






Making the Web W






Inter-Application Data Exchange Made Easy

Introduction

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

Control Reference



-  Properties
-  Methods

-  Events
-  Functions
-  Trappable Errors

Operational Components

-  Using the APPLINK.INI File
 -  App-Link 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 iMS 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 iMS 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 iMS SOFTWARE.

The App-Link iMS Software 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 individual will be accessing the product. The purchaser may not distribute the App-Link iMS Workstation Edition design-time components (specifically: APPLINK.LIC or APPLINK.VBX) under any circumstances. If the purchaser has purchased an App-Link iMS Server Edition license, then the App-Link iMS Client Edition of the product may be distributed by the purchaser to provide design-time support for interfacing to App-Link Server Edition applications.

The purchaser is authorized to install the App-Link iMS communications server component (specifically ALTCP32.EXE or ALTCP16.EXE) on a maximum of 10 (ten) workstations, subject to the terms of this agreement. This restriction also extends to installation on a network, if more than one workstation will be accessing the component in a shared manner (i.e. each workstation sharing the component counts toward the limit of 10). If additional communications server licenses are required, then the purchaser must obtain a separate runtime license directly from SSTI. If you have any questions regarding the terms of App-Link iMS runtime licensing, or how to obtain additional licenses please contact SSTI directly for clarification.

The App-Link iMS software may be included in ActiveX objects as long as the design-time and run-time licensing restrictions identified in this document are adhered to at all times. Use of any ActiveX or OLE control or object containing App-Link iMS constitutes design-time use of App-Link iMS, and requires a design-time license. Further, products or applications using App-Link iMS must use the product to augment system function, and may not be commercially competitive with the App-Link iMS product itself.

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

©1997 Synergy Software Technologies Inc. All rights reserved.
First printing, July 1997

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

Trademarks

App-Link, iMS and HyperActiveX 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.

Cover image copyright 1997 PhotoDisc, Inc.

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!

Synergy Software Technologies would like to thank the beta testers and existing customers that provided input for App-Link iMS. We appreciate your input!

Overview of App-Link

Thank you for purchasing the App-Link Internet Messaging System (iMS). App-Link iMS will help you enable your applications to the Internet easily and efficiently. App-Link iMS provides effortless deployment of communication between applications over the Internet, Intranet, or any TCP/IP network. Consider App-Link iMS to be your own personal data-exchange superhero -- it 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 know by now, most visual programming environments present information to application users on "forms". A form is essentially a window that contains controls, 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.

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 computers on a 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 control icon looks just like a phone jack when you place it on a form!



This is the graphic of the App-Link control object. 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 communications 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. App-Link is fast, reliable and easy to use in a large variety of Windows visual programming environments.

Three Dimensional Data Exchange

App-Link iMS 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, in essence allowing you to create your own events that are fired by the receipt of App-Link based messages.

Between sockets in separate applications 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. 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 network

Perhaps the most exciting use of App-Link iMS, this level provides high-performance data exchange between sockets residing on separate physical computers linked by the Internet, or an Intranet running the TCP/IP protocol. Using App-Link iMS at this level, a programmer can provide full access to the interaction of applications over the network as if they were residing on the same computer! By providing on the IP address/DNS name for the remote host and the name of the destination App-Link socket, data can be sent without any additional programming. Its that simple!

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 can create a unique name for you if you wish.

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

Socket1	A local socket named Socket1, that is assumed to be on the same computer that it is referenced on.
\\Server\Socket1	A remote socket named Socket1 that resides on a remote host computer named Server.
\\126.034.124.261\Socket1	A remote socket named Socket1 residing on a host computer with an IP address of 126.034.124.261.

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 host 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:

```
txtComputer = skt1.GetComputerName()
```

This statement will set the contents of the text box txtComputer to the name of the local host 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 try out the control in your application. 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 iMS control set, you are ready to use the App-Link control objects 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 OCX/ActiveX controls. App-Link adheres to 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 called `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 which 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 structured messages. Because structures can sometimes contain embedded nulls, there is really no way to accurately determine their length by convention, thus the requirement 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 who 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 on a TCP/IP network is only a matter of adding the remote *address* to the socket name in the destination parameter.

The remote address is specified using the standard TCP/IP format (123.234.212.111), or a DNS name from your local HOST file or one that is registered on an Internet DNS server. 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 TCP/IP using App-Link iMS. App-Link iMS 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 same local network (or subnet) 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 same subnet as the sending socket:

```
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 iMS development

We have illustrated a very basic example of how to use App-Link iMS 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 iMS functionality. Playing around with these samples is an excellent way of getting your feet wet with the control.

As you begin to use App-Link, you will discover that the power associated with the control is exciting. With the effortless deployment of communication between applications leveraged by the power of App-Link iMS, the number of applications that can be Internet enabled is staggering.

Some examples of how you could use App-Link iMS are:

Groupware Development

App-Link iMS is the ticket to turning a stand-alone application into a Groupware product running on the Internet/Intranet. Using App-Link iMS, a programmer can easily establish real-time connections between applications over TCP/IP. Since receipt of App-Link messages is event-driven, the program designer can easily integrate Groupware functionality into an existing stand-alone application. 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 iMS 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 iMS 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 controls that allow Visual Basic to interface with PBX cards, telephony application development has skyrocketed. App-Link iMS 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 to remote processes all over the globe.

Networked Data Acquisition and Control Systems

Many DACS hardware manufacturers now provide controls as the interface to their hardware. Using App-Link iMS, the DACs developer can easily develop a distributed DACS application. DACS hardware residing on distributed computers can transfer and receive data over a TCP/IP network using App-Link iMS. Since App-Link iMS 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 over the net.

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 iMS offers a powerful messaging infrastructure, 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. 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. Now your Client-Server applications can span the globe!

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.

HyperActiveX Controls

Being an OLE control, App-Link iMS objects may be embedded in ActiveX objects that are created with Microsofts Visual Basic Version 5. By doing this, you instantly create a messaging capability for your ActiveX control that extends its functionality over the Internet! We call this technology HyperActiveXtm, and its potential is very exciting! Just imagine the ActiveX objects that you can create using App-Link iMS as a messaging conduit to servers that you write. ActiveX controls can be offered to developers that extend their applications to communicate with App-Link enabled servers.

Note: Use of a HyperActiveX control requires an App-Link iMS design-time license. Please familiarize yourself with the App-Link iMS license agreement prior to distributing your HyperActiveX controls.

The potential uses of App-Link iMS 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. Finally, a way to leverage the power of the Internet without having to be a TCP/IP expert.

What's New in App-Link iMS

The following items run down the major updates of App-Link for iMS:

- ✓ **Enhanced 16 and 32-bit control set**

The App-Link iMS control set integrates local communication with connectivity over the Internet, Intranet, or any TCP/IP network.

- ✓ **New 16 and 32-bit TCP/IP servers**

Our new iMS servers provide full access to remote applications over TCP/IP as if they were residing on the same computer.

- ✓ **New technology for encoding/decoding user types in messages**

This release features a technology called Structured Message Support (SMS). Using SMS functions, you will be able to encode and decode any series of data items into and out of variable-length strings.

- ✓ **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.

Installation

To install the App-Link Internet Messaging System, simply place installation diskette #1 in your floppy drive, and execute the SETUP.EXE program. The setup program presents a series of dialogs that lead you through the installation process, prompting you for information along the way.

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
VALIMS10\README.WRI	General readme file in Write format
VALIMS10\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\ALTCP16.EXE	16-bit TCP/IP server
VALIMS10\SAMPLES\OCX\	16-bit OCX sample applications
VALIMS10\SAMPLES\VBX\	16-bit VBX sample applications
VALIMS10\REDIST\WIN16\	16-bit 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\ALTCP32.EXE	32-bit TCP/IP server
VALIMS10\SAMPLES\OCX\	32-bit OCX sample applications
VALIMS10\REDIST\WIN32\	32-bit 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 Version 4.0. 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 Visual 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 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 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 (it is invisible at run-time).
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 it. Set the command buttons caption property 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 application. 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. 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.

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 rudimentary 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 though 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 over a TCP/IP network. To do this, simply add the remote IP address or DNS host 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 a remote address of

123.111.111.221, then the new *Dest* property on the sending machine should be:

```
"\\123.111.111.221\\Socket1"
```

You could also use a DNS name instead of the cryptic IP address.

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 iMS 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 properties, 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. When using the ActiveX controls, we strongly encourage the use of control 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 queue.

App-Link uses the Microsoft DDE Management Library (DDEML) as the messaging infrastructure within the local 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 16-bit 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 do 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 address of the target App-Link socket.

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 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 remote host *address*, followed by the destination socket name. If the specification includes the remote host address, a pair of backslash characters (\\) must precede the remote address and a single backslash character (\) must separate the remote address and socket name components.

The remote host address can be specified using an IP address in standard TCP/IP format (111.221.241.124), or a host name that can reside in your local HOST file, or on an Internet Domain Name Server (DNS).

For example, the following assignment specifies a destination socket named *Receiver* on a remote host named *NODEX*.

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

Assuming that *NODEX* is configured with an IP address of 128.121.231.66, the following specification is equivalent to the one above.

```
MySocket.Dest = "\\128.121.231.66\\Receiver"
```

Given the choice between using a host name or an IP address, most programmers will choose to use a host name. A host name is easier to remember, and can convey meaningful information. Since the host name provides a level of indirection to the network address, a computer can be relocated and assigned a different network address without requiring changes to the application. The mapping between host name and network address is all that needs to change. Your application continues to run!

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 iMS 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.

App-Link iMS supports *limited IP broadcast* (also known as local broadcast). Limited IP broadcast means that a router will never forward a message beyond the local network (or sub-network). Only App-Link applications running on the same local network as the sender will receive the broadcast message (including the sending application).

Broadcast should be used very carefully because of the load it places on the network. Every network interface receives broadcast packets, whether or not it has a network application ready to receive the packet. This can introduce excessive overhead for every host connected to the local network.

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 iMS 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 specify the IP address or host name of the remote computer 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 connected to the local network that has an App-Link iMS server loaded (including the sender).

The ActiveX version of App-Link 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 host name of the remote computer 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 iMS 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 queries the *Error* property upon return from a 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 the Trappable Errors section of this online manual.

Data Type

String



ErrorText property example

This example displays a message box containing the 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 send 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 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	DEFAULT_FLAGS The default value is 0 (all flags set to OFF). This setting defines a synchronous, non-queued send without auto-start capability.
&H1	QUEUE_MSG 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 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 Command property description or Methods section of this online manual for more information.
&H2	START_SINGLE_INSTANCE or START_DEST_APP 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. <u>The target socket must initially have the same name as the .EXE file being loaded.</u> 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 remote host *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 a remote computer with an IP address of 111.222.123.221. 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 = "\\111.222.123.221\FIRE"
    skt.Flags = START_SINGLE_INSTANCE
    skt.Command = SENDMSG
End Sub
```

Note that the socket is located on a remote computer. It is important to point out that in order for the application to be automatically loaded on the remote computer, the App-Link iMS [messaging server](#) must be loaded on the receiving machine at the time of the send. App-Link will automatically load the iMS server on the sending end if necessary, but not on the receiving side.

Example 3

This example sends a message to a socket named *PING* on a remote host computer named *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 a remote host computer named *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 causing 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 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 conversion functions. These functions, collectively known as Structured Message Support (SMS) allow you to pack and unpack any series of data items into and out of variable-length strings. 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 from the 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 a remote computer with an IP address of 111.123.111.222. The message will be placed in the queue ahead of messages with a priority greater than 100 and after any other priority 100 messages.

```
Sub SendButton_Click ()
    skt.Msg = "This is a queued, priority 100 message"
    skt.MsgLen = Len( skt.Msg )
    skt.Dest = "\\111.123.111.222\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. App-Link iMS extends protocol support to any TCP/IP capable network.

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
1	TCPIP	TCP/IP protocol

Other settings will be allowed if the appropriate server has been installed. The value for each 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 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 VBs 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 send.

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 iMS 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.

These methods are not available in the VBX version of App-Link.

The following methods are supported in the ActiveX version:

[Broadcast](#)

[GetComputerName](#)

[GetComputerNameEx](#)

[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 iMS 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 iMS 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 specify the IP address or host name of the remote computer 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.

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 Dest property for more information regarding destination formats.
<i>Msg</i>	String	The text of the message to send. See the Msg property 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

App-Link iMS supports *limited IP broadcast* (also known as local broadcast). Limited IP broadcast means that a router will never forward a message beyond the local network (or sub-network). Only App-Link applications running on the same local network as the sender will receive the broadcast message (including the sending application).

Broadcast should be used very carefully because of the load it places on the network. Every network

interface receives broadcast packets, whether or not it has a network application ready to receive the packet. This can introduce excessive overhead for every host connected to the local network.

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.



Broadcast method example

Example 1

This example broadcasts a message to a socket called REGISTER on all remote hosts on the local network having an App-Link iMS server loaded.

```
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 hosts on the local network having an App-Link iMS server loaded. Assume that the string parameter *BinaryData* is a user defined type that has been converted to a string using the App-Link SMS functions. 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
```



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
```



GetComputerNameEx Method

[Example](#)

Description

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

Usage

```
Computer = skt.GetComputerNameEx(Format)
```

Return Value

Computer String Contains the name or address 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

Format Long Specifies the format of the returned value. The following computer name formats are available:

<u>Value</u>	<u>Symbol</u>	<u>Description</u>
1	CNF_HOSTNAME	Format value as host name.
2	CNF_HOSTADDR	Format value as host address.

Remarks

A socket configured for the NetBIOS protocol only supports the host name format option.



GetComputerNameEx method example

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

```
Sub DisplayComputerName()  
    Dim ComputerName As String  
    ComputerName = skt.GetComputerName(CNF_HOSTNAME)  
    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

Msg String Contains the message string if the call is successful. An empty string is returned if the call fails.

Parameters

Orig String Originating computer and socket name. If the message was sent from a local socket, *Orig* contains the socket name only. Refer to the [Dest property](#) for more information on the format of this string.

MsgLen Long Length of the message data.

Priority Integer Priority assigned to the message.

Flags Long Send flags in effect when the message was sent. See the [Flags property](#) 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 Dest property description for more information.
<i>Msg</i>	String	The text of the message to send. See the Msg property 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 Flags property description for more detailed information.



Send method example

Example 1

This example sends the message Hello There to a socket named CATCHER on a remote host with an IP address of 111.114.135.222.

```
Sub SendHello_Click()  
    Dim RetCode As Long  
    RetCode = skt.Send(\\111.114.135.222\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 the remote host computer named 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 the 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 iMS 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 dont 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 (senders IP address 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

It is often desirable to define the layout of a message using user-defined types. Since the App-Link control is designed to send string data, a user-defined type must somehow be converted to a string before App-Link can send it. Likewise, at the receiving end the string must be converted back to a similarly-defined user type.

Previous releases of App-Link have included two routines to perform these conversions: `AplkVBTypeToString` and `AplkStringToVBType`. Although they did the job well in VB3 (and even VB4 with some restrictions applied), we have found that the simple design premise on which these functions were based, no longer holds water in the latest development environments.

This release of App-Link features new functions for encoding and decoding the content of a message. We call this new technology Structured Message Support (SMS). Using SMS functions, you will be able to pack and unpack any series of data items into and out of variable-length strings.

SMS is intended as a replacement for the existing `AplkVBTypeToString` and `AplkStringToVBType` functions that are packaged with the App-Link RADX product. Although we will continue to support the these functions, we encourage you to use the new SMS technology because it is more robust, and is designed to be interoperable between 16/32-bit environments.

User Type to String Conversion Functions

[`AplkVBTypeToString`](#)
[`AplkStringToVBType`](#)

Structured Message Support Functions

[Introduction to SMS](#)
[`AismStoreAll`](#)
[`AismFetchAll`](#)
[`AismStoreItem`](#)
[`AismFetchItem`](#)
[`AismIndex`](#)
[`AismLenType`](#)
[`AismPieceCount`](#)
[`AismPieceItemNo`](#)
[`AismPieceSize`](#)
[`AismItemCount`](#)
[`AismItemType`](#)



AplkVBTypeToString Function

[Example](#)

Description

Copies a Visual Basic user type variable to a string.

Usage

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

Return Value

MsgText String Contains the converted user type as a string.

Parameters

<i>Src</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 *Src*.

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 * 32  
  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

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

Usage

AplkStringToVBType(*Dest* As Any, ByVal *Src* 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>Src</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 *Src* 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
```

Introduction to SMS

Structured Message Support (SMS) consists of a set of functions that allow you to manipulate App-Link messages as user types. Using SMS functions, you can pack and unpack any sequence of data items of varying types -- integers, longs, strings, arrays of any kind -- into *SuperStrings*.

SuperStrings are ordinary variable-length strings into which you have packed a series of data items using the SMS functions. Since a SuperString is nothing more than an encoded variable-length string, it can be sent and received by an App-Link socket.

SMS provides two ways to pack and unpack SuperStrings:

- ◆ All at once

The `AlsmStoreAll` function converts the entire contents of a Type variable into a SuperString. The `AlsmFetchAll` function unpacks a SuperString into the components of an appropriately structured Type variable.

- ◆ Selectively, by field

The `AlsmStoreItem` function writes (or re-writes) an individual data item into a specified position within a SuperString. The `AlsmFetchItem` function reads an individual data item from a specified position within a SuperString.

SuperString Format Descriptions

Since SuperStrings may be packed with any arrangement of data items, you must supply each of the SMS access functions with a *format description string* parameter that specifies the layout of items within a given SuperString. Format description strings are constructed according to specific rules (or syntax). Here are two examples of simple format strings:

<code>%,%,\$,#</code>	Two integers, a variable-length String, and a Double
<code>!,A100%,&</code>	A Single, a 100-element array of Integers, and a Long

Note that in order to unpack data from a SuperString, you must use the same format description that you used to pack it.

There is no restriction on the length of the format description string.

Rules for Constructing Format Description Strings

A format description string is composed of one or more *piece descriptions*. A piece description represents either an individual data item such as an Integer, or an array of individual data items. When constructing a format description string consisting of more than one piece, each piece description may be separated with spaces and/or commas.

Individual data items are represented with the standard type-declaration characters used in Visual Basic as follows:

<code>%</code>	Integer
<code>&</code>	Long
<code>!</code>	Single
<code>#</code>	Double
<code>@</code>	Currency

\$*n Fixed-length string, n bytes long
\$ Variable-length string

Note that there is no representation for type *variant*.

Arrays are represented with the letter **A**, followed by the number of array elements, and the data type (from the list above). For example:

A50% An array of 50 Integers
A50\$*10 An array of 50 fixed-length strings, each 10 bytes long

If you want to pack an array of more than one dimension, multiply the array bounds to arrive at the total number of elements. For example, a two-dimensional array with 6 rows and 4 columns has 24 elements.

In VB3, the Visual Basic *Option Base* statement does not affect arrays within Types. In this case, it is safer to define the bounds of such arrays explicitly -- for example, MyArray(1 To 10).

If you have a series of identical, consecutive pieces in a format description string, you can (optionally) save space by representing such repeating pieces with a *repeat count*. A repeat count is represented with the letter **R**, followed by the number of times the piece description would otherwise appear, and the piece description.

For example:

R4% Equivalent to %, %, %, %
R4A50% Equivalent to A50%, A50%, A50%, A50%
R2A5\$*2 Equivalent to A5\$*2, A5\$*2

Warning: Do not confuse *repeated pieces* with *pieces*. For example, the format description string R5%,#,R10A3@ contains 16 pieces, not three. This is important if you are going to use any of the auxiliary functions.

You may not repeat groups of pieces, nor may you nest repeats. The syntax does not support: R5(%,#,@) or R5R3%. In addition, arrays of variable-length strings are **not** the same as repeats of individual variable-length strings (i.e. R99\$ is not the same as A99\$).

Function Return Codes

The SMS functions return negative values to signify errors. You should make it a practice to test the return value of every function call for errors.

The valid function return codes are:

-1 *BAD FORMAT*

Either the format description string is improperly constructed, or the actual data types are not as described in the format description string. The most common format-mismatch errors are: (1) miscounting data items (and, note that we consider all counts to begin with 1, not 0) and (2) confusing fixed-length strings and variable-length strings.

-2 *BAD PARAMETER VALUE*

One or more of your input parameters has a nonsensical value (e.g., a negative item number).

-3 *GENERAL ERROR*

This error is generated for one of the following reasons: (1) out of memory, (2) internal/coding/system/Windows error or (3) if using the AlsmFetchItem function, the source SuperString was improperly formatted (or null).

-4 *UNINITIALIZED VLSTRING ARRAY*

This special error condition occurs only under the following specific circumstances: (a) You are using AlsmFetchAll, (b) the destination Type variable includes an array of variable-length (not fixed-length) strings, and (c) no element of that array has had anything assigned to it. You can initialize a variable-length string array by assigning anything (even a null string) to any of its elements.



AlsmStoreAll Function

[Example](#)

Description

Converts the entire contents of the source Type into an output SuperString.

Usage

```
Retcode = AlsmStoreAll(SrcType, Format, DestStr)
```

Return Value

Retcode Long Function return code. 0 if ok, otherwise -1, -2, or -3 as described under the Function Return Codes section.

Parameters

SrcType Any The name of an instance of a user-defined Type.

Format String A format description string that describes the structure of the *SrcType*. See Rules for Constructing Format Description Strings for more information.

DestStr String The name of a variable-length string variable. Do not specify a VB object property, such as `TextBox1.Text` for the `DestStr$` parameter.

Remarks

The reason you cannot use a VB object property as the destination is because Visual Basic strips non-printable characters out of object properties, and SuperStrings will contain non-printable characters.



AlsmStoreAll function example

Example

This example demonstrates how to pack a user-defined Type into an output SuperString. The resulting SuperString is passed to the Send method for transmission to the destination socket *Controller* running on a remote host with an assigned IP address of 126.222.321.234.

First, define the user type:

```
'Define the layout of my user type
```

```
Type SensorData  
  TestName As String * 30  
  SensorNumber As Long  
  Reading(1 To 3) As Single  
End Type
```

And then use the *AlsmStoreAll* function to convert the entire type to a SuperString prior to sending it to the destination socket. The example below sends the message when a button called SendSensorData is clicked.

```
Sub SendSensorData_Click()  
  
  Dim Sensor As SensorData  
  Dim SsFormat As String, Ss As String  
  Dim RetCode As Long  
  
  Initialize SensorData structure  
  
  Sensor.TestName = Extreme Voltage  
  Sensor.SensorNumber = 3  
  Sensor.Reading(1) = 10000  
  Sensor.Reading(2) = 20500  
  Sensor.Reading(3) = 99000  
  
  Format description string for SensorData structure  
  
  SsFormat = $"*30,&,A3!  
  
  Notice the call to AlsmStoreAll below. Here the user type is  
  converted to a SuperString prior to using the send method.  
  
  RetCode = AlsmStoreAll(Sensor,SsFormat,Ss)  
  If RetCode <> 0 Then  
    Handle conversion errors  
    x = MessageBox(0,RetCode: & Str$(RetCode),AlsmStoreAll Error,0)  
  Else  
    Because this string may contain embedded nulls, we explicitly  
    provide the length of the message to the send method below.  
    RetCode = skt.Send(\\126.222.321.234\Controller,Ss,Len(Ss),0)  
    If RetCode <> 0 Then
```

```
        x = MessageBox(0,skt.ErrorText,Send Error,0)
    End If
End If
End Sub
```



AlsmFetchAll Function

Example

Description

Reads all data items from the source SuperString and copies them into the corresponding components of the destination Type variable.

Usage

Retcode = **AlsmFetchAll**(*SrcStr*, *Format*, *DestType*)

Return Value

Retcode Long Function return code. 0 if ok, otherwise -1, -2, -3, or -4 as described under the Function Return Codes section.

Parameters

<i>SrcStr</i>	String	The name of a SuperString string variable whose contents were generated by either AlsmStoreAll or AlsmStoreItem.
<i>Format</i>	String	A format description string that describes the internal structure of the source SuperString <i>SrcStr</i> . The format description string must be identical to the format description used to create the SuperString.
<i>DestStr</i>	Any	The name of an instance of a user-defined Type. The structure of this Type variable must correspond exactly to the format description used to create the source SuperString.



AlsmFetchAll function example

Example

This example demonstrates how to unpack the contents of a SuperString into a user-defined Type variable. In the ReceiveMessage event code that follows, the Msg parameter contains the formatted SuperString. Sensor is a Type variable that serves as the destination for the conversion process.

Assuming the user type is defined as follows:

'Define the layout of my user type

```
Type SensorData
  TestName As String * 30
  SensorNumber As Long
  Reading(1 To 3) As Single
End Type
```

Use the *AlsmFetchAll* function to copy the entire contents of the SuperString to the user Type variable.

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

```
  Dim Sensor As SensorData
  Dim SsFormat As String
```

```
  Format description string for SensorData structure
```

```
  SsFormat = $"*30,&,A3!
```

```
  Copy the SuperString to the destination Type variable
```

```
  RetCode = AlsmFetchAll(Msg,SsFormat,Sensor)
```

```
  If RetCode = 0 Then
```

```
    The user type Sensor now contains a copy of Msg data!
```

```
    Now getting the values out of the message is a snap
```

```
    Debug.Print TestName:  & Sensor.TestName
```

```
    Debug.Print SensorNumber:  & Trim$(Str$(Sensor.SensorNumber))
```

```
    Debug.Print Reading 1:  & Trim$(Str$(Sensor.Reading(1)))
```

```
    Debug.Print Reading 2:  & Trim$(Str$(Sensor.Reading(2)))
```

```
    Debug.Print Reading 3:  & Trim$(Str$(Sensor.Reading(3)))
```

```
    Indicate that we processed the message successfully
```

```
    RetCode = 0
```

```
  End If
```

```
End Sub
```



AlsmStoreItem Function

Example

Description

Stores a data item into the specified position in the target SuperString.

Usage

Retcode = **AlsmStoreItem**(*TargetStr*, *Format*, *ItemNo*, *ItemType*, *Data*)

Return Value

Retcode Long Function return code. 0 if ok, otherwise -1, -2, or -3 as described under the Function Return Codes section.

Parameters

<i>TargetStr</i>	String	The name of a SuperString string variable in which you wish to store the data item.
<i>Format</i>	String	A format description string that describes the internal structure of the target SuperString <i>TargetStr</i> .
<i>ItemNo</i>	Long	The sequential number of the individual data item within the target SuperString. When calculating item numbers, consider SuperStrings to be linear sequences of individual data items, starting with item no. 1. For example, if your SuperString consists of an Integer, a 10-element array of Longs, and a Currency, the Currency item number is 12.
<i>ItemType</i>	String	The data type of the selected item, represented with the standard type-declaration characters: %, &, !, #, @, \$*n, or \$. <i>ItemType</i> is used as a consistency check and must agree with the format description and with the data type of the item.
<i>Data</i>	Any	The actual data value to store.

Remarks

AlsmStoreItem only deals with a single data item at a time. Arrays are not considered to be single data items. If the target SuperString is null -- that is, if you've set it to prior to calling AlsmStoreItem -- the function will first initialize the target SuperString. If you are going to use a series of calls to AlsmStoreItem to build a SuperString, be sure to set the string to null () before the first call.

Do not specify a VB object property such as TextBox1.Text or Label1.Caption for the target SuperString.



AlsmStoreItem function example

Example

This example builds a SuperString consisting of two items using a series of calls to *AlsmStoreItem*. We could achieve the same results using a Type variable with the same layout, and using *AlsmStoreAll* to pack it.

```
Sub ResendLastReading_Click()

    Dim Ss As String, SsFormat As String
    Dim RetCode As Long, SensorNumber As Long
    Dim Reading As Single

    Format description string that describes the internal layout of SuperString
    SsFormat = &,!
    Initialize the SuperString prior to calling AlsmStoreItem
    Ss =
    Build the SuperString on-the-fly
    SensorNumber = 5
    RetCode = AlsmStoreItem(Ss,SsFormat,1,&,SensorNumber)
    If RetCode = 0 Then
        Add Reading to SuperString variable
        Reading = 55.32
        RetCode = AlsmStoreItem(Ss,SsFormat,2!,Reading)
    End If
    If RetCode <> 0 Then
        Handle conversion errors
        x = MsgBox(0,RetCode: & Str$(RetCode),AlsmStoreItem Error,0)
    Else
        Because this string may contain embedded nulls, we explicitly
        provide the length of the message to the send method below
        RetCode = skt.Send(\\126.222.321.234\Controller,Ss,Len(Ss),0)
        If RetCode <> 0 Then
            x = MsgBox(0,skt.ErrorText,Send Error,0)
        End If
    End If

End Sub
```



AlsmFetchItem Function

Example

Description

Reads a data item from the specified position in the source SuperString and copies the value into the supplied data variable.

Usage

Retcode = **AlsmFetchItem**(*SrcStr*, *Format*, *ItemNo*, *ItemType*, *Data*)

Return Value

Retcode Long Function return code. 0 if ok, otherwise -1, -2, or -3 as described under the Function Return Codes section.

Parameters

SrcStr String The name of a SuperString string variable that contains the data item you wish to read.

Format String A format description string that describes the internal structure of the source SuperString *SrcStr*.

ItemNo Long The sequential number of the individual data item within the source SuperString. When calculating item numbers, consider SuperStrings to be linear sequences of individual data items, starting with item no. 1. For example, if your SuperString consists of an Integer, a 10-element array of Longs, and a Currency, the Currency item number is 12.

ItemType String The data type of the selected item, represented with the standard type-declaration characters: %, &, !, #, @, \$*n, or \$. *ItemType* is used as a consistency check and must agree with the format description and with the data type of the item.

Data Any The name of a variable to receive the data value.

Remarks

AlsmFetchItem only deals with a single data item at a time. Arrays are not considered to be single data items.

The source SuperString must be a properly formatted SuperString or an error will occur. Be sure to specify a destination data-item variable of the correct data type. If you specify a destination that is shorter than the actual data, the function call can overwrite memory out of bounds.

Do not specify a VB object property such as TextBox1.Text or Label1.Caption for the source SuperString.



AlsmFetchItem function example

Example

The following example presents a scenario for processing structured messages using the first item in the message as a command code. The command code is used to direct the type of processing that executes.

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

    Const UNKNOWN_CMD = 99

    Dim CommandCode As Integer
    Dim SsFormat As String

    Format description string for header
    SsFormat = %
    Extract the command code from the structured message
    RetCode = AlsmFetchItem(Msg,SsFormat,1,%,CommandCode)
    If RetCode = 0 Then
        Select a processing routine based on the command code
        Select Case CommandCode
            Case 1
                RetCode = ProcessCommand1(Msg,MsgLen)
            Case 2
                RetCode = ProcessCommand2(Msg,MsgLen)
            Case Else
                RetCode = UNKNOWN_CMD
        End Select
    End If

End Sub
```



AlsmIsIndex Function

Description

Indicates if the specified item is an index field.

Usage

Retvalue = **AlsmIsIndex**(*Format*, *ItemNo*)

Return Value

Retvalue Long Function return value. A zero (false) indicates the item is not an index field. A non-zero (true) value indicates the item is an index field.

Parameters

Format String The format description string to interrogate.

ItemNo Long The sequential number of the item within the format description string *Format* to examine.

Example

Format description strings \$\$\$X\$%! and \$,\$\$,X\$,%,! are equivalent, with the fourth item being an index field.



AlsmLenType Function

Description

Returns the byte length of the user-defined Type structure represented by the specified format description string.

Usage

Retvalue = **AlsmLenType**(*Format*)

Return Value

Retvalue Long Function return value. Contains the byte length of the structure represented by the given format string.

Parameters

Format String The format description string of the Type structure whose length you wish to calculate.

Examples

%,%,%	Returns 6
R100!	Returns 400
A10&,%,%	Returns 44
R100A15!	Returns 6000



AlsmPieceCount Function

Description

Returns the number of *pieces* in the SuperString described by the specified format description string.

Usage

Retvalue = **AlsmPieceCount**(*Format*)

Return Value

Retvalue Long Function return value. Contains the piece count for the given format description string.

Parameters

Format String The format description string to examine.

Examples

%,%,A10&,%	Describes 4 pieces, 13 items
------------	------------------------------

%,%,A10&R100%

Describes 103 pieces, 112 items

%,%,A10&,R100A99%

Describes 103 pieces, 9912 items



AlsmPieceltemNo Function

Description

Returns the item number of the **first element** of the specified *piece* of the SuperString described by the given format description string.

Usage

Retvalue = **AlsmPieceltemNo**(*Format*, *PieceNo*)

Return Value

Retvalue Long Function return value. Receives the item number of the first element of the specified piece.

Parameters

Format String The format description string to examine.

PieceNo Long The sequential number of the piece.

Example

Given a format description string equal to #,R5\$*123,@,A10\$,\$,R4%,!, the first element of piece number 11 (R4%) is item number 20.



AlsmPieceSize Function

Description

Returns the number of items in the specified *piece*.

Usage

Retvalue = **AlsmPieceSize**(*Format*, *PieceNo*)

Return Value

Retvalue Long Function return value. Receives the number of items in the specified piece.

Parameters

Format String The format description string to examine.

PieceNo Long The sequential number of the piece.

Example

Given a format description string equal to R2@,A50%, the third piece of the corresponding SuperString contains 50 items. Note that the SuperString contains three pieces, not two (R2@,A50% expands to @,@,A50%).



AlsmItemCount Function

Description

Returns the total number of *items* in the given format description string.

Usage

Retvalue = **AlsmItemCount**(*Format*)

Return Value

Retvalue Long Function return value. Receives the item count.

Parameters

Format String The format description string to examine.

Examples

%,%,A10&,%	Contains 13 items
%,%,A10&R100%	Contains 112 items
%,%,A10&,R100A99%	Contains 9912 items



AlsmItemType Function

Description

Returns the data type symbol for the given *item* in the specified format description string.

Usage

Retvalue = **AlsmItemType**(*Format*, *ItemNo*)

Return Value

Retvalue String Function return value. Receives the data type symbol for the specified item.

Parameters

Format String The format description string to examine.

ItemNo Long The sequential number of the item within the given SuperString whose data type you wish to retrieve.

Examples

%,%,A10&,%	ItemNo 4 returns &
%,%,A10&R100%	ItemNo 7 returns \$*5
%,%,A10&,R100A99%	ItemNo 4 returns \$

Trappable Errors

Trappable errors from the App-Link facility can occur while an application is running, either within the development environment or as a stand-alone executable. You can test and respond to trappable errors using the Error property. 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 look through the README.WRI file for any last-minute updates that we may have made.

General Errors

The following errors are generated by any of the App-Link system components:

Error Code	Error Description
20000	System error
20001	Unable to allocate local memory
20002	Unable to load application
20007	Unable to allocate shared memory
20008	Unable to map shared memory
20009	Shared memory access failed
20500	Unable to query computer name
20501	Request aborted
20502	Transaction failure
20503	Unable to locate task instance
20504	Duplicate socket name
20505	Unable to locate socket
20506	Unable to locate conversation
20507	Message queue is empty
20508	Unable to query task information
20509	Unable to register object class
20510	Timeout occurred
20511	Unable to establish connection
20512	Invalid message priority value
20513	Invalid timeout value
20514	Invalid message length
20515	Null message pointer
20523	No answer from partner
20528	Synchronous transaction pending
20529	PostMessage failed
20530	Unable to load communications server
20531	Unable to register notification handler
20532	Unable to create object window
20533	DdeCreateStringHandle failed
20534	Unable to register socket service name
20535	Connection was dropped
20536	Destination is missing
20537	Destination contains blanks
20538	Destination missing separators
20539	Invalid computer name length
20540	Socket name is missing
20541	Invalid socket name length
20542	Invalid separator in socket name
20543	Socket name contains blanks

20544	Application terminated
20545	Unable to load daemon task
20546	Server connection table is full
20547	Socket is disabled
20548	Null parameter pointer
20549	String resource was not found
20550	Input buffer is too small
20551	Unable to query string
20552	Socket property is read-only
20553	Invalid protocol option
20554	Invalid queue storage option
20555	Invalid queue access option
20556	Socket event handler was not specified
20557	Invalid socket event handler option
20558	Type mismatch
20559	Unable to load Netapi library
20560	NetWkstaGetInfo failed
20561	WNetGetUser failed
20562	Unable to query module name
20563	Unable to initialize 32-bit message transport
20564	Unable to verify service user
20565	Invalid computer name format
30000	Unknown method specified

IMS Errors

These errors are generated by the IMS servers:

Error Code	Error Description
20600	Interrupted function call
20602	Permission denied
20603	Bad address
20604	Invalid argument
20605	Too many open files
20606	Operation would block
20607	Operation now in progress
20608	Operation already in progress
20609	Socket operation on nonsocket
20610	Destination address required
20611	Message too long
20612	Protocol wrong type for socket
20613	Bad protocol option
20614	Protocol not supported
20615	Socket type not supported
20616	Operation not supported on socket
20617	Protocol family not supported
20618	Address family not supported by protocol family
20619	Address already in use
20620	Cannot assign requested address
20621	Network is down
20622	Network is unreachable
20623	Network dropped connection on reset
20624	Software caused connection abort

20625 [Connection reset by peer](#)
20626 [No buffer space available](#)
20627 [Socket is already connected](#)
20628 [Socket is not connected](#)
20629 [Cannot send after socket shutdown](#)
20631 [Connection timed out](#)
20632 [Connection refused](#)
20642 [Network subsystem is unavailable](#)
20643 [WinSock DLL version out of range](#)
20644 [Successful WsaStartup not yet performed](#)
20646 [Host not found](#)
20647 [Nonauthoritative host not found](#)
20648 [Nonrecoverable error](#)
20649 [Valid name no data record of requested type](#)

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 executable exists on the computer and that it is located somewhere in the Windows search path.

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 run time service before App-Link run time support was fully initialized. When using the App-Link CAPI, this error is also reported if an invalid application instance handle is 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 name specified in 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 App-Link object window class could not be registered because an internal call to the Windows RegisterClass function failed. This error is most likely due to insufficient window resources.

Timeout occurred

A timeout occurred while waiting for a synchronous transaction to complete. The destination socket may be busy or unable to respond at this time. This error may also result from a timeout issued by the App-Link iMS server while waiting for a reply to a host name resolution query.

Unable to establish connection

The destination socket is not responding to a DDE connection request. The application that contains 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 or method parameter was Null or not assigned.

No answer from partner

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

Synchronous transaction pending

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

PostMessage failed

An internal call to the Windows PostMessage function has failed. This error is most likely caused by a full Windows message queue. See the Remarks section under the Command property for 16-bit configuration options.

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.

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 or method parameter is empty.

Destination contains blanks

The Dest property or method parameter contains embedded blanks.

Destination missing separators

The Dest property or method parameter 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 or method parameter 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 or method parameter is missing.

Invalid socket name length

The length of either the SocketName property or the socket name component of the Dest property or method parameter 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 or method parameter contains invalid separator characters.

Socket name contains blanks

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

Application terminated

A synchronous request was pending when the application terminated. 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. This error may also result from 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 because of a null parameter.

String resource was not found

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

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 invalid. The supported protocols are 0 - NetBIOS or 1 - TCP/IP.

Invalid queue storage option

The queue storage property is invalid. 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

The types of one or more arguments 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 Windows 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 Windows search path.

Unable to verify service user

The thunk layer failed to verify the calling task as a user of App-Link services. This is typically the result of an incomplete initialization of the thunk layer.

Invalid computer name format

The computer name format value is invalid. The valid values are 1 - host name or 2 - host address.

Unknown method specified

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

Interrupted function call

A blocking WinSock operation was canceled.

Permission denied

The requested address is a broadcast address, but the socket is not configured for broadcast support.

Bad address

This error occurs if the length of a buffer parameter is too small, or an invalid pointer value is received by a WinSock function.

Invalid argument

An invalid argument was supplied in a WinSock function call. In some cases, the current state of the socket parameter is invalid. For example, a socket is not bound, a socket already bound to an address or a listen was not invoked prior to calling accept.

Too many open files

The network system has run out of socket handles.

Operation would block

The socket is configured as non-blocking, and the requested operation is not complete at this time.

Operation now in progress

A blocking WinSock operation is already in progress. Only one blocking operation can be outstanding per thread at any one time.

Operation already in progress

An attempt to cancel an asynchronous operation failed because the operation has already completed or has been cancelled.

Socket operation on nonsocket

The socket descriptor is not a valid socket handle or the handle was closed.

Destination address required

A destination address is required in order to connect to or send to a destination socket. This error could be caused by a problem resolving the host name.

Message too long

A message sent on a socket was larger than the internal message buffer or other network limit. This error is typically generated when trying to send a datagram that is too large to fit into the network buffer and is truncated.

Protocol wrong type for socket

The specified port is the wrong type for this socket.

Bad protocol option

The specified socket option is unknown or unsupported.

Protocol not supported

The protocol has not been configured into the system, or no implementation exists for it. The specified port is therefore not supported.

Socket type not supported

The specified socket type is not supported in this address family.

Operation not supported on socket

The attempted operation is not supported by this type of socket. The socket is not a type that supports connection oriented services.

Protocol family not supported

The protocol family has not been configured into the system, or no implementation exists for it.

Address family not supported by protocol family

The address family is not compatible with the referenced sockets protocol.

Address already in use

The 3-tuple: protocol, port number, and IP address specified for the local socket name is already in use. The APPLINK.INI file contains an option named PortNumber that you can modify to allow the iMS server to use a different port. If you change this option, you must ensure that all participating iMS servers are configured to use the new port number.

Cannot assign requested address

The address is not available from the local machine.

Network is down

The network subsystem failed. Check your WinSock DLL, protocol stack, network driver, and network interface card configuration.

Network is unreachable

The network cannot be reached from this host at this time. Try to ping the destination host using the same address. Run traceroute to determine where the failure occurs between your host and the destination host.

Network dropped connection on reset

The host you were connected to crashed and rebooted.

Software caused connection abort

The local network system aborted a connection. This could result from a timeout waiting for an acknowledgement to data sent on a stream socket. Ping the remote host to which you were connected. If it does not respond, the host might be down or there may be a network problem along the route.

Connection reset by peer

A connection was forcibly closed by the remote side. This normally results from a loss of the connection on the remote socket due to a timeout or reboot.

No buffer space available

WinSock was unable to allocate memory to accommodate the function request. This error indicates a shortage of resources on your system. If you are running many applications at one time, try unloading some to reclaim resources.

Socket is already connected

A connection request was made on an already connected socket.

Socket is not connected

A request to send/receive data failed because a connection does not exist between the local and remote host.

Cannot send after socket shutdown

A request to send data failed because the socket is shutdown.

Connection timed out

A connection request failed because the remote host did not respond within the timeout period. Make sure the destination address is correct. Try to ping the destination host using the same address. Run traceroute to determine if all routers are working properly. Check your subnet mask for correctness.

Connection refused

No connection could be made because the target machine refused it. This usually results from trying to connect to a service that is inactive on the remote host. Make sure the destination address is correct. Check the PortNumber setting in the APPLINK.INI file for an incorrect port number.

Network subsystem is unavailable

The WinSock implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable. Check that the WinSock DLL is in the current path and that all necessary components are installed and configured correctly.

WinSock DLL version out of range

The current WinSock implementation does not support the Windows Sockets version requested by the iMS server. There may be an older version of the WinSock DLL in the path ahead of the requested version DLL.

Successful WsaStartup not yet performed

The iMS server failed to initialize the WinSock DLL. The network subsystem could be inactive or configured wrong.

Host not found

The destination host name is unknown. The name is not an official host name or alias. Make sure that you have a name server configured correctly or a local host table that contains the destination host name and address.

Nonauthoritative host not found

This is usually a temporary error that indicates that the local server did not receive a response from an asynchronous service request.

Nonrecoverable error

A nonrecoverable error was received. This problem could be related to errors that were received from the Domain Name System (DNS): Format errors, Request refused or not implemented. Make sure that you have a name server configured correctly or a local host table that contains the destination host name and address. Try to ping the name server to make sure it running.

Valid name no data record of requested type

The requested host name is valid, but does not have an Internet IP address configured at the name server or local host database.

Using the APPLINK.INI File

App-Link iMS 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.

The configuration data is contained in one section in the initialization file. Note that the options listed here may also be specified as command line parameters to the iMS server. See the section on App-Link Communications Servers for more information.

The section is named **[TCP/IP Server]** and contains the following entries:

AutoUnload = 0 | 1

Defines when the iMS 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.

Appearance = 0 | 1 | 2

Specifies how the iMS 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 1.

MessagePopup = 0 | 1

Specifies whether the iMS server is to display a message box if an error is encountered during initialization. If this entry is 1, the server will display a 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 ALTCPnn.ERR located in the same directory that the server was launched from.

The default value is 1.

PortNumber = *number*

Specifies the port number on which the iMS server will register its service. A port number is used to distinguish between multiple services on a single computer. All participating iMS servers must be configured to listen on the same port. Valid ports range from 1025 - 5000. The default port is 3432.

App-Link iMS connections over the Internet may require additional configuration if any of the computers has a direct connection to the Internet over a corporate network. Due to the unregulated nature of the Internet, corporations insulate their corporate networks from unauthorized Internet access with *firewalls*. A firewall prevents unauthorized access by selectively allowing data to pass between the Internet and the corporate network.

To facilitate App-Link connections over the Internet, the iMS server allows configuration of the port number to use when registering its service. To allow App-Link connections through a firewall, your network administrator should open the port that is assigned to the PortNumber option.

App-Link Communications Servers

App-Link communications servers extend the capabilities of the message transport interface by providing connectivity to sockets residing on remote computers. A server receives requests from the message transport interface and is responsible for relaying them to a counterpart server on another computer. The target server hands over the request to the message transport interface on that computer for delivery to the specified socket. Typically, a server handles a specific network protocol (e.g. NetBIOS, TCP/IP).

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 [Protocol property](#) setting of the sending socket, and load it into memory. Once the server is loaded, the message transport interface offloads the 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 start the server from your application load event using the Visual Basic *Shell* function. That way, the server will automatically load when your application is started. The iMS servers that ship with App-Link are named ALTCP16.EXE (16-bit) or ALTCP32.EXE (32-bit). Alternatively, you could send a *dummy* message to a nonexistent computer. If the iMS server is not loaded, the App-Link message transport will load it.

Command Line Specification of Server Options

The App-Link iMS 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 *AutoUnload* option a value of 1 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\ALTCP32.EXE AutoUnload=1;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 appearance of the iMS server icon on the desktop.

```
C:\WINDOWS\SYSTEM\ALTCP32.EXE Appearance=1
```

Network Setup

If you do not intend to use App-Link iMS for remote connectivity, then you can skip this section. However, if you do plan on networking with App-Link iMS, then you are in for some exciting programming! App-Link iMS makes messaging over a TCP/IP network extremely easy, allowing you to concentrate on the design and development of your application. You'll be able to leverage the power of the Internet without having to be a TCP/IP expert.

App-Link iMS on the network is fast, reliable and efficient. Unlike many network controls, App-Link performs sophisticated network connection management under the covers, freeing you from worrying about things like low-level error recovery. All you 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 iMS control set ships with 16 and 32-bit network servers that provide remote connectivity using the TCP/IP protocol. TCP/IP is the network protocol used to connect to wide-area networks and the Internet. Using App-Link iMS controls, you can easily enable data exchange between applications running over the Internet, Intranet, or any TCP/IP network.

App-Link iMS should work in *any* network environment running under Microsoft Windows that supports the TCP/IP protocol and provides at least a level 1.1 compliant Windows Sockets library (WINSOCK.DLL). The WINSOCK.DLL library is an implementation of the Windows Sockets specification that provides an application programming interface for accessing network functionality. The iMS server uses WinSock for all TCP/IP network communications.

Because network setup 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 iMS is the network setup. As such, we provide you with the following procedures that describe the general steps that are required for installing networking components.

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 iMS. Incidentally, we have *never* run into a situation where this was the case, but if we do, we will cheerfully refund your license fee in exchange for the package in return.

Before installing TCP/IP, you need to know whether you can use Dynamic Host Configuration Protocol (DHCP). DHCP provides the best method for ensuring easy and accurate installation of TCP/IP because it automatically configures your local computer with the correct IP address, subnet mask, and default gateway. You can choose this option for configuring TCP/IP if there is a DHCP server installed on your internetwork. Contact your network administrator to find out if this option is available.

If you cannot use DHCP for automatic configuration, obtain the following values from your network administrator to configure TCP/IP manually:

- ◆ The IP address and subnet mask for each installed network adapter
- ◆ The IP address for the default local gateway (IP router)
- ◆ Whether your computer will use Domain Name System (DNS) and, if so, the IP addresses of the DNS servers on the internetwork.

Domain Name System is used to translate computer and domain names into IP addresses. A DNS server maintains a database that maps domain names to IP addresses. Usually, DNS is employed if

you are using TCP/IP to communicate over the Internet or if your Intranet uses DNS to distribute host information. TCP/IP must be installed before you can setup DNS services.

- ◆ IP addresses of WINS servers, if they available on your network

To connect to another computer using App-Link iMS, you must know the computers Internet name (host name) or IP address. However, a computers address on the Internet may vary from session to session. For example, users who dial in to an Internet provider are assigned temporary addresses.

Windows Internet Name Service (WINS) provides a distributed database for registering and querying dynamic computer name-to-IP address mappings. A WINS server allows computers with temporary Internet addresses to connect using permanent names. Even though your address is different each time you connect to the Internet, other computers will be able to address your computer by specifying your WINS name.

The following sections summarize how to install networking support for each of the Microsoft Windows operating environments.

[Windows95](#)

[Windows NT 4.0](#)

[Windows NT 3.5x](#)

[Windows for Workgroups](#)

Windows95 Network Setup

You can install TCP/IP when you run custom setup of Windows 95, or you can install it after Setup by using the Network option in the Control Panel.

Installing TCP/IP

1. In the Network dialog click the Configuration tab, and then click Add.
2. Double-click Protocol in the Select Network Component Type dialog.
3. Select the name of the component manufacturer in the Manufacturers list on the Select Network Protocol dialog, and then select the specific component in the Models list. Click OK to select the protocol.

When you install Dial-Up Networking or another network adapter, Windows 95 will automatically bind TCP/IP to the adapters if it has been installed.

Configuring TCP/IP

For TCP/IP to work on your computer, it must be configured with the IP addresses, subnet masks, and default gateway for each network adapter in the computer. You can configure TCP/IP using two different methods:

- ◆ Automatically, using a Dynamic Host Configuration Protocol server.
- ◆ Manually, if there is no DHCP server available.

Configuring TCP/IP using DHCP

1. Make sure the Obtain an IP Address Automatically option is checked on the IP Address tab of the TCP/IP Properties dialog.
2. When you restart the computer, the DHCP server automatically provides the correct configuration information for your computer.

If you subsequently change the configuration, any manual settings will override the automatic settings provided by the DHCP server.

Configuring TCP/IP Manually

You can manually configure TCP/IP after it is installed if you cannot use automatic DHCP configuration. The following procedure describes the basic configuration steps for configuring valid addressing information.

1. In the Network dialog, double-click the TCP/IP protocol that you wish to configure in the list of installed components.

If your computer has multiple network adapters, you will need to configure each adapter with its own IP address, subnet mask, and default gateway. All other settings apply system-wide.

2. Click the IP Address tab in the TCP/IP Properties dialog.
3. Click the option named Specify An IP Address.
4. Type an IP address and subnet mask in the respective boxes.

The value in the IP Address box identifies the IP address for the network adapter in the local computer.

The value in the Subnet Mask box identifies the network membership and its host ID for the selected network adapter. The system uses this value to separate the IP address into host and network ID components.

5. To specify which network clients are bound to the TCP/IP protocol, click the Bindings tab. A checkmark beside the client name indicates that the network client is bound to the given protocol.
6. Click the Gateway tab. Type at least one IP address for the default gateway (IP router) on the network, and then click the Add button.

To specify an IP address for another gateway, type the IP address in the New Gateway box, and then click the Add button.

The first gateway in the list is the default gateway. Windows 95 attempts to connect to other gateways only if the default gateway is unavailable.

7. Click OK, and then restart the computer for your changes to take effect.

If you are using Dial-Up Networking to connect to the Internet, you can manually configure DNS and IP addresses for each connection that you define.

Configuring TCP/IP to Use DNS

On a local computer, the HOSTS file can be used to list known IP addresses mapped with corresponding host names. The HOSTS file is used as a local DNS equivalent to resolves host names to IP addresses.

Windows 95 can also be configured to use a DNS name server. The name server must be configured and available on the network. The DNS name server replaces the functionality of the HOSTS file by providing a *dynamic* mapping of IP addresses to host names. Before you can use TCP/IP to connect to the Internet, you need to configure a computer to recognize DNS information.

Setting the DNS IP Address with a Direct LAN Connection to the Internet

1. In the Network dialog, click the Configuration tab.
2. Double-click TCP/IP for the given network adapter in the Network Component list.
3. Click the DNS Configuration tab in the TCP/IP Properties dialog.
4. Click Enable DNS, and then type your host name and domain name in the Host and Domain boxes. These names identify who you are on the Internet.
5. Type the address of your LANs DNS server in the DNS Server Search Order box. If your network has more than one DNS server, type each DNS server and then click Add. DNS settings are global across all installed instances of TCP/IP. The first server listed is the first one searched, then the second, and so on.

Setting the DNS IP Address for each Connection in Dial-Up Networking

1. In Dial-Up Networking, right-click the connection you defined for the Internet, and then click Properties.

2. In the connections properties, click Server Type, and then click TCP/IP Settings.
3. Click the Specify An IP Address option in the TCP/IP Settings dialog, and type in your IP address.
4. Click the Specify Name Server Address option, and then type in the IP address of the DNS server in the Primary DNS box.

Because IP addresses identify nodes on an interconnected network, each host on the internetwork must be assigned a unique IP address for that network.

Windows NT 4.0 Network Setup

You can install TCP/IP when you run custom setup of Windows NT 4.0, or you can install it after Setup by using the Network option in the Control Panel.

Installing TCP/IP

1. Double-click the Network icon in the Control Panel.
2. In the Network dialog click the Protocols tab, and then click Add.
3. Select TCP/IP Protocol in the Network Protocol list, and then click OK.
4. Click Yes to use DHCP to configure TCP/IP automatically, or Click No to manually configure TCP/IP.
5. Type the full path to the Windows NT distribution files and click Continue.

All necessary files are copied to your hard disk.

When you install Dial-Up Networking or another network adapter, Windows NT will automatically bind TCP/IP to the adapters if it has been installed.

Configuring TCP/IP

For TCP/IP to work on your computer, it must be configured with the IP addresses, subnet masks, and default gateway for each network adapter in the computer. You can configure TCP/IP using two different methods:

- ◆ Automatically, using a Dynamic Host Configuration Protocol server.
- ◆ Manually, if there is no DHCP server available.

Configuring TCP/IP using DHCP

1. Make sure the Obtain an IP Address Automatically option is checked on the IP Address tab of the TCP/IP Properties dialog.
2. When you restart the computer, the DHCP server automatically provides the correct configuration information for your computer.

You cannot use DHCP configuration for a server that you are installing as a DHCP server or a WINS server. In these cases, you must manually configure TCP/IP settings per instructions in the section Configuring TCP/IP manually.

If you subsequently change the configuration, any manual settings will override the automatic settings provided by the DHCP server. As a general rule, you should not change the automatic settings configured by DHCP.

Configuring TCP/IP Manually

You can manually configure TCP/IP after it is installed if you cannot use automatic DHCP configuration. The following procedure describes the basic configuration steps for configuring valid addressing information.

You must be logged on as a member of the Administrators group for the local computer to install and configure TCP/IP.

1. In the Network dialog, click TCP/IP Protocol in the Network Protocol list, and then click Properties.

If your computer has multiple network adapters, you will need to configure each adapter with its own IP address, subnet mask, and default gateway. All other settings apply system-wide.

2. Click the IP Address tab in the TCP/IP Properties dialog.
3. In the Adapter list, select the network adapter that you want to configure.
4. For each bound network adapter, type values in the IP Address and Subnet Mask boxes.

The value in the IP Address box identifies the IP address for the selected network adapter.

The value in the Subnet Mask box identifies the network membership and its host ID for the selected network adapter. The system uses this value to separate the IP address into host and network ID components.

5. For each network adapter, type the IP address of the default gateway in the Default Gateway box.
6. Click OK on the TCP/IP Properties dialog.
7. Click OK on the Network dialog.

The settings take effect after you restart your computer.

If you are using Dial-Up Networking to connect to the Internet, you can manually configure DNS and IP addresses for each connection that you define.

Configuring TCP/IP to Use DNS

On a local computer, the HOSTS file can be used to list known IP addresses mapped with corresponding host names. The HOSTS file is used as a local DNS equivalent to resolves host names to IP addresses.

Windows NT 4.0 can also be configured to use a DNS name server. The name server must be configured and available on the network. The DNS name server replaces the functionality of the HOSTS file by providing a *dynamic* mapping of IP addresses to host names. Before you can use TCP/IP to connect to the Internet, you need to configure a computer to recognize DNS information.

To Configure TCP/IP DNS Connectivity

1. In the Network dialog, click TCP/IP Protocol in the Network Protocol list, and then click Properties.
2. Click the DNS tab in the TCP/IP Properties dialog.
3. Optionally, type in your host name and domain name in the Host Name and Domain boxes. These names identify who you are on the Internet.
4. Click the Add button under DNS Server Search Order.

If you are using DHCP, it may be set up to automatically configure the DNS Server Search Order. If not go to step 5.

5. Type the IP address of the DNS server that will provide name resolution, and then click the Add button. This will move the address to the DNS Server Search Order box.

You can add up to three IP addresses for DNS servers. The DNS servers will be queried in the order

listed.

6. Click the Add button under Domain Suffix Search Order.
7. Type the DNS domain suffix to append to host names during name resolution, and then click the Add button. This list can contain up to six DNS domain suffixes. The suffixes will be appended in the order listed.
8. Click OK on the TCP/IP Properties dialog.
9. Click OK on the Network dialog.

The settings take effect after you restart your computer.

Windows NT 3.5x Network Setup

You must be logged on as a member of the Administrators group for the local computer to install and configure TCP/IP.

Installing TCP/IP

1. Double-click the Network icon in the Control Panel to display the Network Settings dialog.
2. Choose Add Software to display the Add Network Software dialog.
3. Select TCP/IP Protocol And Related Components from the Network Software box, and then click the Continue button.
4. In the Windows NT TCP/IP Installation Options dialog, select the options for the TCP/IP components you want to install. When you have selected the options you want, click the Continue button. Windows NT Setup displays a message prompting for the full path to the distribution files.
5. In the Windows NT Setup dialog, enter the full path to the distribution files, and then click the Continue button.

All necessary files are copied to your hard disk.

6. Choose the OK button on the Network Settings dialog.

Configuring TCP/IP

For TCP/IP to work on your computer, it must be configured with the IP addresses, subnet masks, and default gateway for each network adapter in the computer. You can configure TCP/IP using two different methods:

- ◆ Automatically, using a Dynamic Host Configuration Protocol server.
- ◆ Manually, if there is no DHCP server available.

Configuring TCP/IP using DHCP

1. Make sure the Enable Automatic DHCP Configuration option is checked in either the Windows NT TCP/IP Installation Options dialog or the TCP/IP Configuration dialog.
2. When you restart the computer, the DHCP server automatically provides the correct configuration information for your computer.

You cannot use DHCP configuration for a server that you are installing as a DHCP server or a WINS server. In these cases, you must manually configure TCP/IP settings per instructions in the section Configuring TCP/IP manually.

If you subsequently change the configuration, any manual settings will override the automatic settings provided by the DHCP server.

Configuring TCP/IP Manually

You can manually configure TCP/IP after it is installed if you cannot use automatic DHCP configuration. The following procedure describes the basic configuration steps for configuring valid addressing information.

1. Double-click the Network option in Control Panel to display the Network Settings dialog.
2. In the Installed Network Software box, select TCP/IP protocol, and then click the Configure button to

display the TCP/IP Configuration dialog.

3. In the Adapter box, select the network adapter that you want to configure.

The Adapter list contains all network adapters to which IP is bound on this computer. The list includes all adapters installed on the computer. If your computer has multiple network adapters, you will need to configure each adapter with its own IP address, subnet mask, and default gateway.

4. For each bound network adapter, type values in the IP Address and Subnet Mask boxes.

The value in the IP Address box identifies the IP address for the local computer, or the network adapter card selected in the Adapter box if more than one network card is installed in the computer.

The value in the Subnet Mask box identifies the network membership and its host ID for the selected network adapter. The system uses this value to separate the IP address into host and network ID components.

5. For each network adapter on the computer, type the IP address for the default gateway (IP router) in the Default Gateway box. This parameter is only required for systems on internetworks. If this parameter is not provided, IP functionality is limited to the local subnet.

To specify IP addresses for additional gateways, select the Advanced button to display the Advanced TCP/IP Configuration dialog. Select the network adapter for which you want to specify advanced configuration options. In the Default Gateway box, type the IP address of another gateway that the selected adapter can use, and then choose the Add button. This list specifies up to five additional default gateways to use.

6. Choose the OK button to complete each configuration dialog. When the Network Settings dialog reappears, choose the OK button to complete TCP/IP configuration. Restart the computer for changes to take effect.

Configuring TCP/IP to Use DNS

On a local computer, the HOSTS file can be used to list known IP addresses mapped with corresponding host names. The HOSTS file is used as a local DNS equivalent to resolves host names to IP addresses.

Windows NT can also be configured to use a DNS name server. The name server must be configured and available on the network. The DNS name server replaces the functionality of the HOSTS file by providing a *dynamic* mapping of IP addresses to host names. Before you can use TCP/IP to connect to the Internet, you need to configure a computer to recognize DNS information.

To Configure TCP/IP DNS Connectivity

1. Double-click the Network option in Control Panel to display the Network Settings dialog.
2. In the Installed Network Software box, select TCP/IP protocol, and then click the Configure button to display the TCP/IP Configuration dialog.
3. Click the DNS button to display the DNS Configuration dialog. Names are displayed in the Host Name and Domain Name boxes.
4. Complete one or both of the following optional tasks:

Type a new name in the Host Name box. By default, this value is the Windows NT computer name,

but you can assign another host name without affecting the computer name. The host name is used to identify the local computer for authentication by utilities.

Type a new name in the Domain Name box. The DNS Domain Name is usually an organization name followed by a period and an extension that indicates the type of organization, such as microsoft.com. The domain name is used with the host name to create a fully-qualified domain name for the computer. A DNS domain is not the same as a Windows NT domain.

5. In the Domain Name System Search Order box, type the IP address of the DNS server that will provide name resolution, and then click Add. You can add up to three IP addresses for DNS servers. The DNS servers will be queried in the order listed.
6. In the Domain Suffix Search Order box, type the domain suffixes to add to your domain suffix search list, and then click the Add button. This list specifies the DNS domain suffixes to append to host names during name resolution, and can contain up to six domain suffixes.
7. When you are finished setting DNS options, click the OK button.
8. Click OK on the TCP/IP Configuration dialog.
9. Click OK on the Network Settings dialog.

You must restart your computer for the new settings to take effect.

Windows for Workgroups Network Setup

You use the Network Setup application to install Microsoft TCP/IP-32.

Installing TCP/IP

1. Double-click the Network Setup icon in the Network program group.
2. Choose the Drivers button in the Network Setup dialog to display the Network Drivers dialog.
3. In the Network Drivers list, select the adapter that you want to run TCP/IP on, and then choose the Add Protocol button. The list shows all of the network adapters and protocols installed on your computer.
4. Select Unlisted Or Updated Protocol in the Add Network Protocol dialog, and then click OK.
5. In the Install Driver dialog, specify the drive letter and path for your Microsoft TCP/IP-32 Windows for Workgroups disk, and then click OK.
6. In the Unlisted or Updated Protocol dialog, the Protocols list shows the options available on the installation disk. Select Microsoft TCP/IP-32, and then choose OK.

All necessary files are copied to your hard disk.

7. Configure your IP address, subnet mask, and other settings. See the section on Configuring TCP/IP for more details.

You must restart your computer for the changes to take effect.

Configuring TCP/IP

For TCP/IP to work on your computer, it must be configured with the IP addresses, subnet masks, and default gateway for each network adapter in the computer. You can configure TCP/IP using two different methods:

- ◆ Automatically, using a Dynamic Host Configuration Protocol server.
- ◆ Manually, if there is no DHCP server available.

Configuring TCP/IP using DHCP

1. Make sure the Enable Automatic DHCP Configuration option is checked in the Microsoft TCP/IP Configuration dialog.
2. When you restart the computer, the DHCP server automatically provides the correct configuration information for your computer.

If you subsequently change the configuration, any manual settings will override the automatic settings provided by the DHCP server.

Configuring TCP/IP Manually

You can manually configure TCP/IP after it is installed if you cannot use automatic DHCP configuration. The following procedure describes the basic configuration steps for configuring valid addressing information.

1. Select Microsoft TCP/IP-32 in the Network Drivers dialog, and then choose the Setup button.
2. In the Adapter list, select the network adapter that you want to configure.

The Adapter list contains all network adapters to which IP is bound on this computer. Initially, this list includes all adapters installed on the computer. If your computer has multiple network adapters, you will need to configure each adapter with its own IP address, subnet mask, and default gateway.

3. For each bound network adapter, type values in the IP address and Subnet Mask boxes.

The value in the IP Address box identifies the IP address for the local computer, or the network adapter card selected in the Adapter box if more than one network card is installed in the computer.

The value in the Subnet Mask box identifies the network membership and its host ID for the selected network adapter. The system uses this value to separate the IP address into host and network ID components.

4. For each network adapter on the computer, type the IP address for the default gateway (IP router) in the Default Gateway box. This parameter is only required for systems on internetworks. If this parameter is not provided, IP functionality is limited to the local subnet.

To specify IP addresses for additional gateways, select the Advanced button to display the Advanced TCP/IP Configuration dialog. Select the network adapter for which you want to specify advanced configuration options. In the Default Gateway box, type the IP address of another gateway that the selected adapter can use, and then choose the Add button. This list specifies up to five additional default gateways to use.

5. Choose the OK button to complete each configuration dialog. When the Network Drivers dialog reappears, choose the OK button to complete TCP/IP configuration. Restart the computer for changes to take effect.

Configuring TCP/IP to Use DNS

On a local computer, the HOSTS file can be used to list known IP addresses mapped with corresponding host names. The HOSTS file is used as a local DNS equivalent to resolves host names to IP addresses.

WFW can also be configured to use a DNS name server. The name server must be configured and available on the network. The DNS name server replaces the functionality of the HOSTS file by providing a *dynamic* mapping of IP addresses to host names. Before you can use TCP/IP to connect to the Internet, you need to configure a computer to recognize DNS information.

To Configure TCP/IP DNS Connectivity

1. Select Microsoft TCP/IP-32 in the Network Drivers dialog, and then choose the Setup button. The TCP/IP Configuration dialog appears.
2. Click the DNS button to display the DNS Configuration dialog. Names are displayed in the Host Name and Domain Name boxes.
3. Complete one or both of the following optional tasks:

Type a new name in the Host Name box. By default, this value is the Windows for Workgroups computer name, but you can assign another host name without affecting the computer name. The host name is used to identify the local computer for authentication by utilities. Other TCP/IP-based utilities can use this value to learn the name of the local computer.

Type a new name in the Domain Name box. The DNS Domain Name is usually an organization name followed by a period and an extension that indicates the type of organization, such as

microsoft.com. The domain name is used with the host name to create a fully-qualified domain name for the computer.

4. In the Domain Name System Search Order box, type the IP address of the DNS server that will provide name resolution, and then click Add. You can add up to three IP addresses for DNS servers. The DNS servers will be queried in the order listed.
5. In the Domain Suffix Search Order box, type the domain suffixes to add to your domain suffix search list, and then click the Add button. This list specifies the DNS domain suffixes to append to host names during name resolution, and can contain up to six domain suffixes.
6. When you are finished setting DNS options, click the OK button.
7. Click OK on the TCP/IP Configuration dialog.
8. Click OK on the Network Drivers dialog.

You must restart your computer for the new settings to take effect.

Distribution of App-Link Applications

Okay, you've debugged your App-Link application and are ready to distribute it. In order to run the application though, you'll 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 for distribution.

The easiest method of determining which App-Link run-time files to distribute is to have our App-Link iMS Distribution Wizard tell you what files are required. The Distribution wizard is a small setup-like program that automates the process of selecting the required App-Link iMS run-time files from the REDIST subdirectories described below.

The Distribution wizard presents a series of questions for you to answer regarding your application and target environment, and then creates a distribution directory containing the required App-Link iMS run-time files. All you need to do is incorporate the selected files into your custom installation.

To access the Distribution wizard, simply click on the Distribution Wizard program item within the App-Link Internet Messaging System 1.0 group. The Distribution wizard executable file is named ALIMSDWZ.EXE and can be found in the App-Link iMS REDIST subdirectory (\ALIMS10\REDIST).

If you decide not to use the Distribution wizard, you can manually copy the App-Link iMS run-time files from the REDIST subdirectories yourself. To do this, 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. It's 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 \ALIMS10\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 TCP/IP network:

ALTCP16.EXE

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

Copy all of the files under the \ALIMS10\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 TCP/IP network:

ALTCP32.EXE

Under no circumstances should you distribute the design-time version of the App-Link iMS Workstation Edition VBX or the license file (specifically: APPLINK.LIC or APPLINK.VBX), 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. If you purchased an App-Link iMS Server Edition license, then you may freely distribute the App-Link iMS Client to provide design-time support. Under the standard agreement, you are authorized to install a maximum of 10 (ten) run-time versions of the App-

Link iMS communications server (specifically: ALTCP32.EXE or ALTCP16.EXE). If additional licenses are required, you must obtain a separate run-time license directly from Synergy Software Technologies Inc. Please be sure to review the App-Link iMS Software License Agreement at the front of this online manual for additional information regarding our licensing policies.

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](#)

[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 DDEML, 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 does not contain the correct host name or IP address of the remote computer that you intend to receive the message..

Make sure that the host 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.

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

In order for App-Link to communicate over a TCP/IP network, it is necessary for each computer using App-Link for network 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 TCP/IP networks are named ALTCP16.EXE (16-bit) and ALTCP32.EXE (32-bit). If you intend to perform network communications with App-Link, then the best approach is to start the server from your application load event using the Visual Basic *Shell* function. Alternatively, you could send a *dummy* message to a nonexistent computer. If the iMS server is not loaded, the App-Link message transport will load it.

App-Link doesn't work with my network

Ensure that a TCP/IP stack is installed. Also check the configuration of your WinSock DLL, protocol stack, network driver, and network interface card.

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 running on 16-bit Windows, and is highly message-bound (i.e. it is sending messages at a high rate in a tight loop), then you may be overflowing 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. Often times an inordinate amount of message traffic is created when it is not really necessary.

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 and specify either the remote IP address or host name.

App-Link iMS supports *limited IP broadcast* (also known as local broadcast). Limited IP broadcast means that a router will never forward a message beyond the local network (or sub-network). Only App-Link applications running on the same local network as the sender will receive the broadcast message (including the sending application).

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. We will be here to help you through any difficulties that may arise, and are dedicated to resolving all issues brought forth.

The following describes the technical support options available to you.

◆ Phone Support

Technical phone support is available for a period of 60 days from the time that you place your first technical support phone call. You can contact the technical support department at (802) 878-8514. You will need to provide your App-Link serial number in order to receive phone support. The serial number can be found on installation diskette #1.

You can optionally purchase a one-year support contract directly from us. Please call Synergy or visit our web site for details regarding this offering.

◆ Other Services

These services are available to you free of charge:

Fax Number	(802) 878-4055
Web Site	http://www.synergysw.com
CompuServe	GO SYNSOFT
E-Mail	sst@synergysw.com

Before calling Synergy Software Technologies for technical support, please make sure that you have read through the troubleshooting section of the manual first. In addition, we encourage you to visit our web site or CompuServe forum. 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 give us a call. Please have the following information available:

- 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

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

