCode = skt.Send(Dest, Msg [,MsgLen][,Flags])
```

### Return Value

*RetCode*    Long    Contains the return code from the send operation upon completion.

### Parameters

<i>Dest</i>	String	The name of the destination socket. See the <a href="#">Dest property</a> description for more information.
<i>Msg</i>	String	The text of the message to send. See the <a href="#">Msg property</a> description for more information.
<i>MsgLen</i>	Long	An optional parameter that specifies the length of the message. If omitted, App-Link determines the length of the text by looking for the first null character in the string. If your message contains binary data, specify its length using this parameter.
<i>Flags</i>	Long	An optional parameter that controls the behavior of the send method. Refer to the <a href="#">Flags property</a> description for more detailed information.



## Send method example

### Example 1

This example sends the message Hello There to a socket named CATCHER on computer PC32.

```
Sub SendHello_Click()  
    Dim RetCode As Long  
    RetCode = skt.Send(\\PC32\CATCHER,Hello There)  
    If RetCode <> 0 Then  
        Process error condition  
        x = MessageBox(0,skt.ErrorText,Send Error,0)  
    End If  
End Sub
```

### Example 2

This example sends a queued message to a socket named LOGGER on computer DIAG001.

```
Sub SendDiagMessage_Click()  
    Dim RetCode As Long  
    Note the use of the double comma in the parameter list below. This  
    informs VB that we do not care to pass the optional MsgLen parameter.  
    RetCode = skt.Send(\\DIAG001\LOGGER,LOG Alarm #0001,,QUEUE_MSG)  
    If RetCode <> 0 Then  
        Process error condition  
        x = MessageBox(0,skt.ErrorText,Send Error,0)  
    End If  
End Sub
```

In comparison, the following code performs the same operation using the VBX properties.

```
Sub SendDiagMessage_Click()  
    skt.Msg = "LOG Alarm #0001"  
    skt.MsgLen = Len(skt.Msg)  
    skt.Dest = "\\DIAG001\LOGGER"  
    skt.Flags = QUEUE_MSG  
    On Error Resume Next  
    skt.Command = SENDMSG  
    If skt.Error <> 0 Then  
        Process error condition  
        x = MessageBox(0,skt.ErrorText,Send Error,0)  
    End If  
End Sub
```

## Events

Events are used by custom controls to inform the application that a particular action related to the control has taken place. The event is marked by the calling of a routine to handle the action.

App-Link supports the following events:

Error

NotifyMessage

ReceiveMessage





## Error Event

### Example

### Description

The Error event is fired as a result of an asynchronous App-Link error that takes place when no application code is being executed.

### Usage

Sub *skt\_Error*(*Number* As Integer, *Description* As String, *Scode* As Long, *Source* As String, *HelpFile* As String, *HelpContext* As Long, *CancelDisplay* As Boolean)

### Parameters

<i>Number</i>	Integer	The App-Link error number.
<i>Description</i>	String	Contains the textual description of the error.
<i>Scode</i>	Long	OLE status code.
<i>Source</i>	String	The name of the object or application that generated the error.
<i>HelpFile</i>	String	Fully-qualified path to the help file.
<i>HelpContext</i>	Long	Help context ID.
<i>CancelDisplay</i>	Boolean	True - application handles error, False - App-Link displays the error message in a message box.

### Remarks

If you don't code an event procedure for the *Error* event, App-Link displays the message associated with the error.



## Error event example

### Example

This example provides application-specific processing of asynchronous error events. The error message is displayed in a Windows message box.

```
Sub skt_Error(Number As Integer, Description As String, Scode As Long, Source As String, HelpFile As String,  
    HelpContext As Long, CancelDisplay As Boolean)
```

```
    Build error message string
```

```
    Dim Prompt As String
```

```
    Prompt = _  
        The following socket error has occurred: & vbLf & vbLf & _  
        Description & vbLf & vbLf & _  
        Return Code = & Ltrim$(Str$(Number))
```

```
    x = MessageBox(0,Prompt,Me.Caption,0)
```

```
    Tell App-Link that we handled the error event
```

```
    CancelDisplay = True
```

```
End Sub
```



## NotifyMessage Event

### [Example](#)

### **Description**

This event is fired when a message is placed in a sockets internal message queue.

### **Usage**

Sub skt\_NotifyMessage()

### **Parameters**

None

### **Remarks**

You can use this event to trigger message handling the moment a message is queued. Previously, an application had to use a timer or DoEvents loop to poll the sockets [QueueCount property](#) in order to know when a message arrived.



## NotifyMessage event example

### Example 1

This example refreshes a label control with the number of pending messages in the queue.

```
Sub skt_NotifyMessage()  
    label = LTrim$(Str$(skt.QueueCount))  
End Sub
```

### Example 2

This example fires the *ReceiveMessage* event the moment a message is queued at the socket. This is the equivalent of sending a non-queued message and having App-Link fire the *ReceiveMessage* event for you.

```
Sub skt_NotifyMessage()  
    Dim RetCode As Long  
    RetCode = skt.ReceiveNextQueuedMessage()  
    If RetCode <> 0 Then  
        Process error condition  
        x = MsgBox(0,skt.ErrorText,NotifyMessage Error,0)  
    End If  
End Sub
```



## ReceiveMessage Event

### Example

### Description

This event is fired when a message is received from another socket. There is one ReceiveMessage event handler for each socket placed on a form.

### Usage

```
Sub skt_ReceiveMessage(Orig As String, MsgLen As Long, Msg As String, RetCode As Long)
```

### Parameters

<i>Orig</i>	String	The originator, or name of the socket that sent the message. The originator name will be fully-qualified (computer name plus socket name) if the message came from a remote socket.
<i>MsgLen</i>	Long	The length of the message in bytes.
<i>Msg</i>	String	The actual message content.
<i>RetCode</i>	Long	Event return code. The value assigned to this variable is returned to the originating socket ( <i>Orig</i> ) if the message was sent synchronously.

### Remarks

The event occurs when a message is received from a socket, either from a message sent directly to the socket, or receiving a queued message in code by setting the value of the *Command* property (pseudo-method) to *RECVMSG* or by using the *ReceiveNextQueuedMessage* method.



## ReceiveMessage event example

### Example

The following example presents a scenario for processing incoming messages as command strings. The first portion of the message contains the command to execute followed by a semicolon. The semicolon delimits the command name from the rest of the message. The remaining portion of the message provides the parameters required by the command.

Once command execution is complete (i.e. the ReceiveMessage subroutine exits), the originating socket receives the return code placed in the *RetCode* variable which indicates the results of execution.

```
Sub skt_ReceiveMessage(Orig As String, MsgLen As Long, Msg As String, RetCode As Long)
```

```
    Const UNKNOWN_CMD = 99
```

```
    'Locate the command delimiter within the message
```

```
    DelimPos% = InStr(1,Msg,":")
```

```
    If DelimPos% > 0 Then
```

```
        Command$ = Left$(Msg,DelimPos%-1)
```

```
    Else
```

```
        Command$ = ""
```

```
    End If
```

```
    Select a processing routine based on the command name
```

```
    Select Case Command$
```

```
        Case Command1
```

```
            RetCode = ProcessCommand1(Msg,MsgLen)
```

```
        Case CommandX
```

```
            RetCode = ProcessCommandX(Msg,MsgLen)
```

```
        Case Else
```

```
            RetCode = UNKNOWN_CMD
```

```
    End Select
```

```
End Sub
```

## Functions

This section defines the functions provided with the App-Link package.

Often times it is desirable to use user-defined types within messages. This makes organizing data prior to and following data transfer easy to manage. Since the App-Link control is designed to send string data, a user-defined type must be converted to a string prior to sending it with App-Link. On the other end, the string must be converted back to its original user-defined type. Two functions are provided with the App-Link package that allow you to easily perform these conversions.

[AplkVBTypeToString](#)

[AplkStringToVBType](#)



## AplkVBTypeToString Function

### Example

#### Description

Converts a Visual Basic user type variable to a string.

#### Usage

*MsgText* = **AplkVBTypeToString**(*Source* As Any, ByVal *cb* As Long)

#### Return Value

*MsgText*    String    Contains the converted user type as a string.

#### Parameters

<i>Source</i>	Any	A pointer to the Visual Basic user type variable.
<i>cb</i>	Long	Length of the user type. The length is typically obtained via a call to the Len function.

#### Remarks

Creates a string of length *cb* bytes and initializes the string with the contents of the memory location pointed to by *Source*.

User-defined type elements declared as strings must have a fixed length. You cannot use variable-length strings or variants within a user type that you intend to convert to a string with this function.

---





## AplkVBTypeToString function example

### Example

This example demonstrates how user type variables may be sent as messages. A message containing the user type *MailPacket* is sent to the destination socket *RdrList* on computer MAIL. To see how the message is converted back to the user type, see the example under the description of the *AplkStringToVBType* function.

First, define the user type:

'Define the layout of my user type

```
Type MailPacket
  From As String * 30
  DeptNum As Long
  Subject As String * 40
  Class As Long
  Text As String * 1024
End Type
```

And then use the *AplkVBTypeToString* function to convert the type to a string prior to sending it to the destination socket. The example below sends the message when a button called SendButton is clicked.

```
Sub SendButton_Click()

  Dim MailMsg As MailPacket
  Dim MyMsg as String
  Dim RetCode as Long

  Initialize mail message structure

  MailMsg.From = "Fred Flintstone"
  MailMsg.DeptNum = 500
  MailMsg.Subject = "Vacation"
  MailMsg.Class = 3
  MailMsg.Text = "I'll be on vacation from 10/20-10/22."

  Notice the call to AplkVBTypeToString below. Here the user
  type is converted to a string prior to using the send method.

  MyMsg = AplkVBTypeToString(MailMsg,Len(MailMsg))

  Because this string may contain embedded nulls, we explicitly
  provide the length of the message to the send method below.

  RetCode = skt.Send("\\MAIL\RdrList",MyMsg,Len(MyMsg),DEFAULT_FLAGS)

  If RetCode <> 0 Then
    x = MessageBox(0,skt.ErrorText,Send Error,0)
  End If
```

End Sub



## AplkStringToVBType Function

### Example

#### Description

Converts a string created with the *AplkVBTypeToString* function to a Visual Basic user type.

#### Usage

**AplkStringToVBType**(*Dest* As Any, ByVal *Source* As String, ByVal *cb* As Long)

#### Return Value

None

#### Parameters

<i>Dest</i>	Any	A pointer to the Visual Basic user type variable that serves as the destination of the conversion process.
<i>Source</i>	String	String data to convert.
<i>cb</i>	Long	Length of the string data.

#### Remarks

Copies the contents of the memory location pointed to by *Source* to the memory location pointed to by *Dest* for a length of *cb* bytes.



## AplkStringToVBType function example

### Example

This example shows how messages containing user types might be processed in the ReceiveMessage event procedure.

```
Sub skt_ReceiveMessage(Orig As String, MsgLen As Long, Msg As String, RetCode As Long)

    Dim MailMsg As MailPacket

    'Copy the contents of Msg to the user type variable MailMsg

    AplkStringToVBType MailMsg, Msg, MsgLen

    'The user type MailMsg now contains a copy of Msg data!
    Now getting the values out of the message is a snap

    Debug.Print "From: " & MailMsg.From
    Debug.Print "Dept: " & Trim$(Str$(MailMsg.DeptNum))
    Debug.Print "Subject: " & MailMsg.Subject
    Debug.Print "Class: " & Trim$(Str$(MailMsg.Class))
    Debug.Print "Text: " & MailMsg.Text

    Indicate that we processed the message successfully

    RetCode = 0

End Sub
```

## Trappable Errors

Listed below are the error codes and associated messages that can be returned from the App-Link facility. If you encounter an error and are unable to diagnose the problem, please refer to the *Troubleshooting* section of the manual prior to calling for technical support. Also, be sure to read the README.WRI file for added messages.

Error Code	Error Description
20000	<a href="#"><u>System error</u></a>
20001	<a href="#"><u>Unable to allocate local memory</u></a>
20002	<a href="#"><u>Unable to load application</u></a>
20007	<a href="#"><u>Unable to allocate shared memory</u></a>
20008	<a href="#"><u>Unable to map shared memory</u></a>
20009	<a href="#"><u>Shared memory access failed</u></a>
20500	<a href="#"><u>Unable to query computer name</u></a>
20501	<a href="#"><u>Request aborted</u></a>
20502	<a href="#"><u>Transaction failure</u></a>
20503	<a href="#"><u>Unable to locate task instance</u></a>
20504	<a href="#"><u>Duplicate socket name</u></a>
20505	<a href="#"><u>Unable to locate socket</u></a>
20506	<a href="#"><u>Unable to locate conversation</u></a>
20507	<a href="#"><u>Message queue is empty</u></a>
20508	<a href="#"><u>Unable to query task information</u></a>
20509	<a href="#"><u>Unable to register object class</u></a>
20510	<a href="#"><u>Timeout occurred</u></a>
20511	<a href="#"><u>Unable to establish connection</u></a>
20512	<a href="#"><u>Invalid message priority value</u></a>
20513	<a href="#"><u>Invalid timeout value</u></a>
20514	<a href="#"><u>Invalid message length</u></a>
20515	<a href="#"><u>Null message pointer</u></a>
20516	<a href="#"><u>Invalid local session number</u></a>
20517	<a href="#"><u>No remote resources available</u></a>
20518	<a href="#"><u>Network session was closed</u></a>
20519	<a href="#"><u>Network command canceled</u></a>
20523	<a href="#"><u>No answer from partner</u></a>
20524	<a href="#"><u>Unable to find network name</u></a>
20525	<a href="#"><u>Too many network commands</u></a>
20526	<a href="#"><u>Network adapter malfunction</u></a>
20528	<a href="#"><u>Synchronous transaction pending</u></a>
20529	<a href="#"><u>PostMessage failed</u></a>
20530	<a href="#"><u>Unable to load communications server</u></a>
20531	<a href="#"><u>Unable to register notification handler</u></a>
20532	<a href="#"><u>Unable to create object window</u></a>
20533	<a href="#"><u>DdeCreateStringHandle failed</u></a>
20534	<a href="#"><u>Unable to register socket service name</u></a>
20535	<a href="#"><u>Connection was dropped</u></a>
20536	<a href="#"><u>Destination is missing</u></a>
20537	<a href="#"><u>Destination contains blanks</u></a>
20538	<a href="#"><u>Destination missing separators</u></a>
20539	<a href="#"><u>Invalid computer name length</u></a>
20540	<a href="#"><u>Socket name is missing</u></a>
20541	<a href="#"><u>Invalid socket name length</u></a>
20542	<a href="#"><u>Invalid separator in socket name</u></a>

20543	<u>Socket name contains blanks</u>
20544	<u>Application terminated</u>
20545	<u>Unable to load daemon task</u>
20546	<u>Server connection table is full</u>
20547	<u>Socket is disabled</u>
20548	<u>Null parameter pointer</u>
20549	<u>String resource was not found</u>
20550	<u>Input buffer is too small</u>
20551	<u>Unable to query string</u>
20552	<u>Socket property is read-only</u>
20553	<u>Invalid protocol option</u>
20554	<u>Invalid queue storage option</u>
20555	<u>Invalid queue access option</u>
20556	<u>Socket event handler was not specified</u>
20557	<u>Invalid socket event handler option</u>
20558	<u>Type mismatch</u>
20559	<u>Unable to load Netapi library</u>
20560	<u>NetWkstaGetInfo failed</u>
20561	<u>WNetGetUser failed</u>
20562	<u>Unable to query module name</u>
20563	<u>Unable to initialize 32-bit message transport</u>
20564	<u>Unable to verify service user</u>
30000	<u>Unknown method specified</u>

## System error

An internal error has occurred in the system.

Unable to allocate local memory

A local memory allocation failed.



## Unable to load application

The operating system was unable to load the destination application. Verify that the application program exists and is located in a directory specified in the PATH statement.

Unable to allocate shared memory

A shared memory allocation failed.

Unable to map shared memory

Could not map a view of shared memory to the process address space.

## Shared memory access failed

An attempt to open a shared memory object for read/write access failed.

Unable to query computer name

Could not obtain the computer name from the operating system.

## Request aborted

A request was prematurely terminated before a reply was received from the destination socket.

## Transaction failure

An unknown error occurred within the system. The request was terminated.

## Unable to locate task instance

An application attempted to use an App-Link service prior to the initialization of App-Link run time support or an invalid task instance identifier was passed to a run time function.



## Duplicate socket name

An attempt to name a socket failed. The name is already used by another socket on the computer.

Unable to locate socket

The socket specified by the Dest property could not be found.

## Unable to locate conversation

A request was received for a DDE conversation that no longer exists. The request was discarded.

Message queue is empty

An attempt was made to receive a message from an empty message queue.

Unable to query task information

An internal call to the Windows TaskFindHandle function failed.

## Unable to register object class

An internal call to the Windows RegisterClass function failed while attempting to register the App-Link object window class.

Timeout occurred

A timeout occurred before an acknowledgement was received from the destination socket.

## Unable to establish connection

The destination socket is not responding to a DDE connection request. The application represented by the destination socket may not be loaded.



## Invalid message priority

The message priority number is out of range. The value must be in the range 100 to 32767 inclusive.

## Invalid timeout value

The timeout value is out of range. The value must be in the range -1 to 2147483647 inclusive.

## Invalid message length

The message length value is out of range. The value must be in the range 1 to 64000 inclusive.

## Null message pointer

The Msg property was specified as a Null value or not assigned.

## Invalid local session number

A network command was issued for a network session that no longer exists.

## No remote resources available

A network session could not be established with a remote server because there is insufficient room in its session table.

Network session was closed

A network session was closed prematurely by the session partner.

## Network command canceled

A network command was canceled before the command could complete.



No answer from partner

No response was received from a session connection request or the remote network server is not available at this time.

## Unable to find network name

The name of a remote network server could not be found in the network name table. The network server on the remote computer may not be loaded.

## Too many network commands

The number of network commands currently pending equals the maximum number configured for the network adapter. The command cannot be completed at this time.

## Network adapter malfunction

The network adapter has experienced an internal error. The adapter may need to be reset.

## Synchronous transaction pending

An attempt to send a synchronous message failed because another synchronous message is already pending. Only one synchronous message can be sent at a time.

PostMessage failed

An internal call to the Windows PostMessage function has failed.

## Unable to load communications server

An error was encountered while trying to load an App-Link communications server. Check to make sure that the server executable exists on the computer and that it is located somewhere in the Windows search path. In addition, the server itself may have encountered an internal error while trying to initialize. In this case, check the server .ERR file located in the same directory as the one from which the server was started for more details on the source of the problem. An internal call to the Windows PostMessage function has failed.

## Unable to register notification handler

An error was encountered while trying to register a notification callback with Windows. The application cannot be initialized at this time.



## Unable to create object window

An internal call to the Windows CreateWindow function failed while attempting to create an App-Link object window. This error could be due to insufficient Windows resources.

DdeCreateStringHandle failed

An internal call to the Windows DdeCreateStringHandle function has failed.

Unable to register socket service name

An internal call to the Windows DdeNameService function has failed.

Connection was dropped

A connection was dropped prematurely. The transaction in progress was terminated.

Destination is missing

The Dest property is empty.

Destination contains blanks

The Dest property contains embedded blanks.

## Destination missing separators

The Dest property appears to contain both computer name and socket name components, however the specification is missing one or more separators.

## Invalid computer name length

The length of the computer name component specified in the Dest property is invalid. The length must be 1 - 14 bytes inclusive.



Socket name is missing

The SocketName property is empty or the socket name component of the Dest property is missing.

### Invalid socket name length

The length of either the SocketName property or the socket name component of the Dest property is invalid. The length must be 1 - 40 bytes inclusive.

## Invalid separator in socket name

The SocketName property or the socket name component of the Dest property contains invalid separator characters.

## Socket name contains blanks

The SocketName property or the socket name component of the Dest property contains embedded blanks.

## Application terminated

A synchronous request was pending when the application decided to terminate. The request was aborted.

## Unable to load daemon task

An error was encountered while trying to load the App-Link daemon program APLKDAEM.EXE. Verify that the executable exists on the computer and that it is located in the Windows search path. Another problem may be a lack of conventional memory. If this is the case, unload any other applications that are running and try to start the application again.

## Server connection table is full

A request to establish a connection with a remote computer failed because the communication servers connection table is full. Increase the value for the Connections setting in the App-Link INI file (or on the servers command line) and restart the server.

Socket is disabled

A request was rejected by the destination socket because it is in a disabled state.



## Null parameter pointer

An internal call to an App-Link API failed attempting to process a null parameter.

## String resource was not found

The textual description for an App-Link error code could not be found. Verify that the error for which error text is requested is a valid App-Link error code.

Input buffer is too small

An internal App-Link error was encountered. The buffer used to obtain the value of a socket property is too small.

Unable to query string

Unable to retrieve the text string associated with a global string handle.

Socket property is read-only

An attempt was made to assign a value to a read-only socket property. The request was rejected.

## Invalid protocol option

The protocol property is out of range. At this time, the only supported protocol is NetBIOS which has a property value of 0.

## Invalid queue storage option

The queue storage property is out of range. At this time, the only supported storage option is memory-based which has a property value of 0.

### Invalid queue access option

The queue access property is out of range. The valid values are 0 - priority based, 1 - FIFO or 2 - LIFO.



Socket event handler was not specified

An internal call to the App-Link open socket function failed because an event handler was not specified or is invalid.

## Invalid socket event handler option

The event handler type setting is out of range. The valid values are 1 - callback or 2 - window.

## Type mismatch

One or more argument types do not match the expected types of an App-Link method.

## Unable to load Netapi library

Unable to load the Windows NetApi dynamic link library. Verify that the file NETAPI.DLL exists on the computer and that it is located somewhere along the search path.

NetWkstaGetInfo failed

An internal call to the Windows NetWkstaGetInfo function has failed.

WNetGetUser failed

An internal call to the Windows WNetGetUser function has failed.

Unable to query module name

Could not obtain the module name for the calling process.

## Unable to initialize 32 bit message transport

The thunk layer was unable to initialize the 32-bit App-Link message transport library. Verify that the file ALAPI32.DLL exists on the computer and that it is located somewhere along the search path.



Unable to verify service user

The thunk layer failed to verify the calling task as a user of App-Link services.

## Unknown method specified

The Command property was assigned an invalid value. The valid values are 1 - *Send* or 2 - *Receive Next Queued Message*.

## Using the APPLINK.INI File

App-Link uses a private initialization file to hold system configuration information. The file is named APPLINK.INI and it is copied to the Windows subdirectory during installation.

At present, the initialization file contains one section that describes setup information for the NetBIOS server. Note that the options listed here may also be specified as command line parameters to the NetBIOS server. See the section on App-Link Communications Servers for more information.

The section in the INI file is named **[NetBIOS Server]** and contains the following entries:

### **AutoUnload = 0 | 1**

Defines when the NetBIOS server is shutdown. If this entry is 1, the server is automatically unloaded after the last socket within a computer closes. A value of 0 instructs the server to remain loaded after the last socket closes. In such a case, the server must be manually closed via the Task Manager, or by shutting down Windows. The default value is 1.

### **LanaNum = *number***

Identifies the network protocol to network adapter binding that NetBIOS should use to process network commands. The default is 0.

### **MachineName = *name***

Specifies the computer's name. Messages are sent to a destination computer using this name. If a name is not provided or this option is removed from the INI file, the server will attempt to ascertain the name from the operating system. This is the default behavior.

If you are using the NetBIOS emulator under NOVELL, then you should use this parameter to name your computer. Alternatively, you could also use the equivalent command line parameter when invoking the NetBIOS communications server.

### **Appearance = 0 | 1 | 2**

Specifies how the NetBIOS server icon is represented on the Windows desktop. Valid values are:

- 0 = Display icon, include entry in Task Manager
- 1 = Do not display icon, include entry in Task Manager
- 2 = Do not display icon, do not include entry in Task Manager

The default value is 0.

### **MessagePopup = 0 | 1**

Specifies whether the NetBIOS server is to display a message box if an error is encountered during initialization. If this entry is 1, the server will display a system-modal message box containing a description of the error. A value of 0 disables the message box. In either case, the error will also be posted to the server's log file APLKNB.ERR located in the same directory that the server was launched from. The default value is 1.

### **Connections = *number***

Specifies the maximum number of concurrent connections maintained by the server. If the maximum

number of connections is reached, the server will begin to break least-used, idle connections in order to make room for new ones. The default is 10.

**PerConnectReceiveBuffers = *number***

Specifies the number of network control blocks (NCBs) available per connection for message reception. Typically, the default setting of 2 is adequate for most network needs. If you experience frequent timeouts you can try increasing this value; however, network control blocks are a limited global resource. Most NetBIOS implementations provide for an absolute maximum of 255 NCBs. A full connection table will use (Connections\*PerConnectReceiveBuffers) NCBs.

**ReceiveBuffers = *number***

Specifies the total number of pending receive network control blocks that are available to service requests from other computers. Usually the default setting of 5 is adequate for most situations. If your application is of a server-type, that receives many requests at once, then you may need to increase this value. Unlike PerConnectReceiveBuffers, this option defines a pool of receive NCBs to be used by any connection. In fact, PerConnectReceiveBuffers and ReceiveBuffers are mutually-exclusive options. If both options are specified, PerConnectReceiveBuffers will be used.

**Listens = *number***

Specifies the number of network control blocks that are available to service requests for new connections. The default setting of 2 is usually sufficient for most needs. If you experience frequent 'No answer from partner' errors, and you are sure that network connections are working between the computers, the errors could be related to insufficient pending listen NCBs. In this case, try increasing the Listens value; however keep in mind that increasing the value will require more of your system's resources.

**BroadcastReceiveBuffers = *number***

Specifies the number of pending receive network control blocks that are available to service broadcast requests. Usually, the default setting of 2 is adequate for most situations. If your application is broadcast-intensive, you may need to increase this value if you find that messages are being lost frequently. However, keep in mind that broadcast communication is a best-effort delivery scheme, that is, not guaranteed.

**SendBuffers = *number***

Specifies the total number of network control blocks that are available for sending messages. Outgoing messages are queued internally until an NCB from this pool becomes available. The default setting of 2 is usually sufficient for most application needs.

**IdleTimeSweepFrequency = *number***

Defines the frequency (in seconds) at which the server sweeps the connection table to 'age' idle connections. Each connection has an associated idle time that specifies the amount of time since the last message sent/received over the connection. The server uses the idle time value to determine which connections to break when either the connection table becomes full or the disconnect idle time is exceeded (as described below). In the case of a full connection table, the server will break the least-used idle connection, that is, the connection with the most idle time. (You should not need to change this entry.)

**DisconnectIdleTime = *number***

Specifies the number of minutes a connection can remain idle before the connection is broken by the

server and removed from the connection table. A value of 0 will disable the server's disconnect-idle-time processing, and prevent connections from being broken in this manner. The server will continue to break least-used idle connections when the connection table becomes full and a new connection request is received.

You may find that the advanced protocol settings for Maximum Sessions and NCBs found under Network Setup for the Microsoft NetBEUI driver are insufficient to support your configuration of the above options.

---

To change the default advanced protocol settings, open up the Network group within Program Manager and double click the Network Setup icon. From the Network Setup window, click the Drivers... button. This causes the Network Drivers window to appear. In the Network Drivers window, select Microsoft NetBEUI and then click the Setup... button. The advanced protocol settings window will appear.

## Communications Servers

App-Link communications servers extend the capabilities of the message transport interface by providing connectivity to sockets residing on other computers. A server receives requests from the message transport interface and is responsible for relaying them to a counterpart server on another computer. The destination server then hands off the request to the message transport interface on that computer for delivery to the specified socket. A server is usually written to handle a specific network protocol, for example NetBIOS.

A communications server is only used for network communications. The message transport interface handles all *local* communications within the same computer. Because the communications server is not needed for local communications, it is not loaded right away. The load process is delayed until the first message requiring remote communications is sent. At that time, the message transport interface will select a communications server based on the setting of the [Protocol property](#) for the sending socket, and load it into memory. Once the server is loaded, the message transport interface offloads the original message to the server for delivery to the remote socket.

**In order for a request to be delivered to a remote socket, the communications server on the destination computer must already be running.** App-Link supports dynamic loading of the server on the *sending* computer, but there is no way to load a remote server on-the-fly. In order to ensure that a server is ready and waiting to process incoming messages from remote computers, you must load it prior to actually running your application.

The easiest way to do this is to add the App-Link communications server to your Windows start-up group. That way, the server will automatically load when Windows is started. Alternatively, your application can start the server using the Visual Basic *Shell* function. The NetBIOS servers that ship with App-Link are named APLKNB.EXE (16-bit) or ALNB32.EXE (32-bit). If these files are placed in your Windows startup group, then any App-Link application running on your computer will be able to intercept and process App-Link messages off the LAN.

## Command Line Specification of Server Options

The App-Link NetBIOS server includes support for specifying any of the server INI options defined in the [APPLINK.INI File](#) via the server command line. *Server options found on the command line override corresponding settings found in the App-Link INI file.* To specify an option via the command line, include the option name followed by an equals sign and the actual value. The semi-colon character is used to delimit multiple server options. For example, the following command line assigns the *MachineName* option a value of SERVER1 and the *Appearance* option a value of 2 (no icon or Task Manager entry). Please take note of the fact that the semi-colon character was used to delimit the two option specifications.

```
C:\WINDOWS\SYSTEM\APLKNB.EXE MachineName=SERVER1;Appearance=2
```

It is not necessary to include the delimiter when only one option is specified on the command line. For example, the following command line may be used to specify the *LanaNum* associated with the NetBIOS stack.

```
C:\WINDOWS\SYSTEM\APLKNB.EXE LanaNum=0
```

## Network Setup

If you do not intend to use App-Link for communications over a Local Area Network, then you can skip this section. However, if you do plan on networking with App-Link RADX, then you are in for some exciting programming! App-Link makes network messaging extremely easy, which allows you to concentrate on the design and development of your application. In a word, using App-Link for network messaging is cool, and you'll be amazed at how much fun it can be.

App-Link on the network is very fast, reliable and efficient. The performance of App-Link is about 20x that of NetDDE, and is also much more reliable and easy to use. Unlike many network controls, App-Link performs some very sophisticated network connection management under the covers, freeing you from worrying about things like low-level error recovery. All you really need to do is tell App-Link what the message is and where to send it. Also, synchronous messages under App-Link are reliable, meaning that you know for sure whether or not a message arrived at its destination. This is not typical of many network messaging facilities.

The App-Link RADX control set is shipped with 16 and 32-bit network servers that support the NetBIOS protocol. Other servers are under development, and support for TCP/IP and native IPX/SPX will be available soon. For now, the key point to remember is that App-Link should work in *any* network environment running under Windows that supports the NetBIOS protocol. This includes the IPX/SPX NetBIOS emulator that ships with Novell.

Because network set-up is performed differently depending upon the version of Windows used, as well as the Network Operating System (NOS) employed, the most difficult part of using App-Link (if you can even call it difficult) is the network set-up. As such, we provide you with the following sections to explain the network set up procedure for environments that people seem to have the most questions about.

If, after going through this section and the chapter on troubleshooting, you are unable to achieve messaging over the network, then please give us a call. We will be happy to assist you, and won't be satisfied until you are either up and running, or we determine that your environment is incompatible with App-Link. Incidentally, we have *never* run into a situation where this was the case, but if we do, we will cheerfully accept the package in return and refund your license fee.

If your environment is not covered in the following sections, then fear not.

Ninety percent of the time there is really no need to even reference this section, as App-Link requires no special setup. It has been our experience however, that these environments seem to cause some confusion.

When you test App-Link on the network, please use the sample SEND and RECV programs that we ship with the product. We know that internally these applications use App-Link properly, thus it helps to eliminate the possibility of a programming problem and allows us to focus on network connectivity issues.

## Windows95

Under Windows95 you can run NetBIOS via these means:

### 1. Microsoft NetBEUI

This is the simplest configuration. If this is your only protocol, then you should be all set. If you have multiple protocols loaded, then Microsoft NetBEUI should be specified as your default protocol. To set Microsoft NetBEUI as your default protocol, do the following:

1. Right mouse click on the Network Neighborhood icon on your desktop, and select Properties.

2. The Network dialog should now be displayed. Select the Configuration tab, then select NetBEUI in the network component list.
3. Click on the Properties button, which will then present the NetBEUI Properties dialog.
4. Select the Advanced tab, then verify that the check box in the lower left hand corner reading Set this protocol to be the default protocol is checked.
5. Click on OK to save the NetBEUI properties.
6. Click on OK to close the Network Dialog.
7. At this point, you will need to restart your system to make the change take effect.

#### 1. **Novell IPX/SPX with the NetBIOS compatible transport**

Assuming that the components for Novell are already installed:

1. Right mouse click on the Network Neighborhood icon on your desktop, and select Properties.
2. The Network dialog should now be displayed. Select the Configuration tab, then select IPX/SPX - compatible Protocol in the network component list.
3. Click on the Properties button, which will then present the IPX/SPX - compatible Protocol Properties dialog.
4. Select the NetBIOS tab.
5. Make sure that the check box stating I want to enable NetBIOS over IPX/SPX is checked.
6. Select the Advanced tab, then verify that the check box in the lower left hand corner reading Set this protocol to be the default protocol is checked.
7. Click on OK to save the IPX/SPX - compatible Protocol properties.
8. Click on OK to close the Network Dialog.
9. At this point, you will need to restart your system to make the change take effect.

## **Finding the name of your computer**

Sometimes it is confusing trying to figure out what your computer name is under Windows95. This is a common cause of No answer from partner errors. Fortunately, finding the correct name is easy. Simply run the WHOAMI.EXE program that is included with the package. This is a simple VB program that uses the App-Link *GetComputerName* method to query the local computers name and displays it in a small window.

## **Windows NT**

In order to run App-Link on Windows NT, follow these steps:

1. Go to your Main group window and double-click the Control Panel icon.
2. In the Control Panel group window, double-click the Network icon. At this point you should see the Network Settings dialog box.
3. First, make sure that you have NetBIOS support configured. You should see an entry for the NetBEUI Protocol, and another for NetBIOS Interface listed in the Installed Network Software list. If not, you need to install these components. Once this is done, proceed.
4. Select the NetBIOS Interface list item and click on the Configure... button. This will bring up another dialog box that shows network routing information for this component. You should see a dropdown list of Network Routes and an entry field for the associated Lana Number. You need to make sure that the route that starts with Nbf>... is associated with Lana Number 0. If it is not, then change it using the following procedure:

To load from DOS:

- a. Select the entry that is currently assigned to Lana 0. Change the Lana Number from 0 to the next available Lana Number. Note: If you try to assign it to one that currently



- exists, you'll get an error to the effect of duplicate route definition.
- b. Now select the route for Nbf>... and change the Lana Number to 0.
  - c. Click the OK button on this dialog to save your changes.
  - d. Click OK again on the Network Settings dialog. NT will tell you that changes have been made to the network configuration. At this point, you should shutdown and then restart NT.

You should be all set once these changes have been made. Just make sure that the LanaNum setting in your APPLINK.INI file is set to 0 as well. Unfortunately, for some reason the NetBIOS Interface route must be associated with Lana Number 0 under Windows NT.

## Banyan Vines

The setup involves configuration of a Vines server, as well as, the individual workstations.

### Setting up the Vines Server

You will need to configure the NetBIOS Naming Service on one of your servers. NNS as it is referred to, provides centralized management of NetBIOS node names. Each workstation must register their NetBIOS node name with this server. This is done using the SETNETB command (more on that when we get to the workstation configuration section). To configure NNS:

1. Use the Vines MSERVICE or OPERATE command to add a server-based service.
2. Select the server that will perform the NNS registration service.
3. Select the NNS option from the list of available server options.

Note that the name that you give this service will be used as one of the parameters in the SETNETB command on the workstation.

### Setting up a Vines Workstation

1. Load the PCNETB TSR. You can do this by enabling it with the PCCONFIG program or loading it from DOS after logging in to Vines.

To load from DOS:

- a. Login to Vines by loading BAN.EXE.
- b. Execute PCNETB from the command line. PCNETB supports the following options:

**/SES:n**     Number of concurrent NetBIOS sessions. Range 1-100, default 6. Configure this at 24 or higher. For example, /SES:24.

**/CMD:n**     Number of outstanding commands. Range 1-100, default 12. You should set this option at 48 or so. For example, /CMD:48.

**/PKT:n**     Packet size. Range 400-5600, default 400. The book suggests a setting of 1500 for Ethernet, and 4096 for Token Ring.

**/NAM:n**     Number of local machine names. Range 1-100, default 16. The default is fine.

Proposed command line: **PCNETB /SES:24 /CMD:48**

2. Load SETNETB. This program allows you to specify the NNS service to connect to. Make sure you load SETNETB **after** you load PCNETB. The following options are supported:

<b>/PERMID:hex#</b>	This is the permanent node ID used by NNS. It is a 12 digit hex # that is required if one work station must have multiple NetBIOS sessions at once. Try running SETNETB without this at first.
<b>/NAME:machine name</b>	This is the NetBIOS name that you wish to give to the workstation. When using App-Link to send messages between remote workstations, you use this name in the first part of the Dest specification. The only requirement here is that the name be unique. The NNS server will reject a name if it is already being used by another workstation. If you do not specify the /NAME option, Vines will create one based on users Login StreetTalk name.
<b>/NONAME:</b>	Implies that no machine name is to be specified. Do not use this!
<b>service-name</b>	This option specifies the name of the Vines NNS service configured at the server. A workstation must specify the service name that handles the naming for a particular application.

Proposed command line: **SETNETB /NAME:MyName Netbios@Authors@NRP**

This assumes that the NNS service name configured at the server is Netbios@Authors@NRP.

The following are Vines file requirements for using NetBIOS. You should ensure that these requirements are met:

#### **DOS/Windows (Loading the software before starting Windows)**

REDIR4.EXE  
PCNETB.EXE  
SETNETB.EXE

#### **Windows w/ WINSTART.BAT**

REDIRNP4.EXE  
PCNETB.EXE  
SETNETB.EXE

#### **Other environments**

If you are having difficulty with an environment not listed, then the first thing to do is ensure that NetBIOS support is up and running. Try to run the sample App-Link SEND and RECV programs, and go through the Troubleshooting guide in the manual.

If you still experience difficulties, then call our technical support line or send us an E-Mail that describes the problem. We'll do our best to get you going as soon as possible!

## Distribution of App-Link Applications

Okay, youve debugged your App-Link application and are ready to distribute it. In order to run the application though, youll need to include the App-Link run-time files, but which ones? This section guides you through the process of selecting the correct set of App-Link run-time files to distribute.

Select the description that best describes the environment your application will run in from the list below, then follow the instructions beneath your selection to copy the required App-Link run-time files. Its that simple.

***Windows 3.x, WFW, or Windows95 / Windows NT (running 16-bit App-Link applications exclusively, with no plans for running 32-bit App-Link applications in these Win32 environments):***

Copy all of the files under the \AL20\REDIST\WIN16 subdirectory.

If your application uses the 16-bit OCX version of the control, you need to distribute AL16.OCX. Otherwise, if you are using the VBX control you need to include APPLINK.DLL. Be sure to rename APPLINK.DLL to APPLINK.VBX or your application will not load properly.

The following files are necessary if you are using App-Link for communications over a NetBIOS network:

APLKNB.EXE  
APLKNBP.DLL

***Windows95 or Windows NT running 16-bit or 32-bit applications or both:***

Copy all of the files under the \AL20\REDIST\WIN32 subdirectory.

If your application uses the 16-bit OCX version of the control, you need to distribute AL16.OCX. If your application uses the 32-bit OCX, you need to distribute AL32.OCX.

The following file is necessary if you are using App-Link for communications over a NetBIOS network:

ALNB32.EXE

Under no circumstances should you distribute the design-time version of the VBX or the license file (APPLINK.LIC), as that is a violation of the license agreement. The design-time components are licensed to a single user, and should not be included in the distribution of your application.

---

## Common Questions and Answers

The following sections answer common questions received about the use of App-Link. Most difficulties may be eliminated via the means outlined in this section. You may also want to review the README file included with the package for information that did not make it into the manual.

[Duplicate socket name error](#)

[Synchronous transaction pending error](#)

[Unable to load remote application](#)

[No answer from partner error](#)

[App-Link doesn't work with my network](#)

[App ends abruptly after socket error](#)

[Unable to load daemon task error](#)

[Losing messages](#)

[Network session was closed error](#)

[Broadcast messages not being delivered](#)

## Duplicate socket name error

Each socket on the same computer, even if they are in different applications, must have a unique name. Otherwise, App-Link does not know which one to send the message to. Remember, socket names can be changed during run-time by assigning the SocketName property a value.

The most common circumstance causing this error is when iterative messages are sent using the START\_MULTI\_INSTANCE flag. When this flag is used, the target socket must have the same name as the application to load *until the application has loaded*. After the application has loaded, the socket should be renamed if there is a chance that another instance of the same program is to be loaded using the START\_MULTI\_INSTANCE flag. If the socket is not renamed and there is an attempt to load another instance of the same program, then the "Duplicate socket name" error will occur.

One easy solution to this problem is to let App-Link name the socket for you. This may be accomplished by assigning a blank to the SocketName property. App-Link will then automatically generate a unique socket name that your application can query by reading the SocketName property value.

## Synchronous transaction pending error

When a message is sent in "WAIT" mode, control does not return to the command execution line until a return code has been received from the receiving socket. That is, the receiving socket's [ReceiveMessage procedure](#) is exited. Due to restrictions in DDE, only one synchronous operation can happen at a time. So, for example, if you have a button that allows the user to send a message in WAIT mode and he clicks on it fast and repetitively, chances are he is going to fire off another synchronous message request while one is still being processed, thus the "Synchronous transaction pending" error.

Fortunately, there is an easy solution to this problem. Just prior to sending a message in WAIT mode, disable the controls that would allow the user to send another message while the current one is being processed. Just after the send command line (where control returns) insert a command to re-enable the commands allowing messages to be sent once again. Unless there is some kind of problem on the receiving end of the message causing a timeout, the disabled state period should be very brief.

## Unable to load remote application

In order to load a remote application, the App-Link network server must be running on the remote computer. Check to be sure that the server is running on the remote computer and try again. If it still does not work, make sure that the target socket name is exactly the same as the .EXE file name of the application that you wish to load, and that the application has a socket with the same name. Also, it's a good idea to rename the socket via program control after it loads in order to avoid a "Duplicate socket name" error on a subsequent load of the same application.

## No answer from partner error

If you receive this error, chances are that one of three things are wrong , and all are easy to fix:

1. *The Dest property is not exactly equal to the name of the computer you intend to receive the message.*

Make sure that that the name you provide to the *Dest* property is correct. A common mistake is to include leading or trailing blanks in the name assigned to the *Dest* property, which would not be recognized as a match. Also, be sure that you have identified the name of your computers correctly. If you are using Windows for Workgroups (WFW), then the computer name may be found in the SYSTEM .INI file as assigned to the MachineName option.

If you have a Novell Network network, then you must first run the NETBIOS.EXE program that is shipped with Novell prior to using App-Link for network communications. The best approach for this environment is to include the NETBIOS.EXE program in the AUTOEXEC.BAT file. NETBIOS.EXE allows Novell Network to emulate the NetBIOS protocol used by the NetBIOS server shipped with App-Link. Because Novell does not provide facilities for naming a computer, App-Link provides this capability via the MachineName option in the APPLINK.INI file, or as a parameter on the APLKNB.EXE (NetBIOS Server) invocation. If applicable, be sure that the MachineName parameter value for the receiving computer matches the value assigned to the *Dest* property on the sending computer.

2. *The App-Link NetBIOS server is not loaded on the receiving end.*

In order for App-Link to communicate over a LAN, it is necessary for each computer using App-Link for LAN communications to run a small program in the background called the App-Link communications server. This program is responsible for acting as a transfer agent between the network and the App-Link message transport. Note that it is only required for network communications. Thus, the server will only be loaded *automatically* if a program requests a send with a remote node specified in the *Dest* property. On the receiving end, however, the server must be explicitly loaded in order for the receiving socket to get the message off the network.

The App-Link servers that support NetBIOS networks are named APLKNB.EXE (16-bit) and ALNB32.EXE (32-bit). If you intend to perform network communications with App-Link, then the best approach is to place the server executable file in the Windows STARTUP group so that it will be loaded each time Windows starts. Alternatively, your App-Link application could use the Visual Basic Shell function to load the server. At any rate, it must be loaded in order for network communication to occur.

3. *The LANAn setting in PROTOCOL.INI does not match the LANANUM setting in APPLINK.INI.*

The easiest solution for this problem is to set Microsoft NetBEUI as the default protocol on your network. This action will cause the LANA0 value in the Windows PROTOCOL.INI to be set to NetBEUI, which is the default for App-Link. This problem should normally only happen if you are running multiple network protocols. The bottom line is that the LANAn (where *n* is 0,1,2...) setting in PROTOCOL.INI for the NetBIOS protocol must be the same *n* as the LANANUM setting in APPLINK .INI. Note that LANANUM may also be specified as a parameter setting on the invocation of the App-Link NetBIOS communications server.



## App-Link doesn't work with my network

The basic App-Link control supports only NetBIOS networks. Make sure that your LAN is NetBIOS capable. Synergy Software Technologies Inc. is developing servers for other network protocols such as TCP/IP and IPX/SPX. Call for more details on availability and pricing.

## Application ends abruptly after socket error

Visual Basic applications will end immediately upon detection of a trappable error. You should write an error handler routine in order to trap the error, notify the user, and return control to your program. Refer to the sample SEND application for a simple example of how to do this.

## Unable to load daemon task error

App-Link was unable to load APLKDAEM.EXE. This background processor is needed by the 16-bit App-Link controls. Either APLKDAEM.EXE is not in the execution path for the computer or there is not enough conventional memory available to load it. Check the path to ensure that it is available, and if so take a look at your use of conventional memory to see if you can free up some space.

## Losing messages

If your application is highly message-bound (i.e. it is sending messages at a high rate in a tight loop), then you may be overriding the Windows message queue. Setting the WIN.INI DefaultQueueSize option from its default of 8 to a larger size should fix the problem. You should also examine your application from a design perspective and determine if the message rate you are using is necessary or not. Often times an inordinate amount of message traffic is used when it is not really necessary.

## Network session was closed error

The version 2.0 NetBIOS server is not compatible with the its 1.0 predecessor (unfortunately, this was unavoidable). Thus, all messages received from version 1.0 servers will be rejected. The sending server will receive a '*Network session was closed*' error (for messages that were not broadcast). Also, the receiving version 2.0 server will log an error to the file APLKNB.ERR that specifies the name of the offending computer, and the fact that a non-compatible message was received . View the APLKNB.ERR file to verify that this is the problem. If it is, then simply replace the back-level (1.0) version of the server files, consisting of APLKNB.EXE, APPLINK.INI and APLKNBP.DLL with their version 2.0 counterparts.

## Broadcast messages not being delivered

Broadcast support is a "connectionless" service. That is to say, delivery of a message when using broadcast is not guaranteed by the network. If your application requires guaranteed delivery of messages, then you should use synchronous sends.

## Technical Support

The majority of problems encountered during App-Link usage can be resolved by following the guidance of the Troubleshooting section of this manual. Should the options offered there fail to resolve your problem, don't despair. App-Link Technical Support is just a phone call away, and, unlike many other software providers, it's free. All we ask is that you pay for the phone call (please do not use our 800 sales line for technical support or questions.) We'll be here to help you through any difficulties that may arise, and are dedicated to resolving all issues brought forth.

Before calling Synergy for technical support, please make sure that you have read through the troubleshooting section of the manual first. We also encourage you to use the Synergy Software Technologies forum on Compuserve (GO SYNSOFT). If you post a message there we'll get back to you within 24 hours. Also, other App-Link users may be able to offer you suggestions regarding different ways to approach your particular problem. If you find yourself without a solution after exhausting these means, then please give us a call. Also, please have ready the following information:

- Your App-Link serial number (on the installation diskette).
- The version of Microsoft Windows you are running.
- The type of local area network you are using.
- The type and version of the network software you are using.
- The nature of your problem.

We can be reached via any of the following ways:

*Support:* (802) 878-8514

*Fax:* (802) 878-4055

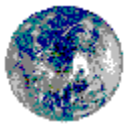
*Compuserve:* 74777,757 (visit our forum on CIS: type GO SYNSOFT)

*Internet:* 74777.757@COMPUSERVE.COM

*Mail:* Synergy Software Technologies Inc.  
159 Pearl Street  
Essex Junction, VT 05452

Synergy is best reached between the hours of 8am and 5pm, Eastern Standard Time.

Oh yeah, check out our World Wide Web site at:



[www.synergysw.com](http://www.synergysw.com)

