



## Pipex Documentation

© 2001 ... Slofstra Software Inc.

# Table of Contents

Foreword	0
<b>Part I Contents</b>	<b>4</b>
<b>Part II Overview</b>	<b>4</b>
1 Concepts.....	5
Elements of a Pipex network .....	5
Pipex Designer example .....	6
Node .....	7
Application .....	8
Queue .....	9
Node Routing .....	10
Logical Nodes .....	11
2 Design Principles .....	11
Queuing .....	11
Serialization .....	12
Acknowledgement .....	12
Process Independence .....	12
Error Handling .....	12
Security .....	12
Deployment .....	13
3 Components of Pipex .....	13
4 Supported Platforms and Technologies .....	14
<b>Part III Creating a Pipex network</b>	<b>14</b>
1 Contents.....	14
2 Starting Pipex Designer .....	15
3 Pipex Designer .....	18
Creating a node .....	18
Changing or deleting a node .....	20
Creating a queue .....	20
Changing, deleting, copying a queue .....	22
Creating an application .....	22
Changing, deleting, copying an application .....	23
Creating, changing, deleting a route .....	24
Creating, changing, deleting a logical node .....	25
Preparing files for deployment .....	26
4 Pipex API Setup .....	26
5 Pipex Queue Handler for OLE DB .....	27
6 Deployment .....	29
<b>Part IV Pipex Server Operators' Guide</b>	<b>30</b>
1 Contents.....	30
2 Starting and stopping the server .....	30

3 Viewing error messages and events .....	32
4 Enabling and disabling clients, viewing and flushing messages .....	33
<b>Part V Tutorials</b>	<b>34</b>
1 Building your first Pipex network .....	34
Step 1 - Create a new design .....	35
Step 2 - Create nodes and routes .....	37
Step 3 - Create a queue .....	39
Step 4 - Create an application .....	42
Step 5 - Writing Application Code .....	44
Step 6 - Testing .....	46
Step 7 - Deployment .....	48
<b>Part VI Programming Pipex</b>	<b>49</b>
1 Introduction .....	49
2 Pipex Programming Model .....	49
3 Connecting to Pipex .....	50
4 Sending messages to a queue .....	51
5 Sending messages to the OLE DB Handler .....	51
6 Creating your own queue handler .....	52
Initialization .....	52
Configure parameters when called from the Designer .....	53
Connect to the Pipex server .....	54
Receive next message, Acknowledge processed message, Report an error .....	54
Disconnect from the Pipex server .....	56
Polling Pipex for messages .....	56
7 Referencing the Pipex API Control in your Visual Basic Project or Microsoft Access Application	
<b>Part VII Reference</b>	<b>58</b>
1 Error messages .....	58
Error messages - Pipex API .....	58
Can't find Windows system directory .....	60
Invalid client type specified .....	60
Application name parameter is missing .....	60
Application name parameter is not required .....	60
Error while reading the registry .....	60
Unknown reply received .....	60
Non-numeric sequence number .....	60
Parameters are not set up in the registry .....	61
Command-line parse error .....	61
Unable to read temporary parameter file .....	61
Unable to write temporary parameter file .....	61
2 Pipex API Control Reference .....	61
Initialization and connecting .....	61
Initialize() method .....	61
ConfigurationRequested property .....	63
Host property .....	63
Port property .....	64
Node property .....	64

Queue property .....	65
Application Property .....	65
Password property .....	66
Connect() method .....	66
Disconnect() method.....	67
<b>Parameters</b> .....	<b>67</b>
Parameters property .....	67
EditParameters() method.....	68
SaveParameters() method.....	68
SetParameter() method.....	69
GetParameter() method.....	69
<b>Message Processing</b> .....	<b>70</b>
Send() method.....	70
Receive() method.....	71
Ack() method.....	73
ReportError() Method.....	73
NoOp() method .....	75
<b>Polling Pipex</b> .....	<b>75</b>
Standby() method .....	75
Notified property .....	75

**Index****0**

# 1 Contents



**Innovation through diversity**

## **Manual**

Pipex version 4.1 Manual release 1.3.

# 2 Overview

## 2.1 Concepts

### 2.1.1 Elements of a Pipex network

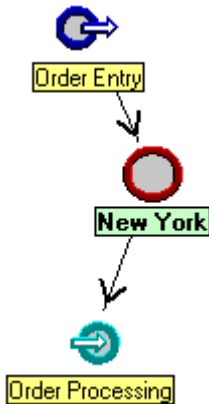
Here is a quick overview of the elements of a Pipex network. Each of these elements will be explained in some detail in the sections that follow.

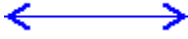
A minimal Pipex network has the following components:


- A [Node](#) 
- An [Application](#) 
- A [Queue](#) 

Messages always originate in an application. Nodes receive messages from applications, and either forward the message to another node, or to a queue for processing.

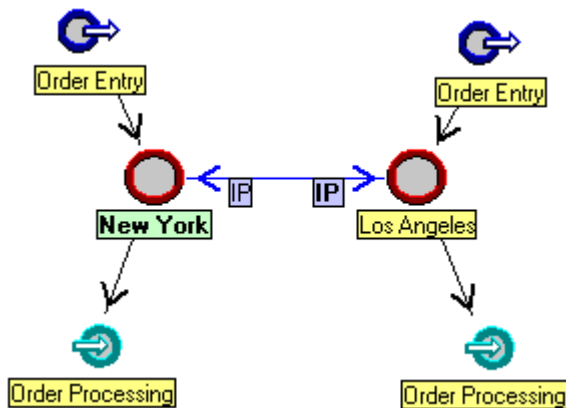
An example of a minimal Pipex network is illustrated below:



Pipex networks usually contain multiple nodes. [Routes](#)  must be set up between nodes that define how messages are forwarded from one node to another. Routes are handled by programs called **Transports**. Pipex comes with a standard TCP/IP-based transport that allows forwarding messages through an IP (Internet Protocol) network. Routes must be defined both ways.

The [Logical Node](#)  object provides a pseudonym for either a single Pipex node or a group of nodes.

An example of a more complex network is illustrated below:

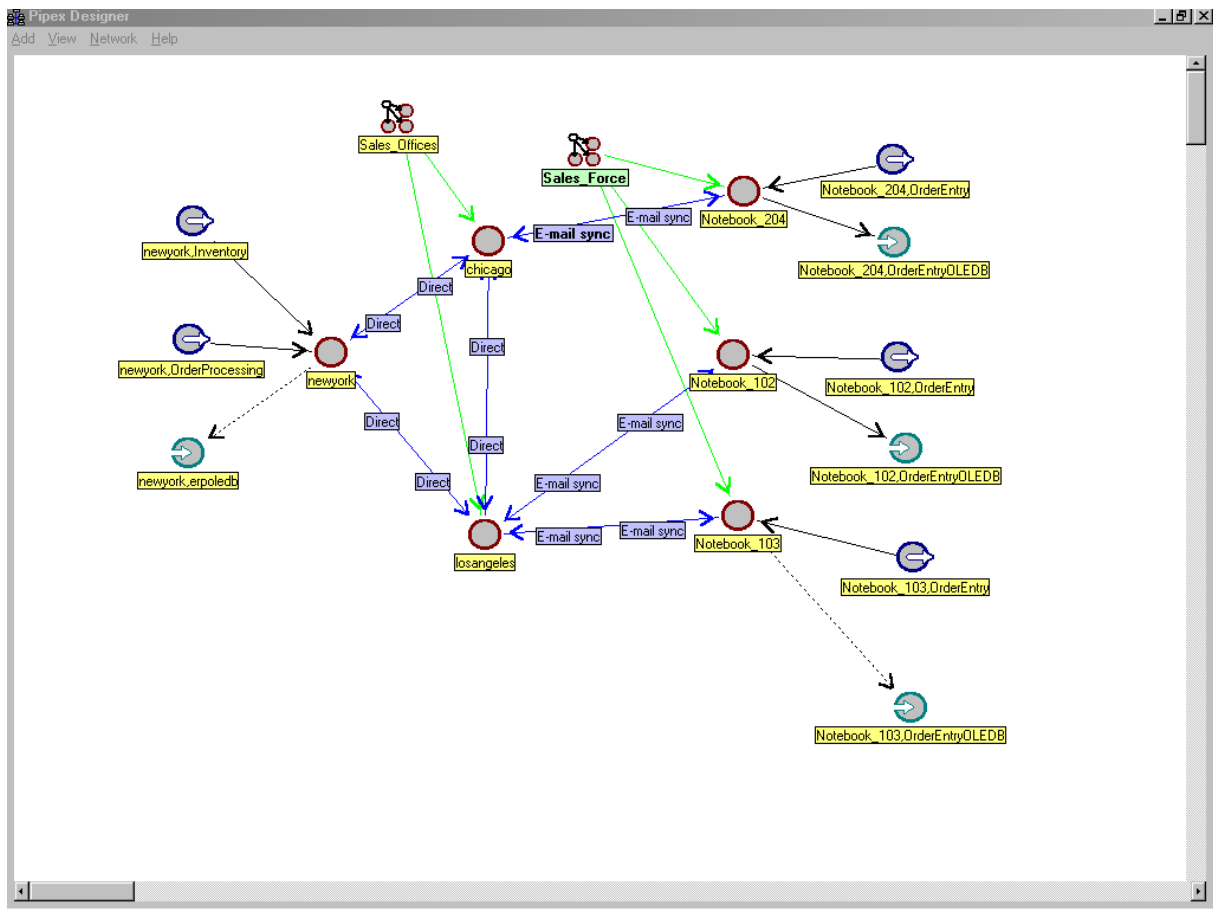


Pipex networks are created using [Pipex Network Designer](#) .

### 2.1.2 Pipex Designer example

Here is an example of a Pipex network. Don't be concerned if you can't see it all in your help system. This is provided for an overall impression, and the following sections will break down the diagram as we introduce each new topic.

The organization pictured has a Head Office in New York, with branch offices in Chicago and L.A. Each of these locations is a single Pipex node, although all have an extensive local area network. A small geographically dispersed sales force of three sales people each have their own notebook. In our example, the sales people connect to the head office using e-mail to send in new quotes and orders. The sales force also receives pricing, product and customer updates using Pipex. Pipex will also support intermittent connection across a VPN to obtain updates. The main system is an ERP package that runs on a server in New York. Inventory is maintained at branch locations, but inventory records for all locations are centralized in New York. Replicas of the inventory balance records are sent to the branch locations as the central database is updated. Thus each location has access to inventory balance information for all locations. Of course, there are other ways to handle this problem depending on the situation. Pipex provides full flexibility and control over how replicas are managed.



### 2.1.3 Node

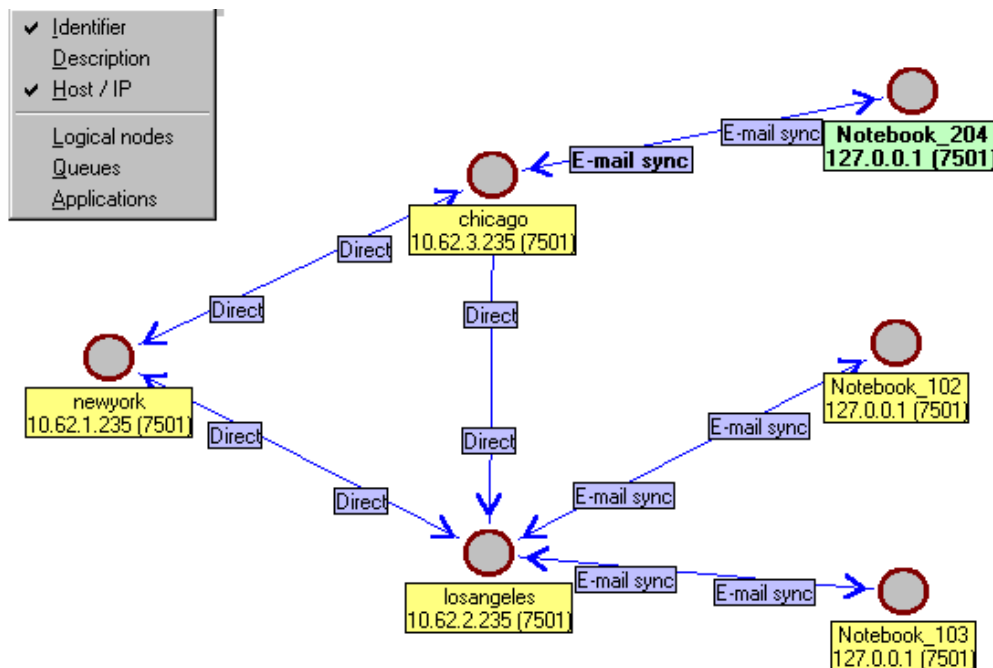
Each computer that sends or receives Pipex messages is termed a **node**. Each node must run a small executive program called **pipex.exe** (Windows environment) which handles all message passing and queuing required. The Pipex server does not process the messages itself, but passes messages on to other nodes or to queue handlers.

Each node must be assigned a **unique name**, and if running on an IP network, a dedicated IP address or a resolvable host name.

The diagram below is a partial view of our Pipex network example, showing only nodes and routes (or transports). Nodes have been defined for a main office in New York, and branch offices in Chicago and Los Angeles. Each of these nodes handles an entire LAN. As well, nodes are defined for each salesperson's notebook. These notebooks are disconnected from the LAN so each is a distinct node on the Pipex network.

The diagram also shows how nodes are connected through **routes**, which are described in the Node Routing section.

*The diagram below is an actual screen display created using the Pipex designer. The top left corner indicates the View menu settings in effect for the display.*



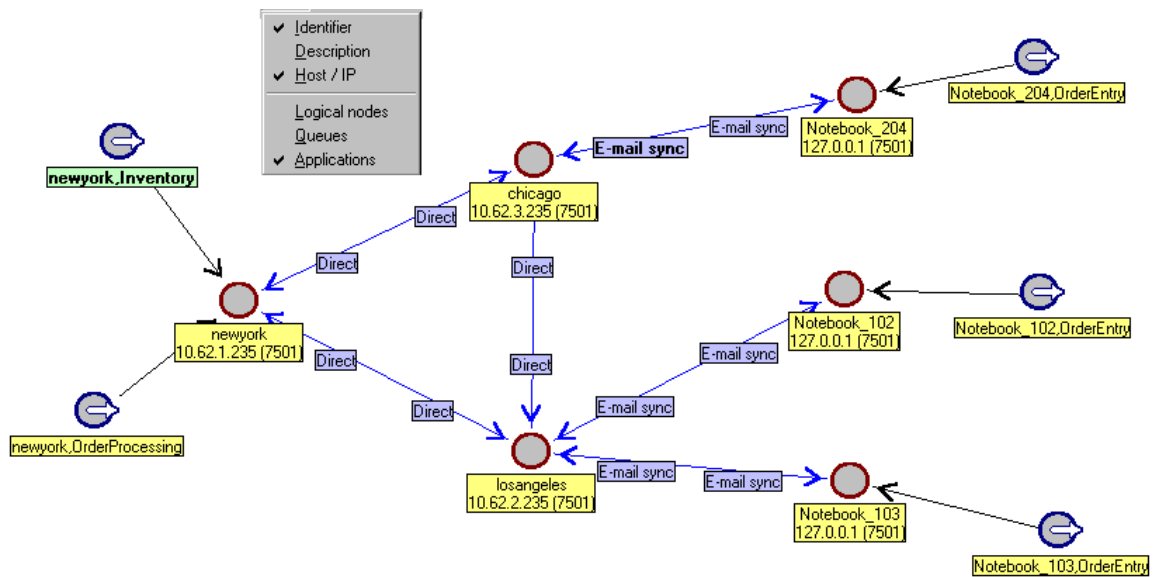
## 2.1.4 Application

Messages originate in an application program written in a standard programming language and are presented to the Pipex server through a programmer's API. The interface can be established through an ActiveX control (OCX).

The ActiveX control can communicate with a Pipex server running on another computer, thus a given local area network needs to run only one Pipex server.

Messages consist of an address and body. The message address includes the name of the recipient node, and the name of a queue on the recipient node. The message body can be any sequence of binary characters. Its content is governed by convention between the application originator and message handler. In many applications this property relieves the application program from having to translate to a complex universal data format such as XML.

The example network now shows both nodes and applications. Messages can originate from the Inventory and Order Processing applications run in New York. These messages could be transactions containing customer, product, pricing and inventory balance information for the remote salespersons' notebook computers. Messages can also originate from the Order Entry applications run on the salespersons' notebooks. These messages could be new orders or quotations to be processed by the main system in New York.



## 2.1.5 Queue

The address portion of a message sent through the Pipex network must include both a node name and a queue name. The queue is associated with a message handler process. Message handler software to update OLE DB-compliant databases is provided "out of the box" with Pipex. Thus application programmers can send database updates to any remote database which supports OLE DB, including Microsoft Access, ORACLE, and Microsoft SQL Server. Third party and optional message handlers to update non-OLE DB databases such as IBM Universe are also available. Such handlers typically also provide a field mapping capability.

Programmers can add message handlers that function at a level higher than database maintenance. For example, a queue handler can process inventory transactions that updates inventory balances, recalculates running totals, updates a transaction history, and so on.

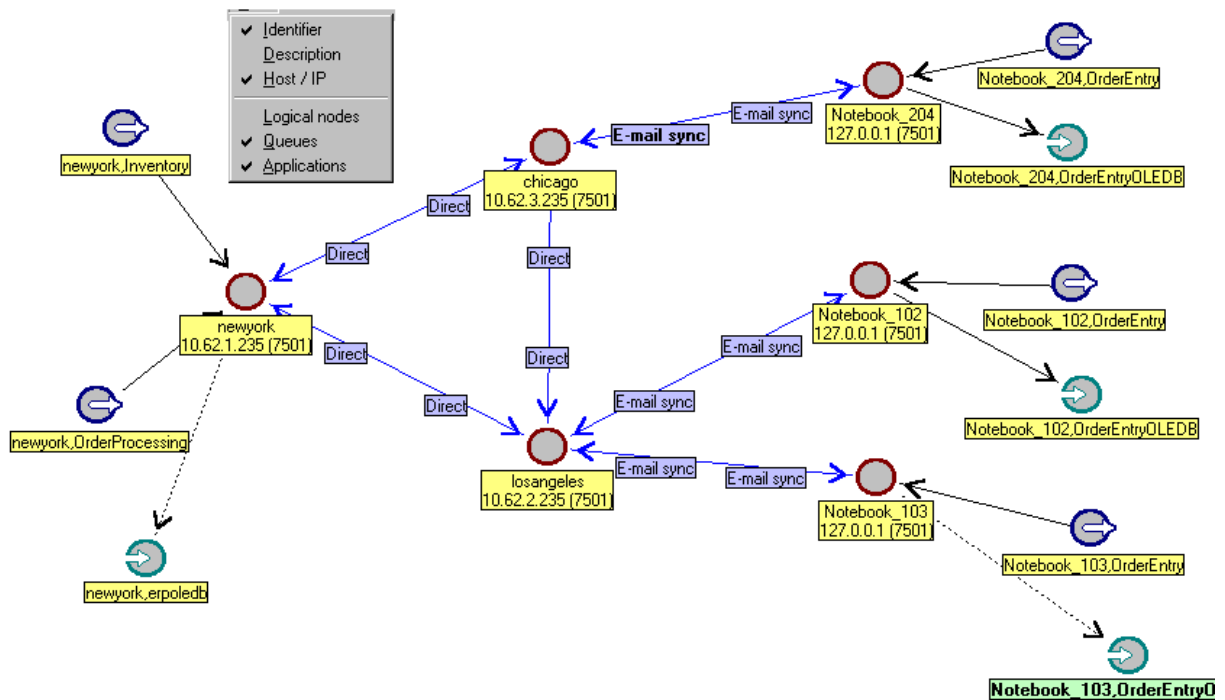
Message handlers can be run:

- **By a user or on a scheduled basis**, in which case the Pipex executive queues messages until contacted by the message handler.
- **On demand.** The Pipex server can wake up the message handler whenever there are messages to process.

Whatever way the message handler is started, it can either quit immediately after processing available messages or it can stay connected and sleep until more messages are available. The Pipex API provides functions that allow polling the Pipex server with minimal CPU utilization.

The example network now shows nodes, applications, and queues. Queues have been defined to handle Order Processing updates to the ERP system in New York, and to handle pricing, customer and product updates on each salesperson's notebook.

Note that adding another notebook to the network is a simple matter. The Pipex designer provides functions to copy an existing node to a new one, including any applications and queues.



## 2.1.6 Node Routing

The node routing specification determines how messages are passed from node to node. Out of the box two mechanisms or **transports** are provided for connecting nodes. These are: **IP message passing** (direct) and **file transport**. IP message passing makes possible message transfer between any two nodes on an IP network. The two nodes need not be simultaneously active; the originating node will hold or queue messages for the recipient.

To minimize the number of connections on the network, every node need not be connected to every other node. As long as a path exists through intermediate nodes, Pipex will determine the best path for directing messages.

The file transport is used to create a message file for presentation to an off-line node. A utility (pipemapi) is provided (open-source) to take the file of messages and create an e-mail attachment. Another utility (pipebriefcase) allows a file of outstanding messages to be copied to removable storage media. Note that Pipex keeps a separate file of messages for each destination node supported by the file transport.

The resulting e-mail attachment or message file must be presented to the Pipex server at the destination node. This could be as simple as having the user double-click the e-mail attachment OR inserting a floppy disk and running a shortcut on the desktop. The underlying process is actually a file transport program (supplied with Pipex), which connects with the Pipex server and presents the new transactions.

**(Disclaimer:** while version 4.1 supports the file transport feature, the implementation details are not well-documented.)

Pipex will ensure that messages directed to it from the file transport are processed in the exact sequence created. If the user at the remote end omits an e-mail or removable disk (and this will happen) Pipex will refuse to process the messages, and provide the user with status information. Further, Pipex safeguards the integrity of system data by retaining all messages at the source end until an acknowledgement is received from the target queue handler that messages have been properly processed. A necessary implication is that the 'file transport' must move data in both directions between two nodes.

A recovery option is provided to re-create all messages that have not been acknowledged from a target node back to a sender node. This would be used in the event that a disk or e-mail

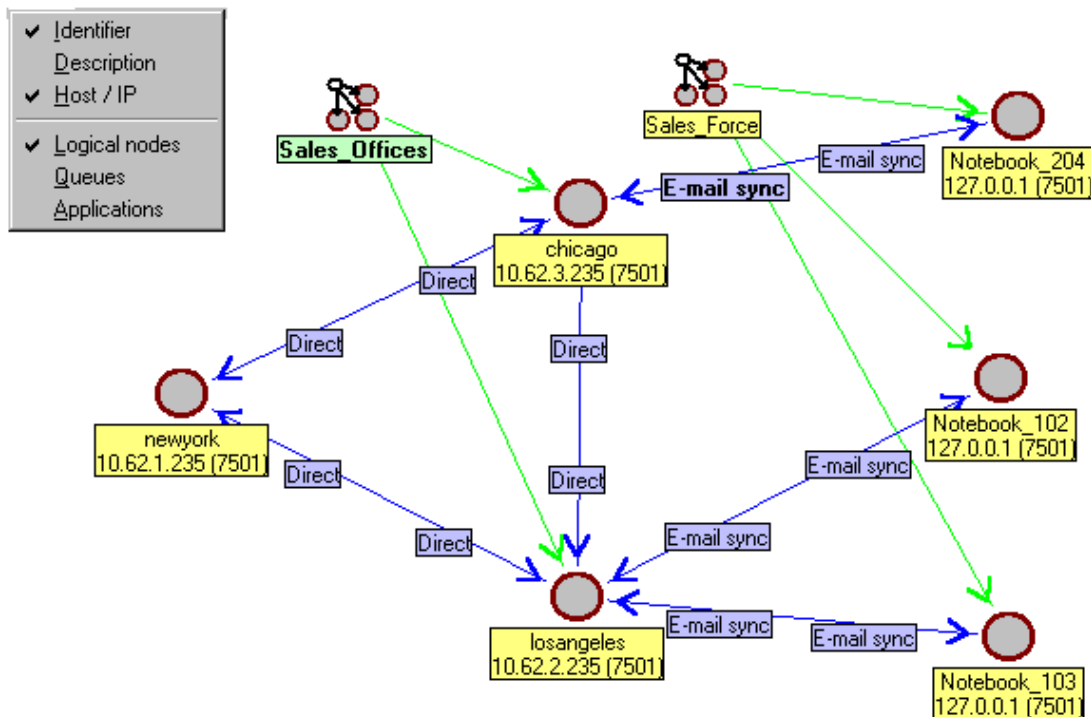
attachment has been lost. Finally, note that Pipex will just ignore messages presented to it that have been previously processed.

The current version of Pipex does not provide a mechanism for 'broadcasting' updates to a variety of users (publish and subscribe). Each node on the Pipex network must be defined and known.

## 2.1.7 Logical Nodes

Logical nodes are an essential concept used to isolate the application code from the construction of the Pipex network. A logical node provides a pseudonym for either a single Pipex node or a group of nodes.

In the example, the Order Processing application can direct messages to a logical node called "Sales\_Force". The Pipex message server will send the message to each node referenced by the logical node definition, that is to "Notebook\_102", "Notebook\_103", "Notebook\_204", and so on. Nodes for salespeople can be added or removed from the network without having to change application programming.



## 2.2 Design Principles

### 2.2.1 Queuing

The goal of the Pipex network is to guarantee delivery of messages passed to it, even if the intended recipient is not active on the network. This requires that Pipex 'store' messages until the intended recipient activates. It also means that there is a degree of latency between the sending of message and processing by the recipient. While messages sent to a single queue will always be processed in the order sent, messages sent across multiple queues may be processed out of sequence. This should be considered in designing higher level application

queue handlers. It's best to handle all transactions for a set of related files through one queue handler.

## 2.2.2 Serialization

All messages passed across the network are identified with a serial number. This includes messages passed to Pipex from an application, messages passed from one node to another, and messages passed to a queue handler. If a Pipex server or handler detects a gap in the sequence, an error will be logged and that node or queue handler will be shut down until the reason is determined. The most common reason for serial number failure is that a Pipex file has been restored from backup, potentially violating the integrity of the system, especially in a data replication scenario. Pipex does provide administrator level functions to reset serial counters and restart the related processes.

## 2.2.3 Acknowledgement

Pipex will re-send messages where transports are not totally reliable, as for example, on a serial port. Pipex will not delete a message until the destination queue handler acknowledges that the message has been reliably processed. If the queue handler posts an error in processing the message, for example, for an access violation, then on restart Pipex will automatically re-send the most recent unacknowledged message to the queue handler.

The acknowledgement process is central to maintaining the integrity of the file transfer mechanism, and any other disconnected routing mechanism. Further details are discussed in the [Node Routing](#) section.

## 2.2.4 Process Independence

The main Pipex server on a node typically runs unattended all the time. If a node's server process is disconnected from the Pipex network or the server goes down, the remainder of the Pipex network continues unimpaired. Messages intended for the downed process will be held until the node server process re-connects to the Pipex network.

Each Pipex server provides a status display of the nodes and queues with which it is maintaining a connection. If a connection goes down, the problem is logged in the central event log.

Queue handlers can be started on demand by the Pipex server, or started externally by a scheduler program.

## 2.2.5 Error Handling

All errors are logged centrally, and an event log viewer is provided. An API is provided so that queue handlers added to Pipex can make use of this feature.

Pipex is very robust and detects over 100 different possible errors. The central principle is that if a message cannot be processed by a queue handler, an error will be logged and that queue will be shut down until the problem is remedied by network support. A queue handler is re-enabled through the Pipex server console. The queue handler will attempt to re-process the message that caused the error condition. If the underlying condition (e.g. permission, database error) has been remedied, the Pipex process continues. The network administrator can also 'flush' the offending message, and continue with the next.

## 2.2.6 Security

Whether deployed on a wide area network, disparate PCs, or on a VPN, Pipex is a closed restricted-user network. Access to the Pipex network is password-restricted if run on the open Internet, but traffic is not encrypted. A VPN based deployment is quite secure. Pipex also securely supports disparate PCs which connect intermittently to a LAN or a central server using dial-up or Internet based connections to synchronize files.

Password keys are set and controlled by the network designer using the Pipex designer program. When a new node is deployed, the deployment module writes unique password keys into the registry of the PC running the node software. The password keys are presently supplied by the network designer; a future release will allow automatic generation of strong passwords. A Pipex server process will only accept messages from processes which can supply an authorized password key. The registry and configuration file where passwords are stored must be protected using the security mechanisms offered by the operating system, whether Windows or UNIX. This would involve setting file permissions and user log ins to restrict access to registry and configuration files to authorized users and developers.

Like all security systems, Pipex security relies on proper management of logins and passwords. If users share passwords, use weak passwords, or never change passwords, security can be breached more easily. The potential impact of a security breach is limited to the methods that a queue handler exposes to the Pipex network. If using general purpose handlers, such as OLE DB, the designer is able to restrict access to certain tables. The use of special purpose transaction handlers can limit or nullify the potential damage of compromised security. When deploying Pipex across the Internet a common approach is to allow Internet access only to public replicas of a private database. Pipex can also be used to synchronize replicas of the private database.

## 2.2.7 Deployment

The underlying design of the Pipex network is apparent by examining the file system. All network configuration information is stored in one configuration file (pipexconfig.mdb under Windows) which must reside locally for each Pipex process. The network designer maintains the original Pipex configuration file on her computer. A folder is kept for each node on the network with the respective configuration files for each node. This configuration file must be copied to each node whenever the network configuration is changed. Future Pipex versions will replicate the configuration information automatically. Until this feature is added, Pipex administration may not be practical for large-scale networks containing many nodes. Each PC and server accessing the Pipex network also requires registry settings to help locate node processes and to store password keys.

Pipex provides several tools to assist in network deployment. The **prepare files for deployment** function will update the network node folders on the designer's computer. Configuration files, registry setting files (.reg) are created for each individual node. The deployment wizard is used the first time the network design is copied to a new node. Subsequent network configuration changes can be copied to network nodes during a convenient network down period without use of the wizard.

Pipex configuration files do not have to remain absolutely consistent to retain network integrity. Not every node needs to know about all changes on the network. However, certain inconsistencies can create recoverable errors at run time. For example, if one node addresses a queue at another node that does not exist, an error will be logged.

## 2.3 Components of Pipex

Pipex is **message queuing** middleware that is used to integrate disparate computer processes by passing messages between them. The disparate processes may be connected across an IP network such as a local area network or the Internet, but need not be. Pipex can also pass messages by e-mail or by means of removable storage media such as a floppy disk or CD, with user interaction.

Messages are often used to move data file updates, business transactions, but could be used for other purposes. Pipex consists of the following elements:

- A message passing process, the Pipex server, ('pipex.exe') that must be reachable through IP by every computer creating Pipex messages. Typically, Pipex server will run on a network server to serve an entire LAN. Remote disconnected computers creating Pipex messages will each require a Pipex server process to run. Each Pipex server process represents a **node**.

- A set of configuration files and registry settings that describe the Pipex network. The configuration files are managed using the Pipex network designer and are stored locally for use by each **node**. The configuration files are identical. A deployment wizard is supplied to assist in creating and deploying the configuration files.
- An Activex control that can be dropped onto a VB and Access form to permit passing messages to the Pipex API. A program that creates messages is termed a message originator or simply an **application**.
- Built in OLE DB handler to respond to Pipex messages to update (add, update, delete) records in any database which supports OLE DB. A message destination is called a **queue**. The process that handles the messages in the queue is called a **queue handler**.
- An interface to Pipex which will allow the programmer to create custom queue handlers for unique data file formats, or at a higher transaction functional level. For example, a queue handler could be written to support a set of inventory transactions, such as receipts, issues and shipments. Thus a centralized inventory could be accessed from anywhere on the Pipex network, with updated balances broadcast back to locally stored databases at remote sites. This interface is supported by the Pipex Activex control.
- OLE DB support through an ActiveX DLL which assists in building OLE DB messages for transmission to the OLE DB handler through Pipex.
- Message transport processes for IP and file transfer (e-mail, removable media). Pipex passes messages from node to node through a **route** between the nodes. The route is supported by a transport process. Pipex includes transport processes for routing across IP networks and through file transfer. Intermediate nodes can be used to minimize the number of open connections between nodes on a network. Pipex will automatically determine the best routing to use in moving messages.
- A simple process scheduler, for running message originators or queue handlers unattended and automatically. Queue handlers can also be started on demand, whenever a transaction needs to be processed.

## 2.4 Supported Platforms and Technologies

Pipex version 4.1 runs under **Windows** NT, 2000 or XP, 98 and NTWS. A gateway to older versions of Pipex running under **SCO UNIX** and **Linux** is available. Out of the box, Pipex queue handlers are provided for all **OLE DB** and ODBC databases including **Oracle** and **MS SQL Server**. Optional queue handlers and an application sender interface is provided for Multi-value databases including ULTI-PLUS and **IBM Universe**.

## 3 Creating a Pipex network

### 3.1 Contents

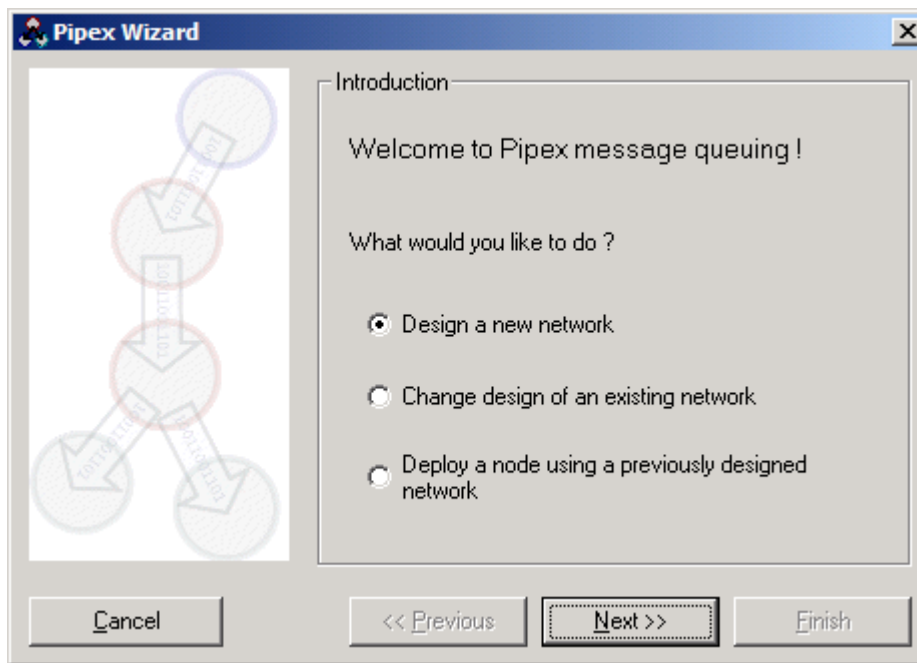
Creating a working Pipex network involves three main stages:

- Design
  - [Components of a Pipex network](#)
  - [Designing a network using the Pipex Network Designer](#)
- Deployment
  - Preparing files for deployment, copying configuration files
  - [Deploying nodes using the Pipex Wizard](#)
  - Configuring client computers

- Running  
Changing the design of a running network

## 3.2 Starting Pipex Designer

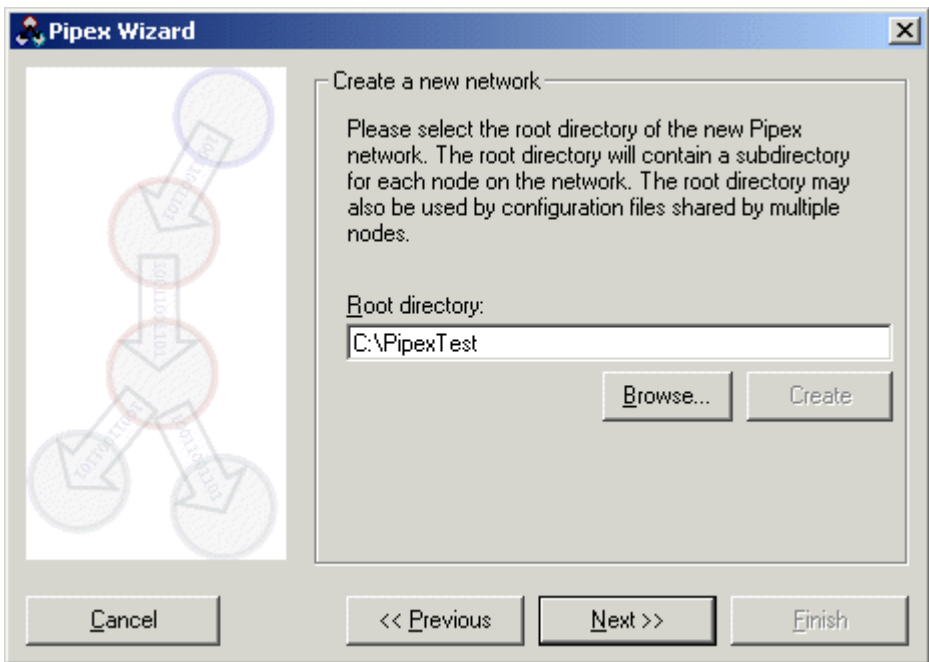
When Pipex is installed, a shortcut to **Pipex Wizard** is created in the Start menu.



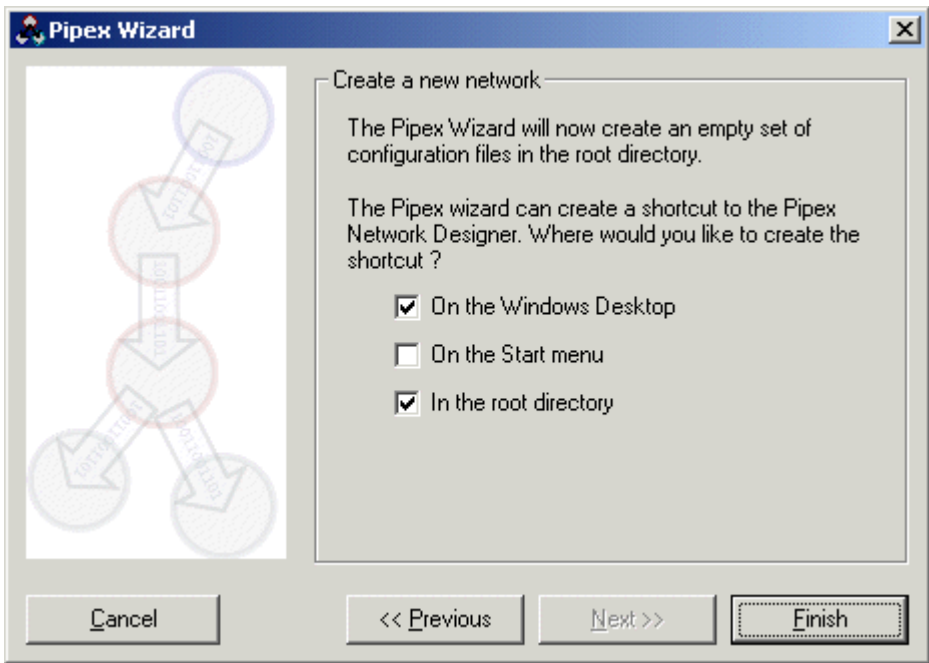
- Use "**Design a new network**" to create a completely new network design on your computer. The Pipex Wizard will create a new empty design for you in the selected subdirectory, and a shortcut to the [Pipex Network Designer](#) .
- Use "**Change design of an existing network**" to recall an existing network design and make changes.
- Use "**Deploy a node using a previously designed network**" if you have already created a network and you are about to deploy one of the nodes. DO NOT use this wizard if you already have the node running on this computer, unless you are deploying a new, different node.

### Creating a new network

Each Pipex network design that you create must be located in its own directory. This is called the 'Root directory' for your Pipex network. Each node created will have its own subdirectory in the root directory.



The root directory may also be used by configuration files that are shared between nodes. Press **Create** to create the root directory if it does not exist.

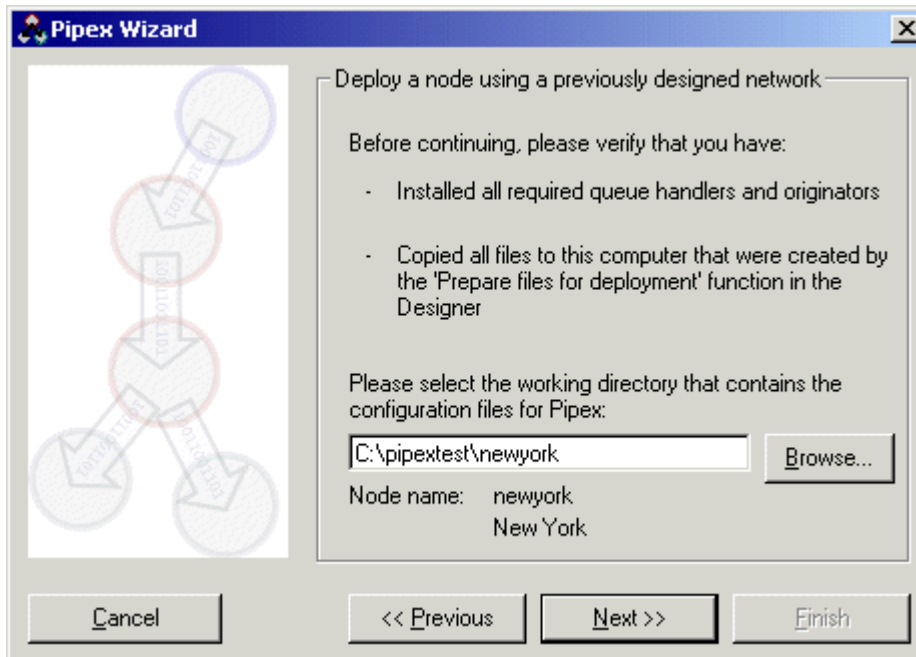


Press **Finish** to create a new empty network design and to create a shortcut to the Pipex Network designer.

**Deploying a node**

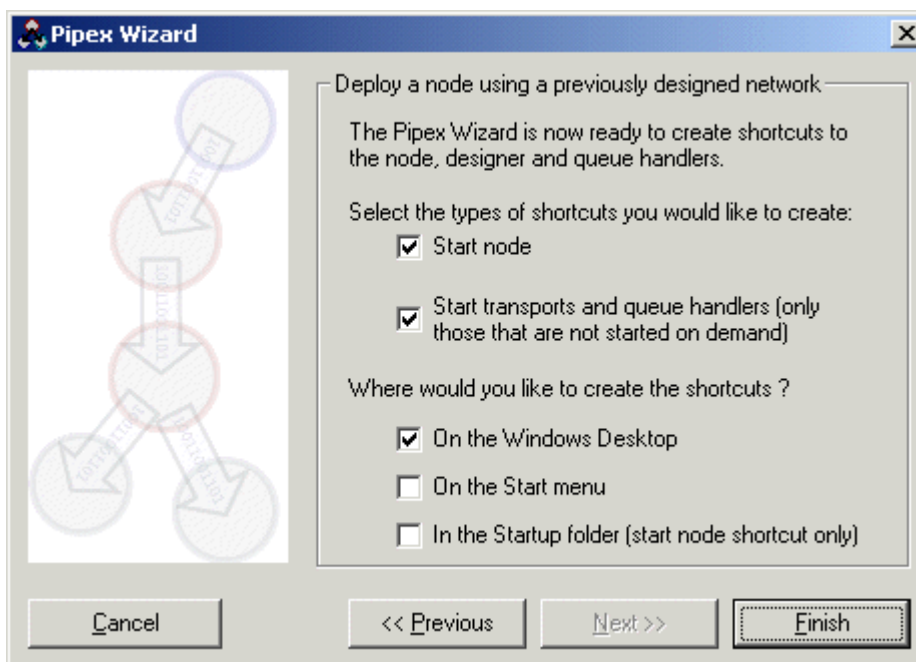
Once the network design is complete, the configuration files are prepared by selecting

'Utilities' / 'Prepare files for deployment' from Pipex Network Designer. After preparation, step by step instructions are displayed on how to deploy each node. These instructions include installing Pipex and copying configuration files to computers. After copying the configuration files, Pipex Wizard can create the shortcuts needed to start the Pipex server:



First specify where the configuration files have been copied. As soon as a valid directory is selected, 'Node name :' appears and the 'Next' button becomes available.

Then specify where you want to create shortcuts:

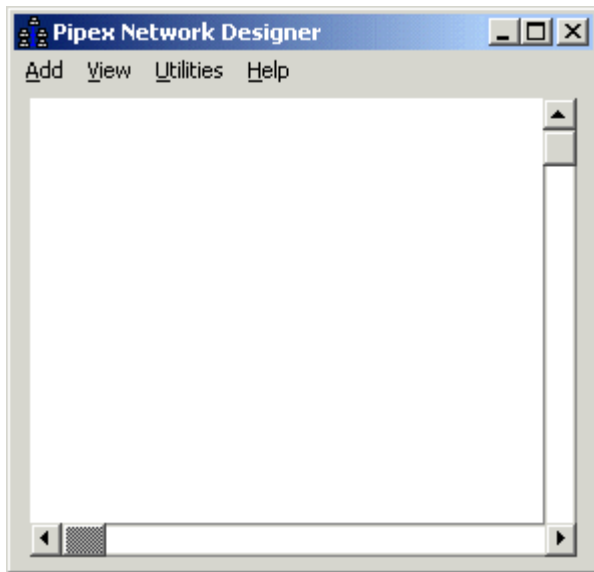


If the 'Startup' folder is specified, the Pipex server will be started as soon as the user logs on

to the computer.

### 3.3 Pipex Designer

The Pipex Network Designer is used to create [all components](#) of the Pipex network. It can also be used to configure applications, queue handlers and transports. Once a Pipex network is designed, it can prepare configuration files for deployment.



**Add menu:** Create nodes and logical nodes or close the Pipex Network Designer.

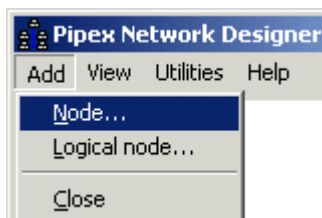
**View menu:** Specify components of the network that are shown. Use this menu to hide or show logical nodes, queues or applications. You hide or display object identifiers or description. You can show or hide IP addresses or host names of nodes.

**Utilities:** Prepare files for deployment or run the Pipex API Setup. The Pipex API Setup can be used to configure applications that run on your machine.

There is no **'Save'** function in the Pipex Network Designer. Any time a change is made, it is automatically saved.

#### 3.3.1 Creating a node

Add a new node by selecting **Node** from the **Add menu**



Enter properties of the new node:

**Name:** Identifier of the new node. Applications will refer to this name in the program code. The node identifier must not contain any special characters, but it may contain spaces or underscore (\_) characters.

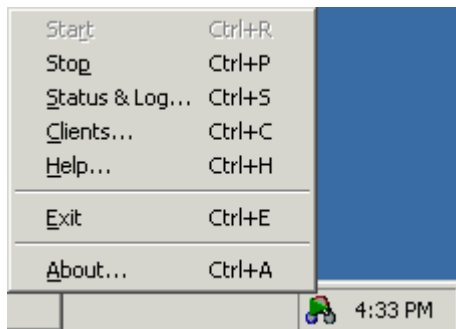
**Description:** Description of the node. This description will appear in the designer.

**Hostname / IP:** Host-name or IP address of the computer that will be running the Pipex Server program.

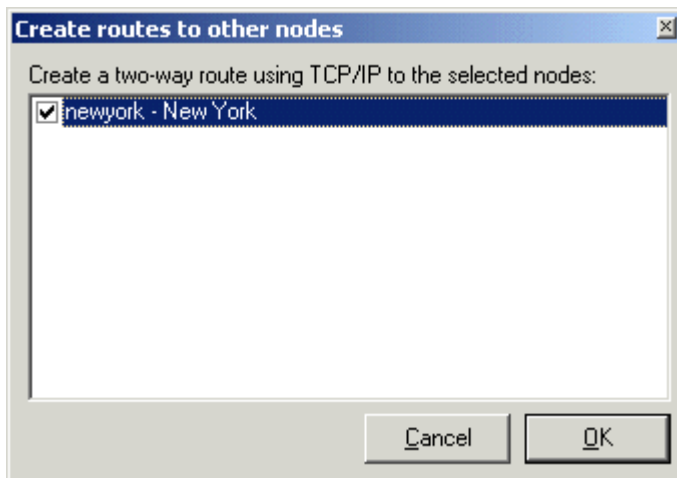
**Port:** TCP Port number to listen on. When multiple servers run on a single computer (such as a test environment), unique port numbers must be specified for each server.

**Password:** Password that a transport must provide to be able forward messages to or from this node.

**Error handler node:** By default, all error messages are kept on the node where they were generated. Select a different node here if you want all error messages to be routed there. You can display error messages and events by right-clicking on the Pipex icon on the system tray and selecting *Status & Log*:



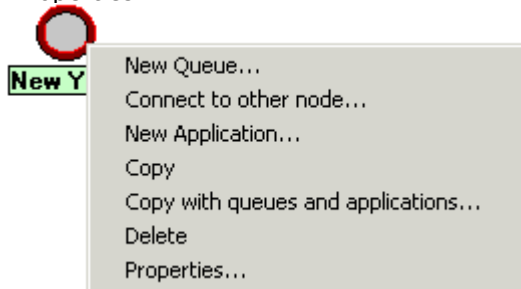
If other nodes already exist on the network, Pipex will offer to create a two-way TCP/IP based route to the other nodes:



Select Cancel if you do not want to create any two-way routes at this time. Routes can later be created by right-clicking on the node and selecting 'Connect to other node'. If routes are set up this way, please remember that a route **must** be created in both directions.

### 3.3.2 Changing or deleting a node

A node can be changed or deleted by right-clicking on the icon of the node and selecting 'Properties':



Select 'Properties' to change the node or select 'Delete' delete the node. When a node is deleted, all related applications, queues, routes or node group entries will automatically be deleted.

The node identifier may not be changed on an existing node. You must delete and re-create the node to assign a different identifier.

### 3.3.3 Creating a queue

A queue is always associated with a node. Right-click on the icon of a node and select 'New Queue':

**Node:** Identifier of the node where the new queue is created.

**Queue Name:** Queue identifier. Applications will refer to this identifier in the program code. The queue identifier must not contain any special characters, but it may contain spaces or underscore ( \_ ) characters.

**Description:** Description of the queue. This description will appear in the designer.

**Password:** Password that the queue handler program must provide to download and process messages.

**Start queue handler on demand:** Check this option if the queue handler will be started by the Pipex server. If this option is checked, 'Queue handler executable' must also be specified.

**Command-line parameters:** This is not an editable field. It displays the command-line parameters that will be passed to the queue handler when started by the Pipex server. This field can be used by programmers who wish to debug a queue handler. It may also be used if the queue handler is executed on another computer or by an external task scheduler.

**Queue handler executable:** Specifies the location and name of the queue handler executable file. It can either be a standard queue handler shipped with Pipex or a custom queue handler developed in Visual Basic. For information on how to create a custom queue handler, read ['Creating your own queue handler'](#) .

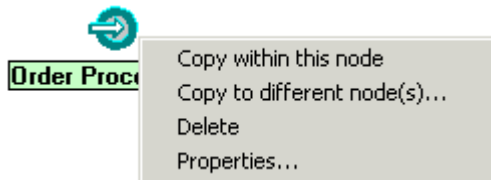
**Configuration file:** Configuration file to be used by the queue handler. Consult the queue handler documentation for information on the format and location of configuration files. Some queue handlers may not require a configuration. Others can create the files automatically (such as [OLE-DB Queue Handler](#) ), when first started by pressing the 'Build' button at 'Queue handler parameters'.

**Queue handler parameters:** Consult the queue handler documentation for information on

supported parameters. Press 'Build' to configure the queue handler by executing it in 'configuration' mode.

### 3.3.4 Changing, deleting, copying a queue

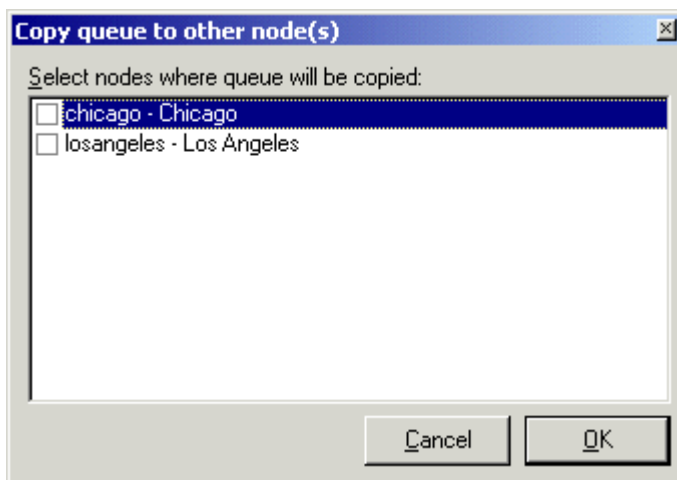
A queue can be changed by right-clicking on the icon of the queue and selecting 'Properties':



Select 'Delete' to delete the queue. The configuration file associated with this queue will not be deleted.

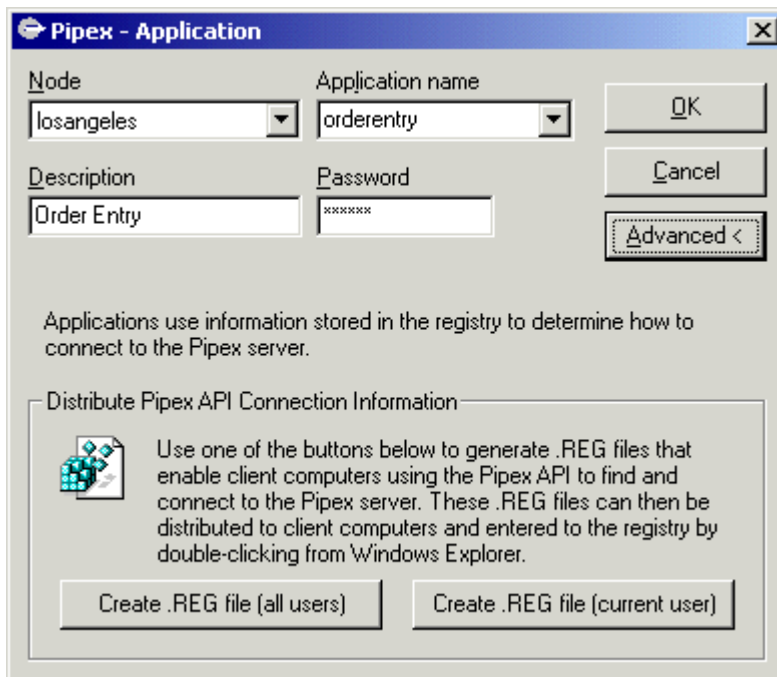
Select 'Copy within this node' to create a copy of the queue on the same node. The program will prompt for a new identifier.

Select 'Copy to different node(s)...' to copy the queue to other nodes. The program will prompt for a list of nodes to copy to. All properties, including the identifier will be copied.



### 3.3.5 Creating an application

A new application can be added by right-clicking on a node, and selecting 'New Application':



**Node:** Identifier of the node where the new application is created.

**Application Name:** Application identifier. Applications refer to this identifier in the program code when calling the `Initialize()` method. The application identifier must not contain any special characters, but it may contain spaces or underscore (`_`) characters.

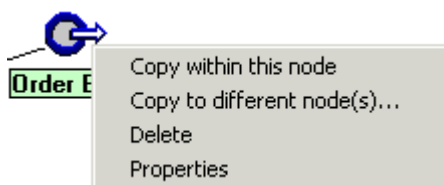
**Description:** Description of the application. This description will appear in the designer.

**Password:** Password that the application program must provide to be able to send messages through Pipex.

**Create .REG file:** Create .REG file that when entered into the registry, will allow an application to determine how to connect to the Pipex server. The registry entries can either apply to all users or only to the current user.

### 3.3.6 Changing, deleting, copying an application

An application can be changed by right-clicking on the icon of the application and selecting 'Properties':

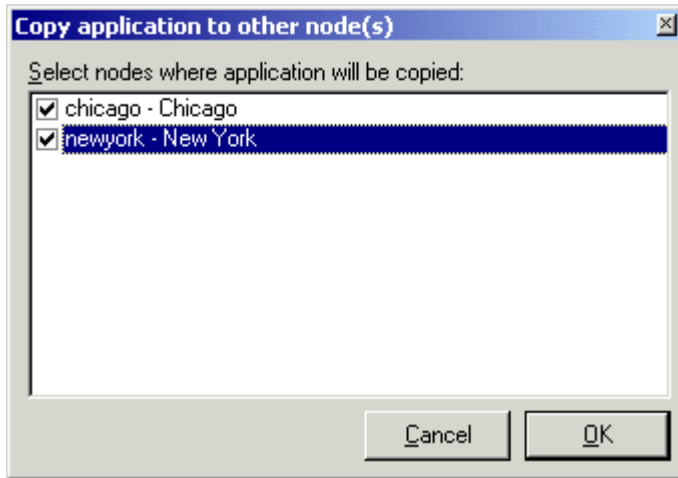


Select 'Delete' to delete the application.

Select 'Copy within this node' to create a copy of the application on the same node. The program will prompt for a new identifier.

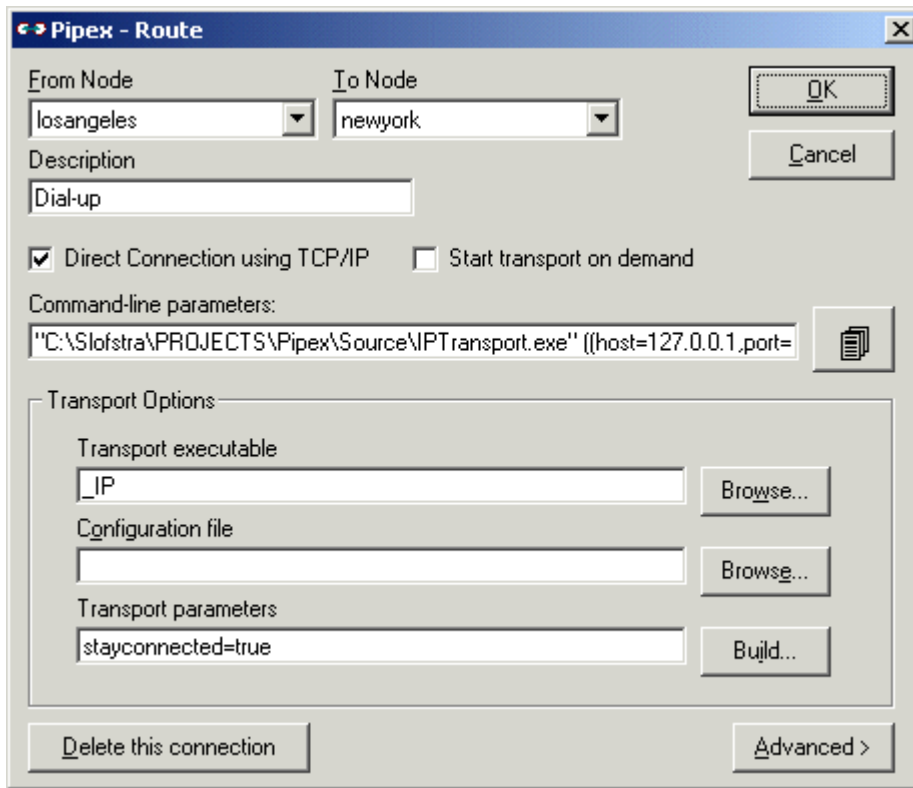
Select 'Copy to different node(s)...' to copy the application to other nodes. The program will

prompt for a list of nodes to copy to. All properties, including the identifier will be copied.



### 3.3.7 Creating, changing, deleting a route

Whenever a **new node is created**, the Pipex Network Designer offers to automatically create a two-way IP-based route to any of the existing nodes, handled by the IP transport that is shipped with Pipex. To use another type of transport or to override the default parameters, right-click on the node and select '**Connect to other node**'. Change 'To Node' to change the destination node.



**From Node, To Node:** Endpoints of route to be created. Remember that a route must be

created to both directions.

**Description:** Description of the route or transport. This description will appear in the designer.

**Direct Connection using TCP/IP:** Use the default TCP/IP-based transport installed with Pipex (IPTransport.exe).

**Start transport on demand:** Check this option if the transport will be started by the Pipex server. If this option is checked, 'Transport executable' must also be specified.

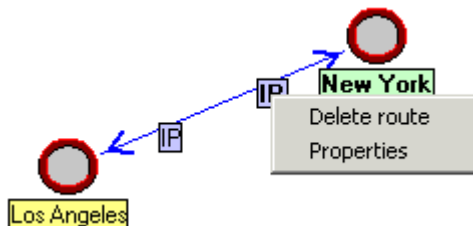
**Command-line parameters:** This is not an editable field. It displays the command-line parameters that will be passed to the transport when started by the Pipex server. This field can be used by programmers who wish to debug a transport. It may also be used if the transport is executed on another computer or by an external task scheduler.

**Transport executable:** Specifies the location and name of the transport executable file. It can either be a standard transport shipped with Pipex or a custom transport developed in Visual Basic. (Contact SSI for further information on creating your own transport.)

**Configuration file:** Configuration file to be used by the transport. Consult the transport documentation for information on the format and location of configuration files. Some transports may not require a configuration file, while others can create the files automatically when first started by pressing the 'Build' button at 'Transport parameters'.

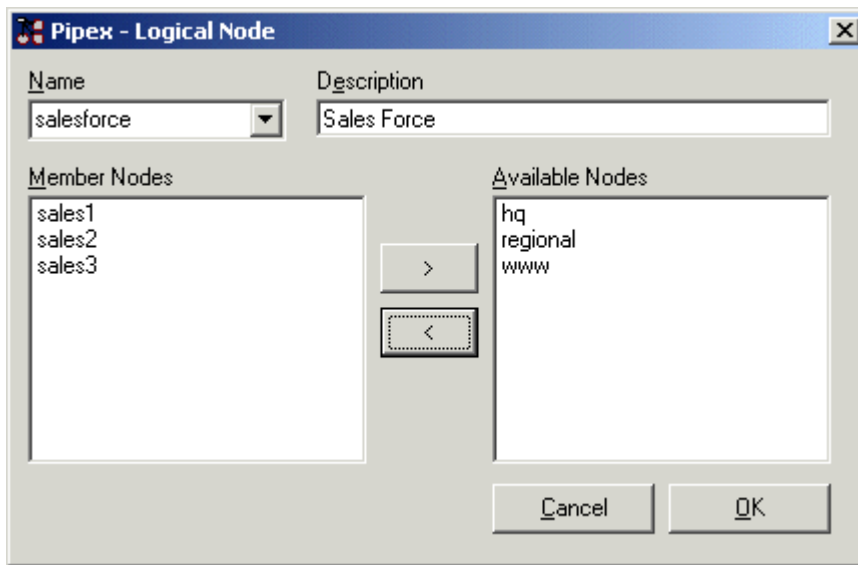
**Transport parameters:** Press 'Build' to configure the transport by executing it in 'configuration' mode. The parameter "stayconnected=true" (for IP transport) will keep the transport running even when there are no messages to process.

You can **change** or **delete** a route by right-clicking the label on the lines, then selecting **Delete** or **Properties**:



### 3.3.8 Creating, changing, deleting a logical node

Select **'Add', 'Logical Node'** from the Pipex Network Designer menu.



**Name:** Identifier of the new logical node. Applications will refer to this name in the program code. The logical node identifier must not contain any special characters, but it may contain spaces or underscore (\_) characters.

**Description:** Description of the logical node. This description will appear in the designer.

Use the < and > buttons to add and remove member nodes.

See also [Concepts - Logical Nodes](#) .

### 3.3.9 Preparing files for deployment

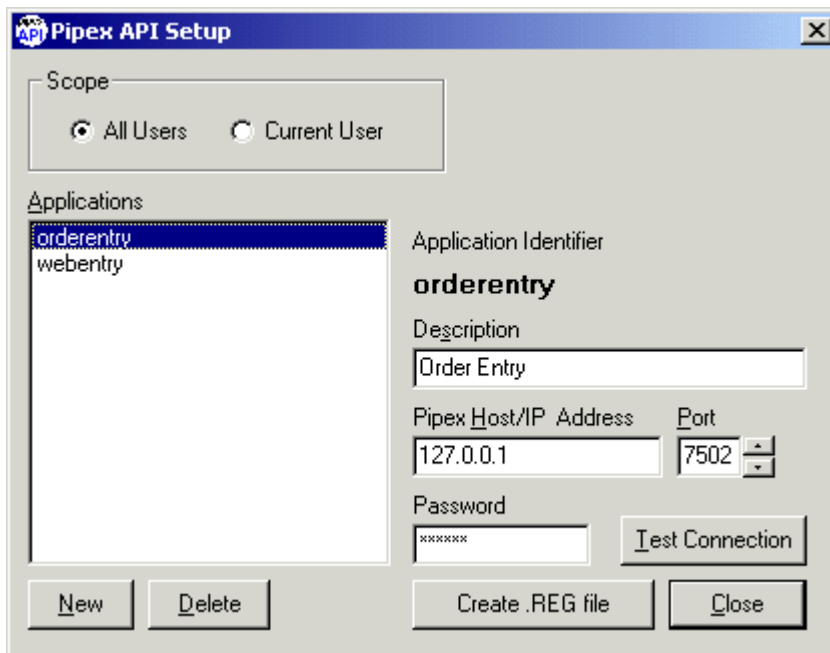
Once a network design is complete, configuration files must be prepared for deployment. Select **Utilities, Prepare Files for Deployment** from the Pipex Network Designer menu.

This function will create a copy of the current configuration file for each node and two .REG (registry) files for each application on each node.

After being copied to the destination computer, these files are used by the Pipex Wizard to deploy a node. The .REG files are used on computers running applications.

## 3.4 Pipex API Setup

The Pipex API Setup is a small utility that allows viewing, maintaining and testing connection information in the registry used by [Applications](#) .



**Scope:** Select 'All Users' to maintain registry entries that apply to all users or 'Current User' to edit entries applying to the current user. The current user is the user logged on to the computer.

If there are entries with the same identifier in both scopes, the entry in 'Current User' will override the entry in 'All Users'.

**New:** Create a new application entry.

**Delete:** Delete application entry.

**Test Connection:** Test connection to Pipex server.

**Create .REG file:** Save connection information into a .REG file. This registry file can be used to copy connection information to another computer.

**Description:** Optional description of the application.

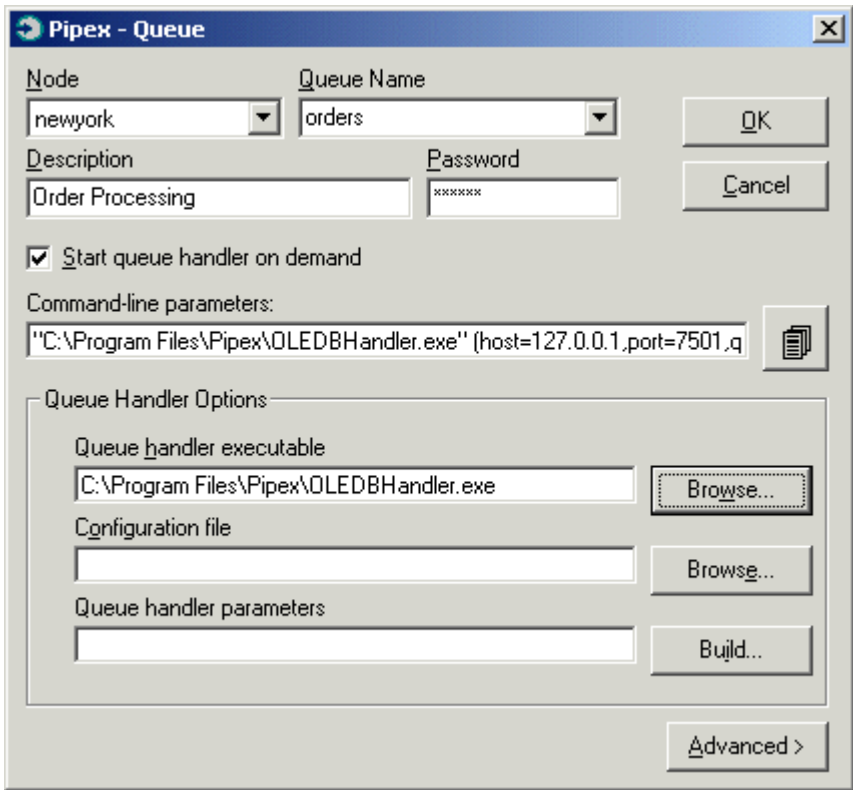
**Pipex Host/IP Address:** Host name or IP address of the computer where the Pipex server is running.

**Port:** TCP Port number used by the Pipex server.

**Password:** Application password entered here must be the same as what is defined in the [Pipex Network Designer](#).

### 3.5 Pipex Queue Handler for OLE DB

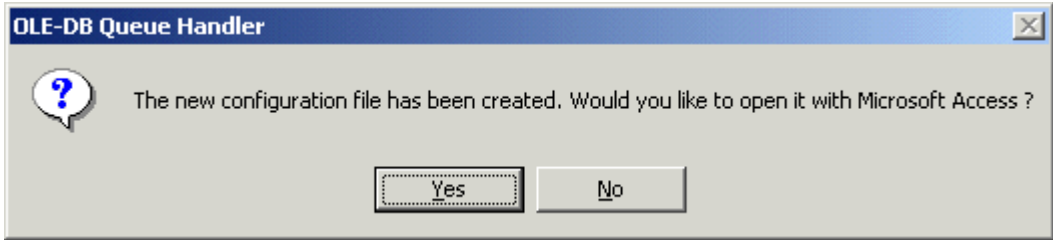
This is the standard properties box for queue handlers as displayed in the Pipex designer.



The Pipex OLE DB queue handler is invoked by specifying OLEDBHandler.exe executable as the "Queue handler executable". This is best accomplished by pressing "Browse..." and locating the 'queue handler executable' in your Program Files directory.

Pressing "Build ..." will set up an OLE DB configuration file for this specific queue in the Pipex design directory. The "build" function will prompt for a file name and place this file in the configuration directory for the queue. It is suggested that by convention the configuration file name contain the letters 'oledb'.

The Pipex designer will permit you to open the configuration file using Microsoft Access:



Within Microsoft Access, a form is provided to allow access to each table to be updated by Pipex messages.

TABLES : Table							
TableName	TargetType	Server	Database	Username	Password	TargetTable	
OrderHeaders	0		C:\Program Files\Pipex\Tutorial\OrderProcessing.mdb				
OrderLines	0		C:\Program Files\Pipex\Tutorial\OrderProcessing.mdb				
*	0						

Record: 1 of 2

**TableName:** The table name that will be used in all messages sent to the queue handler.

**TargetType:** 0 for MS Access, 1 for SQL Server

**Server:** Server name (for SQL Server only)

**Database:** Path name and file name for an MDB, data base name for SQL Server

**Username:** For SQL Server only

**Password:**

**TargetTable:** Actual table name in MS Access or SQL Server

Later, when you wish to make changes to the OLE DB configuration table, press "Build .." in the properties form of the relevant queue handler, and this form will appear.

**Stay connected to Pipex:** If checked, the queue handler task will not shut down after completing outstanding transaction. It will wait for more transactions.

**Create from template:** This can be used to create a new OLE DB configuration file. This function will prompt you for a new file name.

## 3.6 Deployment

### Install Pipex

Run the Pipex setup program on each computer.

### Install queue handlers and originators

It is recommended that additional queue handlers be installed in the Pipex program file directory (usually C:\Program Files\Pipex).

#### Copy files from Pipex root directory

The Pipex root directory may contain configuration files that are shared by multiple nodes.

#### Copy files - subdirectories for each node

A subdirectory is created when a new node is added in the designer. When files are prepared for deployment, various configuration files are created in this subdirectory. Copy this subdirectory to the machine where the Pipex node will be installed.

#### Start Pipex Wizard on each target computer and select 'Deploy a node from a previously designed network'

Follow the instructions: select the subdirectory where the configuration files were copied. Then the wizard will create shortcuts to start the Pipex server. Start the Pipex server by double-clicking on the newly created shortcuts.

#### Install registry files on client computers.

On the computers that will run applications (originators) certain registry settings must be set. Copy the .REG files on these computers then double-click to add the settings. The 'AllUsers' or 'CurrentUsers' suffix in the file name indicates what the scope of the settings is.

## 4 Pipex Server Operators' Guide

### 4.1 Contents

#### **Pipex Server Operators' Reference**

[Starting and stopping the server](#)

[Viewing error messages and events](#)

[Enabling and disabling clients, viewing and flushing messages](#)

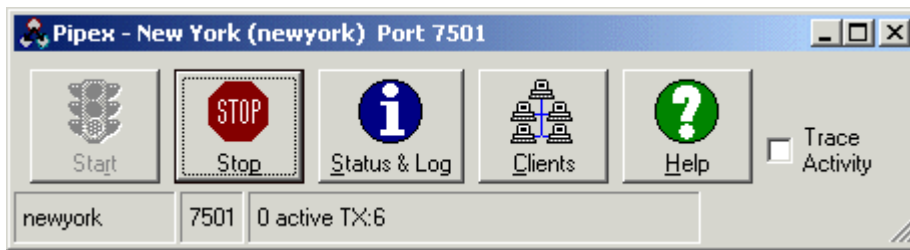
### 4.2 Starting and stopping the server

Shortcuts to start the Pipex server are created by the Pipex Wizard. Double-click on one of these shortcuts to start the Pipex server:



The server will start automatically and the Pipex icon will appear on the system tray: .

The server control panel can be opened by double-clicking on the system tray icon:




### Start

Start Pipex server.

### Stop

Stop Pipex server. When Pipex is stopped, it stops accepting connections from applications, queue handlers and scheduled events, it forcefully closes all existing connections and stops performing scheduled events.

The system tray icon changes to  when the server is stopped.

### Status & Log

Display status information about the server, display error messages and events. See also [Viewing error messages and events](#) .

### Clients

List all clients known by the server, connected or not connected and provide various control-functions. See also [Enabling and disabling clients, viewing and flushing messages.](#)

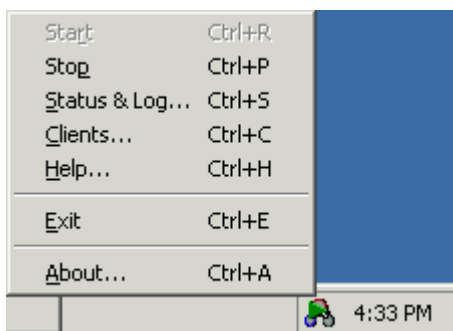
### Help

Provide on-line help.

### Trace Activity

Trace all traffic between the server and clients. Useful for debugging.

Many of the functions accessible from the control panel are also available by simply right-clicking on the system tray icon of Pipex:



## 4.3 Viewing error messages and events

### Sources of errors and events

Errors and events are generated by:

- The Pipex server
- Other Pipex servers that define this server as an error handler node
- Programs utilizing Pipex report errors using an error-reporting facility built into the [Pipex API](#)

### Reporting

When a new error message is reported, Pipex indicates it by flashing one of the following icons on the system tray:



Each one of these symbols represent a certain type of error:

- warning/information
- transient error
- permanent error
- fatal error
- unknown error

If multiple error messages are reported, the symbol of the most severe type will be flashing.

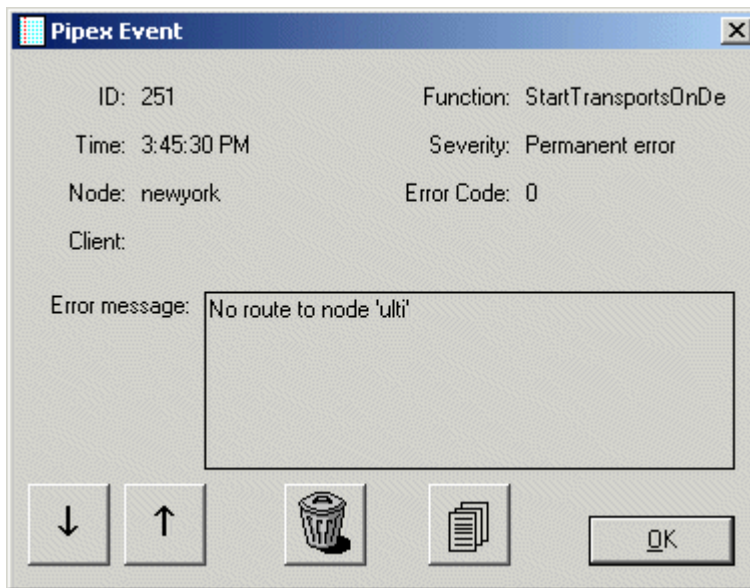
### Effects of errors


If a certain number of transient errors (defined in the desinger) are reported by a certain client, Pipex will disable that client. When a permanent or fatal error occurs, the client is immediately disabled. Warnings are simply logged but have no effect on the status of the client.

### Viewing events

Click on the **Status & Log** button, then select the **Events** tab.

Double-click on an event to view its details:



Press the Up/Down arrows to navigate between events, press the trash button to delete the event. Press the **'Copy to clipboard'** button  to copy the text of the event to the clipboard, which then can be pasted to another program such as an e-mail client.

#### Deleting events

Delete events by selecting an event and pressing the **Delete** (Del) key.

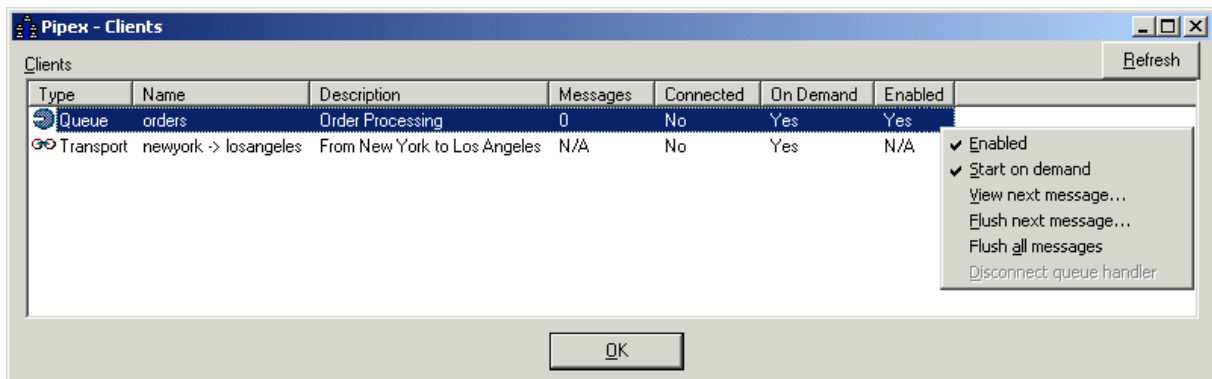
Delete all events by pressing **'Delete all events'**.

## 4.4 Enabling and disabling clients, viewing and flushing messages

Any application, queue handler or transport can be either in an **Enabled** or **Disabled** state. When a client is disabled, the Pipex server will not permit it to connect to it.

A client can be disabled manually by the server operator (for the purpose of maintenance for example) or automatically by the server if too many transient errors occur. The number of transient errors that caused the client to flip to Disabled state can be defined in the Pipex Network Designer.

Press **Clients** to bring up the list of clients known to the server, then right-click on a client to show the context-menu for that client. Check or uncheck the **Enabled flag** to enable or disable the client:



When an error occurs in a queue handler, it may be required to flush the message that causes the error. The next message can be viewed by selecting **View next message**. Viewing the message may provide important information on why the queue handler failed to process that message. The 'bad' message can be **flushed** by selecting **Flush next message** from the context-menu. It is possible to flush all messages lining up for this queue by selecting **Flush all messages**. Once the 'bad' message is flushed, the queue handler can be re-enabled so it can continue to download the following messages.

The **Start on demand** flag indicates to Pipex that it should start the queue handler or transport as soon as messages are available to process.

## 5 Tutorials

### 5.1 Building your first Pipex network

This tutorial will walk you through creating a simple working Pipex network that you can create and test on your computer. The tutorial will describe how to create a testing/development environment and then how to deploy the Pipex network on servers.

The sample program will send order records from a node in Los Angeles to a node in New York, using Pipex.

Access 2000 or newer is required to configure the OLE-DB Queue Handler.

[Step 1 - Create a new design](#)

[Step 2 - Create nodes and routes](#)

[Step 3 - Create queues](#)

[Step 4 - Create an application](#)

[Step 5 - Writing application code](#)

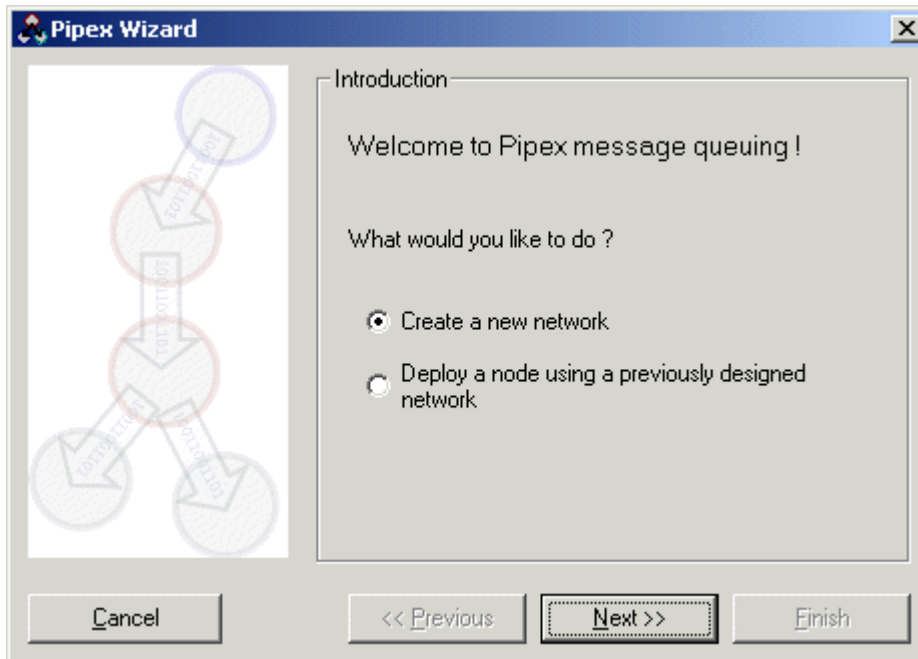
[Step 5 - Testing](#)

[Step 6 - Deployment](#)

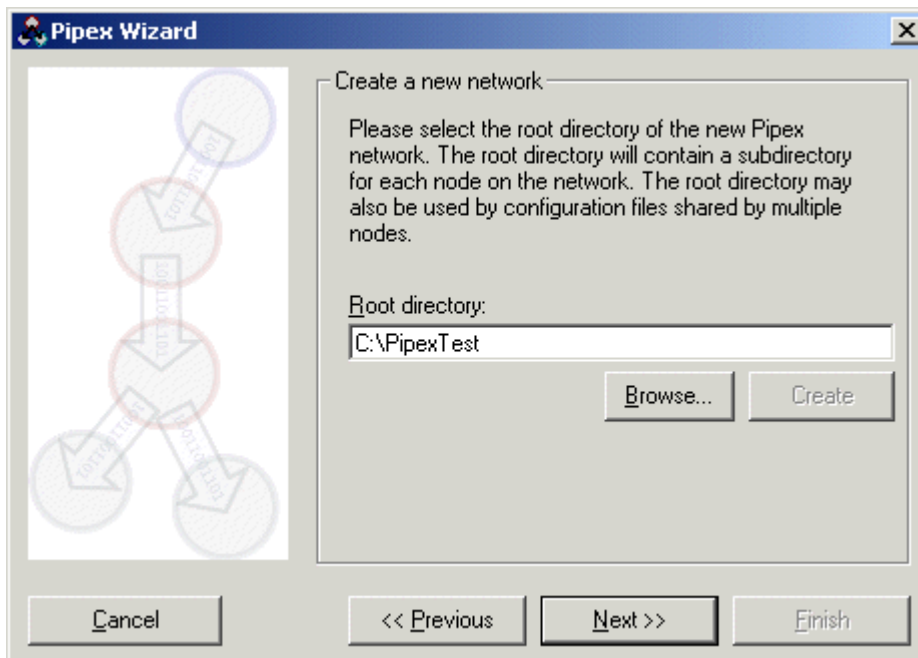
### 5.1.1 Step 1 - Create a new design

#### [Next Step](#)

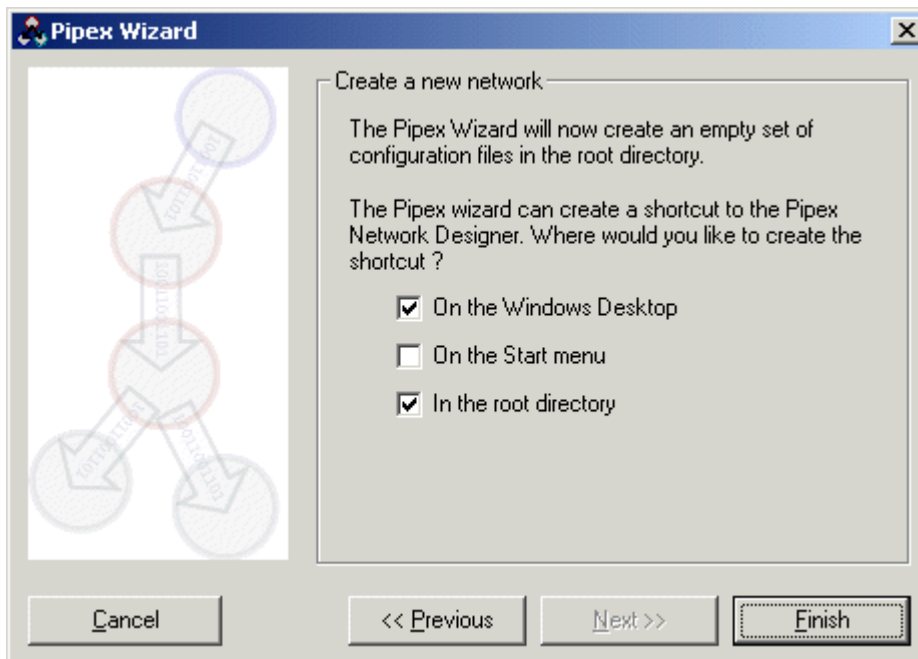
Create a new network using the [Pipex Wizard](#) :



Select Create **New Network**, then click **Next**.



Set the new network's root directory to **C:\PipexTest**. Click **Next**.

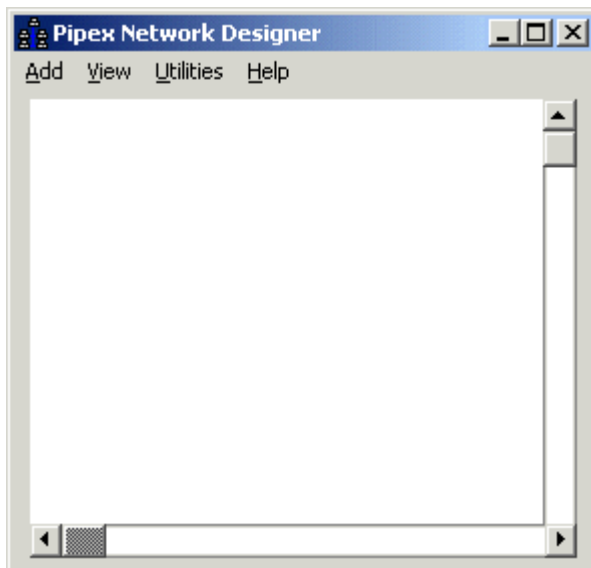


Specify where you want to create shortcuts to the Pipex Network Designer, then click on **Finish**.

A new shortcut will appear on your desktop:



Double-click on the new shortcut to start the [Pipex Network Designer](#).

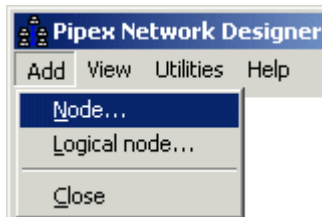


[Next Step](#)

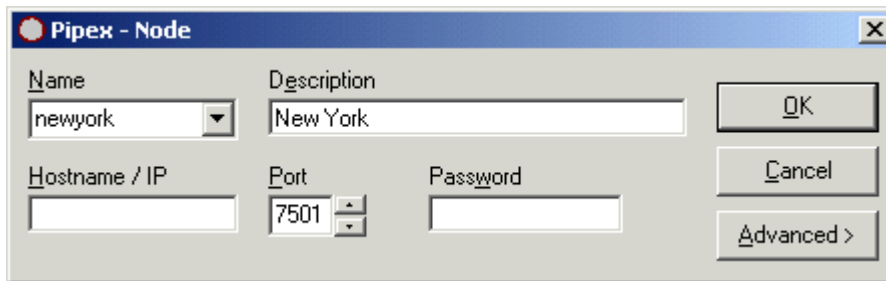
## 5.1.2 Step 2 - Create nodes and routes

[Next Step](#)    [Previous Step](#)

Add a new node by selecting **Node** from the **Add menu**



Create a new node called 'newyork':



### Tips:

- Once a node is added, it can be changed by right-clicking on its icon and selecting **Properties**.
- Nodes can be deleted by right-clicking on the node and selecting **Delete** from the drop down menu.
- Leaving the host name blank means that the node will run and will be accessed only on this computer. Once testing is done, the actual host name or IP address can be specified. If you are running the developer edition or if you intend to test the network on a single PC, the host name should be left blank.

The node icon can be moved by clicking and holding the left button and dragging it on the screen.

Next, create another node, and call it 'losangeles':

**Pipex - Node**

Name: losangeles Description: Los Angeles

Hostname / IP: Port: 7502 Password:

Error handler node: newyork

OK Cancel Advanced <

Change the port number to **7502**. A unique port number is required for each server, because multiple instances of the Pipex server will run on one computer during testing.

Change the error handler to **'newyork'**. By changing the error handler node to 'newyork', node 'losangeles' will forward all errors and events to node 'newyork'. This allows viewing the error messages and events from a central location.

Press OK to create the new node. The Pipex Network Designer will offer to create a two-way TCP/IP based route to the other nodes:

**Create routes to other nodes**

Create a two-way route using TCP/IP to the selected nodes:

- newyork - New York

Cancel OK

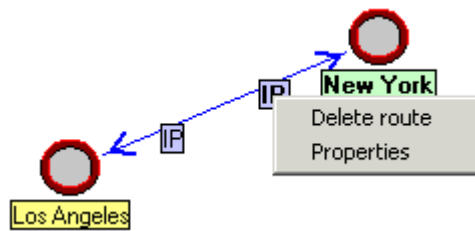
Check **'newyork'** and press 'OK'.


Drag the icon for 'newyork' away so the routes become visible. An arrow between the icons symbolizes the route between nodes.



**Tip:**

You can change or delete a route by right-clicking the label on the lines, then selecting Delete or Properties:



In general, a context-menu is available in the designer window when the cursor changes to . You can activate the context-menu by right-clicking.

[Next Step](#)

[Previous Step](#)

### 5.1.3 Step 3 - Create a queue

[Next Step](#)

[Previous Step](#)

Next, set up a [queue](#) for the purpose of updating tables in a Microsoft Access database. The OLE-DB Queue Handler will be used, which is included with Pipex.

Right-click on the **'New York'** node and select **'New Queue'**. Most of the fields required will be defaulted automatically by the network designer. A few entries are required as follows:

Name the new queue '**orders**'. This name will be used by programs that send messages to this queue.

Change the default description to '**Order Processing**'.

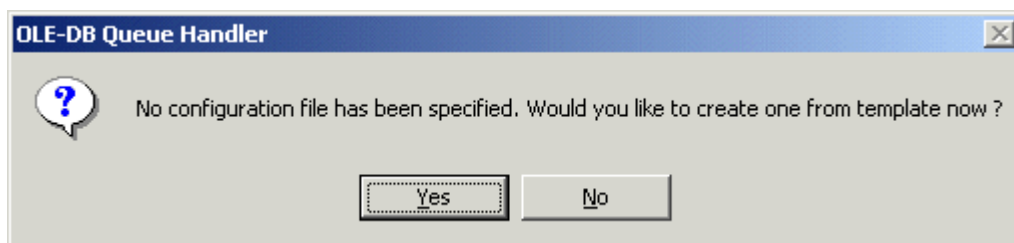
You can read more on the purpose of passwords in the [Security](#) section. For the purpose of the tutorial, choose an easily remembered password.

Check '**Start queue handler on demand**'. The OLE-DB Handler program will be executed automatically by Pipex when a message is received for this queue. (The alternative is to initiate execution by a user or by a scheduler utility, such as the Windows Task Scheduler. This approach is useful if the queue handler requires user interaction, runs on a different computer than the Pipex server).

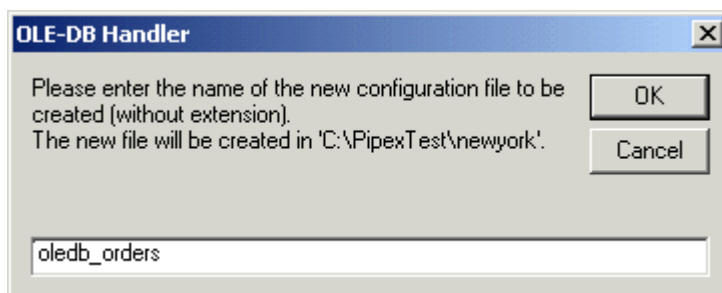
Click on '**Browse**' by the the '**Queue handler executable**' field and find '**OLEDBHandler.exe**'. Normally, it is in C:\Program Files\Pipex unless you have installed the software somewhere else.

Click on '**Build**' by the 'Queue handler parameters' field to configure the OLE-DB Queue Handler:

OLE-DB Handler will offer to create a **new empty configurationfile** for this queue.



Select **Yes** to create one. Then name the new configuration file '**oledb\_orders**':

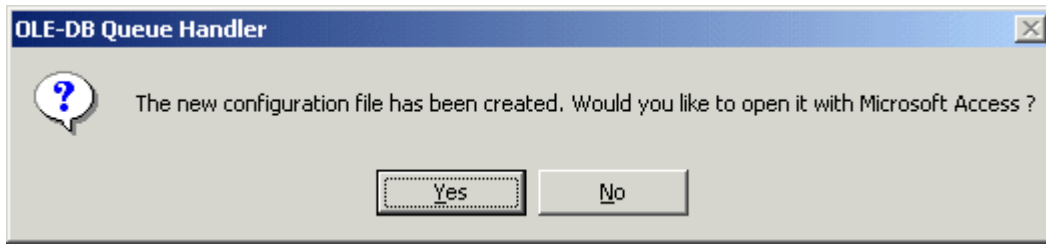


The Pipex designer will now create an empty configuration file called '**oledb\_orders.mdb**' in the '**newyork**' subdirectory within the Pipex directory containing your design. This file contains information used by the OLE DB handler in the node '**newyork**'.



**Tip:** Configuration files may be re-used for other queues, or even copied to be used on another node.

You can immediately open the configuration file using Microsoft Access:



Select **Yes**. Fill out the **Tables** table as follows:

TableName	TargetType	Server	Database	Username	Password	TargetTable
OrderHeaders	0		C:\Program Files\Pipex\Tutorial\OrderProcessing.mdb			
OrderLines	0		C:\Program Files\Pipex\Tutorial\OrderProcessing.mdb			
*	0					

Record: 1 of 2

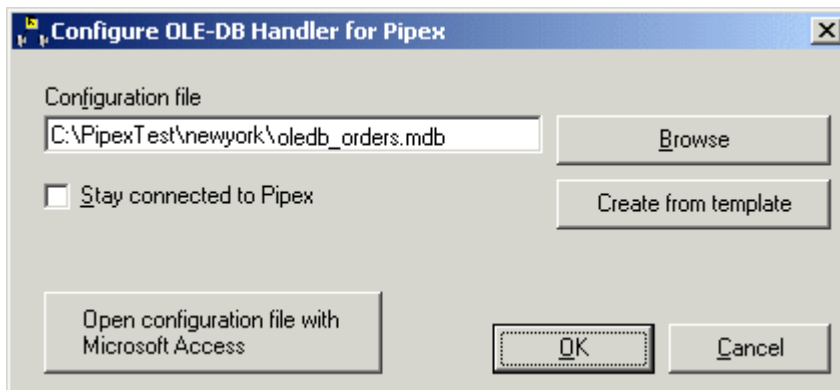
The Tables table defines where each OLE DB table is physically located on the node. Only tables that are listed here may be updated by the OLE-DB queue handler. Messages from other processes will refer to these physical tables using the name in the '**TableName**' column.



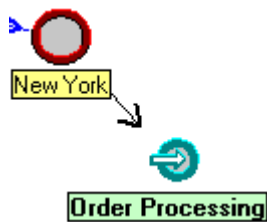
**Tip:**

- The queue handler will update multiple tables within a transaction. These tables do not necessarily have to reside in the same database or even have the same database type.
- You can access the OLE DB table in future by right-clicking on the '**newyork**' icon, and pressing the Build button.

Exit Microsoft Access, then press OK in the OLE-DB Handler configuration window:



Press **OK** in the queue properties dialog box. The new queue will appear in the designer:



**Tip:** You can change the queue by right-clicking on the queue's icon and selecting 'Properties'. Select 'Delete' to delete the queue.

The sample database referenced above is installed with Pipex. To understand the sample source code provided later, you may inspect the database using Microsoft Access or refer to the table-design below:

#### OrderHeaders:

	Field Name	Data Type
🔑	OrderNumber	Number
	Customer	Text
	PurchaseOrderNumber	Text
	DateShipped	Date/Time
	DateEntered	Date/Time
	Remarks	Memo

#### OrderLines:

	Field Name	Data Type
🔑	OrderNumber	Number
🔑	LineNumber	Number
	Product	Text
	UnitPrice	Currency
	Quantity	Currency
	Amount	Currency

[Next Step](#)    [Previous Step](#)

### 5.1.4 Step 4 - Create an application

[Next Step](#)    [Previous Step](#)

This chapter will describe how to create an application that creates and sends messages to a queue. The following sample application will send an order to the 'orders' queue, handled by the OLE-DB Queue Handler:

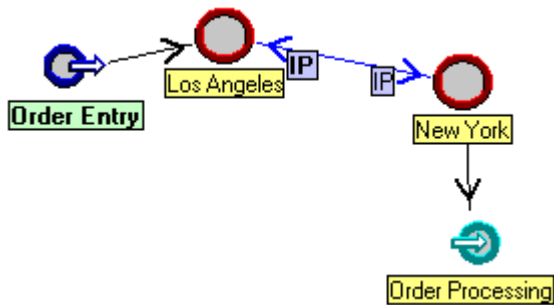
Right-click on the '**Los Angeles**' node and select '**New Application**!':

The screenshot shows the 'Pipex - Application' dialog box. It has a title bar with a Pipex icon and a close button. The dialog contains the following fields and buttons:

- Node:** A dropdown menu with 'losangeles' selected.
- Application name:** A dropdown menu with 'orderentry' selected.
- Description:** A text box containing 'Order Entry'.
- Password:** A text box containing '\*\*\*\*\*'.
- Buttons:** 'OK', 'Cancel', and 'Advanced >'.

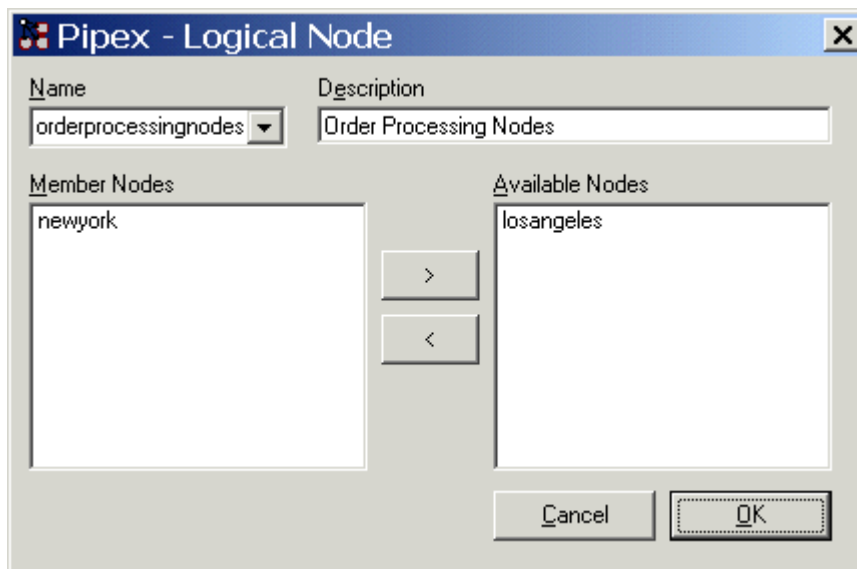
Press **OK**. The new application will appear in the designer. The network layout should look like

this:



The application defined here will send order records to the 'orders' queue on node 'newyork'. The destination node and queue names have to be known by the calling program. To avoid having to hard-code physical network details in the program, a logical node name called **'orderprocessingnodes'** will be used instead. This allows using a single source code that will work on any Pipex network. The physical details of the Pipex network are described in the Designer.

Select **Add, Logical node** from the menu, then add a new logical node called 'orderprocessingnodes' with one member node, 'newyork':



**Tip:** A logical node may represent multiple nodes.

Exit Pipex Network Designer, by selecting **'Close'** from the **'Add'** menu.

All required components in the Pipex Network Designer are now in place.

[Next Step](#)

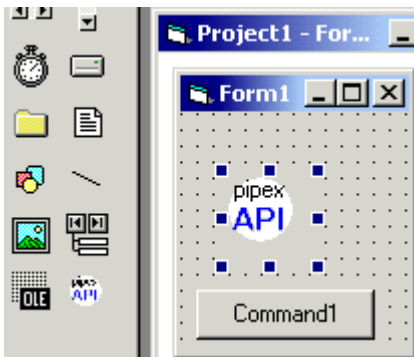
[Previous Step](#)

## 5.1.5 Step 5 - Writing Application Code

[Next Step](#)      [Previous Step](#)

Next, a sample application program will be written, using **Visual Basic**. The code below will also work in other VBA-enabled applications such as Microsoft Access or Microsoft Excel, with minor modifications.

1. Create a new Visual Basic Project (Standard EXE).
2. Select Project / Components... from the Visual Basic menu.
3. Find and check "**Pipex API Control**", then press OK.
4. Select Project / References... from the Visual Basic menu.
5. Find and check "**Pipex OLE-DB Originator**", then press OK.
6. Create a new command-button on the form
7. Drop a new PipexConnection control on the form:



8. Double-click on the command button, then add code to the event procedure. You may copy and paste the code provided below:

```
Private Sub Command1_Click()
'
Dim Tran As PipexOLEDB.Transaction      'Transaction
Dim Key As PipexOLEDB.FieldList        'Key
Dim Fields As PipexOLEDB.FieldList     'Field list
'
' Following statement defines a new transaction to be sent from
' the Order Entry program to the database server.
Set Tran = New PipexOLEDB.Transaction
'
' A 'fieldlist' class can contain a set of key fields or data record fields.
Set Key = New PipexOLEDB.FieldList
'
Key.Add "OrderNumber", 120
'
Set Fields = New PipexOLEDB.FieldList
'
Fields.Add "Customer", "Joe Monroe"
Fields.Add "PurchaseOrderNumber", "PO-10293"
Fields.Add "DateShipped", Date - 1

```

```

Fields.Add "DateEntered", Date
Fields.Add "Remarks", "Assembly requested by customer"
'
' OrderHeaders is the name of the target table in our tutorial app.
Tran.Add "OrderHeaders", Key, Fields 'Add record to transaction
'
'Order line 1
Set Key = New PipexOLEDB.FieldList
'
Key.Add "OrderNumber", 120
Key.Add "LineNumber", 1
'
Set Fields = New PipexOLEDB.FieldList
'
Fields.Add "Product", "KT-112"
Fields.Add "UnitPrice", 240.5
Fields.Add "Quantity", 1
Fields.Add "Amount", 240.5
'
' OrderLines is the second target table in our tutorial app.
Tran.Add "OrderLines", Key, Fields
'
' Order line 2
Set Key = New PipexOLEDB.FieldList
'
Key.Add "OrderNumber", 120
Key.Add "LineNumber", 2
'
Set Fields = New PipexOLEDB.FieldList
'
Fields.Add "Product", "KC-201"
Fields.Add "UnitPrice", 45.1
Fields.Add "Quantity", 4
Fields.Add "Amount", 45.1 * 4
'
Tran.Add "OrderLines", Key, Fields
'
' Send message to Pipex
'
' Normally, you would initialize, connect and disconnect in
' program initialization and close down segments instead of here.
PipexConnection1.Initialize pctApplication, "orderentry"
PipexConnection1.Connect
PipexConnection1.Send "", "orderprocessingnodes", "orders", "write",
"create=yes", Tran.PipexData
PipexConnection1.Disconnect
End Sub

```

The above example builds a transaction for the OLE-DB Queue Handler, then sends that transaction to the 'orders' queue on the 'orderprocessingnodes' node (which is translated to 'newyork' by the Pipex server). The **PipexOLEDB.Transaction** object is used to create a message string that is understood by the OLE-DB Queue Handler. Save the project before moving to the next step - testing your application.

[Next Step](#)

[Previous Step](#)

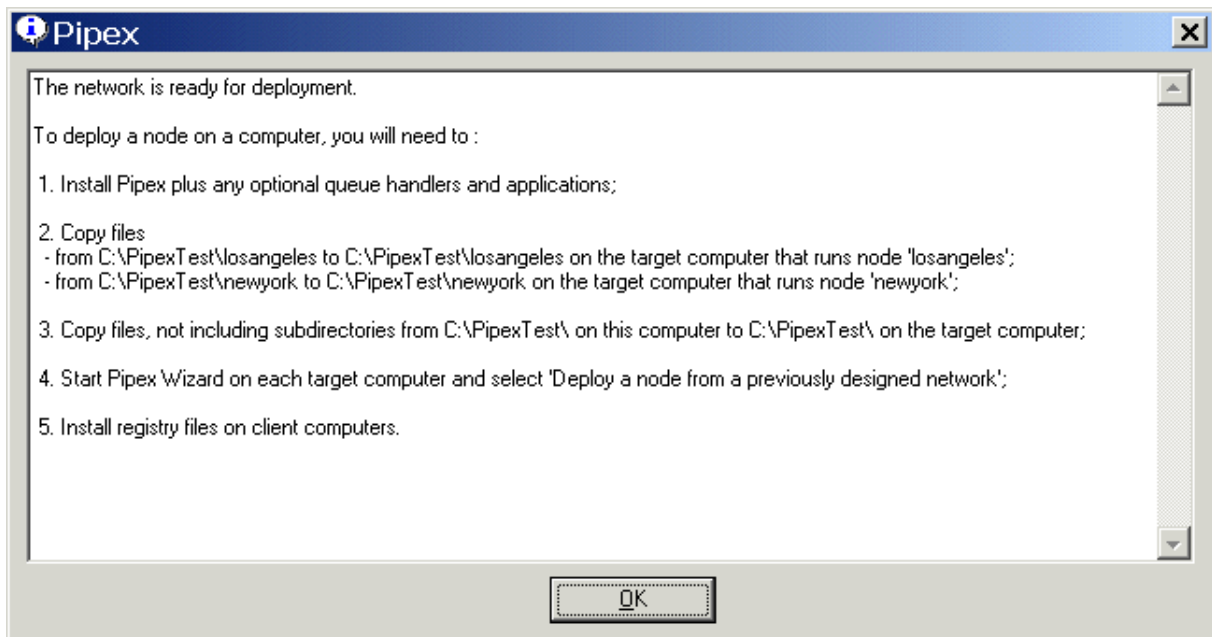
## 5.1.6 Step 6 - Testing

[Next Step](#)      [Previous Step](#)

If you try to run the application program you've just created and press the command-button, you get an error message saying *"Run-time error 107: Parameters are missing for application 'orderentry'"*. To be able to test the program, you will need to **deploy a test network**, then **start the Pipex servers**.

### Deploy a test network

Start the Pipex Network Designer again, and select **Utilities / Prepare files for deployment**. Once the files are prepared, you get instructions on how to deploy the network:

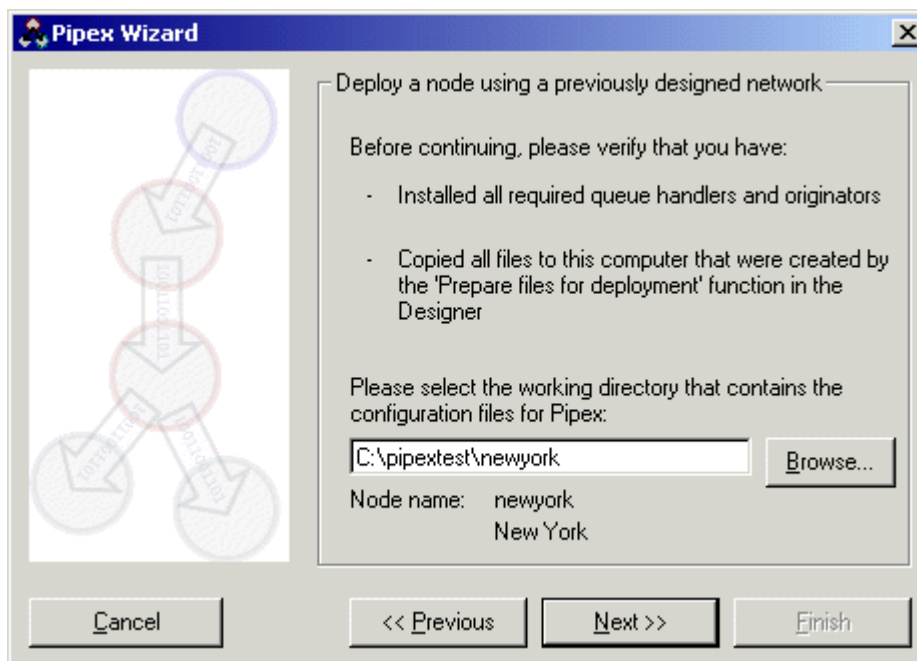


For testing purposes you can simulate the activity for both New York and Los Angeles on a single PC. In this tutorial we will deploy both nodes on the same computer where you created the network design.

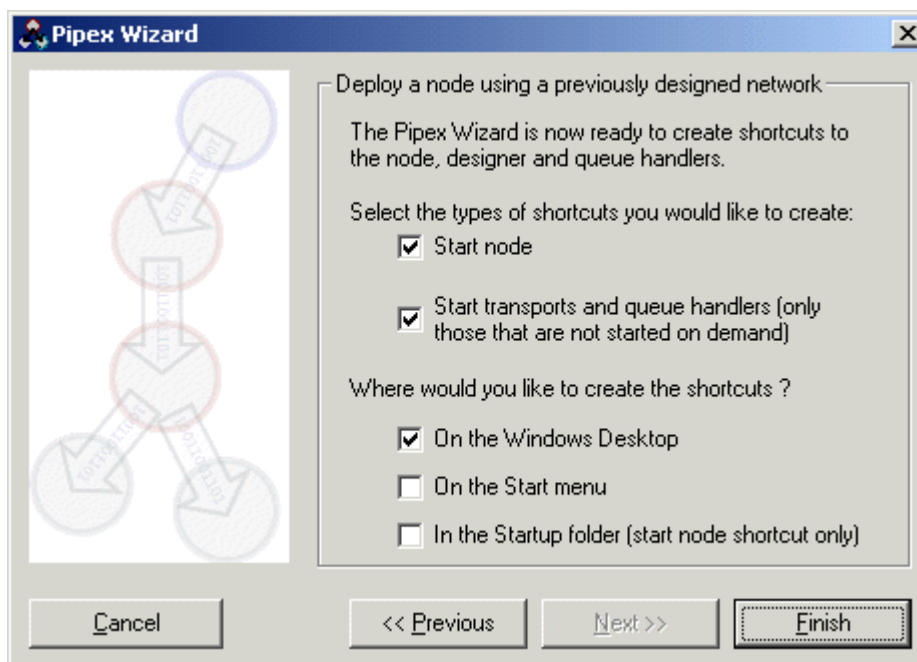
Because you have already installed Pipex on your tutorial computer, steps 1 to 3 above are not necessary; you need to complete only steps 4 and 5.

Deploy each node on your test PC using the Pipex Wizard

Start the [Pipex Wizard](#) and select '**Deploy a node from a previously designed network**'. Then specify the working directory that will be used by node 'newyork':



Click **'Next'** then specify where you want to create shortcuts that start the Pipex server:



This step will place a shortcut on your desktop for starting the Pipex server for node 'newyork'. Repeat these steps for node 'losangeles'. Now you will have shortcuts for both Pipex servers on your desktop. It is possible to test your new Pipex network on a single PC, even though eventually it will run on multiple PCs.

#### Install registry files on test PC

Registry files for the 'orderentry' application can be found in 'C:\pipetest\newyork'. There are two '.reg' files named 'pipex\_newyork\_CurrentUser.reg' (for Windows NT without administrator

privileges only) and 'pipex\_newyork\_orderentry\_AllUsers.reg' (for Windows 95 and 98 and NT with administrator privileges). Double-click the appropriate registry file for your test computer. This will copy the connection information to the registry where it will be accessible to the 'orderentry' application through the [Initialize\(\)](#) method.

### Testing the application

Start both Pipex servers using the shortcuts created by Pipex Wizard. Two new icons will appear on the System Tray:



Run the test application and press the command button on the form. The program will send a transaction to node 'newyork', queue 'orders'. As soon as the OLE-DB Queue handler processes the transaction, the new records appear in the database file 'Orders.mdb'.

Behind the scenes, the following happens:

1. Node 'losangeles' accepts the message from the test program
2. It launches the TCP/IP based transport program that establishes a connection between the two nodes
3. The transport program downloads the message from 'losangeles' and uploads it to 'newyork'
4. The 'newyork' node executes the queue handler for queue 'orders', which is the OLE-DB queue handler
5. The queue handler processes the message by adding the transaction to the 'orders.mdb' database
6. The queue handler sends an ACK message to the Pipex server
7. The ACK message is passed back to the originator node ('losangeles') through the TCP/IP transport
8. The originator node clears the processed message from its internal queue

[Next Step](#)      [Previous Step](#)

## 5.1.7 Step 7 - Deployment

[Previous Step](#)

Once a Pipex network is tested, it can be deployed to servers. In this example, the two servers will be connected through a TCP/IP connection.

### Assigning server computers

A single Pipex server program typically serves all PCs running on a local area network or high-speed WAN. It can run on any Windows 95/NT based computer that is assigned for this purpose. For servers that will serve a large number of client PCs, a dedicated computer running Windows NT 4.0, 2000 or XP is recommended.

### Server addresses

It is important that the server computers' addresses are accessible from the whole network. They should either have fixed IP addresses, or their addresses should be resolvable by name.

### Configure live properties

Open Pipex Network Designer and **change all properties to live values:**

- Change the IP addresses/host names
- Assign passwords to nodes and queues

Example:

- Review the 'Start on demand' flag on queue handlers and transports. If the queue handler is executed by the user, 'Start on demand' should be off.
- Change queue handler parameter values to live values

### Prepare files for deployment

From the Pipex Network Designer select **Utilities / Prepare files for deployment**. Once the files are prepared, you get instructions on how to deploy the network.

## 6 Programming Pipex

### 6.1 Introduction

Pipex provides an easy to use API (Application Programming Interfaces) to access Pipex servers. This chapter will describe the different types of programs that use Pipex and how to write them. A complete reference to the Pipex API ActiveX control is also provided in the Reference section of the manual.

### 6.2 Pipex Programming Model

Please be sure that you are familiar with the following terms before continuing:

- [Node](#)
- [Logical Node](#)
- [Queue](#)
- [Node Routing](#)
- [Application](#)

Programming Pipex involves connecting to Pipex server, then sending/receiving/forwarding messages. Programs using Pipex belong to one of these three categories:

- Originators (also known as Applications)
- Queue Handlers
- Transports

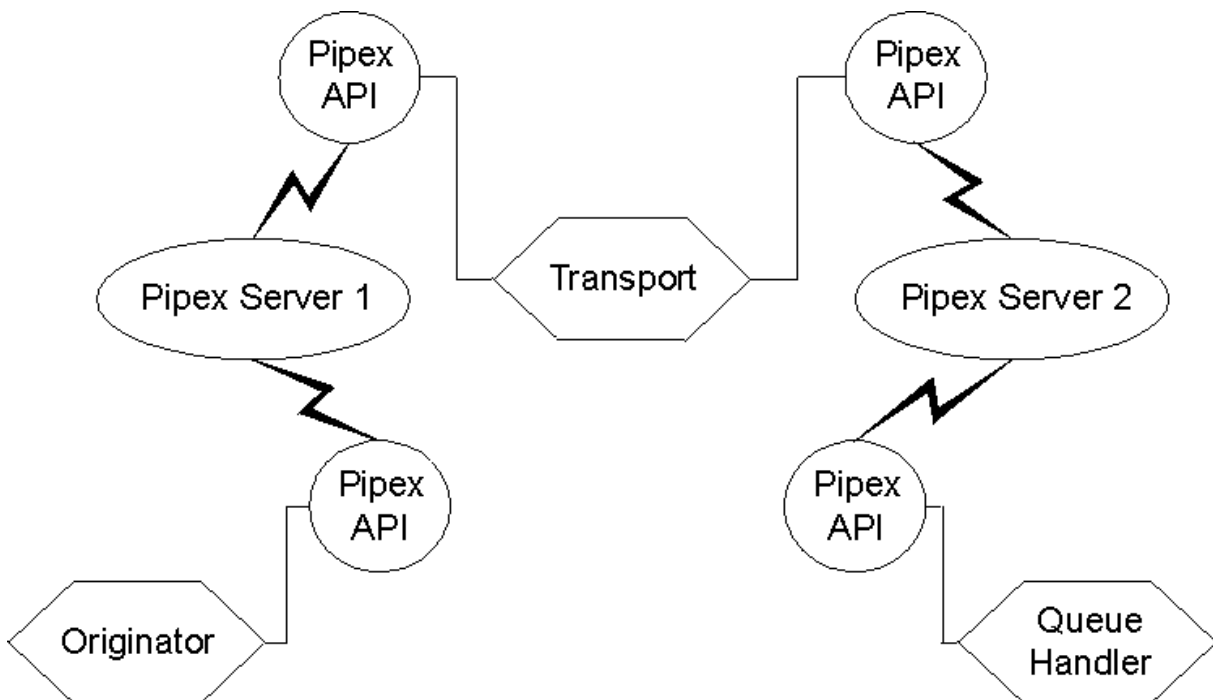
All of these programs use the Pipex API ActiveX control to access the Pipex server. The ActiveX control uses the TCP/IP network protocol to connect to the Pipex server.

Originators are programs that originate (and send) messages to a queue. Queue handlers are programs that process messages sent by originators. Transports forward messages from one node to another.

Queue handlers and transports typically run on the same computer as the Pipex server. They are started by the Pipex server if "Start On Demand" is specified in the Pipex Network Designer.

Originators (applications) run on client computers and connect to a Pipex server on the local network using the Pipex API ActiveX control.

The following diagram illustrates the role of the Pipex API:



### 6.3 Connecting to Pipex

The [Pipex API Control](#) provides the functionality to deliver messages through Pipex. It is an ActiveX control that needs to be [referenced](#) from your **Visual Basic** project or **Visual Basic for Applications** application.

The application that originates messages connects to the Pipex server using the [Connect\(\)](#) method. Connection information such as address of the server, username and password must be specified prior to calling `Connect()`.

*Example:*

```

PipexConnection1.Application = "orderentry"
PipexConnection1.Password = InputBox("Please enter password : ")
PipexConnection1.Host = "192.168.1.1"
PipexConnection1.Connect
  
```

Instead of having to hard-code connection information in your program, the above parameters

can be automatically obtained by using the [Initialize\(\)](#) method:

*Example:*

```
PipexConnection1.Initialize    pctApplication, "orderentry"
PipexConnection1.Connect
```

Once connected, the [Send\(\)](#) method can be used to send messages to a queue on the local or on a remote node. When all messages are sent, the [Disconnect\(\)](#) method must be used to close the connection to the Pipex server.

At any time while the application is connected, errors can be reported by using the [ReportError\(\)](#) method. Errors reported by the application will be displayed by the Pipex error handler.

## 6.4 Sending messages to a queue

Once connected, message originators (applications) may send messages to a queue on the Pipex network. The destination of a message is defined by a node name and a queue name. Every queue is handled by a '**queue handler**' as specified in the Pipex Network Designer. The format of the message sent is irrelevant to Pipex, but it must be understood by the queue handler.

*Example:*

```
RemoteNode = "newyork"
RemoteQueue = "chat"
FunctionName = "show"
Message = "Hello World"

PipexConnection1.Send "", RemoteNode, RemoteQueue, FunctionName, "", Message
```

## 6.5 Sending messages to the OLE DB Handler

The Pipex OLE DB control can be used to simplify the programming of messages sent to a Pipex OLE DB queue handler.

The class for the message as a whole is `PipexOLEDB.Transaction`. A transaction consists of multiple pairs of class `PipexOLEDB.Fieldlist`. Each fieldlist pair contains a record key fieldlist, and a data record fieldlist.

The `fieldlist.Add` function can be used to add new fields to a field list, and takes the form:

*fieldlist.Add fieldname, value.*

The `transaction.Add` function is used to add records to the transaction, and takes the form:

*transaction.Add tablename, key fieldlist, data record fieldlist*

`transaction.Add` can be called numerous times with various table updates to build a single committable transaction set.

`PipexAPIControlSend` Source, TargetNode, TargetQueue, Func, FuncOptions, Data, DataOptions, SequenceNumber

`PipexAPIControlSend` is used to send the transaction to the target queue handler.

The [Send](#) command parameters are specified as follows:

Source	Leave blank.
TargetNode	The logical name of the target node(s).
TargetQueue	The name of the target queue.
Func	'add' for a new key, 'write' for an existing key, 'delete' to delete an existing key.
FuncOptions	'overwrite=yes' to overwrite existing records, 'create=yes' to create new keys if missing.
Data	The <i>transaction</i> object.
DataOptions	Leave empty
SequenceNumber	Set to -1.

The tutorial provides a [complete example](#) for sending messages to the OLE DB handler.

## 6.6 Creating your own queue handler

Pipex comes with a queue handler called 'OLE-DB queue handler'. This queue handler can update databases that provide an OLE-DB interface, such as Microsoft Access or SQL Server.

In some cases, more complex processing is required by a queue handler. The Pipex API interface provides all the necessary functions to create your own queue handler.

Structure of a custom queue handler program:

```
Initialization

If Configuration Requested (called from the designer) Then
  Configure and save parameters
Else
  Process messages:

  Connect to Pipex

  Receive next message until SeqNo = -1
  For each received message:
    Process message
    Send 'Ack' back if successfully processed,
    otherwise report error and exit loop

  Disconnect from Pipex
End If
```

### 6.6.1 Initialization

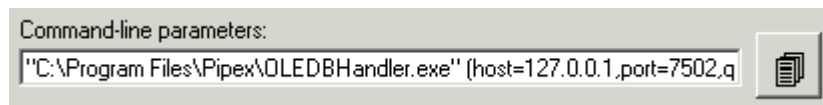
A queue handler can be executed three different ways:

- By the Pipex server, on demand. The connection parameters are passed through the

command-line, which is parsed by the [Initialize\(\) method](#) . The Initialize() method automatically sets the [Host](#) , [Port](#) , [Queue](#) and [Password](#) properties.

- By the Pipex designer. The configuration parameters are passed through the command-line, which is parsed by the [Initialize\(\) method](#) . The Initialize() method sets the [Parameters property](#) , which is a collection of configuration parameters.
- By the user. If no parameters are passed through the command-line, the [Host](#) , [Port](#) , [Queue](#) and [Password](#) properties must be set instead of initializing with the Initialize() method.

It is also possible to pass the Pipex-generated command-line to a manually started queue handler, which allows the programmer to use the Initialize() method. The Pipex-generated command-line settings are available from the Pipex Network Designer. Right-click on the queue, select Properties then copy and paste the contents of the **'Command-line parameters' field**:



*Example, using the Initialize() method:*

```
On Error Resume Next
Err.Clear
PipexConnection1.Initialize   pctQueueHandler, , , Command()
If Err.Number <> 0 Then
    'Handle error
    MsgBox Err.Description, vbCritical
    Exit Sub
End If
```

*Example, by setting each configuration parameter:*

```
PipexConnection1.Host   = "192.168.1.1"
PipexConnection1.Port   = 7501
PipexConnection1.Queue  = "orders"
PipexConnection1.Password = InputBox("Please enter password:")
```

## 6.6.2 Configure parameters when called from the Designer

When you press the 'Build' button next to the 'Queue handler parameters' field, the Pipex Network Designer launches the queue handler in configuration mode. The queue handler program must catch this request by checking the [ConfigurationRequested property](#) :

*Example:*

```
If PipexConnection1.ConfigurationRequested Then
    'Use default configuration handler
    frmMain.PipexConnection1.EditParameters
Else
    ...
```

The queue handler may simply return (if there are no configurable parameters) or display the default parameter editor by calling the [EditParameters](#) method. To do more advanced configuration, it can display a custom form to change the [Parameters](#) collection, then save the changes by calling [SaveParameters\(\)](#) .

### 6.6.3 Connect to the Pipex server

Once all the connection parameters are set, the queue handler must establish a connection to the Pipex server using the [Connect\(\)](#) method. It is very important to enable error-trapping before calling [Connect\(\)](#). An error may be raised for various reasons: remote host is unreachable, the Pipex server is down or the authentication has failed.

```
On Error Resume Next
Err.Clear
PipexConnection1.Connect
If Err.Number <> 0 Then
Else
    'Continue processing
End If
```

### 6.6.4 Receive next message, Acknowledge processed message, Report an error

The following example illustrates how a typical queue handler's mainline should be programmed. The subroutine assumes that the connection to Pipex has been previously opened using the [Connect\(\)](#) method (refer to the section above).

The subroutine downloads the next available message ([Receive\(\) method](#)), or returns -1 if there are no more available. The queue handler then has to try to process this message. If the message is processed successfully, it must send an acknowledgement to Pipex ([Ack\(\) method](#)).

```
Sub Sample_Process()
'
Dim Source As String
Dim TargetNode As String
Dim TargetQueue As String
Dim Func As String
Dim FuncOptions As String
Dim Data As String
Dim DataOptions As String
Dim SeqNo As Long
'
Dim OK As Boolean
'
Dim EN As Long      'Error number
Dim ES As String    'Error source
Dim ED As String    'Error description
'
On Error GoTo Sample_Process_Error      'Set default error handler
'
OK = True
'
'***
' Custom initialization code can be called here
'***
'
Do
    'Download messages until there are no more or until an error occurs
    SeqNo = 0
    Do Until SeqNo = -1 Or (Not OK)
        '
        ' Receive next message
        ' SeqNo returns -1 if there are no more messages
        On Error Resume Next
```

```

Err.Clear
'
PipexConnection1.Receive _
    Source, TargetNode, TargetQueue, Func, FuncOptions, Data, DataOptions,
SeqNo
'
If Err.Number <> 0 Then
    OK = False
    EN = Err.Number
    ES = Err.Source
    ED = Err.Description
    ReportError "Sample_Process", EN, ES, ED, pesPermanentError
End If
On Error GoTo Sample_Process_Error
'
' Process received message
'
If (SeqNo <> -1) And OK Then
    'Process one message
    OK = ProcessOneMessage(Func, FuncOptions, Data, DataOptions)
    '
    If OK Then
        ' Send acknowledgement
        '
        On Error Resume Next
        Err.Clear
        '
        ' Send acknowledgement to Pipex
        PipexConnection1.Ack (SeqNo)
        '
        If Err.Number <> 0 Then
            OK = False
            EN = Err.Number
            ES = Err.Source
            ED = Err.Description
            ReportError "Sample_Process", EN, ES, ED, pesPermanentError
        End If
    End If
    On Error GoTo Sample_Process_Error
'
End If
Loop
'
'***
' Custom cleanup code can be called here
'***
'
Exit Sub
'
' Catch and report unhandled errors
Sample_Process_Error:
    EN = Err.Number
    ES = Err.Source
    ED = Err.Description
    ReportError "OLEDBProcess", EN, ES, ED, pesPermanentError
End Sub
'
' Process one transaction, return True on success, False on failure

```

```
' Replace this code with your own custom processing code
'
Function ProcessOneMessage( _
    Func As String , _
    FuncOptions As String , _
    Data As String , _
    DataOptions As String ) As Boolean
'
    ProcessOneMessage = False
'
    If MsgBox( _
        "Do you want to accept this message ?" & vbCrLf & Func & ":" & Data, _
        vbQuestion + vbYesNo) Then
        ProcessOneMessage = True
    End If
End Function
```

If processing a message takes a long time (several minutes), the connection to Pipex may time out. To avoid this, call the [NoOp\(\) method](#) periodically (every minute or so) to notify Pipex that the program is still 'alive'.

### 6.6.5 Disconnect from the Pipex server

The connection to Pipex must be closed after processing. This is especially important for queue handlers, as the Pipex server only allows one queue handler to be logged in at a time for the same queue.

```
On Error Resume Next
Err.Clear
PipexConnection1.Disconnect
If Err.Number <> 0 Then
Else
    'Continue processing
End If
```

### 6.6.6 Polling Pipex for messages

The example above demonstrated a simple queue handler that processes all the messages available, then quits. This is practical if the program is started by the user, or if messages do not arrive frequently. However, sometimes it is better if a queue handler stays connected to Pipex all the time, so it can respond to receiving a message immediately, without having to start up, connect to Pipex, re-initialize, etc. The Pipex API provides methods to efficiently poll the Pipex server and to allow the program to immediately respond to a new message.

*Mainline of a queue handler that stays connected to Pipex:*

#### Initialization

```
If Configuration Requested (called from the designer) Then
    Configure and save parameters
Else
    Process messages:

    Connect to Pipex

    While there are no errors:
```

```

    Receive next message until SeqNo = -1
    For each received message:
    Process message
    Send 'Ack' back if successfully processed,
    otherwise report error and exit loop

    If there are no errors:
    Go to standby mode
    Sleep until notified or there is an error

    Disconnect from Pipex
End

```

### Go to standby mode:

Calling the StandBy() method notifies the Pipex server that the queue handler is idle and needs to be notified if a new message arrives.

```

'
' Receive loop
'
' ...
' go to standby mode when no more messages are available
Err.Clear
PipexConnection1.Standby
If Err.Number <> 0 Then
    OK = False
    EN = Err.Number
    ES = Err.Source
    ED = Err.Description
    ReportError "Sample_Process", EN, ES, ED, pesTransientError
End If
'
' Loop until server sends notification
If OK Then
    Notified = False
    Do Until Notified Or (Not OK)
        Err.Clear
        Notified = PipexConnection1.Notified
        If Err.Number <> 0 Then
            OK = False
            EN = Err.Number
            ES = Err.Source
            ED = Err.Description
            ReportError "Sample_Process", EN, ES, ED, pesTransientError
        End If
    '
    If OK Then
        DoEvents
        Sleep 1000 'Check in every second
    End If
    '
    Loop
End If
'
' ... If there were no errors, go back to receive/process messages again
'

```

## 6.7 Referencing the Pipex API Control in your Visual Basic Project or Microsoft Access Application

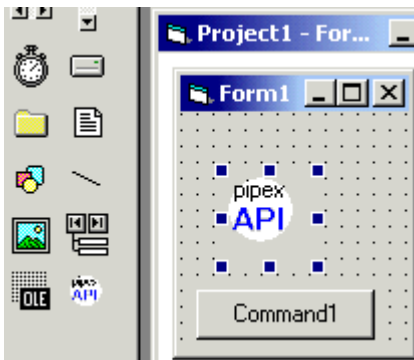
The Pipex API control is a standard ActiveX® control that can be used by any developer tool that utilizes the ActiveX technology. The Pipex API control is installed as a file named **PipexAPI.OCX** in your System directory.

The Pipex API control have been tested using the following developer tools:

- Microsoft Visual Basic® 6.0
- Microsoft Access® 97
- Microsoft Access® 2000

### Example - referencing the Pipex API Control in Visual Basic:

1. Select Project / Components... from the Visual Basic menu.
2. Find and check "**Pipex API Control**", then press OK.
3. Drop a new PipexConnection control on one of the forms:



### Limitations

No more than 8 instances of the Pipex API control can be created in your program.

## 7 Reference

### 7.1 Error messages

#### 7.1.1 Error messages - Pipex API

##### Pipex API Error Messages

PE_WSASStartupFailed = 1	WSASStartup() failed
PE_SocketFailed = 2	socket() failed
PE_BindFailed = 3	bind() failed
PE_GethostbynameFailed = 4	gethostbyname() failed:
PE_ConnectFailed = 5	connect() failed:
PE_ClosesocketFailed = 6	closesocket() failed
PE_InvalidIPAddress = 7	invalid IP address (not used)
PE_ErroronRecvInPipexAuthenticate = 8	Error on recv() in

pipexauthenticate		
PE_SendErrorReturnedZero	= 9	send() error: returned zero
PE_InternalAuthenticationError	= 10	Internal error: got to unknown state during authentication
PE_ServiceNotReady	= 11	Service not ready
PE_LoginIncorrect	= 12	Login incorrect
PE_SendFailed	= 13	send() failed
PE_SendFailedinPipexSend	= 14	send() failed in pipexsend
PE_ErrorOnRecvInPipexSend	= 15	Error on recv() in pipexsend
PE_InternalErrorUnknownStateInPipexSend	= 16	Internal error: got to unknown state in pipexsend
PE_DataRejectedByServer	= 17	Data rejected by server -
PE_ServerIsNotReadyforDATACommand	= 18	Server is not ready for DATA command
PE_ErrorOnSendInPipexSend	= 19	Error on send() in pipexsend
PE_SendFailedinPipexlibClose	= 20	send() failed in PipexlibClose()
PE_SendFailedinPipexReceive_Send	= 21	send() failed in pipexreceive_send
PE_ErrorOnRecvInPipexReceive	= 22	Error on recv() in pipexreceive
PE_UnknownStateInPipexReceive	= 23	Internal error: got to unknown state in pipexreceive
PE_UnexpectedReplyReceivedAfterSendingRECV	= 24	Unexpected reply received after sending RECV:
PE_SocketIsNotConnected	= 25	Socket is not connected
PE_ErrorOnRecvInPipexAck	= 26	Error on recv() in pipexack
PE_SendFailedinPipexAck	= 27	send() failed in pipexack
PE_ACKInvalidSequenceNumber	= 28	ACK: Invalid sequence number
PE_ACKUnexpectedCommand	= 29	ACK: Unexpected command
PE_PipexAckInternalError	= 30	pipexack: internal error
PE_PipexAckUnexpectedReplyReceivedFromServer	= 31	pipexack: unexpected reply received from server
PE_LoginError	= 32	Login error:
PE_SendFailedinPipexStandby	= 33	send() failed in pipexstandby
PE_ErrorOnRecvInPipexStandby	= 34	Error on recv() in pipexstandby
PE_PipexStandbycannotSend	= 35	pipexstandby: can't send
PE_InternalErrorUnknownStateinPipexStandby	= 36	Internal error: got to unknown state in pipexstandby
PE_ErrorOnSelect	= 37	Error on select()
PE_ErrorOnRecvInPipexlibNotified	= 38	Error on recv() in PipexlibNotified()
PE_PipexlibNotifiedServerErrorInStandbymode	= 39	PipexlibNotified: server returned an error in standby mode
PE_SendFailedinPipexReportError_Send	= 40	send() failed in pipexreporterror_send
PE_ErrorOnRecvInPipexreportError	= 41	Error on recv() in pipexreporterror
PE_PipexreportErrorcannotSend	= 42	pipexreporterror: can't send
PE_UnknownStateInPipexReportError	= 43	Internal error: got to unknown state in pipexreporterror
PE_SendFailedWhileExecutingNoOpCommand	= 44	send() failed while executing NOOP command
PE_NoWindowsSystemDir	= 100	Can't find Windows system directory
PE_InvalidClientType	= 101	Invalid client type specified
PE_MissingApplicationName	= 102	Application name parameter is missing
PE_ApplicationNameNotRequired	= 103	Application name parameter is not required
PE_ErrorReadingRegistry	= 104	Error while reading the registry
PE_UnknwonReply	= 105	Unknown reply received
PE_NonNumericSeqNo	= 106	Non-numeric sequence number
PE_APIParametersMissing	= 107	Parameters are not set up in the registry
PE_CommandLineParseError	= 108	Command-line parse error (queue handlers or transports)
PE_UnableToReadParameterFile	= 109	Unable to read temporary parameter file

PE\_UnableToWriteParameterFile = 110 Unable to write temporary parameter file

## 7.1.2 Can't find Windows system directory

### 5100 - Can't find Windows system directory

## 7.1.3 Invalid client type specified

### 5101 - Invalid client type specified

The Initialize method was called with an invalid Client Type parameter.

## 7.1.4 Application name parameter is missing

### 5102 - Applicationname parameter is missing

The application name parameter must be specified when calling the Initialize method with Client Type = *pctApplication*.

## 7.1.5 Application name parameter is not required

### 5103 - Applicationname parameter is not required

The Application Name parameter was specified even though the Client Type is not *pctApplication*.

## 7.1.6 Error while reading the registry

### 5104 - Error while reading the registry

An error occurred while trying to read the registry. The Pipex API reads the following sections of the registry:

- HKEY\_CURRENT\_USER\SOFTWARE\Slofstra Software\Pipex\API\Applications
- HKEY\_LOCAL\_MACHINE\SOFTWARE\Slofstra Software\Pipex\API\Applications

Please verify if the process using the Pipex API has the appropriate permissions to read these sections.

## 7.1.7 Unknown reply received

### 5105 - Unknown reply received

An unknown reply was received from the Pipex server.

## 7.1.8 Non-numeric sequence number

### 5106 - Non-numericsequence number

A message with a non-numeric sequence number was received from the Pipex server.

## 7.1.9 Parameters are not set up in the registry

### 5107 - Parameters are not set up in the registry

The Initialize method was called by the application, but connection parameters for this application are missing from the registry.

- Use the Pipex API Setup utility to inspect the applications configured on this computer and add new entries if necessary.
- Add registry entries generated by the *Prepare Files for Deployment* function of the Pipex designer. Refer to Deployment for more information.

## 7.1.10 Command-line parse error

### 5108 - Command-line parse error

Queue handlers and transports obtain the connection information through the command-line. An error occurred while trying to parse the command-line.

This error can not occur if the queue handler or transport is executed by on demand. If the queue handler or transport is executed manually or by an external task scheduler, please compare the command-line parameters to what appears in the Pipex Network Designer in the *Command-line parameters* field.

## 7.1.11 Unable to read temporary parameter file

### 5109 - Unable to read temporary parameter file

When a queue handler or transport is configured from the Pipex Network Designer, it uses an intermediate temporary file created in the Windows temporary folder to pass parameters. Both the queue handler or transport and the Pipex server must have read/write access to the Windows temporary folder.

## 7.1.12 Unable to write temporary parameter file

### 5110 - Unable to write temporary parameter file

When a queue handler or transport is configured from the Pipex Network Designer, it uses an intermediate temporary file created in the Windows temporary folder to pass parameters. Both the queue handler or transport and the Pipex server must have read/write access to the Windows temporary folder.

## 7.2 Pipex API Control Reference

### 7.2.1 Initialization and connecting

#### 7.2.1.1 Initialize() method

Programs utilizing the Pipex API rely on information about the closest Pipex server, the usernames and passwords to use, etc. Transports and queue handlers obtain this information through command-line parameters, while applications (data originators) obtain the information from the registry. The Initialize() method initializes all properties on the control that are required to be able to successfully connect to the Pipex server.

Initialize() sets connection parameters such as [Host](#), [Port](#), [Node](#), [Queue](#), [Application](#) and [Password](#) that are required before calling the Connect() method. It also sets the [ConfigurationRequested](#) and [Parameters](#) properties when a queue handler or transport is called by the Pipex designer.

### Syntax

*object*.Initialize(CT, [ApplicationID], [Index], [CommandLine] )

*object*            A Pipex API Control object

### Parameters

#### *CT*

Client type. It must be one of the following values:

*pctApplication = 0*

The calling program is an [application](#). When this client type is selected, the *ApplicationID* parameter is mandatory.

*pctQueueHandler = 1*

The calling program is a [queue handler](#).

*pctTransport = 2*

The calling program is a [transport](#).

#### *ApplicationID*

Required if *CT = pctApplication*, otherwise it must be omitted.

#### *Index*

Index of the connection parameter set to be retrieved. It must be one of the following values:

*ppsiFromNode = 1*

Initialize connection parameters for 'from node'.

*ppsiToNode = 2*

Initialize connection parameters for 'to node'.

The default value is *ppsiFromNode* which refers to the 'from node'. This parameter is only used by [transports](#).

#### *CommandLine*

Command-line parameters of the queue handler or transport. Pass the return value of the Command() Visual Basic function. Required if *CT = pctQueueHandler* or if *CT = pctTransport*, otherwise it must be omitted.

### Requirements

Applications obtain connection parameter information from the registry which is set up by the [Pipex API Setup program](#).

Queue handlers and transports **retrieve** this information from the command-line. The command-line parameters are automatically assigned by the Pipex server if the queue handler or the

transport is set to "Start on demand".

### Error Handling

A run-time error is generated if no command-line parameters are given (queue handlers and transports) or if there is no entry in the registry for the selected application (applications). Error trapping must be enabled before calling this method.

### Example

```
'Initialize an application called 'orderentry'  
PipexConnection1.Initialize   pctApplication,   "orderentry"  
  
'Initialize an queue handler  
PipexConnection1.Initialize   pctQueueHandler,   ,   , Command()
```

### See Also

[Host](#), [Port](#), [Node](#), [Queue](#), [Application](#), [Password](#), [Connect\(\)](#)

#### 7.2.1.2 ConfigurationRequested property

The ConfigurationRequested property is set to **True** if the queue handler or transport was called from the Pipex designer to configure its parameters. The property is **read-only**.

#### Syntax

*object*.ConfigurationRequested

*object*            A Pipex API Control object

#### Requirements

The Initialize method must be called prior to reading this property.

#### See also

[Initialize\(\) method](#)

#### 7.2.1.3 Hostproperty

The Host property specifies the **host name or IP address** of the Pipex server to connect to. It must be specified prior to calling the [Connect\(\)](#) method. If initialization is done using [Initialize\(\)](#), this property is automatically set. Refer to [Connecting to Pipex](#) to learn about different ways of connecting.

#### Syntax

*object*.Host = [value]

*object*            A Pipex API Control object  
*value*            Alphanumeric value

### Remarks

Simple host names, fully qualified domain names or IP addresses can be used. If the connection is made to the local host, the property can be set to "" (empty string).

### See also

[Port property](#) , [Connect\(\) method](#)

#### 7.2.1.4 Port property

The Port property specifies the **TCP port number** of the Pipex server to connect to. It must be specified prior to calling the [Connect\(\)](#) method. If initialization is done using [Initialize\(\)](#), this property is automatically set. Refer to [Connecting to Pipex](#) to learn about different ways of connecting.

### Syntax

*object*.Port = [value]

*object*            A Pipex API Control object  
*value*            Numeric value (Long)

### Remarks

The default Port number is **7501** . This number is generally used when multiple instances of the Pipex server are running on one machine. Each instance needs to have a unique port number. Permitted values are **from 7501 to 7599** .

### See also

[Host property](#) , [Connect\(\) method](#)

#### 7.2.1.5 Node property

The Node property specifies the name of the node to deliver messages for. This property is used by [transports](#) . It must be specified prior to calling the [Connect\(\)](#) method. If initialization is done using [Initialize\(\)](#), this property is automatically set. Refer to [Connecting to Pipex](#) to learn about different ways of connecting.

### Syntax

*object*.Node = [value]

*object*            A Pipex API Control object  
*value*            Alphanumeric value

### Remarks

This property is used to specify for the Pipex server which node the transport delivers for. If the Pipex server requires a password to deliver messages, the [Password property](#) must also be specified.

For example if a transport delivers between node 'office' and node 'plant' it would connect to node 'office' using the node name 'plant'.

**See also**

[Password property](#) , [Connect\(\) method](#)

**7.2.1.6 Queue property**

The Queue property specifies the name of the queue to process messages for. This property is used by [Queue handlers](#) . It must be specified prior to calling the [Connect\(\)](#) method. If initialization is done using [Initialize\(\)](#), this property is automatically set. Refer to [Connecting to Pipex](#) to learn about different ways of connecting.

**Syntax**

*object.Queue* = [*value*]

*object*            A Pipex API Control object  
*value*            Alphanumeric value

**Remarks**

This property is used to specify to the Pipex server which queue the queue handler processes messages for. If the Pipex server requires a password to process messages, the [Password property](#) must also be specified.

**See also**

[Password property](#) , [Connect\(\) method](#)

**7.2.1.7 Application Property**

The Application property specifies the name of the [application](#) that generates Pipex messages. It must be specified prior to calling the [Connect\(\)](#) method. If initialization is done using [Initialize\(\)](#), this property is automatically set. Refer to [Connecting to Pipex](#) to learn about different ways of connecting.

**Syntax**

*object.Application* = [*value*]

*object*            A Pipex API Control object  
*value*            Alphanumeric value

**Remarks**

If the Pipex server requires a password to generate messages, the [Password property](#) must also be specified. Setting the application property will not automatically load the connection parameters from the registry. Use the [Initialize method](#) to load the connection parameters.

**See also**

[Password property](#) , [Connect\(\) method](#)

### 7.2.1.8 Password property

The Password property allows specifying passwords for transports, queue handlers and applications. This property can be automatically set by calling the [Initialize\(\)](#) method. The default value is no password.

#### Syntax

```
object.Password = [value]
```

*object*            A Pipex API Control object  
*value*            Alphanumeric value

#### See also

[Node Property](#) , [Queue Property](#) , [User Property](#) , [Connect\(\) method](#)

### 7.2.1.9 Connect() method

Establish a connection to the [Pipex server](#) .

#### Syntax

```
object.Connect()
```

*object*            A Pipex API Control object

#### Remarks

Prior to calling the Connect() method, one of [Node](#), [Queue](#) or [Application](#) properties must be specified. Depending on which one of these three is specified, the client program will be considered a [transport](#) , a [queue handler](#) or an [application](#) . Properties [Host](#) and [Port](#) specify the Pipex server's host name and TCP port number. The [Password](#) property must be set if the Pipex server requires a password for delivering, processing or sending messages.

#### Error Handling

Typically, error trapping should be turned on before calling Connect(). A run-time error is generated when the remote host is unreachable, the Pipex server is down or if the authentication failed. The following is an example of how to use error trapping:

```
PipexConnection1.Application = "orderentry"  
  
On Error GoTo Open_Error  
  
    PipexConnection1.Connect  
  
On Error GoTo 0  
...  
  
Open_Error :  
  
    MsgBox "Connection to pipex could not be established : " & Err.Description,  
vbCritical
```

**See also**

[Disconnect\(\) method](#)

**7.2.1.10 Disconnect()method**

Disconnect from the [Pipex server](#) .

**Syntax**

*object*.Disconnect()

*object*            A Pipex API Control object

**Error Handling**

Typically, error trapping should be turned on before calling Disconnect(). Refer to [Connect\(\)](#) for information on how to use error trapping in your code.

**See also**

[Connect\(\) method](#)

**7.2.2 Parameters****7.2.2.1 Parameters property**

The Parameters property holds the list of parameters that were passed by Pipex to the **queue handler** or **transport**

If the [ConfigurationRequested](#) property is true, the queue handler or transport was started from the Pipex designer and it needs to configure and save parameters before exiting using the [SaveParameters](#) method.

The Parameters property points to an instance of the **Collection object** The **Add** and **Remove** methods can be used to add or remove values. The **Count** property returns the number of parameters in the collection. The entire list of parameters can be iterated using the **For Each...Next** statement. Each value in the list is of the **PipexAPIParameter** type which has two properties: **Name** and **Value**.

Parameters can also be set and read by name using the [SetParameter](#) and [GetParameter](#) methods.

**Syntax**

*object*.Parameters (*index*)

*object*            A Pipex API Control object  
*index*            Parameter name

**'configurationfile'parameter**

If a transport or queue handler uses a configuration file, the **configurationfile** item is set to the name of the file. This parameter can be defined in the Pipex designer. The following code demonstrates how to retrieve the configuration file name:

```
Dim Par As PipexAPIControl.PipexAPIParameter
Dim cfn As String

For Each Par In PipexConnection1.Parameters
Select Case Par.Name
    Case "configurationfile"
        cfn = Par.Value
    ...
    Case Else
        MsgBox "Unknown parameter '" & Par.Name & "'", vbExclamation
End Select

Or

cfn = PipexConnection1.GetParameter("configurationfile")
```

### See also

[ConfigurationRequested property](#) , [GetParameter method](#) , [SetParameter method](#)

#### 7.2.2.2 EditParameters() method

Bring up the standard parameter editor for queue handlers and transports. This method should be called when the queue handler or transport starts up and the [ConfigurationRequested](#) property returns **True**. The [ConfigurationRequested](#) property is True when the queue handler or transport is called from the Pipex designer.

This method saves the parameter list if the user presses "OK" on the parameter editor form.

### Syntax

*object*.EditParameters()

*object*            A Pipex API Control object

### Requirements

The Initialize method must be called prior to calling this method.

### See also

[Parameters property](#) , [ConfigurationRequested property](#)

#### 7.2.2.3 SaveParameters() method

Save the list of parameters stored in the [Parameters](#) collection. The parameters need to be saved after the parameters have been changed by a custom parameter editor or configuration form.

When the Pipex designer starts the queue handler or transport, the queue handler or transport has two options. It can either call the standard parameter editor by calling [EditParameters\(\)](#) , or it can call its own parameter editor which changes the [Parameters](#) collection. The updated Parameters collection then needs to be saved using [SaveParameters\(\)](#).

### Syntax

*object*.SaveParameters()

*object*            A Pipex API Control object

### Requirements

The Initialize method must be called prior to calling this method.

### See also

[Parameters property](#) , [ConfigurationRequested property](#) , [Creating your own queue handler](#)

#### 7.2.2.4 SetParameter() method

This method allows setting configuration parameters by name. See [Parameters](#) property for more information on parameters. This function is typically called by a custom parameter editor or configuration form. Before exiting, the program needs to call [SaveParameters](#) to update parameters in the Pipex Designer.

If the parameter specified does not exist in the parameter collection, it will be created.

### Syntax

*object*.SetParameter(  
    Name As String,  
    Value As String)

*object*            A Pipex API Control object  
*Name*            Name of parameter to set (case insensitive)  
*Value*            New parameter value

### Requirements

The Initialize method must be called prior to calling this method.

### See also

[Parameters property](#) , [ConfigurationRequested property](#) , [Creating your own queue handler](#)

#### 7.2.2.5 GetParameter() method

This method allows reading a parameter's value by name. See [Parameters](#) property for more information on parameters.

### Syntax

*object*.GetParameter(  
    Name As String)  
    As String

*object*            A Pipex API Control object  
*Name*            Name of parameter to read (case insensitive)

### Return Value

Returns the value of the specified parameter. Returns empty if the parameter does not exist in the parameter collection.

### Requirements

The Initialize method must be called prior to calling this method.

### See also

[Parameters property](#) , [ConfigurationRequested property](#)

## 7.2.3 Message Processing

### 7.2.3.1 Send() method

Send a message to [queue](#) or a [node](#). Used by [applications](#) and [transports](#), not strictly allowed for a [queue handler](#) session. However, a queue handler program may open an application session.

### Syntax

```
object.Send(
    Source As String,
    TargetNode As String,
    TargetQueue As String,
    Func As String,
    FuncOptions As String,
    Data As String,
    DataOptions As String,
    SequenceNumber As Long)
```

*object*            A Pipex API Control object

### Parameters

#### *Source*

Name of the node where the message is being delivered from.

This parameter **must be left blank** when Send() is called **by an application**. Transports simply need to forward the value of the Source parameter set by a [Receive\(\)](#) call.

#### *TargetNode*

Name of the node where the message needs to be delivered. *TargetNode* and *TargetQueue* together **specify the final destination of the message**. This field is required.

#### *TargetQueue*

Name of the queue where the message needs to be delivered. This field is required for applications and must be left blank by transports.

#### *Func*

This parameter can be used to specify a function name or code for the queue handler. Refer to the queue handler's documentation for information about valid functions. Examples are 'write', 'delete', 'post', etc. The 'ack' function is used internally by Pipex to send acknowledgements back to the sender node.

#### *FuncOptions*

Options related to the *Func* parameter. For a list of available options refer to the

documentation of the queue handler.

#### *Data*

This parameter can be used to provide additional data for the queue handler.

#### *DataOptions*

DataOptions can be used to specify options related to the Data parameter. Examples are the type of encoding, checksum, language, etc.

#### *SequenceNumber*

The sequence number is assigned by the first Pipex node that the message is sent to. Sequence numbers are used to ensure that no messages are lost and all messages are delivered to the queue handler once and only once.

The sequence number **must be set to -1 by applications** and **may not be changed by transports**.

**Note:** Any of the *Func*, *FuncOptions*, *Data* or *DataOptions* parameters can be left blank. However, the queue handler that handles the target queue may require some or all of these parameters to be specified.

### Remarks

Prior to calling the Send() method, a connection must be established to the Pipex server using the [Connect\(\)](#) method.

### Error Handling

Typically, error trapping should be turned on before calling Send(). A run-time error is generated when the connection to the Pipex server is lost or if the message is rejected by the server. Refer to [Connect\(\)](#) for more information on using error trapping in your code.

### See also

[Receive\(\) method](#), [Connect\(\) method](#), [Delivering messages though Pipex from your application](#)

## 7.2.3.2 Receive() method

Receive a message from the Pipex server. The received message must be processed by a queue handler or delivered by a transport.

### Syntax

```
object.Receive(
    ByRef Source As String,
    ByRef TargetNode As String,
    ByRef TargetQueue As String,
    ByRef Func As String,
    ByRef FuncOptions As String,
    ByRef Data As String,
    ByRef DataOptions As String,
    ByRef SeqNo As Long)
```

*object*            A Pipex API Control object

### Return Values

#### *Source*

Name of the node where the message was delivered from.

*TargetNode*

Name of the node where the message needs to be delivered.

*TargetQueue*

Name of the queue where the message needs to be delivered.

*Func*

This parameter can be used to specify a function name or code for the queue handler. Refer to the queue handler's documentation for information about valid functions. Examples are 'write', 'delete', 'post', etc. The 'ack' function is used internally by Pipex to send acknowledgements back to the sender node.

*FuncOptions*

Options related to the *Func* parameter. For a list of available options refer to the documentation of the queue handler.

*Data*

This parameter can be used to provide additional data for the queue handler.

*DataOptions*

DataOptions can be used to specify options related to the Data parameter. Examples are the type of encoding, checksum, language, etc.

*SequenceNumber*

The sequence number is used by queue handlers to identify the received messages. As soon as a message is processed, the queue handler **must call the [Ack\(\) method](#)** with this sequence number to acknowledge the processing of the message. The sequence number is **set to -1 if there are no more messages to download**.

**Additional remarks:** If Ack() is not called or the Ack() call fails, Pipex **re-sends the message again** when the queue handler or transport reconnects. This behaviour poses a problem to queue handlers that may work incorrectly if they receive messages more than once. For example, an inventory-update module that is implemented as a queue handler may post a shipment twice and set the on hand quantity incorrectly. To avoid incorrect operation, the queue handler should store the last sequence number processed before the Ack() method is called. If the Ack() method fails, the queue handler can simply ignore the re-sent message by checking its sequence number against the stored one. Another way to avoid this problem is to use transactions. The queue handler must not commit the transaction until the Ack() method is successfully called.

Read more about writing a queue handler mainline in [Creating your own queue handler](#) .

**Note:** Transports must leave the above values unchanged and must forward them to the target node by using the Send() method.

## Remarks

Prior to calling the Receive() method, a connection must be established to the Pipex server using the [Connect\(\)](#) method. As soon as a message is received using Receive(), the [Ack\(\)](#) method must be called to acknowledge that the message has been processed. Continue calling the Receive() method until it returns with **SequenceNumber = -1**.

## Error Handling

Typically, error trapping should be turned on before calling Receive(). Refer to [Connect\(\)](#) for more information on using error trapping in your code.

## See also

[Ack\(\) method](#), [Send\(\) method](#), [Connect\(\) method](#), [Creating your own queue handler](#)

### 7.2.3.3 Ack()method

Acknowledge the processing of messages received by [Receive\(\)](#). When a queue handler or a transport downloads a message using [Receive\(\)](#), it must also call [Ack\(\)](#) if the message is processed successfully. If a message is not acknowledged, Pipex re-sends that message to the queue handler or transport during the next session, and keeps re-sending it until it is acknowledged. If a message is not processed, the reason must be reported as an error using the [ReportError\(\)](#) method.

#### Syntax

*object.Ack(SeqNo As Long)*

*object*            A Pipex API Control object

#### Parameters

*SeqNo*

Sequence number of the last message successfully processed. All messages with this or lower sequence numbers will be acknowledged.

#### Remarks

A connection to the Pipex server must be established using the [Connect\(\)](#) method before calling [Ack\(\)](#). It is legal (although not recommended) to send only one acknowledgement after processing multiple messages. This technique is only safe if used by transports, as Pipex doesn't re-process messages that are sent again because of an error.

#### Error Handling

It is important to trap errors and handle them when using the [Ack\(\)](#) method. Refer to [Connect\(\)](#) for more information on using error trapping in your code. Also read more about writing a safe queue handler mainline in '[Creating your own queue handler](#)'.

#### See also

[Receive\(\) method](#), [Creating your own queue handler](#)

### 7.2.3.4 ReportError()Method

Report an error to the Pipex server. Depending on the error severity, [ReportError\(\)](#) can also cause Pipex to temporarily stop executing the queue handler or transport.

Because queue handlers and transports generally do not interact with the user interface, they must use [ReportError\(\)](#) to report errors. Although applications often do have a user interface, they are allowed to use [ReportError](#) as well.

#### Syntax

*object.ReportError(  
    FunctionName As String,  
    Severity As ErrorSeverity,  
    ErrorCode As Long,  
    ErrorDescription As String)*

*object* A Pipex API Control object

### Parameters

*FunctionName*

Name of the function where the error occurred.

*Severity*

Severity of the error. The following values are valid:

`pesFatalError = 1`

Fatal error. The queue handler or the transport had to stop execution immediately. Pipex will stop executing the queue handler or transport after receiving a fatal error notification.

Examples: Out of memory, Disk error, Network error

`pesPermanentError = 2`

Permanent error. The error will occur again if the message is re-sent. Pipex will stop executing the queue handler or transport after receiving a permanent error notification.

Example: Message format error

`pesTransientError = 3`

Transient error. The error will likely not occur again if the message is re-sent. Pipex will try to execute the queue handler or transport again and re-send the message.

Examples: Record is locked, Remote host is not responding

`pesWarning = 4`

A warning can indicate that although the processing was successful, an unusual event has occurred. Warnings are ignored by Pipex but listed on the error list.

Examples: Disk space low, Record already exists

*ErrorCode*

A numeric error code

*ErrorDescription*

Description of the error

### Remarks

Using `ReportError()` is the best way to let the Pipex server (and the operators) know about an error. If a queue handler or a transport downloads messages without acknowledging them, and does this several times, Pipex will eventually generate an error message and disable executing the transport or queue handler.

### Error Handling

The `ReportError()` function may be called at any time while the program is connected to the Pipex server. However, `ReportError()` generates a run-time error if the connection is lost, and this error must be trapped. Refer to [Connect\(\)](#) for more information on using error trapping in your code.

**See also**

[Connect\(\) method](#) , [Disconnect\(\) method](#)

**7.2.3.5 NoOp() method**

No operation. This function can be used to prevent the connection from timing out. This method is useful for programs that receive a message but do not acknowledge it for a long time. NoOp() notifies Pipex that the queue handler or transport is still 'alive'.

**Syntax**

*object.Ack(SeqNo As Long)*

*object*            A Pipex API Control object

**Error Handling**

The NoOp() function may be called at any time while the program is connected to the Pipex server. However, NoOp() generates a run-time error if the connection is lost, and this error must be trapped. Refer to [Connect\(\)](#) for more information on using error trapping in your code.

**7.2.4 Polling Pipex****7.2.4.1 Standby() method**

Put a queue handler or transport in standby mode. A queue handler or transport may go to standby mode once all the messages have been downloaded but the queue handler or transport does not want to exit and close the connection to the Pipex server. This is useful for queue handlers or transports that are started on-demand by Pipex and receive messages frequently. Read more about [Polling the Pipex server](#) .

**Syntax**

*object.Standby()*

*object*            A Pipex API Control object

**Usage**

Once in standby mode, the queue handler or transport needs to periodically check the value of the [Notified](#) property. If it is set to true, the program has to try to receive messages. If the program does not receive messages, the connection to the Pipex server may time out.

**See also**

[Notified property](#) , [Receive\(\) method](#)

**7.2.4.2 Notified property**

Set to True if Pipex notified the queue handler or transport that is in standby mode. A notification occurs when there are new messages or when the Pipex server checks if the clients are alive.

---

The property is read-only and may not be used by applications.

### Syntax

*object*.Notified

*object*            A Pipex API Control object

### Remarks

The Notified property may only be read if the Standby() method is called first and the queue handler or transport successfully goes in standby mode.

### Error Handling

Reading this property may fail if the connection is lost or because of other errors. Error trapping must be turned on before reading this property.

### See also

[Standby\(\) method](#)