NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

USING MULTI-USER PROGRAMMING IN IMPORT-EXPORT DEPARTMENT IN COMPANY

Graduation Project
COM-400

Student: Amin Ali Khamis (20010701)

Supervisor: Mr.Umit Soyer

Nicosia- 2005

ACKNOWLEDGMENT

"First, I would like to and foremost to thank Allah whom its accomplishment would not have been possible

Second, I would like to deeply thank my supervisor Prof. Dr UMIT SOYER for his invaluable advice and belief in my work and my self over the course of this graduation project

Third I am deeply indebted to my parents, brothers, and sisters for their love and their support. They have always encouraged me to pursue my interests and ambition throughout life.

Last but in no way least, I would also like to thank all of my friends especially Eng.Imad Ahmed Dahdoh and Eng.Rami Kamel Aljundi they were always available for my assistance throughout this project."

ABSTRACT

The purpose of this project is the development of import and export information system, this program designed to help the user easy and sufficient record the information about the stocks and companies and more details which we are importing and exporting.

All the screens which used in this program will be illustrated later, and the stocks, companies, transportation, payment types, import-export, reports, and help will be briefly discussed in chapter three.

TABLE OF CONTENTS

INTRODUCTION

The aim of the project is development of import and export information system using Delphi programming.

The intended audience for this project includes the follow:

1. Codes-any codes that are responsible for creating and maintaining the data elements and file description specified in this project.
2. Screens-those individuals who wish to view the data collected and processed as part of the development import and export.

In Delphi programming language there are many things that we can use to create any kind of project, but in this project I use some standards components and database to create this project.

In this project the user can easily use the program which makes the information more clear and allow him to find any data he wants, also allow him to insert, delete and update the information about the imported and exported stocks and the companies which related together.

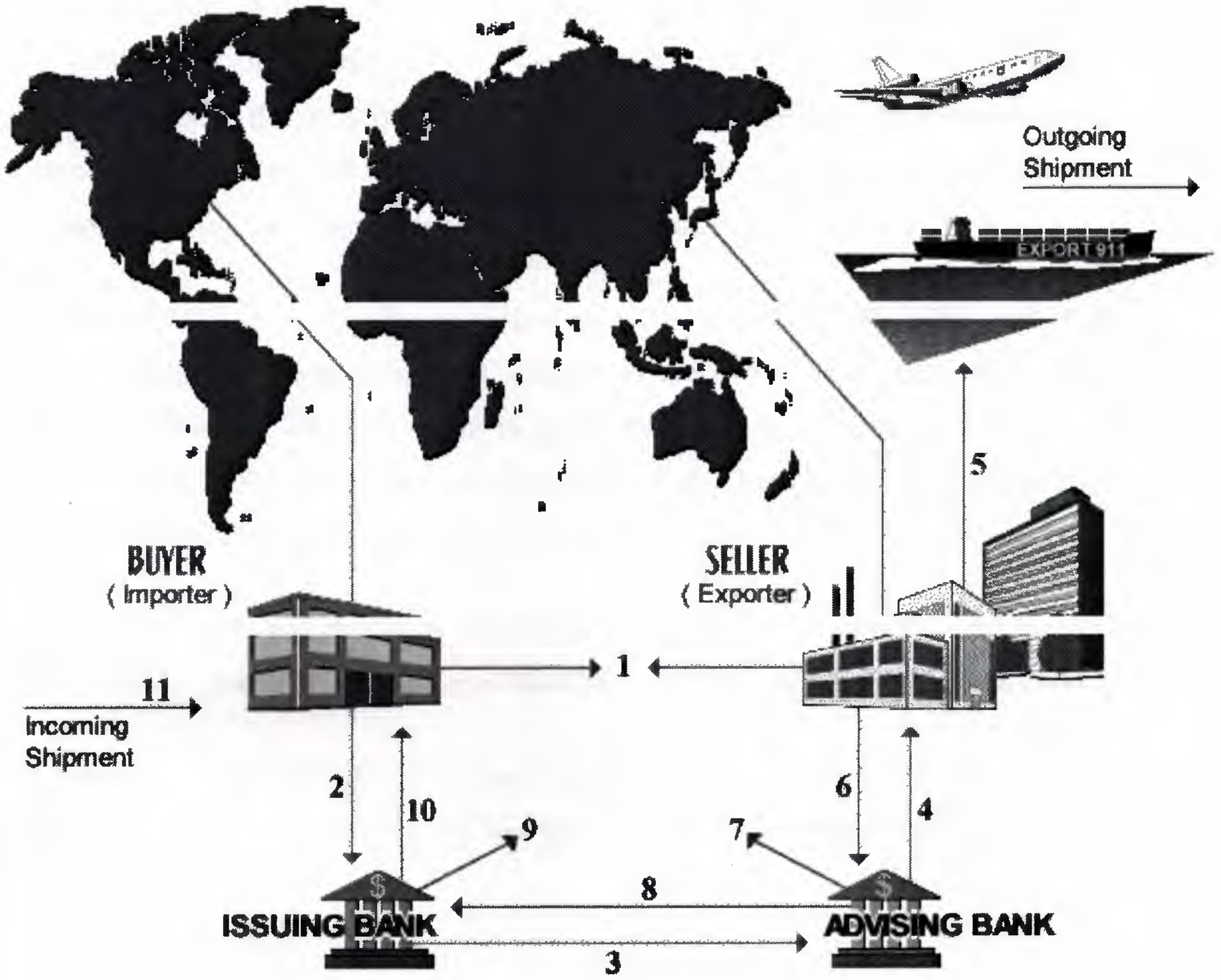
In this program we can use it by a single user or by multi-users by using Linux or UNIX operating system which explained with more details in chapter four.

CHAPTER 1
INTERNATIONAL TRADE

1.1. Export-Import Procedure

1	Seller and Buyer conclude a sales contract, with method of payment usually by letter of credit (documentary credit).
2	Buyer applies to his issuing bank, usually in Buyer's country, for letter of credit in favor of Seller (beneficiary).
3	Issuing bank requests another bank, usually a correspondent bank in Seller's country, to advise, and usually to confirm, the credit.
4	Advising bank, usually in Seller's country, forwards letter of credit to Seller informing about the terms and conditions of credit.
5	If credit terms and conditions conform to sales contract, Seller prepares goods and documentation, and arranges delivery of goods to carrier.
6	Seller presents documents evidencing the shipment and draft (bill of exchange) to paying, accepting or negotiating bank named in the credit (the advising bank usually), or any bank willing to negotiate under the terms of credit.
7	Bank examines the documents and draft for compliance with credit terms. If complied with, bank will pay, accept or negotiate.
8	Bank, if other than the issuing bank, sends the documents and draft to the issuing bank.
9	Bank examines the documents and draft for compliance with credit terms. If complied with, Seller's draft is honored.
10	Documents release to Buyer after payment, or on other terms agreed between the bank and Buyer.

11	Buyer surrenders bill of lading to carrier (in case of ocean freight) in exchange for the goods or the delivery order.



1.2. International Commercial Terms (INCOTERMS):

The INCOTERMS (International Commercial Terms) is a universally recognized set of definitions of international trade terms, such as FOB, CFR and CIF, developed by the International Chamber of Commerce (ICC) in Paris, France. It defines the trade contract responsibilities and liabilities between buyer and seller. It is invaluable and a cost-saving tool. The exporter and the importer need not undergo a lengthy negotiation about the conditions of each transaction. Once they have agreed on a commercial term like FOB, they can sell and buy at FOB without discussing who will be responsible for the freight, cargo insurance, and other costs and risks.

The INCOTERMS was first published in 1936---INCOTERMS 1936---and it is revised periodically to keep up with changes in the international trade needs. The complete definition of each term is available from the current publication---INCOTERMS 2000. The publication is available at your local Chamber of Commerce affiliated with the International Chamber of Commerce (ICC).

Many importers and exporters worldwide are accustomed to and may still use the INCOTERMS 1980, the predecessor of INCOTERMS 1990 and INCOTERMS 2000.

Under the INCOTERMS 2000, the international commercial terms are grouped into E, F, C and D, designated by the first letter of the term (acronym), as follows:

International Commercial Terms
(INCOTERMS)

GROUP	**TERM**	**Stands for**
E	**EXW**	**Ex Works**
F	**FCA**	**Free Carrier**
	FAS	**Free Alongside Ship**

	FOB	**Free On Board**
	CFR	**Cost and Freight**
	CIF	**Cost, Insurance and Freight**
	CPT	**Carriage Paid To**
	CIP	**Carriage and Insurance Paid To**
D	**DAF**	**Delivered At Frontier**
	DES	**Delivered Ex Ship**
	DEQ	**Delivered Ex Quay**
	DDU	**Delivered Duty Unpaid**
	DDP	**Delivered Duty Paid**

In practice, trade terms are written with either all upper case letters (e.g. FOB, CFR, CIF, and FAS) or all lower case letters (e.g. fob, cfr, cif, and fas). They may be written with periods (e.g. F.O.B. and c.i.f.).

In international trade, it would be best for exporters to refrain, wherever possible, from dealing in trade terms that would hold the seller responsible for the import customs clearance and/or payment of import customs duties and taxes and/or other costs and risks at the buyer's end, for example the trade terms DEQ (Delivered Ex Quay) and DDP (Delivered Duty Paid). Quite often, the charges and expenses at the buyer's end may cost more to the seller than anticipated. To overcome losses, hire a reliable customs broker or freight forwarder in the importing country to handle the import routines.

Similarly, it would be best for importers not to deal in EXW (Ex Works), which would hold the buyer responsible for the export customs clearance, payment of export customs charges and taxes, and other costs and risks at the seller's end.

1.2.1. EXW {+ the named place}

Ex Works

Ex means from. Works means factory, mill or warehouse, which is the seller's premises. EXW applies to goods available only at the seller's premises. Buyer is responsible for loading the goods on truck or container at the seller's premises, and for the subsequent costs and risks.

In practice, it is not uncommon that the seller loads the goods on truck or container at the seller's premises without charging loading fee.

In the quotation, indicate the named place (seller's premises) after the acronym EXW, for example EXW Kobe and EXW San Antonio.

The term EXW is commonly used between the manufacturer (seller) and export-trader (buyer), and the export-trader resells on other trade terms to the foreign buyers. Some manufacturers may use the term Ex Factory, which means the same as Ex Works.

1.2.2. FCA {+ the named point of departure}

Free Carrier

The delivery of goods on truck, rail car or container at the specified point (depot) of departure, which is usually the seller's premises, or a named railroad station or a named cargo terminal or into the custody of the carrier, at seller's expense. The point (depot) at origin may or may not be a customs clearance center. Buyer is responsible for the main carriage/freight, cargo insurance and other costs and risks.

In the air shipment, technically speaking, goods placed in the custody of an air carrier is considered as delivery on board the plane. In practice, many importers and exporters still use the term FOB in the air shipment.

The term FCA is also used in the RO/RO (roll on/roll off) services.

In the export quotation, indicate the point of departure (loading) after the acronym FCA, for example FCA Hong Kong and FCA Seattle.

Some manufacturers may use the former terms FOT (Free On Truck) and **1.2.3. FOR** (Free On Rail) in selling to export-traders.

1.2.4. FAS {+ the named port of origin}

Free Alongside Ship

Goods are placed in the dock shed or at the side of the ship, on the dock or lighter, within reach of its loading equipment so that they can be loaded aboard the ship, at seller's expense. Buyer is responsible for the loading fee, main carriage/freight, cargo insurance, and other costs and risks.

In the export quotation, indicate the port of origin (loading) after the acronym FAS, for example FAS New York and FAS Bremen.

The FAS term is popular in the break-bulk shipments and with the importing countries using their own vessels.

1.2.5. FOB {+ the named port of origin}

Free On Board

The delivery of goods on board the vessel at the named port of origin (loading), at seller's expense. Buyer is responsible for the main carriage/freight, cargo insurance and other costs and risks.

In the export quotation, indicate the port of origin (loading) after the acronym FOB, for example FOB Vancouver and FOB Shanghai.

Under the rules of the INCOTERMS 1990, the term FOB is used for ocean freight only. However, in practice, many importers and exporters still use the term FOB in the air freight.

In North America, the term FOB has other applications. Many buyers and sellers in Canada and the U.S.A. dealing on the open account and consignment basis are accustomed to using the shipping terms FOB Origin and FOB Destination.

FOB Origin means the buyer is responsible for the freight and other costs and risks. FOB Destination means the seller is responsible for the freight and other costs and risks until the goods are delivered to the buyer's premises, which may include the import customs clearance and payment of import customs duties and taxes at the buyer's country, depending on the agreement between the buyer and seller.

In international trade, avoid using the shipping terms FOB Origin and FOB Destination, which are not part of the INCOTERMS (International Commercial Terms).

1.2.6. CFR {+ the named port of destination}

Cost and Freight

The delivery of goods to the named port of destination (discharge) at the seller's expense. Buyer is responsible for the cargo insurance and other costs and risks. The term CFR was formerly written as C&F. Many importers and exporters worldwide still use the term C&F.

In the export quotation, indicate the port of destination (discharge) after the acronym CFR, for example CFR Karachi and CFR Alexandria.

Under the rules of the INCOTERMS 1990, the term Cost and Freight is used for ocean freight only. However, in practice, the term Cost and Freight (C&F) is still commonly used in the air freight.

1.2.7. CIF {+ the named port of destination}

Cost, Insurance and Freight ,he cargo insurance and delivery of goods to the named port of destination (discharge) at the seller's expense. Buyer is responsible for the import customs clearance and other costs and risks.

In the export quotation, indicate the port of destination (discharge) after the acronym CIF, for example CIF Pusan and CIF Singapore.

Under the rules of the INCOTERMS 1990, the term CIF is used for ocean freight only. However, in practice, many importers and exporters still use the term CIF in the air freight.

1.2.8. CPT {+ the named place of destination}

Carriage Paid To

The delivery of goods to the named place of destination (discharge) at seller's expense. Buyer assumes the cargo insurance, import customs clearance, payment of customs duties and taxes, and other costs and risks.

In the export quotation, indicate the place of destination (discharge) after the acronym CPT, for example CPT Los Angeles and CPT Osaka.

1.2.9. CIP {+ the named place of destination}

Carriage and Insurance Paid To

The delivery of goods and the cargo insurance to the named place of destination (discharge) at seller's expense. Buyer assumes the import customs clearance, payment of customs duties and taxes, and other costs and risks.

In the export quotation, indicate the place of destination (discharge) after the acronym CIP, for example CIP Paris and CIP Athens.

1.2.10. DAF {+ the named point at frontier}

Delivered At Frontier

The delivery of goods to the specified point at the frontier at seller's expense. Buyer is responsible for the import customs clearance, payment of customs duties and taxes, and other costs and risks.

In the export quotation, indicate the point at frontier (discharge) after the acronym DAF, for example DAF Buffalo and DAF Welland.

1.2.11. DES {+ the named port of destination}

Delivered Ex Ship

The delivery of goods on board the vessel at the named port of destination (discharge), at seller's expense. Buyer assumes the unloading fee, import customs clearance, payment of customs duties and taxes, cargo insurance, and other costs and risks.

In the export quotation, indicate the port of destination (discharge) after the acronym DES, for example DES Helsinki and DES Stockholm.

1.2.12. DEQ {+ the named port of destination}

Delivered Ex Quay

The delivery of goods to the quay (the port) at destination at seller's expense. Seller is responsible for the import customs clearance and payment of customs duties and taxes at the buyer's end. Buyer assumes the cargo insurance and other costs and risks.

In the export quotation, indicate the port of destination (discharge) after the acronym DEQ, for example DEQ Libreville and DEQ Maputo.

1.2.13. DDU {+ the named point of destination}

Delivered Duty Unpaid

The delivery of goods and the cargo insurance to the final point at destination, which is often the project site or buyer's premises, at seller's expense. Buyer assumes the import customs clearance and payment of customs duties and taxes. The seller may opt not to insure the goods at his/her own risks.

In the export quotation, indicate the point of destination (discharge) after the acronym DDU, for example DDU La Paz and DDU Ndjamena.

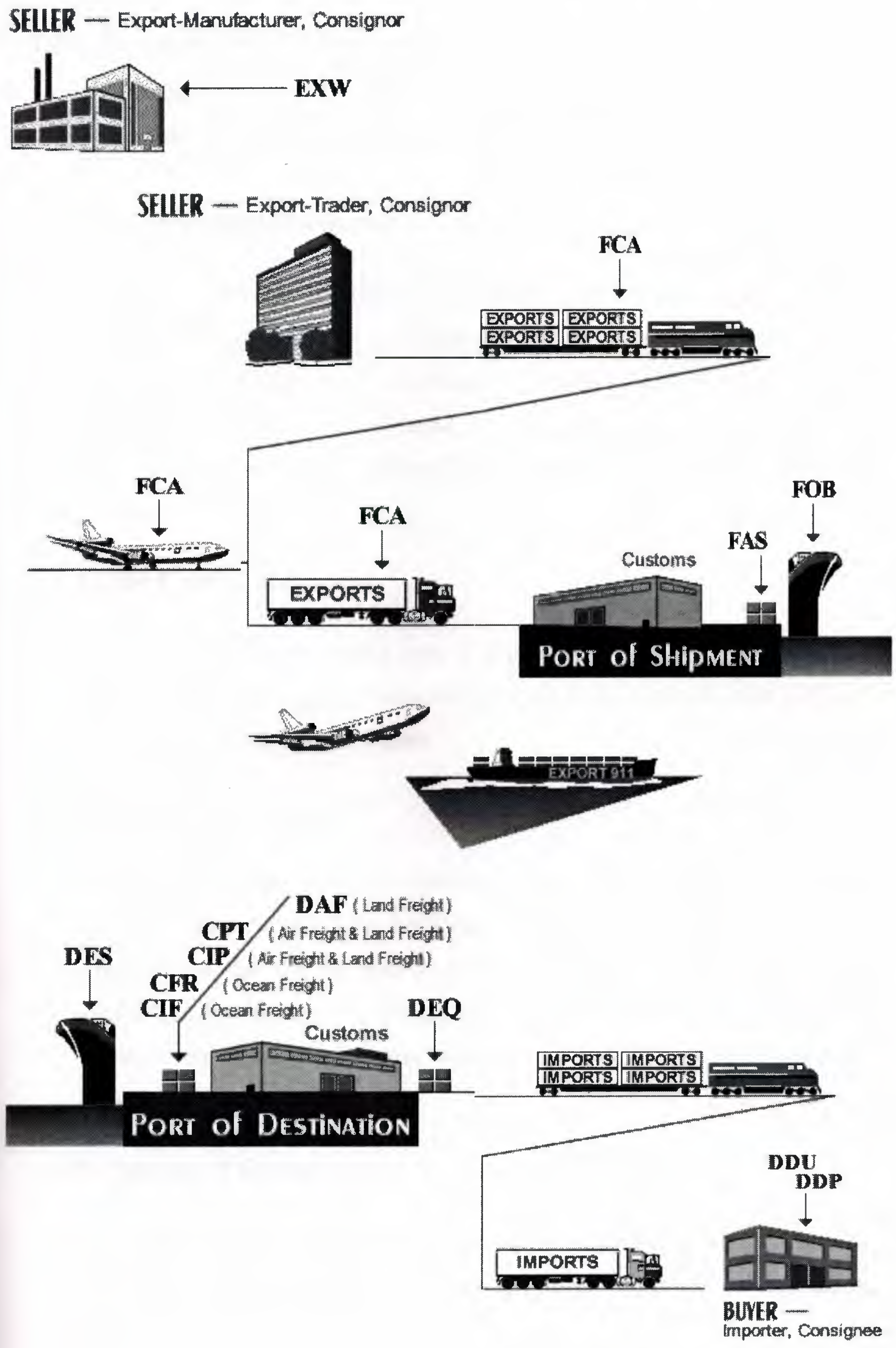
1.2.14. DDP {+ the named point of destination}

Delivered Duty Paid

The seller is responsible for most of the expenses, which include the cargo insurance, import customs clearance, and payment of customs duties and taxes at the buyer's end, and the delivery of goods to the final point at destination, which is often the project site or buyer's premises. The seller may opt not to insure the goods at his/her own risks.

In the export quotation, indicate the point of destination (discharge) after the acronym DDP, for example DDP Bujumbura and DDP Mbabane.

Diagram: International Commercial Terms



1.3. Export Documentary Requirements

1.3.1. Issuance Date of Documents

A document that is dated before the issuing date of letter of credit (L/C) is acceptable, unless otherwise stipulated in the L/C.

1.3.2. The Original Documents

Unless otherwise stipulated in the letter of credit (L/C), a document is considered original if it is produced or appears to have been produced by reprographic (i.e., document reproduced by electronic techniques, for example photocopy), computerized systems, or as carbon copies, provided the document is marked as "Original", and appears to be signed where the signature is needed.

1.3.3. The Copy Documents

A document is considered copy if it is marked as "Copy" or there is no "original" marking, unless otherwise stipulated in the letter of credit (L/C). A copy document need not be signed.

1.3.4. Multiple Documents

A letter of credit (L/C) that requires multiple documents, such as "five (5) copies", "quintuplicate", "five fold" and the like, is satisfied by presenting one original and the rest in copies, except where the document is marked otherwise. the commercial invoice and the packing list both require five copies, meaning one original and four copies of each document are necessary for presentation to the bank.

1.3.5. Signing of Documents

A document may be signed by means of handwriting, stamp, facsimile signature, perforated signature, or by any other electronic or mechanical means.

A copy document need not be signed. Nevertheless, there is no harm in signing the copy(ies) of the commercial invoice and the packing list. In practice, the copies of these two documents are often signed. The letter of credit it is stipulated "signed commercial invoice ...". Therefore, the UVW Exports must sign the original and the copies of the commercial invoice. Since such L/C did not stipulate signing of the packing list, UVW Exports does not have to sign it, but there is no harm done if it is signed.

1.4. Authentication of Documents

Unless otherwise stipulated in the letter of credit (L/C), a document that is required by the L/C to be authenticated, validated, certified, legalized, visaed, or a similar requirement is called for, such condition is deemed to be complied with by any stamp, signature, seal or label on the face of such document that appears to satisfy the requirement.

1.5. Unspecified Issuers or Contents of Documents

When documents other than commercial invoices, transport documents (the bill of lading and the waybill) and insurance documents (the insurance policy and certificate) are required, unless the letter of credit (L/C) stipulates the issuing party and the wording or data content of the documents, the bank will accept them as presented, provided the data content of the documents is consistent with any other stipulated documents presented to the bank.

1.6. Unspecified Documents

The bank will not examine documents not stipulated in the letter of credit (L/C). If unspecified documents are presented, the bank returns them or passes them on without responsibility.

If an L/C contains conditions but does not state the documents to be presented, such conditions are considered not stated and they are disregarded by the bank.

1.7. Standard Cargo Insurance ---

Three Basic Policies (in the Old Cargo Clauses)

1.7.1. Institute Cargo Clauses (All Risks)

The term All Risks is misleading as not all the risks are covered. The All Risks (A.R.) is the broadest form of coverage commonly encountered in exporting. It covers all risks of physical loss or damage from any external causes irrespective of percentage.

If the assured wishes to be covered against the risks of war, strikes, riots, and civil commotions, the insurer deletes the exclusions in the Institute Cargo Clauses and endorses the special clauses, that is, the Institute War Clauses and Institute Strike Clauses, on the insurance policy and the assured pays an additional premium.

1.7.2. Institute Cargo Clauses (With Average)

The With Average (W.A.) is sometimes called the With Particular Average. In insurance parlance, the word "particular" means partial, and the word "average" means loss. As such, the words "with average" and "with particular average" mean including partial loss.

The With Average (W.A.) is a less inclusive form of coverage than the All Risks. It covers against total loss and partial loss caused by the perils of the sea (i.e., the vessel has been stranded, sunk, burnt or been in a collision with other vessels or external substances other than water, such as ice), jettison of cargo, barratry (i.e., negligence, fraud or wrongful acts of the ship's master and/or crew resulting in injury or loss to the ship's owner), and other like perils.

The partial loss, however, is subject to a franchise being written into the policy. The percentage of franchise can be 3% (or other percentage as specified) of the value of the shipment as agreed between the insurer and the assured. If the loss is less than the indicated franchise of 3% (or other percentage as specified) the assured cannot claim the loss. However, if the loss is equal to or more than the indicated franchise, the assured can claim the loss in full amount without any deduction from the insurer.

Instead of a franchise the insurer and the assured may agree on an excess (deductible). The percentage of excess can be 3-10%. If the loss is equal to or less than the indicated excess, the assured bears the loss, that is, cannot claim the loss. However, if the loss is more than the indicated excess, the assured can claim the loss minus the deduction of the percentage of excess specified. In other words, the assured will always shoulder a percentage of the loss regardless of the amount of the loss.

1.7.3. Institute Cargo Clauses (Free of Particular Average)

In insurance parlance, the words "free of" mean the insurer (the insurance company) is not liable for whatever follows the words "free of". The words "particular average" mean partial loss. As such, the words "free of particular average" mean excluding partial loss.

The Free of Particular Average (F.P.A.) is the narrowest form of coverage. It covers against total loss. When partial loss is specifically covered in the policy, it is recoverable from the insurer only if the loss is the result of the carrying vessel being stranded, sunk or burnt, on fire, or in collision.

1.8. Methods and Tools of Payment in Exporting and Importing

The process of exporting is incomplete without receipt of payment. Export income is considered earned only when payment has been received.

1.9. Letter of Credit (L/C)

The most popular and a safer method of payment is by a confirmed irrevocable letter of credit at sight.

1.9.1. Documentary Collections

- Documents Against Payment (D/P)
- Documents Against Acceptance (D/A)

1.10. Cheque and Bank Draft

In exporting to the offshore countries, payment by cheque and bank draft occur more often in a small order, ranging from a few hundred to a couple of thousand U.S. dollars. Cheques and bank drafts are often used in open account and consignment trade arrangements.

Both large and small companies may default in their payments, regardless of the amount involved. In times of economic uncertainty, both large and small companies may go out of business. It is important to receive the cheque or bank draft before releasing the shipment. Unless the integrity of the importer is known, it is very important to wait until the cheque or bank draft has cleared before the shipment. International clearing of cheques and bank drafts takes 3 to 4 weeks usually (except in a sight draft with a paying bank in the seller's country).

Not all cheques and bank drafts are genuine, and not all genuine cheques carry a cash value

1.11. Trade Arrangements Using the Cheque and Bank Draft

1.11.1. Open Account

In an open account trade arrangement, the goods are shipped to a buyer without guarantee of payment. Quite often, the buyer does not pay on the agreed time. Unless the buyer's integrity is unquestionable, this trade arrangement is risky to the seller.

1.11.2. Consignment

In a consignment trade arrangement, the seller ships the goods to the buyer when there is no purchase made. The buyer is obliged to pay the seller for the goods when sold. The seller retains title to the goods until the buyer has sold them.

1.11.3. Cash In Advance (CID)

The cash in advance, which is the safest term of payment, most often is affected using the cheque or bank draft. In some cases, the CID term is paid using the telegraphic transfer (T/T).

1.12. Telegraphic Transfer (T/T)

The telegraphic transfer---cable transfer or wire transfer---is the equivalent of a cash payment that can be credited directly to the seller's account (the name and address of the seller's bank and the seller's bank account number are required by the buyer's bank). It is fast and safe. Unlike a payment by cheque or bank draft, in which the mailing time alone may take several days to few weeks, plus the clearing time of 3 to 4 weeks for a total of about 4 to 6 weeks before the seller may receive the cash, by means of T/T the seller may receive the cash in a few hours or days.

It is important to wait until the T/T has been received before making the shipment, especially when the integrity of the buyer is unknown.

1.13. Combination of Letter of Credit and Telegraphic Transfer

A combination of letter of credit (L/C) and telegraphic transfer (T/T) is a popular means of payment in the undervalue arrangement. The under value is an illegal way of reducing or avoiding the import duties and taxes by under declaring the price of imported goods. It is a sneaky way of bringing the landed cost of imported goods to a competitive level. The under value is being practiced in certain less developed countries, usually involving items whose import duties are relatively high. There is no need to undervalue the goods if the import duty is 10% or less. Sometimes, an item having a 15% rate of duty may not need to be undervalued too, depending on the method of import duty and sales tax calculations in the importing country.

The undervalue arrangement is highly risky. To avoid trouble the exporter should refrain from using this arrangement. Governments do not encourage exports by undervalue. If an exporter does not violate the foreign exchange control and tax laws of the exporting country and international laws such as copyright and patent, the government of the exporting country usually will not step into the exporter's way in the undervalue arrangement.

The undervalue arrangement uses two sets of documents. For example, an importer contracted 1,000 pieces of product X at FOB US$8 each for a total of US$8,000. The importer may want to declare 25% only (10% to 50% of contract price is declared usually in the undervalue arrangement) or at US$2 each for a total of US$2,000. One set of documents will show 1,000 pieces of product X at US$2 each for a total of US$2,000, while the other set shows the true value.

The importer opens an L/C for US$2,000 and remits the US$6,000 balance by T/T. Following the foreign exchange control procedures on exports, the exporter must surrender a total of US$8,000 inward remittances to the government. While at the destination port, the importer pays the duties and taxes based on US$2,000, plus the ancillary expenses required in the arrangement. If the importer is caught at the port of destination, shipments may be seized by the customs.

The importer has to buy the dollar from the black market and remit it by T/T through a third country. Most often the T/T will not reach the exporter on the agreed time. Quite often, the shipment date arrives before the T/T reaches the exporter.

The undervalue arrangement hinges on mutual trust between exporter and importer. The importer has to be very careful because there is a danger that the exporter may run off after receiving the T/T. In the event of a sour relationship, the importer may run the risk of being blackmailed by the exporter through threat of exposing the private arrangement.

With the growing free trade around the world, the undervalue practice is diminishing.

1.14. Commercial Invoice

The commercial invoice is a record or evidence of transaction between the exporter and the importer. It is similar to an ordinary sales invoice, except some entries specific to the export-import trade are added

1.15. Specific Language Requirements in the Commercial Invoice

Certain importing countries may require that the commercial invoice and the packing list be made out in, or translated to, the language of the importing country, for example, in French for shipment to France, in Italian to Italy, and in Spanish to Mexico and Venezuela.

1.16. Declaration on Commercial Invoice

The declaration on the commercial invoice for some countries must be in a specified wording. The exporter may check the wording with the customs broker, the government external trade department, or the foreign government trade office concerned in the exporting country.

The content of a typical declaration includes a sworn statement from the exporter indicating that the goods in question are manufactured in the exporting country, and that the amount shown in the invoice is the true and correct value.

1.17. Certification and/or Legalization of Commercial Invoice

The letter of credit (L/C) from certain importing countries, in particular from the Middle East, requires the certification and/or legalization of the commercial invoice.

The certification, which usually is performed by the local Chamber of Commerce of the exporting country, is to confirm that the invoice and declaration (in the invoice) are correct.

The legalization, which is done by The Consulate or The Commercial Section of the Embassy of the importing country, is to verify that the invoice is correct.

The certification and legalization are most often satisfied with a stamp or a seal on the invoice and payment of a fee. The processing time may take one week.

1.18. Corrections or Changes in the Commercial Invoice

Any visible corrections or changes made in the commercial invoice must be initialed. In practice, the initial usually is done using a rubber stamp bearing the word "CORRECTION".

1.19. signature and/or stamp

The commercial invoice and packing list need not be signed, unless otherwise stipulated in the letter of credit (L/C). In practice, the original and the copy of the commercial invoice and packing list are often signed.

1.20. Description of Goods

The description of the goods in the commercial invoice must correspond with the description in the letter of credit (L/C). In all other documents, the description can be in general terms provided it is not inconsistent with the description in the L/C.

For example, the description of goods in the commercial invoice should be "'ABC' Brand Pneumatic Tools, 1/2" drive, complete with hose and quick couplings", the description in all other documents can be in general terms like "'ABC' Brand Pneumatic Tools". However, if any other documents indicate "'ABC' Brand Air Tools", it is inconsistent with the description in the L/C for using the word "air", which should be "pneumatic", despite both terms being technically the same. Consequently, there is a discrepancy and the bank will reject the documents.

1.20.1. Quantity

If the letter of credit (L/C) does not stipulate the quantity in a stated number of units (i.e., it does not state in units such as piece, set, box, dozen, or gross), or unless the L/C stipulates that the quantity of the goods specified must not be exceeded or reduced, a tolerance of 5% more or 5% less quantity is permitted, provided the total amount does not exceed the amount of the L/C.

In the L/C the stated quantity is 100 Sets, thus the quantity in the invoice must be 100 Sets. If such sample L/C does not state the quantity, the UVW Exports can ship between 95 sets and 100 sets of pneumatic tools, but not over 100 sets as the total amount will exceed the L/C amount of US$25,000. If such L/C does not state the quantity and the L/C amount is US$26,250 or more, the exporter may ship between 95 and 105 sets.

If the L/C quantity is indicated using the words "about", "approximately", "circa" or similar expressions, the quantity in the invoice cannot exceed 10% more or 10% less than the quantity indicated in the L/C. For example, if the L/C quantity is "about 100 sets", the quantity in the invoice can be any quantity between 90 sets and 110 sets, provided the total amount does not exceed the amount of the L/C.

1.20.2. Unit Price

If the letter of credit (L/C) unit price is indicated using the words "about", "approximately", "circa" or similar expressions, the unit price in the invoice cannot exceed 10% more or 10% less than the unit price indicated in the L/C. For example, if the L/C unit price is "about US$250", the unit price in the invoice can be any unit price between US$225 and US$275, provided the total amount does not exceed the amount of the L/C.

1.20.3. Amount

Unless otherwise stipulated in the letter of credit (L/C), the amount must not exceed the amount permitted by the L/C. If the L/C amount is indicated using the words "about", "approximately", "circa" or similar expressions, the amount of the invoice cannot exceed 10% more or 10% less than the amount indicated in the L/C. For example, if the L/C amount is "approximately US$10,000", the amount of invoice can be any amount between US$9,000 and US$11,000.

1.21. Customs Brokers

The customs broker---broker or customhouse broker or customs house broker---is an individual or company licensed to clear export and import goods through customs.

In general, the role of brokers is the same worldwide. Besides clearing of goods through customs, other export services a broker renders include booking of space for ocean, air and land freight, canvassing and providing the freight cost, and preparation of export documents and sending them to the bank for negotiation. In certain countries, it is a business practice that exporters prepare their own export documents. The broker also renders the forwarding services as a freight forwarder.

Few large exporters have their own in-house licensed customs broker.

The broker basically handles any export goods. At times, it is necessary to retain the service of a broker experienced in the exporter's line of product and the port of destination.

It is important to select a reliable customs broker. The exporter normally has to sign an authorization paper allowing the broker to handle the customs declaration. In case the broker commits an error, the exporter is held liable.

The brokerage fee varies from country to country. The broker may collect a basic service fee on top of other charges, such as documentation charges and port fees. In certain countries, the broker collects a uniform base fee, plus a small percentage of the value of shipment. In a country where the brokerage fee is not regulated, it may vary considerably among brokers. The exporter must check with different brokers in order to get the best offer. It is important to request the broker to show the breakdown of charges on the billing.

A good and honest broker can help new exporters with certain export routines and help them save money.

1.22.. Freight Forwarders or Consolidators

The freight forwarder---forwarder---is an individual or firm who renders cargo delivery services. In domestic (local) freight forwarding, it is the delivery of goods usually from the exporter's premises to the local customs in exporting, and vice versa in importing. The customs broker also renders local freight forwarding for exporters and importers.

International (foreign) freight forwarding is the delivery of goods from the exporter's premises (or from the port or point of origin) to the port or point of destination (or to the importer's premises).

The freight consolidator---consolidator or group age operator---is an individual or firm who accepts less than container load (LCL) shipments from individual shippers, and then combines them for delivery to the carrier in full container load (FCL) shipment.

The services of a forwarder are usually available in a consolidator, and the forwarder often engages in the consolidation of cargo. Hence, the term forwarder is often used synonymously with the consolidator.

The forwarder provides a wide range of services. Besides all of the export services available from a customs broker, the forwarder may also arrange for the insurance, export packing and trucking.

The forwarder usually receives the forwarder's charges from the exporter. In addition, it may receive a commission from the carrier---Freight Company (ocean, air, truck and rail).

In the ocean shipment, the forwarder may 'buy' the shipping space, in a special arrangement with the carrier, and 'resell' the space to individual shippers, instead of receiving a commission. In such an arrangement, the forwarder functions as an independent distribution or logistical company known as the NVOCC (no vessel operating common carrier) or NVO (no vessel owner or no vessel owning carrier), or commonly referred to as the ocean freight consolidator.

CHAPTER 2

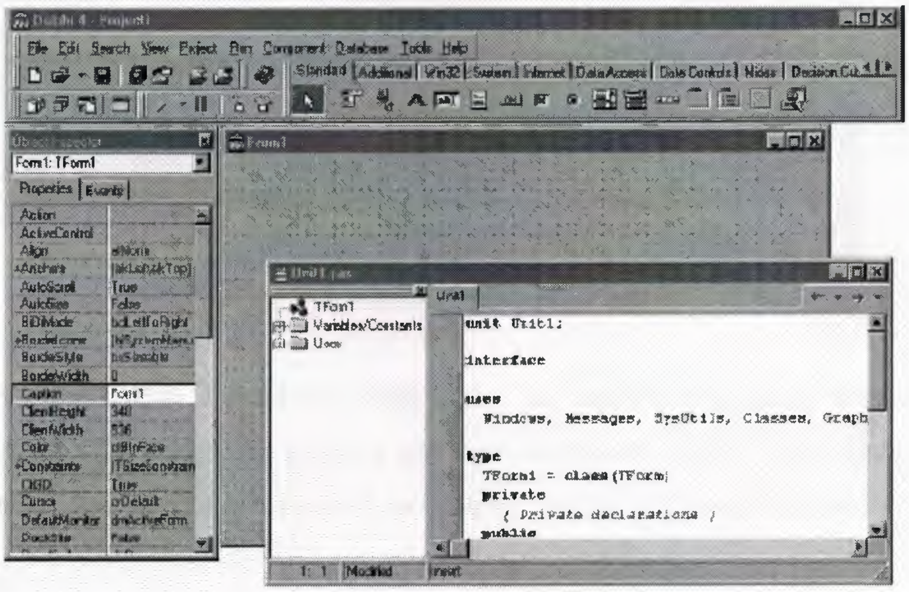
INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

2.1 INTODUCTION

One of the most difficult aspects of learning how to use a new programming environment is finding your way around: getting to know the basic menu structure, what all the options do, and how the environment works as a whole.

2.2 The Delphi IDE

Definition: Integrated Development Environment. This is the user interface (GUI) where you can design, compile and debug your Delphi projects.
So, without further ado, take a look at Figure 1.1 and let's get on with it. if you have used Delphi before, you might find this chapter elementary.


File Edit Search View Project Run Component Database Tools Help
Standard Additional Win32 System Internet Data Access Data Controls Midas Decision Cube
Object Inspector
Form1
Unit1.pas
TForm1
Variables/Constants
Uses
Unit1
1: 1
Modified
Insert

Form1: TForm1

Properties | Events

Action	
ActiveControl	
Align	alNone
+Anchors	[akLeft,akTop]
AutoScroll	True
AutoSize	False
BiDiMode	bdLeftToRight
+BorderIcons	[biSystemMenu,
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	340
ClientWidth	536
Color	clBtnFace
+Constraints	[TSizeConstrain
Ctl3D	True
Cursor	crDefault
DefaultMonitor	dmActiveForm
DockSite	False

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graph

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
```

Figure 2.1 The Delphi IDE

1. The Delphi IDE consists of these main parts:
2. The main menu and toolbars l
3. The Component palette l
4. The Form Designer l
5. The Code Editor l
6. The Object Inspector l
7. The Code Explorer l
8. The Project Manage

2.3 A Quick Look at the Delphi IDE

This section contains a quick look at the Delphi integrated development environment (IDE). Because you are tackling Windows programming, I'll assume you are advanced enough to have figured out how to start Delphi. When you first start the program, you are presented with both a blank form and the IDE, as shown in Figure 2.2.

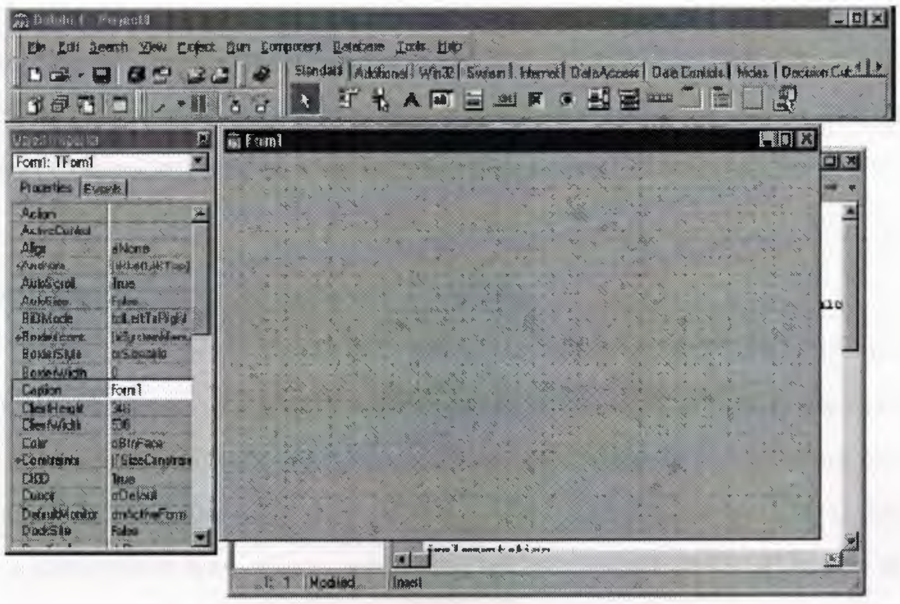

File Edit Search View Project Run Component Database Tools Help
Standard Additional Win32 System Internet DataAccess Data Controls Midas Decision Cube
Form1: TForm1
Properties Events
Form1

Figure2.2 The Delphi IDE and the initial blank form

The Delphi IDE is divided into three parts. The top window can be considered the main window. It contains the toolbars and the Component palette. The Delphi toolbars give you one-click access to tasks such as opening, saving, and compiling projects.

The Component palette contains a wide array of components that you can drop onto your forms. (Components are text labels, edit controls, list boxes, buttons, and the like.) For convenience, the components are divided into groups. Go ahead and click on the tabs to explore the different components available to you. To place a component on your form, you simply click the component's button in the Component palette and then click on your form where you want the component to appear. When you are done exploring, click on the tab labeled Standard, because you'll need it in a moment.

2.4 The Delphi Workspace

The main part of the Delphi IDE is the workspace. The workspace initially displays the Form Designer. It should come as no surprise that the Form Designer enables you to create forms. In Delphi, a form represents a window in your program. The form might be the program's main window, a dialog box, or any other type of window. You use the Form Designer to place, move, and size components as part of the form creation process.

Hiding behind the Form Designer is the Code Editor. The Code Editor is where you type code when writing your programs. The Object Inspector, Form Designer, Code Editor, and Component palette work interactively as you build applications.

2.5 The Delphi Main Menu and Toolbar.

The Delphi main menu has all the choices necessary to make Delphi work. Because programming in Delphi is a highly visual operation, you might not use the main menu as much as you might with other programming environments. Still, just about anything you need is available from the main menu if you prefer to work that way. The Delphi toolbars provide a convenient way of accomplishing often-repeated tasks. A button is easier to locate than a menu item, not to mention that it requires less mouse movement. The Delphi main window toolbars are illustrated in Figure 2.3.

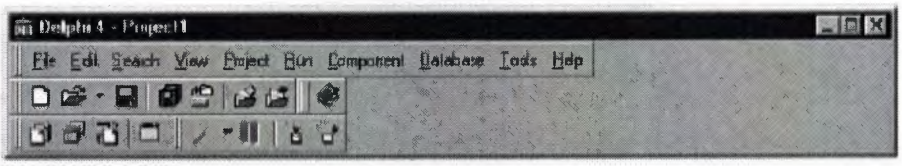

Delphi 4 - Project1
File Edit Search View Project Run Component Database Tools Help

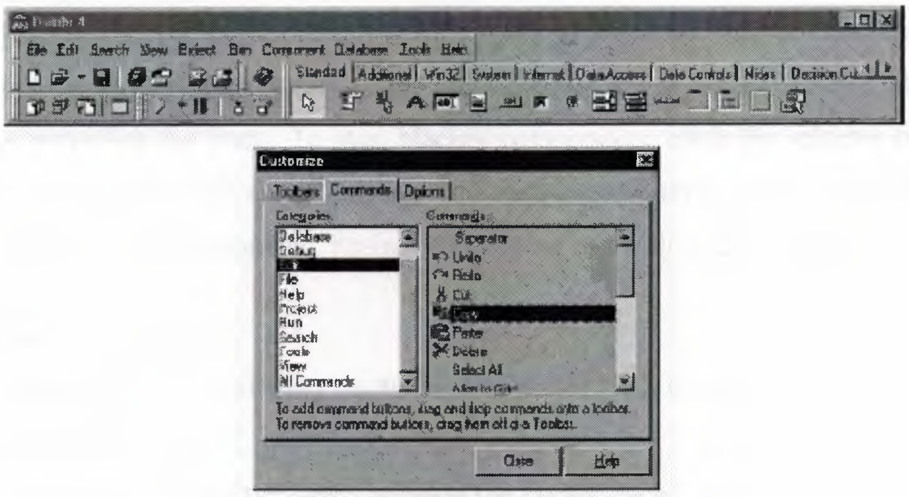
Figure 2.3 The Delphi main window

Delphi enables you to add buttons to the toolbar, remove buttons, and rearrange buttons however you see fit. To configure a toolbar, right-click on the toolbar to display the context menu. Choose Customize from the context menu. When you choose this menu item, the Customize dialog box is displayed.

The Customize dialog box contains three tabs:

The first tab, Toolbars, shows you the toolbars available with a check mark next to toolbars that are currently visible. You can add or remove existing toolbars or reset the toolbars to their original default settings.

The second tab, labeled Commands, shows all the available toolbar buttons. To add a button to the toolbar, just locate its description in the Commands list box and drag it to the place you want it to occupy on any toolbar. To remove a button from a toolbar, grab it and drag it off the toolbar. It's as simple as that. Figure 1.4 shows the act of adding a button to a toolbar. If you really make a mess of things, simply go back to the Toolbars page and click the Reset button. The toolbar will revert to its default settings.


Delphi 4
File Edit Search View Project Run Component Database Tools Help
Standard Additional Win32 System Internet DataAccess DataControls Midas Decision Cube
Customize
Toolbars Commands Options
Categories:
Database
Debug
Edit
File
Help
Project
Run
Search
Tools
View
All Commands
Commands:
Separator
Undo
Redo
Cut
Copy
Paste
Delete
Select All
Align to Grid
Close
Help

To add command buttons, drag and drop commands onto a toolbar.
To remove command buttons, drag them off of a Toolbar.

Figure 2.4 customizing the toolbar

The third tab, Options, contains options such as whether the tooltips are displayed and how they are displayed.

2.6 Using the Component Palette

The Delphi Component palette is used to select a component or other control (such as an ActiveX control) in order to place that control on a form. The Component palette is a multipage window. Tabs are provided to enable you to navigate between pages.

Placing a component on a form is a two-step process. First, go to the Component palette and select the button representing the component you want to use.

Then click on the form to place the component on the form. The component appears with its upper-left corner placed where you clicked with the mouse.

2.7 Placing Multiple Copies of a Component

So far you have placed only one component at a time on a form. You can easily place multiple components of the same type without selecting the component from the Component palette each time. To place multiple components on a form, press and hold the Shift key as you select the component from the Component palette. After you select the component, you can release the Shift key.

The component's button on the Component palette will appear pressed and will be highlighted with a blue border. Click on the form to place the first component. Notice that the button stays pressed on the Component palette. a new component will be placed each time you click the form. To stop placing components, click the selector button on the Component palette (the arrow button). The component button pops up to indicate that you are done placing components. Seeing is believing , so follow these steps:

1. Create a new project.

2. Press and hold the Shift key on the keyboard and click the Label component button in the Component palette.

3. Click three times on the form, moving the cursor each time to indicate where you want the new component placed. A new Label is placed on the form each time you click.

4. Click the arrow button on the Component palette to end the process and return to form design mode.

2.8 About Delphi Forms

Before I continue with the discussion about the Delphi IDE, I need to spend some time explaining forms. You have seen several forms in action as you have worked through this chapter. You need some more background information on forms.

2.8.1 Main Window Forms

Forms are the main building block of a Delphi application. Every GUI application has at least one form that serves as the main window. The main window form might be just a blank window, it might have controls on it, or it might have a bitmap displayed on it. In a typical Windows program, your main window would have a menu. It might also have decorations such as a toolbar or a status bar. Just about anything goes when creating the main window of your application. Each application is unique, and each has different requirements.

2.8.2 Creating the Main Window Form

First you'll create the main window form. The main window for an MDI application must have the FormStyle property set to fsMDIForm. You also need to add a menu to the application, as well as File Open and File Save dialog boxes. Follow these steps:

1. Start Delphi and choose File | New Application from the main menu.
2. For the main form, change the Name property to MainForm.
3. Change the Caption property to Picture Viewer.
4. Change the Height to 450 and the Width to 575 (or other suitable values)
5. Change the FormStyle to fsMDIForm.

Now you've got the main part of the form done. Next you'll add a menu to the form, you will take the easy route to creating a menu. To do that, you can take advantage of a Delphi feature that enables you to import a predefined menu, as follows:

1. Click the Standard tab of the Component palette and click the MainMenu button.
2. Click on the form to place a MainMenu component on the form. It doesn't matter where you place the component because the icon representing the menu is just a

placeholder and won't show on the form at runtime. This is how nonvisual com-ponents appear on a form.

3. Change the Name property to MainMenu.

4. Double-click the MainMenu component. The Menu Designer is displayed.

5. Place your cursor over the Menu Designer and click your right mouse button. Choose Insert from Template from the context menu. The Insert Template dialog box appears.

6. Choose MDI Frame Menu and click OK. The menu is displayed in the Menu Designer.

7. Click the system close box on the Menu Designer to close it.

Now you should be back to the main form. You can click on the top-level items to see the full menu. Don't click on any menu subitems at this point--you'll do that in a minute.

Now you need to prepare the File Open and File Save dialog boxes:

1. Click the Dialogs tab on the Component palette. Choose an Open Picture Dialog component and place it on the form.

The Open Picture Dialog component's icon can be placed anywhere on the form.

2. Change the Name property of the Open dialog box to Open Picture Dialog.

3. Change the Title property to open a Picture for Viewing.

4. Add a Save Picture Dialog component.

5. Change the Name property of the component to Save Picture Dialog and the Title property to Save a Picture.

Your form should now look like the one shown in Figure 1.5.

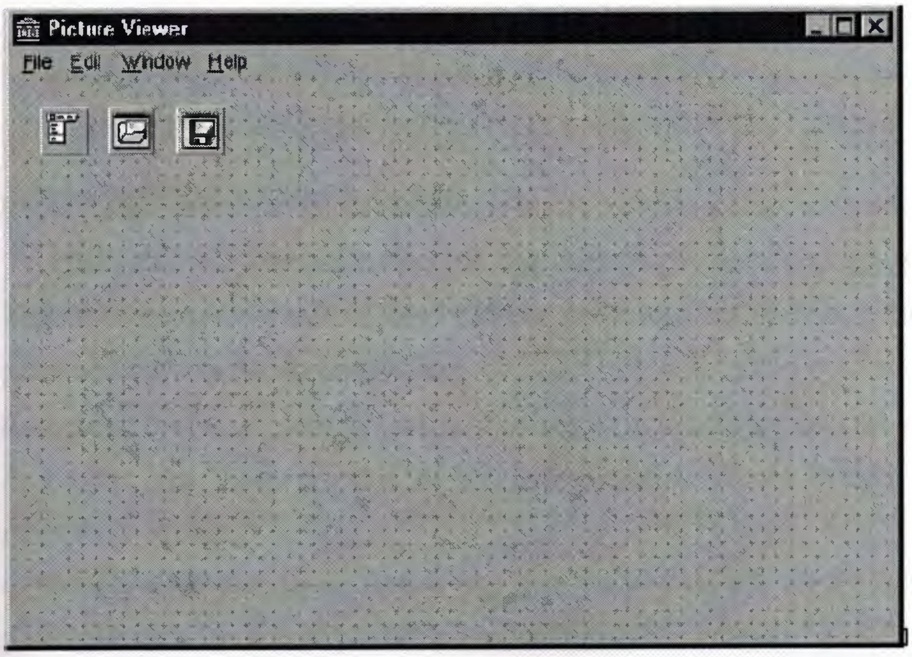


Figure 2.5 Form designed

2.8.3 Dialog Box Forms

Forms are also used for dialog boxes. In fact, to the user there is no difference between a Delphi form acting as a dialog box and a true dialog box.Dialog boxes usually have several traits that distinguish them from ordinary windows:

- Dialog boxes are not usually sizable. They usually perform a specific function, and sizing of the dialog box is neither useful nor desirable.

- Dialog boxes almost always have an OK button. Some dialog boxes have a button labeled Close that performs the same task. A simple dialog box such as an About dialog box typically has only the OK button. Dialog boxes can also have a Cancel button and a Help button.

- Dialog boxes typically have only the system close button on the title bar. They do not usually have minimize and maximize buttons.

2.8.4 Creating a Dialog Form

First you'll add a button to the form that displays the about dialog box:

1. Bring the main form into view. Choose the Button component from the Component palette and drop a button on the form.

2. Arrange the two buttons that are now on the form to balance the look of the form.

3. Change the Name property of the new button to About Button and the Caption property to about.

4. Double-click the About Button you just created on the form. The Code Editor is displayed with the cursor placed in the event-handler function. Add this line of code at the cursor:

about Box. Show Modal, You haven't actually created the About box yet, but when you do you'll name it About Box, so you know enough to type the code that will display the About box.

2.9 A Multiple-Form Application

To illustrate how Delphi uses units, you can create an application with multiple forms. You'll create a simple application that displays a second form when you click a button:

1. Create a new project by choosing File | New Application from the main menu.

2. Change the Name property to MainForm and the Caption property to Multiple Forms Test Program.

3. Save the project. Save the unit as Main and the project as Multiple.

4. Now place a button on the form. Make the button's Name property ShowForm2 and the Caption property Show Form 2.

5. Choose File | New Form from the main menu to create a new form. At this point, the new form has a name of Form1 and is placed exactly over the main form. You want the new form to be smaller than the main form and more or less centered on the main form.

6. Size and position the new form so that it is about 50 percent of the size of the main form and centered on the main form. Use the title bar to move the new form. Size the form by dragging the lower-right corner.
7. Change the new form's Name property to SecondForm and the form's Caption property to A Second Form.
8. Choose File | Save from the main menu.
9. Choose a Label component and drop it on the new form.

2.9.1Adding Units

Rather than having Delphi prompts you to add a unit to your uses list, you can add units yourself. You can manually type the unit name in the uses list for the form, or you can choose File | Use Unit from the main menu. When you choose the latter method, the Use Unit dialog box is displayed, as shown in Figure 2.6. The Use Unit dialog box shows a list of available units. Choose the unit you want to add and click OK. Delphi will add the unit to the current forms uses list.

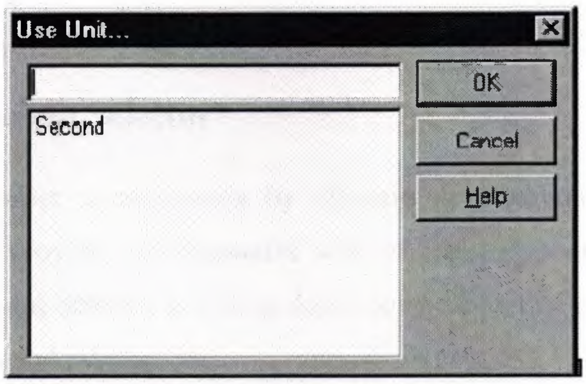


Figure 2.6 The Use Unit dialog box

2.9.2 Some Key Properties for Forms

The TForm class has a lot of properties. Some of these properties are obscure and rarely used others are widely used. I'll touch on the some widely used properties here.

- **Font** The Font property specifies the font that the form uses. The important issue to understand here is that the form's font is inherited by any components placed on the form. This also means that you can change the font used by all components at one time by changing just the form's font.

- **FormStyle** This property is usually set to fsNormal. If you want a form to always be on top, use the fsStayOnTop style. MDI forms should use the fsMDIForm style and MDI child forms should use the fsMDIChild style.

2.10 The Object Inspector

An integral part of the Delphi IDE is the Object Inspector. This window works with the Form Designer to aid in the creation of components.

The Object Inspector is where you set the design-time properties that affect how the component acts at runtime. The Object Inspector has three main areas:

- The Component Selector
- The Properties page
- The Events page

2.10.1The Component Selector

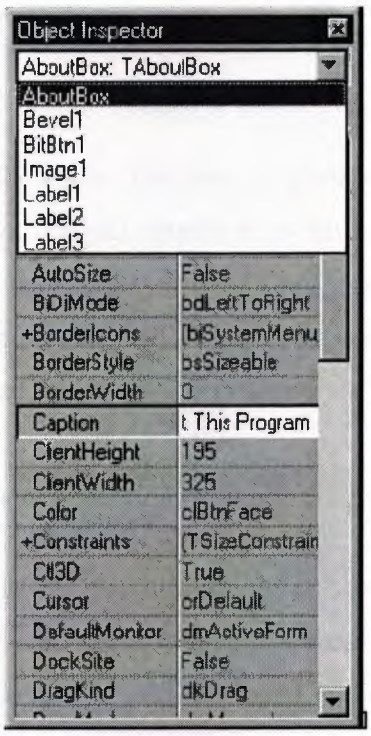
Normally, you select a component by clicking the component on a form. The Component Selector provides an alternative way of selecting a component to view or modify. The Component Selector is a drop-down combo box that is located at the top of the Object Inspector window.

The Component Selector displays the name of the component and the class from which it is derived. For example, a memo component named Memo would appear in the Component Selector as

Memo: TMemo

The class name does not show up in the drop-down list of components, it only appears in the top portion of the Component Selector. To select a component, click the drop-down button to reveal the list of components and then click the one you want to select.

After you select a component in the Component Selector, the component is selected on the form as well. The Properties and Events tabs change to display the properties and events for the selected component. Figure 2.7 shows the Object Inspector with the Component Selector list displayed.



Object Inspector

AboutBox: TAboutBox

AboutBox
Bevel1
BitBtn1
Image1
Label1
Label2
Label3

AutoSize	False
BiDiMode	bdLeftToRight
+BorderIcons	[biSystemMenu
BorderStyle	bsSizeable
BorderWidth	0
Caption	t This Program
ClientHeight	195
ClientWidth	325
Color	clBtnFace
+Constraints	(TSizeConstrain
Ctl3D	True
Cursor	crDefault
DefaultMonitor	dmActiveForm
DockSite	False
DragKind	dkDrag

Figure 2.7 the component selector list

2.10.2The Properties Page

The Properties page of the Object Inspector displays all the design-time properties for the currently selected control. The Properties page has two columns: The Property column is on the left side of the Properties page and shows the property name, the Value column is on the right side of the Properties page and is where you type or select the value for the property.

Properties can be integer values, enumerations, sets, other objects, strings, and other types.

The Object Inspector deals with each type of property according to the data type of the property. Delphi has several built-in property editors to handle data input for the property. For example, the Top property accepts an Integer value. Because the Integer type is a basic data type, no special handling is required, so the property editor is fairly basic. The property editor for this type of property enables you to type a value directly in the Value column for integer properties such as Top, Left, Width, and Height.

2.10.3 The Events Page

The Events page lists all the events that the component is designed to handle. Using the Events page is pretty basic. Delphi creates an event-handling function for you with all the parameters needed to handle that event. The Code Editor is displayed and the cursor is placed in the event handler. All you have to do is start typing code. The name of the function is generated based on the Name property of the component and the event being handled. If, for example, you have a button named OKBtn and are handling the OnClick event, the function name generated would be OKBtnClick.

You can let Delphi generate the name of the event-handling function for you or you can provide the function name for Delphi to use. The Code Editor is displayed, and so is the event-handling function, complete with the name you supplied. After you create an event-handling function for a component, you can use that event handler for any component that handles the same event. Sometimes it's convenient to have several buttons use the same OnClick event.

2.11 Code Templates

This feature lets you insert one of the predefined code templates, such as a complex statement with an inner begin...end block. Code templates must be activated manually, by pressing Ctrl+J to show a list of all of the templates. You can add custom code templates, so that you can build your own shortcuts for commonly used blocks of code. For example, if you use the MessageDlg function often, you might want to add a template for it. To modify templates, go to the Source Options page of the Editor Options dialog box, select Pascal from the Source File Type list, and click the Edit Code Templates button. Doing so opens the new Delphi Code Templates dialog box. At this point, click the Add button, type in a new template name (for example, **mess**), type a description, and then add the following text to the template body in the Code memo control:

```
MessageDlg ('|', mtInformation, [mbOK], 0);
```

Now, every time you need to create a message dialog box, you simply type **mess** and then press Ctrl+J, and you get the full text.

2.12 Writing Code for the File, Open and File, Save As Menu Items.

You haven't created the MDI child form yet, but you know enough about it to write the code for the menu handlers. Keep in mind that the application won't compile until you create the MDI child form. Here you go:

1. On the main form, choose File | Open from the menu. An event handler is created for that menu item and the Code Editor is displayed.

2. Type code so that the event handler looks like this:

```
Procedure TMainForm.Open1Click (Sender: TObject);
Var
Child: TChild;
Begin
if OpenPictureDialog.Execute then begin
Child: = TChild.Create (Self);
With Child.Image.Picture do begin
LoadFromFile (OpenPictureDialog.FileName);
Child.ClientWidth:= Width;
Child.ClientHeight:= Height;
end;
Child.Caption:= ExtractFileName (OpenPictureDialog.FileName);
Child.Show;
end;
end;
```

This code first executes the File Open dialog box and gets a filename. If the Cancel button on the File Open dialog box is clicked, the function returns without doing anything more. If the OK button on the File Open dialog box is clicked, a new TChild object is created (TChild will be the name of the MDI child class you'll create later).

2.13 Writing Code for the Window Menu

Now you can add code to the Window menu. This part is simple:

1. Switch back to the form by pressing F12. Choose Window | Tile from the form's menu.

2. You need to enter only a single line of code for the event handler. The finished event handler will look like this:

```
Procedure TMainForm.Tile1Click (Sender: TObject);
Begin
Tile;
end;
```

3. Switch back to the form and repeat the process for Window | Cascade. The finished function looks like this:

```
Procedure TMainForm.Cascade1Click (Sender: TObject);
Begin
Cascade;
end;
```

4. Repeat the steps for the Window | Arrange All menu item. The single line of code to add for the function body is the following:

```
ArrangeIcons; .
```

CHAPTER 3
Import-Export system

3.1. Database Structure:

First thing as we know Delphi's support for database applications is one of the key feathers of the programming environment. Many programmers spend most of their time writing data-access code, which needs to be the most robust portion of a database application. You can create very complex database applications, starting from a blank form or one generated by Delphi's database from wizard. On computer, permanent data including database data is always stored in files. There are several techniques you can use to accomplish this storage. Delphi can use both approaches, or more precisely, you always refer to a database with its name, which is a sort of a nickname of a database but this reference can be to a database file or to a directory containing files with tables.

3.2. Define Relationships Between Tables:

When we create a relationship, the related fields don't have the same names. However, related field must have the same data type unless the primary key filed is an AutoNumber field. We can match an AutoNmber field with a number field only if the fieldsize property of both of the matching fields is the same. Here we can see the relationships between the tables of this project as shown bellow:

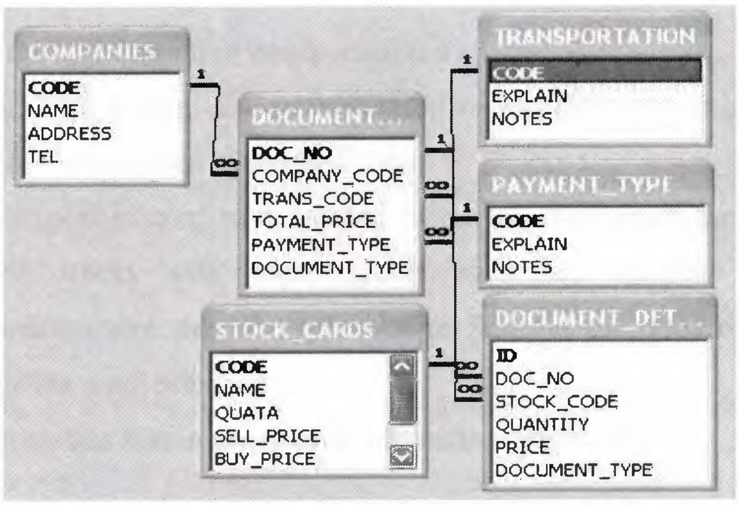


Figure 3.1 relationships between tables.

3.3. Delphi database components:

As we have seen in chapter one Delphi includes a number of components related to database. The data access page pf the component palette contains components used to interact to database. To access s database in Delphi you generally need a data source, identified by data source component. The data source component, however, does not indicate the data directly, it refers to a data set component this can be tables (as in this project) or some other custom data set. As soon as you have placed a table component on the form, you can use the data sets property of the data source component to refer to it. for this property, the object inspector lists available data set of the current form or of other forms connected with the current one(using the file >used form command).

3.4. Layout of the Application:

3.4.1. Main menu screen:

It consists of six buttons. Each button has a specific mission, and these missions will be explaining as follow:

1. File Button: we can use this button to exit from the program.
2. Stock cards button: this button has two sub buttons, these are add new card with its properties, and we can chose one of these sub buttons to do what we want.
3. Company's button: this button used when we want to add new company with its properties, also this button has another sub button which is allow us to update the company's information and research about the company we need.
4. Transportation button: we us this button to insert a new transportation way and update the information of transportation way.
5. Payment-Type button: also in this button we can add a new payment type and update the information of payment type.
6. Import-Export button: here we can insert the data about the exported and imported stocks with more details such as, company name, code, transportation-way, payment type and to find out the price of stock and to calculate the total price .
7. Help button: this button use to give information about me.

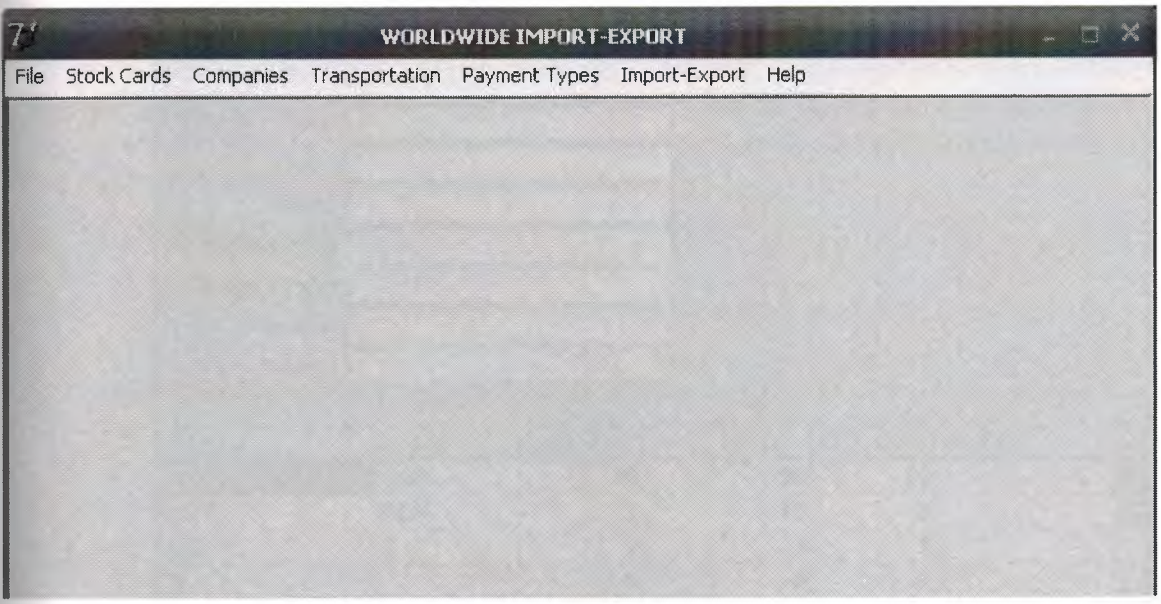
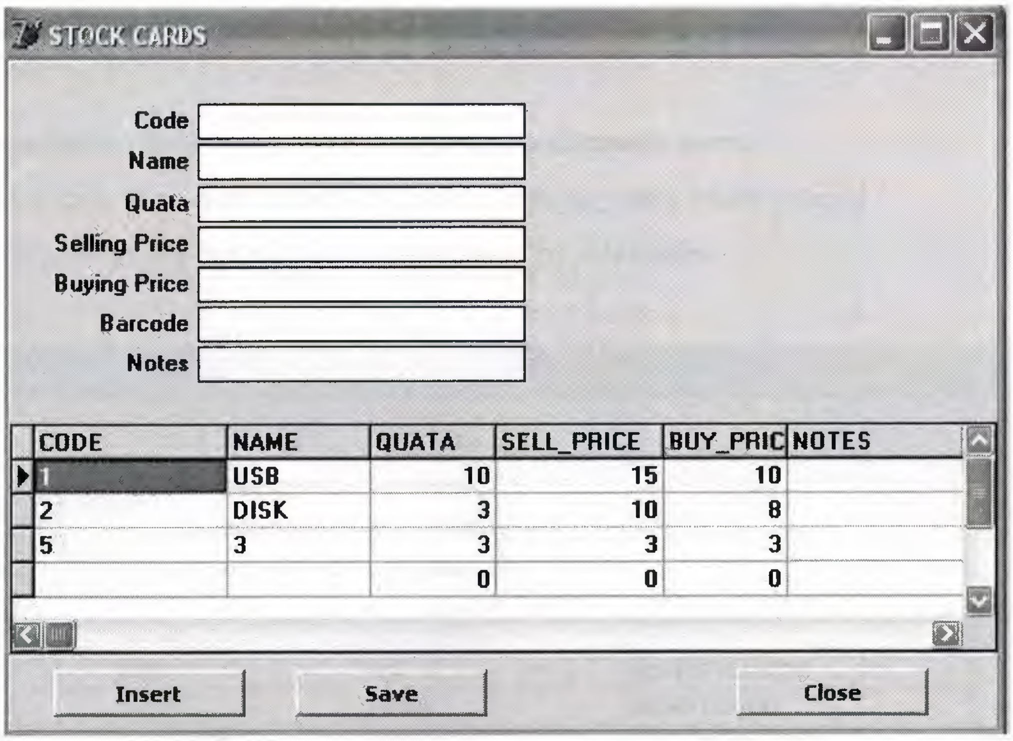


Figure 3.2: Main Menu Screen.

3.4.2. Add new stock card screen:

This screen allows us to enter information about the stock such as: code, name, quata, selling price, buying price, barcode, note, and there are four buttons and their functions are shown below:

1. Insert button: This button used to enter a new stock card.
2. Save button: This button used to keep the stock data which entered.
3. Close button: We can exit from this window by this button.

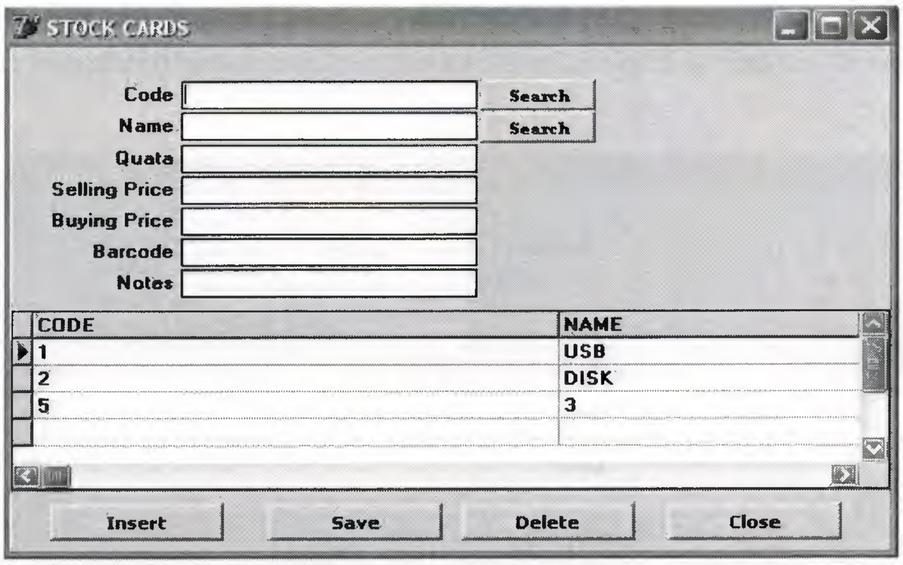


CODE	NAME	QUATA	SELL_PRICE	BUY_PRIC	NOTES
1	USB	10	15	10	
2	DISK	3	10	8	
5	3	3	3	3	
		0	0	0	

Figure3.3: Add stock card Screen.

3.4.3. Update stock card screen:

We use this screen to correct or to update some of stock information, and there are two search buttons which allow us to find the data we need as shown below.

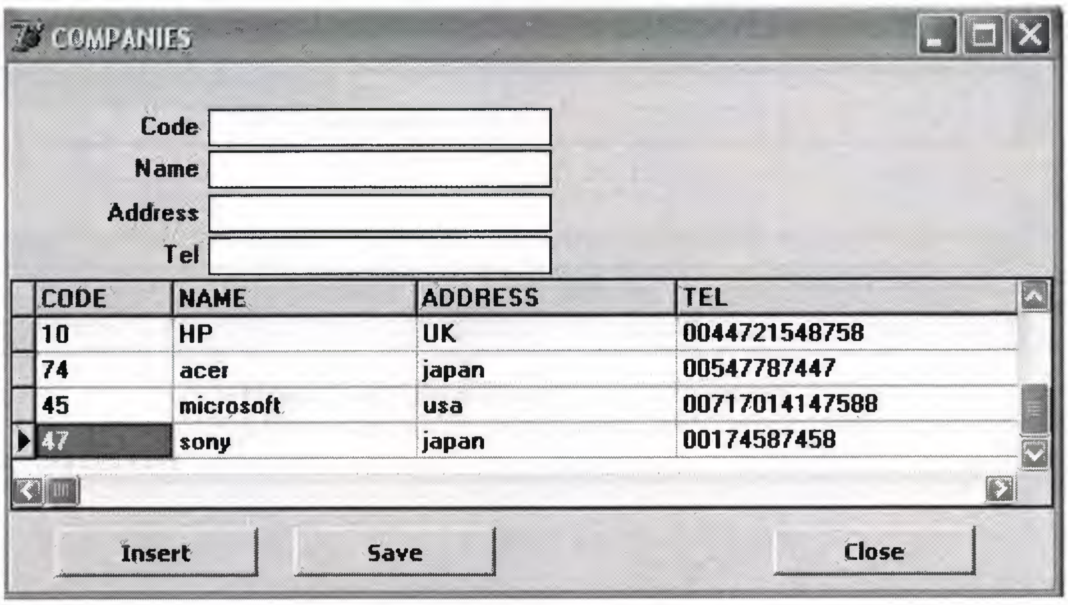


CODE	NAME
1	USB
2	DISK
5	3

Figure 3.3: Update stock card screen.

3.4.4. Add new company screen:

1. Insert button: This button used to enter a new company name.
2. Save button: This button used to keep the company data which entered.
3. Close button: We can exit from this window by this button.

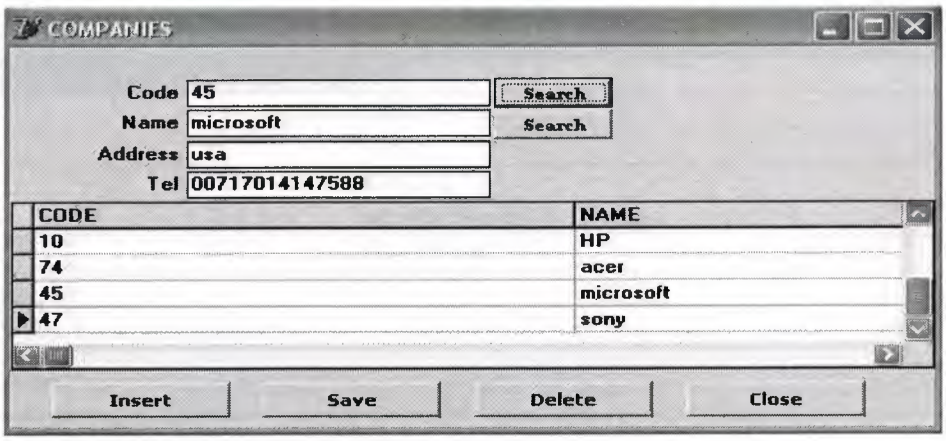


CODE	NAME	ADDRESS	TEL
10	HP	UK	0044721548758
74	acer	japan	00547787447
45	microsoft	usa	00717014147588
47	sony	japan	00174587458

Figure3.4: Add new company Screen

3.4.5. Update stock card screen:

We use this screen to correct or to update some of company information, and there are two search buttons which allow us to find the data we need as shown below.



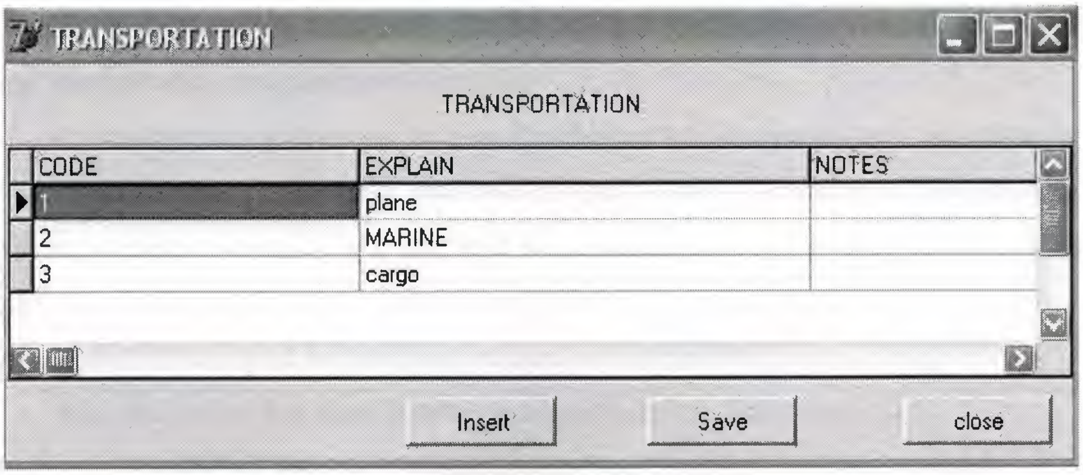
Code 45
Name microsoft
Address usa
Tel 00717014147588

CODE	NAME
10	HP
74	acer
45	microsoft
47	sony

Figure 3.5: Update company screen

3.4.6. Edit transportation way screen:

Here we can edit data about new transportation way or update the informations of transportation which entered before as shown in figure 3.6:



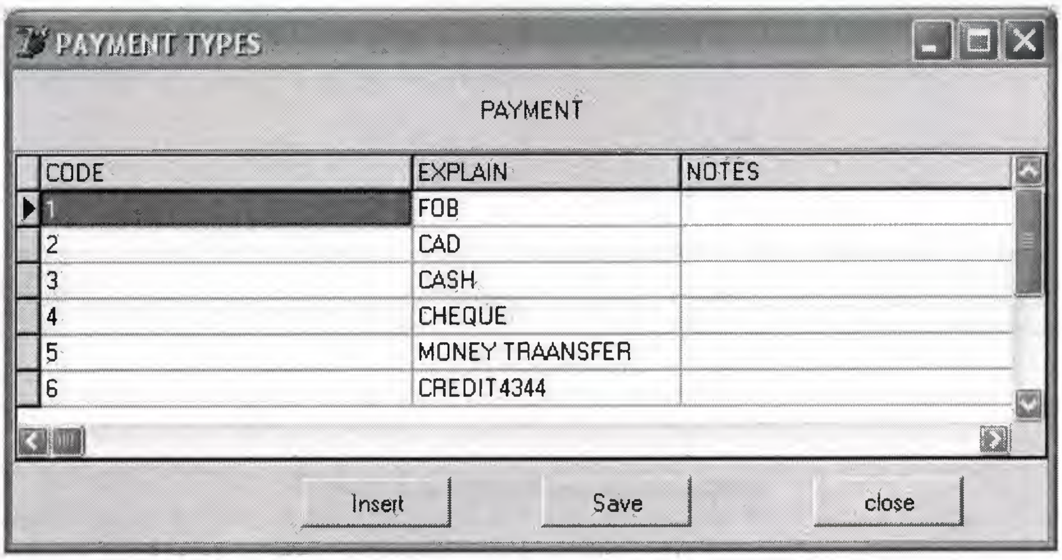
TRANSPORTATION

CODE	EXPLAIN	NOTES
1	plane	
2	MARINE	
3	cargo	

Figure 3.6: Edit transportation way screen.

3.4.7. Edit payments type screen:

Here we can insert new data about payments type or update the informations of payments which entered before as shown in figure 3.7:



PAYMENT

CODE	EXPLAIN	NOTES
1	FOB	
2	CAD	
3	CASH	
4	CHEQUE	
5	MONEY TRAANSFER	
6	CREDIT4344	

Figure 3.7: Edit payments type screen.

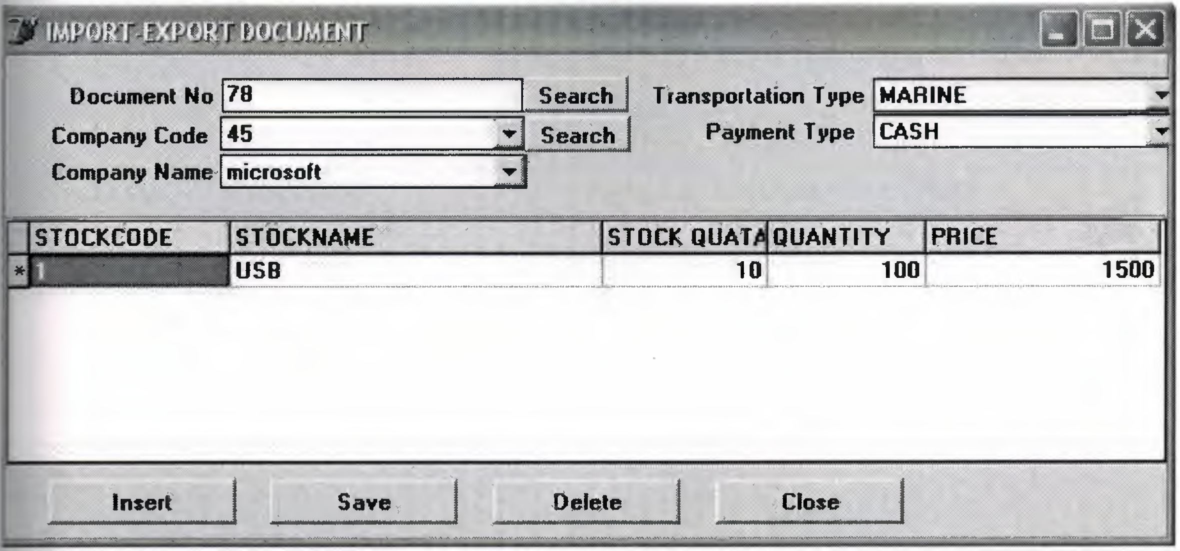
3.4.8. Import documents screen :

In this part we can insert and update all the information about the company, and stocks which imported with more details such as: price, total price, payments type, and transportation way and save it into file

Searching part: it consists of two ComboBox for document no. and company code. By choose one of them or more we can get all the information of that searching which has been choosed. For example if we select company code and we click search button we will get all the data which are giving by company code and it's name, and more details because all the fields are related with each other.

Also this screen has three BitBtn and their functions are shown as:

1. INSERT Button: it's used to reset and clear the data which appear
2. SAVE Button: it's used to keep and save the informations to a file.
3. DELETE Button: it's used to erase the information that we don't need it.
4. CLOSE Button: it's used to exit from the screen.



Document No 78
Company Code 45
Company Name microsoft
Transportation Type MARINE
Payment Type CASH

STOCKCODE	STOCKNAME	STOCK QUATA	QUANTITY	PRICE
1	USB	10	100	1500

Figure 3.8: Import document screen.

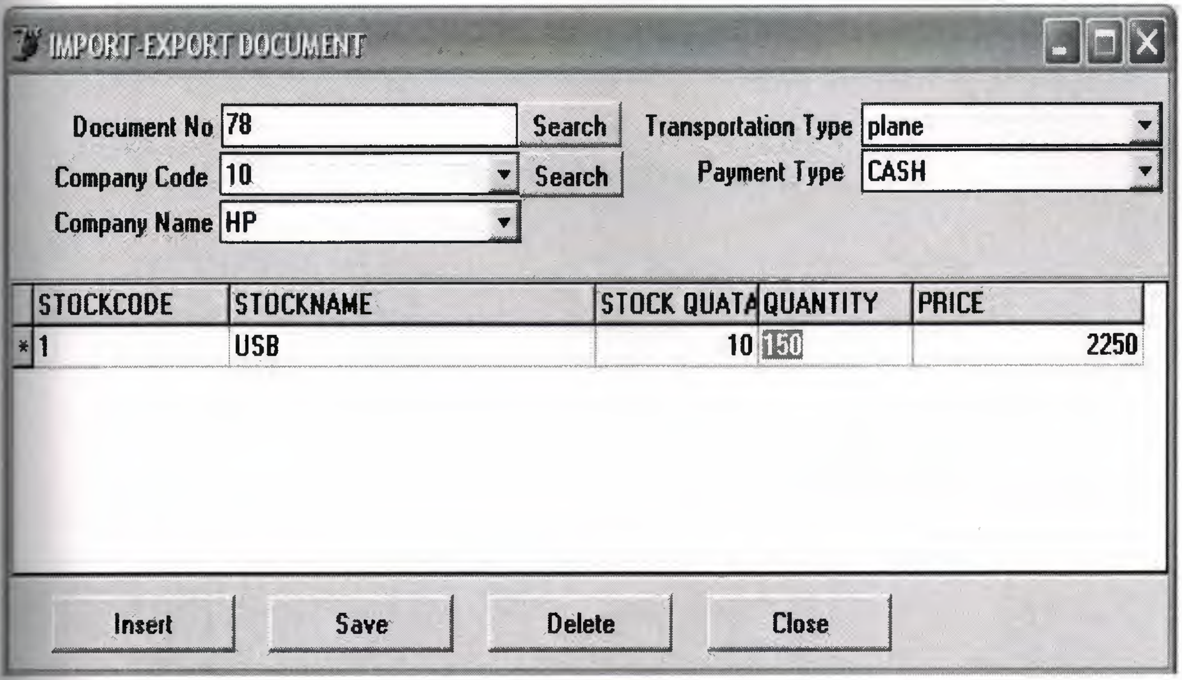
3.4.9. Export documents screen:

In this part we can insert and update all the information about the company, and stocks which exported with more details such as: price, total price, payments type, and transportation way and save it into file

Searching part: it consists of two ComboBox for document no. and company code. By choose one of them or more we can get all the information of that searching which has been choosed. For example if we select company code and we click search button we will get all the data which are giving by company code and it's name, and more details because all the fields are related with each other.

Also this screen has three BitBtn and their functions are shown as:

1. INSERT Button: it's used to reset and clear the data which appear
2. SAVE Button: it's used to keep and save the informations to a file.
3. DELETE Button: it's used to erase the information that we don't need it.
4. CLOSE Button: it's used to exit from the screen.



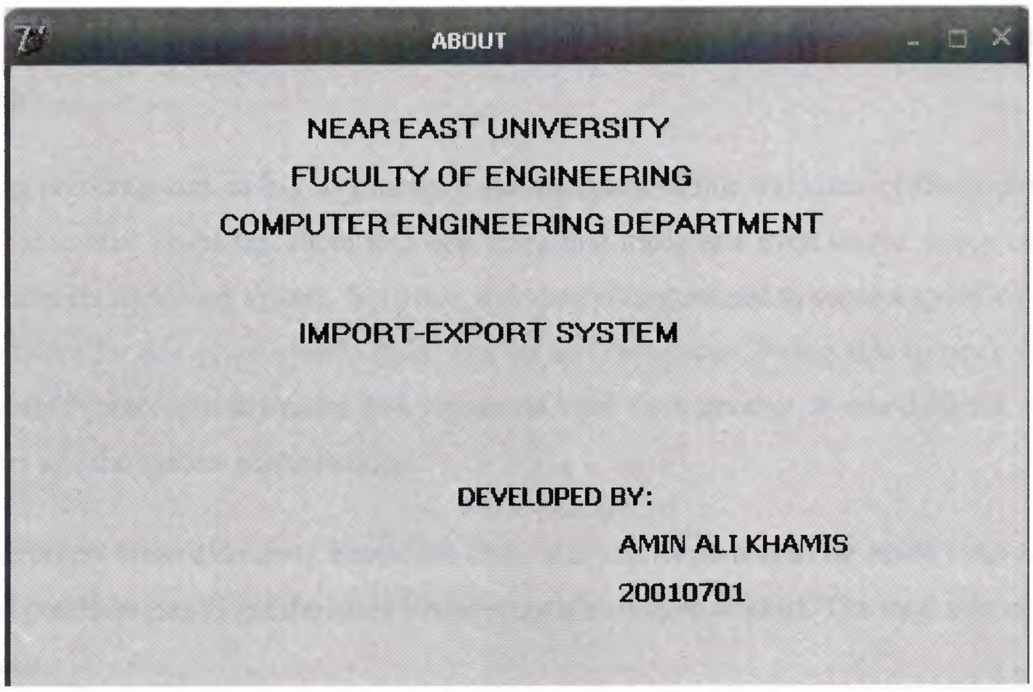
Document No	78
Company Code	10
Company Name	HP
Transportation Type	plane
Payment Type	CASH

STOCKCODE	STOCKNAME	STOCK QUATA	QUANTITY	PRICE
1	USB	10	150	2250

Figure 3.9: Export document screen.

3.4.10. Help screen:

1. ABOUT: in this screen there is information about me as shown below.



NEAR EAST UNIVERSITY

FUCULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

IMPORT-EXPORT SYSTEM

DEVELOPED BY:

AMIN ALI KHAMIS

20010701

Figure 3.10: About screen.

CHAPTER 4

LINUX AND UNIX OPERATING SYSTEM

4.1. Unix

In order to understand the popularity of Linux, we need to travel back in time, about 30 years ago...

Imagine computers as big as houses, even stadiums. While the sizes of those computers posed substantial problems, there was one thing that made this even worse: every computer had a different operating system. Software was always customized to serve a specific purpose, and software for one given system didn't run on another system. Being able to work with one system didn't automatically mean that you could work with another. It was difficult, both for the users and the system administrators.

Computers were extremely expensive then, and sacrifices had to be made even after the original purchase just to get the users to understand how they worked. The total cost of IT was enormous.

Technologically the world was not quite that advanced, so they had to live with the size for another decade. In 1969, a team of developers in the Bell Labs laboratories started working on a solution for the software problem, to address these compatibility issues. They developed a new operating system, which was

- simple and elegant
- written in the C programming language instead of in assembly code
- able to recycle code.

The Bell Labs developers named their project "UNIX."

The code recycling features were very important. Until then, all commercially available computer systems were written in a code specifically developed for one system. UNIX on the other hand needed only a small piece of that special code, which is now commonly named the kernel. This kernel is the only piece of code that needs to be adapted for every specific system and forms the base of the UNIX system.

The operating system and all other functions were built around this kernel and written in a higher programming language, C. This language was especially developed for creating the UNIX system. Using this new technique, it was much easier to develop an operating system that could run on many different types of hardware.

The software vendors were quick to adapt, since they could sell ten times more software almost effortlessly. Weird new situations came in existence: imagine for instance computers from different vendors communicating in the same network, or users working on different systems without the need for extra education to use another computer. UNIX did a great deal to help users become compatible with different systems.

Throughout the next couple of decades the development of UNIX continued. More things became possible to do and more hardware and software vendors added support for UNIX to their products.

UNIX was initially found only in very large environments with mainframes and minicomputers (note that a PC is a "micro" computer). You had to work at a university, for the government or for large financial corporations in order to get your hands on a UNIX system.

But smaller computers were being developed, and by the end of the 80's, many people had home computers. By that time, there were several versions of UNIX available for the PC architecture, but none of them were truly free.

4.2. Linus and Linux

Linus Torvalds, a young man studying computer science at the university of Helsinki, thought it would be a good idea to have some sort of freely available academic version of UNIX, and promptly started to code.

He started to ask questions, looking for answers and solutions that would help him get UNIX on his PC. Below is one of his first posts in comp.os.minix, dating from 1991:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Gcc-1.40 and a posix-question
```

```
   Message-ID: <1991Jul3.100050.9886@klaava.Helsinki.FI>
   Date: 3 Jul 91 10:00:50 GMT
   Hello netlanders,
   Due to a project I'm working on (in minix), I'm interested
in the posix
   standard definition. Could somebody please point me to a
(preferably)
   machine-readable format of the latest posix rules? Ftp-sites
would be
   nice.
```

From the start, it was Linus' goal to have a free system that was completely compliant with the original UNIX. That is why he asked for POSIX standards, POSIX still being the standard for UNIX.

In those days plug-and-play wasn't invented yet, but so many people were interested in having a UNIX system of their own, that this was only a small obstacle. New drivers became available for all kinds of new hardware, at a continuously rising speed. Almost as soon as a new piece of hardware became available, someone bought it and submitted it to the Linux test, as the system was gradually being called, releasing more free code for an ever wider range of hardware. These coders didn't stop at their PC's; every piece of hardware they could find was useful for Linux.

Back then, those people were called "nerds" or "freaks", but it didn't matter to them, as long as the supported hardware list grew longer and longer. Thanks to these people, Linux is now not only ideal to run on new PC's, but is also the system of choice for old and exotic hardware that would be useless if Linux didn't exist.

Two years after Linus' post, there were 12000 Linux users. The project, popular with hobbyists, grew steadily, all the while staying within the bounds of the POSIX standard. All the features of UNIX were added over the next couple of years, resulting in the mature operating system Linux has become today. Linux is a full UNIX clone, fit for use on workstations as well as on middle-range and high-end servers. Today, all the important players on the hard- and software market each have their team of Linux developers; at your local dealer's you can even buy pre-installed Linux systems with official support.

4.3. Current Application of Linux Systems

Today Linux has joined the desktop market. Linux developers concentrated on networking and services in the beginning, and office applications have been the last barrier to be taken down. We don't like to admit that Microsoft is ruling this market, so plenty of alternatives have been started over the last couple of years to make Linux an acceptable choice as a workstation, providing an easy user interface and MS compatible office applications like word processors, spreadsheets, presentations and the like.

On the server side, Linux is well-known as a stable and reliable platform, providing database and trading services for companies like Amazon, the well-known online bookshop, US Post Office, the German army and such. Especially Internet providers and Internet service providers have grown fond of Linux as firewall, proxy- and web server, and you will find a Linux box within reach of every UNIX system administrator who appreciates a comfortable management station. Clusters of Linux machines are used in the creation of movies such as "Titanic", "Shrek" and others. In post offices, they are the nerve centers that route mail and in large search engine, clusters are used to perform internet searches.These are only a few of the thousands of heavy-duty jobs that Linux is performing day-to-day across the world.

It is also worth to note that modern Linux not only runs on workstations, mid- and high-end servers, but also on "gadgets" like PDA's, mobiles, a shipload of embedded applications and even on experimental wristwatches. This makes Linux the only operating system in the world covering such a wide range of hardware.

4.4. The User Interface

4.4.1. Is Linux Difficult?

Whether Linux is difficult to learn depends on the person you're asking. Experienced UNIX users will say no, because Linux is an ideal operating system for power-users and programmers, because it has been and is being developed by such people.

Everything a good programmer can wish for is available: compilers, libraries, development and debugging tools.

These packages come with every standard Linux distribution. The C-compiler is included for free, all the documentation and manuals are there, and examples are often included to help you get started in no time. It feels like UNIX and switching between UNIX and Linux is a natural thing.

In the early days of Linux, being an expert was kind of required to start using the system. Those who mastered Linux felt better than the rest of the "lusers" who hadn't seen the light yet. It was common practice to tell a beginning user to "RTFM" (read the manuals). While the manuals were on every system, it was difficult to find the documentation, and even if someone did, explanations were in such technical terms that the new user became easily discouraged from learning the system.

The Linux-using community started to realize that if Linux was ever to be an important player on the operating system market, there had to be some serious changes in the accessibility of the system.

4.4.2. Linux for non-experienced Users

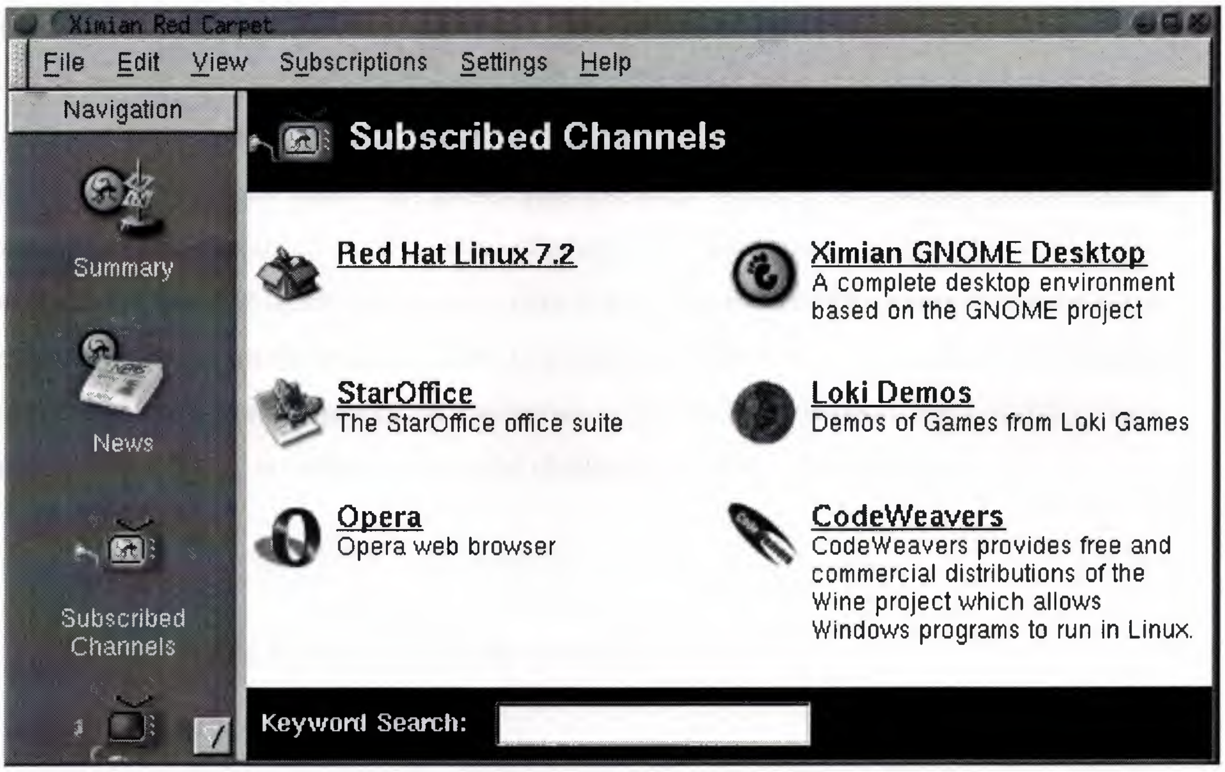
Companies such as RedHat, SuSE and Mandrake have sprung up, providing packaged Linux distributions suitable for mass consumption. They integrated a great deal of graphical user interfaces (GUIs), developed by the community, in order to ease management of programs and services. As a Linux user today you have all the means of getting to know your system inside out, but it is no longer necessary to have that knowledge in order to make the system comply to your requests.

Nowadays you can log in graphically and start all required applications without even having to type a single character, while you still have the ability to access the core of the system if needed. Because of its structure, Linux allows a user to grow into the system: it equally fits new and experienced users. New users are not forced to do difficult things, while experienced users are not forced to work in the same way they did when they first started learning Linux.

While development in the service area continues, great things are being done for desktop users, generally considered as the group least likely to know how a system works.

Developers of desktop applications are making incredible efforts to make the most beautiful desktops you've ever seen, or to make your Linux machine look just like your former MS Windows or MacIntosh workstation. The latest developments also include 3D acceleration support and support for USB devices, single-click updates of system and packages, and so on. Linux has these, and tries to present all available services in a logical form that ordinary people can understand.

The screenshot below shows how each item in the Channel list (RH 7.2, StarOffice, Opera, Ximian Gnome, Loki games and CodeWeavers) can be updated with one mouse click. Adding or removing software packages or keeping the system up to date is simple with tools like this one, called Red Carpet:



Red Hat Linux 7.2

Ximian GNOME Desktop
A complete desktop environment based on the GNOME project

StarOffice
The StarOffice office suite

Loki Demos
Demos of Games from Loki Games

Opera
Opera web browser

CodeWeavers
CodeWeavers provides free and commercial distributions of the Wine project which allows Windows programs to run in Linux.

Figure 4-1. Ximian Red Carpet: automated package management

4.5. Does Linux have a future?

4.5.1. Open Source

The idea behind Open Source software is rather simple: when programmers can read, distribute and change code, the code will mature. People can adapt it, fix it, debug it, and they can do it at a speed that dwarfs the performance of software developers at conventional companies. This software will be more flexible and of a better quality than software that has been developed using the conventional channels, because more people have tested it in more different conditions than the closed software developer ever can.

The Open Source initiative started to make this clear to the commercial world, and very slowly, commercial vendors are starting to see the point. While lots of academics and technical people have already been convinced for 20 years now that this is the way to go, commercial vendors needed applications like the Internet to make them realize they can profit from Open Source. Now Linux has grown past the stage where it was almost exclusively an academic system, useful only to a handful of people with a technical background. Now Linux provides more than the operating system: there is an entire infrastructure supporting the chain of effort of creating an operating system, of making and testing programs for it, of bringing everything to the users, of supplying maintenance, updates and support and customizations, etcetera. Today, Linux is ready to accept the challenge of a fast-changing world.

4.5.2. Ten years of experience at your service

While Linux is probably the most well-known Open Source initiative, there is another project that contributed enormously to the popularity of the Linux operating system. This project is called SAMBA, and its achievement is the reverse engineering of the Server Message Block (SMB)/Common Internet File System (CIFS) protocol used for file- and print-serving on PC-related machines, natively supported by MS Windows NT and OS/2, and Linux.

Packages are now available for almost every system and provide interconnection solutions in mixed environments using MS Windows protocols: Windows-compatible (up to and including Win2K) file- and print-servers.

Maybe even more successful than the SAMBA project is the Apache HTTP server project. The server runs on UNIX, Windows NT and many other operating systems. Originally known as "A PAtCHy server", based on existing code and a series of "patch files", the name for the matured code deserves to be connoted with the native American tribe of the Apache, well-known for their superior skills in warfare strategy and inexhaustible endurance. Apache has been shown to be substantially faster, more stable and more feature-full than many other web servers. Apache is run on sites that get millions of visitors per day, and while no official support is provided by the developers, the Apache user community provides answers to all your questions. Commercial support is now being provided by a number of third parties.

In the category of office applications, a choice of MS Office suite clones is available, ranging from partial to full implementations of the applications available on MS Windows workstations. These initiatives helped a great deal to make Linux acceptable for the desktop market, because the users don't need extra training to learn how to work with new systems. With the desktop comes the praise of the common users, and not only their praise, but also their specific requirements, which are growing more intricate and demanding by the day.

The Open Source community, consisting largely of people who have been contributing for over half a decade, assures Linux' position as an important player on the desktop market as well as in general IT application. Paid employees and volunteers alike are working diligently so that Linux can maintain a position in the market. The more users, the more questions. The Open Source community makes sure answers keep coming, and watches the quality of the answers with a suspicious eye, resulting in ever more stability and accessibility.

Listing all the available Linux software is beyond the scope of this guide, as there are tens of thousands of packages. Throughout this course we will present you with the most common packages, which are almost all freely available. In order to take away some of the fear of the beginning user, here's a screenshot of one of your most-wanted programs.

4.6. Properties of Linux

4.6.1. Linux Pros

A lot of the advantages of Linux are a consequence of Linux' origins, deeply rooted in UNIX, except for the first advantage, of course:

- Linux is free:

As in free beer, they say. If you want to spend absolutely nothing, you don't even have to pay the price of a CD. Linux can be downloaded in its entirety from the Internet completely for free. No registration fees, no costs per user, free updates, and freely available source code in case you want to change the behavior of your system.

Most of all, Linux is free as in free speech:

The license commonly used is the GNU Public License (GPL). The license says that anybody who may want to do so, has the right to change Linux and eventually to redistribute a changed version, on the one condition that the code is still available after redistribution. In practice, you are free to grab a kernel image, for instance to add support for teletransportation machines or time travel and sell your new code, as long as your customers can still have a copy of that code.

- Linux is portable to any hardware platform:

A vendor who wants to sell a new type of computer and who doesn't know what kind of OS his new machine will run (say the CPU in your car or washing machine), can take a Linux kernel and make it work on his hardware, because documentation related to this activity is freely available.

- Linux was made to keep on running:

As with UNIX, a Linux system expects to run without rebooting all the time. That is why a lot of tasks are being executed at night or scheduled automatically for other calm moments, resulting in higher availability during busier periods and a more balanced use of the hardware.

This property allows for Linux to be applicable also in environments where people don't have the time or the possibility to control their systems night and day.

- Linux is secure and versatile:

The security model used in Linux is based on the UNIX idea of security, which is known to be robust and of proven quality. But Linux is not only fit for use as a fort against enemy attacks from the Internet: it will adapt equally to other situations, utilizing the same high standards for security. Your development machine or control station will be as secure as your firewall.

- Linux is scalable:

From a Palmtop with 2 MB of memory to a petabyte storage cluster with hundreds of nodes: add or remove the appropriate packages and Linux fits all. You don't need a supercomputer anymore, because you can use Linux to do big things using the building blocks provided with the system. If you want to do little things, such as making an operating system for an embedded processor or just recycling your old 486, Linux will do that as well.

- The Linux OS and Linux applications have very short debug-times:

Because Linux has been developed and tested by thousands of people, both errors and people to fix them are found very quickly. It often happens that there are only a couple of hours between discovery and fixing of a bug.

4.6.2. Linux Cons

- There are far too many different distributions:

"Quot capites, tot rationes", as the Romans already said: the more people, the more opinions. At first glance, the amount of Linux distributions can be frightening,

or ridiculous, depending on your point of view. But it also means that everyone will find what he or she needs. You don't need to be an expert to find a suitable release.

When asked, generally every Linux user will say that the best distribution is the specific version he is using. So which one should you choose?

Don't worry too much about that: all releases contain more or less the same set of basic packages. On top of the basics, special third party software is added making, for example, TurboLinux more suitable for the small and medium enterprise, RedHat for servers and SuSE for workstations. However, the differences are likely to be very superficial. The best strategy is to test a couple of distributions; unfortunately not everybody has the time for this. Luckily, there is plenty of advice on the subject of choosing your Linux.

- Linux is not very user friendly and confusing for beginners:

In light of its popularity, considerable effort has been made to make Linux even easier to use, especially for new users. More information is being released daily, such as this guide, to help fill the gap for documentation available to users at all levels.

- Is an Open Source product trustworthy?

How can something that is free also be reliable? Linux users have the choice whether to use Linux or not, which gives them an enormous advantage compared to users of proprietary software, who don't have that kind of freedom. After long periods of testing, most Linux users come to the conclusion that Linux is not only as good, but in many cases better and faster that the traditional solutions. If Linux were not trustworthy, it would have been long gone, never knowing the popularity it has now, with millions of users. Now users can influence their systems and share their remarks with the community, so the system gets better and better every day. It is a project that is never finished, that is true, but in an ever changing environment, Linux is also a project that continues to strive for perfection.

4.7. Linux Flavors

4.7.1. Linux and GNU

Although there are a large number of Linux implementations, you will find a lot of similarities in the different distributions, if only because every Linux machine is a box with building blocks that you may put together following your own needs and views. Installing the system is only the beginning of a longterm relationship. Just when you think you have a nice running system, Linux will stimulate your imagination and creativeness, and the more you realize what power the system can give you, the more you will try to redefine its limits.

Linux may appear different depending on the distribution, your hardware and personal taste, but the fundamentals on which all graphical and other interfaces are built, remain the same. The Linux system is based on GNU tools (Gnu's Not UNIX), which provide a set of standard ways to handle and use the system. All GNU tools are open source, so they can be installed on any system. Most distributions offer pre-compiled packages of most common tools, such as RPM packages on RedHat and dpkg packages on Debian, so you needn't be a programmer to install a package on your system. However, if you are and like doing things yourself, you will enjoy Linux all the better, since most distributions come with a complete set of development tools, allowing installation of new software purely from source code. This setup also allows you to install software even if it does not exist in a pre-packaged form suitable for your system.

A list of common GNU software:

- Bash: The GNU shell
- GCC: The GNU C Compiler
- GDB: The GNU Debugger
- Findutils: to search and find files
- Fontutils: to convert fonts from one format to another or make new fonts
- The Gimp: GNU Image Manipulation Program
- Gnome: the GNU desktop environment
- Emacs: a very powerful editor
- Ghostscript and Ghostview: interpreter and graphical frontend for PDF files.
- GNU Photo: software for interaction with digital cameras

- Octave: a program to calculate mathematical functions and images.
- GNU SQL: relational database system
- Radius: a remote authentication and accounting server

Many commercial applications are available for Linux, and for more information about these packages we refer to their specific documentation. Throughout this guide we will only discuss freely available software, which comes (in most cases) with a GNU license.

To install missing or new packages, you will need some form of software management. The most common implementations include RPM, dpkg and Ximian Red Carpet. RPM is the RedHat Package Manager, which is used on a variety of Linux systems, eventhough the name does not suggest this. Dpkg is the Debian package management system, which uses an interface called Apt-Get, that can manage RPM packages as well. Ximian Red Carpet is a third party implementation of RPM with a graphical front-end. Other third party software vendors may have their own installation procedures, sometimes resembling the InstallShield and such, as known on MS Windows and other platforms. As you advance into Linux, you will likely get in touch with one or more of these programs.

4.7.2. GNU/Linux

The Linux kernel (the *bones* of your system, is not part of the GNU project but uses the same license as GNU software. A great majority of utilities and development tools (the *meat* of your system), which are not Linux-specific, are taken from the GNU project. Because any usable system must contain both the kernel and at least a minimal set of utilities, some people argue that such a system should be called a *GNU/Linux* system.

In order to obtain the highest possible degree of independence between distributions, this is the sort of Linux that we will discuss throughout this course. If we are not talking about a GNU/Linux system, the specific distribution, version or program name will be mentioned.

4.7.3. Which distribution should I install?

Prior to installation, the most important factor is your hardware. Since every Linux distribution contains the basic packages and can be built to meet almost any requirement (because they all use the Linux kernel), you only need to consider if the distribution will run on your hardware. LinuxPPC for example has been made to run on MacIntosh and other PowerPCs and does not run on an ordinary x86 based PC. LinuxPPC does run on the new Macs, but you can't use it for some of the older ones with ancient bus technology. Another tricky case is Sun hardware, which could be an old SPARC CPU or a newer UltraSparc, both requiring different versions of Linux.

Some Linux distributions are optimized for certain processors, such as Athlon CPUs, while they will at the same time run decent enough on the standard 486, 586 and 686 Intel processors. Sometimes distributions for special CPUs are not as reliable, since they are tested by fewer people.

Most Linux distributions offer a set of programs for generic PCs with special packages containing optimized kernels for the x86 Intel based CPUs. These distributions are well-tested and maintained on a regular basis, focusing on reliant server implementation and easy installation and update procedures. Examles are RedHat, SuSE and Mandrake, which are by far the most popular Linux systems and generally considered easy to handle for the beginning user, while not blocking professionals from getting the most out of their Linux machines. Linux also runs decently on laptops and middle-range servers. Drivers for new hardware are included only after extensive testing, which adds to the stability of a RedHat system.

While the standard desktop might be Gnome on one system, another might offer KDE by default. Generally, both Gnome and KDE are available for all Linux distributions. Other window and desktop managers are available for more advanced users.

The standard installation process allows to choose between different basic setups, such as a workstation, where all packages needed for everyday use and development are installed, or a server installation, where different network services can be selected. Expert users can install every combination of packages they want during the initial installation process.

4.8. Introduction to unix

4.8.1 The Operating System

Unix is a layered operating system. The innermost layer is the hardware that provides the services for the OS. The operating system, referred to in Unix as the **kernel**, interacts directly with the hardware and provides the services to the user programs. These user programs don't need to know anything about the hardware. They just need to know how to interact with the kernel and it's up to the kernel to provide the desired service. One of the big appeals of Unix to programmers has been that most well written user programs are independent of the underlying hardware, making them readily portable to new systems.

User programs interact with the kernel through a set of standard **system calls**. These system calls request services to be provided by the kernel. Such services would include accessing a file: open close, read, write, link, or execute a file; starting or updating accounting records; changing ownership of a file or directory; changing to a new directory; creating, suspending, or killing a process; enabling access to hardware devices; and setting limits on system resources.

Unix is a multi-user, multi-tasking operating system. You can have many users logged into a system simultaneously, each running many programs. It's the kernel's job to keep each process and user separate and to regulate access to system hardware, including cpu, memory, disk and other I/O devices.

4.8.2. The Unix File System

The UNIX operating system is built around the concept of a filesystem which is used to store all of the information that constitutes the long-term state of the system. This state includes the operating system kernel itself, the executable files for the commands supported by the operating system, configuration information, temporary workfiles, user data, and various special files that are used to give controlled access to system hardware and operating system functions.

Every item stored in a UNIX filesystem belongs to one of four types:

1. **Ordinaryfile**

Ordinary files can contain text, data, or program information. Files cannot contain other files or directories. Unlike other operating systems, UNIX filenames are not broken into a name part and an extension part (although extensions are still frequently used as a means to classify files). Instead they can contain any keyboard character except for '/' and be up to 256 characters long (note however that characters such as *,?,# and & have special meaning in most shells and should not therefore be used in filenames). Putting spaces in filenames also makes them difficult to manipulate - rather use the underscore '_'.

2. **Directories**

Directories are containers or folders that hold files, and other directories.

3. **Devices**

To provide applications with easy access to hardware devices, UNIX allows them to be used in much the same way as ordinary files. There are two types of devices in UNIX - **block-oriented** devices which transfer data in blocks (e.g. hard disks) and **character-oriented** devices that transfer data on a byte-by-byte basis (e.g. modems and dumb terminals).

4. **Links**

A link is a pointer to another file. There are two types of links - a hard link to a file is indistinguishable from the file itself. A soft link (or symbolic link) provides an indirect pointer or shortcut to a file. A soft link is implemented as a directory file entry containing a pathname.

4.9. Typcal Unix Directory Structure

The UNIX filesystem is laid out as a hierarchical tree structure which is anchored at a special top-level directory known as the root (designated by a slash '/').

Because of the tree structure, a directory can have many child directories, but only one parent directory. Fig. 3.2 illustrates this layout.

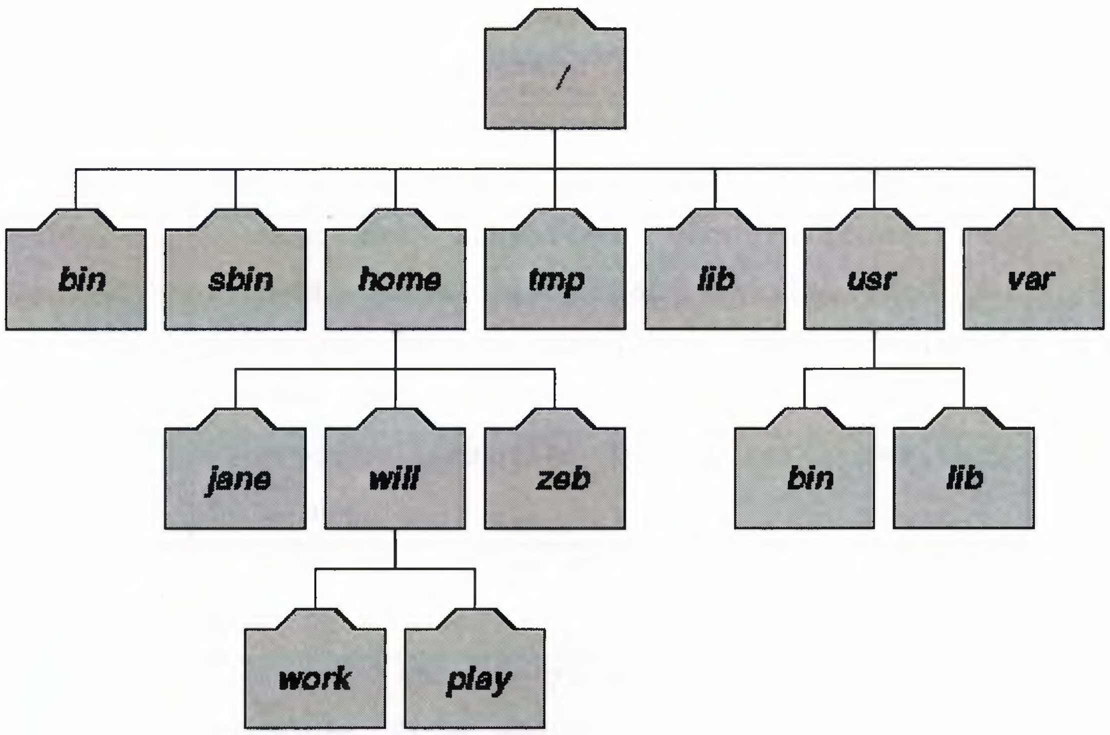


Fig. 4.2: Part of a typical UNIX filesystem tree

To specify a location in the directory hierarchy, we must specify a path through the tree. The path to a location can be defined by an absolute path from the root /, or as a relative path from the current working directory. To specify a path, each directory along the route from the source to the destination must be included in the path, with each directory in the sequence being separated by a slash. To help with the specification of relative paths, UNIX provides the shorthand "`.`" for the current directory and "`..`" for the parent directory. For example, the absolute path to the directory "`play`" is `/home/will/play`, while the relative path to this directory from "`zeb`" is `../will/play`.

Fig. 4.2 shows some typical directories you will find on UNIX systems and briefly describes their contents. Note that these although these subdirectories appear as part of a seamless logical filesystem, they do not need be present on the same hard disk device; some may even be located on a remote machine and accessed across a network.

Directory	Typical Contents
/	The "root" directory
/bin	Essential low-level system utilities

/usr/bin	Higher-level system utilities and application programs
/sbin	Superuser system utilities (for performing system administration tasks)
/lib	Program libraries (collections of system calls that can be included in programs by a compiler) for low-level system utilities
/usr/lib	Program libraries for higher-level user programs
/tmp	Temporary file storage space (can be used by any user)
/home or /homes	User home directories containing personal file space for each user. Each directory is named after the login of the user.
/etc	UNIX system configuration and information files
/dev	Hardware devices
/proc	A pseudo-filesystem which is used as an interface to the kernel. Includes a sub-directory for each active program (or process).

When you log into UNIX, your current working directory is your user home directory. You can refer to your home directory at any time as "~" and the home directory of other users as "~`<login>`". So `~will/play` is another way for user `jane` to specify an absolute path to the directory `/homes/will/play`. User `will` may refer to the directory as `~/play`.

4.10. Directory and File Handling Commands

This section describes some of the more important directory and file handling commands.

- pwd (print [current] working directory)

pwd displays the full absolute path to the your current location in the filesystem.

So

```
 $ pwd ↵
/usr/bin
```

implies that /usr/bin is the current working directory.

- ls (list directory)

ls lists the contents of a directory. If no target directory is given, then the contents of the current working directory are displayed. So, if the current working directory is /,

```
 $ ls ↵
bin  dev  home  mnt   share  usr  var
boot  etc  lib   proc  sbin   tmp  vol
```

Actually, **ls** doesn't show you *all* the entries in a directory - files and directories that begin with a dot (.) are hidden (this includes the directories '.' and '..' which are always present). The reason for this is that files that begin with a . usually contain important configuration information and should not be changed under normal circumstances. If you want to see all files, ls supports the -a option:

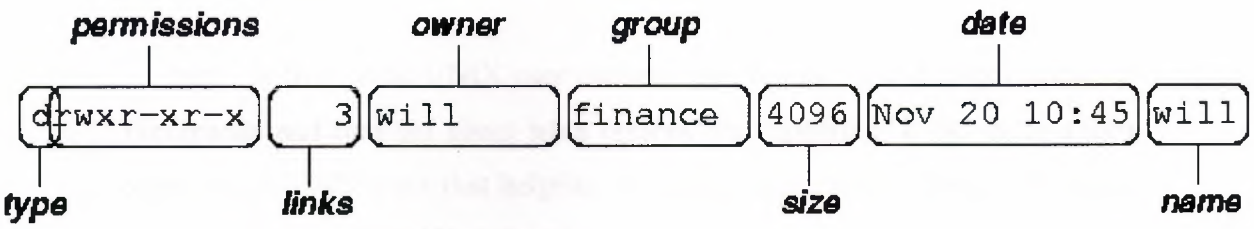
```
 $ ls -a ↵
```

Even this listing is not that helpful - there are no hints to properties such as the size, type and ownership of files, just their names. To see more detailed information, use the -l option (long listing), which can be combined with the -a option as follows:

```
$ ls -a -l ←
```

(or, equivalently,) `$ ls -al ←`

Each line of the output looks like this:



where:

- *type* is a single character which is either 'd' (directory), '-' (ordinary file), 'l' (symbolic link), 'b' (block-oriented device) or 'c' (character-oriented device).
- *permissions* is a set of characters describing access rights. There are 9 permission characters, describing 3 access types given to 3 user categories. The three access types are read ('r'), write ('w') and execute ('x'), and the three users categories are the user who owns the file, users in the group that the file belongs to and other users (the general public). An 'r', 'w' or 'x' character means the corresponding permission is present; a '-' means it is absent.
- *links* refers to the number of filesystem links pointing to the file/directory (see the discussion on hard/soft links in the next section).
- *owner* is usually the user who created the file or directory.
- *group* denotes a collection of users who are allowed to access the file according to the group access rights specified in the permissions field.

- *size* is the length of a file, or the number of bytes used by the operating system to store the list of files in a directory.
- *date* is the date when the file or directory was last modified (written to). The `-u` option display the time when the file was last accessed (read).
- *name* is the name of the file or directory.

`ls` supports more options. To find out what they are, type:

```
$ man ls↵
```

`man` is the online UNIX user manual, and you can use it to get help with commands and find out about what options are supported. It has quite a terse style which is often not that helpful, so some users prefer to the use the (non-standard) `info` utility if it is installed:

```
$ info ls↵
```

- `cd` (change [current working] directory)

```
$ cd path
```

changes your current working directory to *path* (which can be an absolute or a relative path). One of the most common relative paths to use is '..' (i.e. the parent directory of the current directory).

Used without any target directory

```
$ cd↵
```

resets your current working directory to your home directory (useful if you get lost). If you change into a directory and you subsequently want to return to your original directory, use

```
$ cd - ↵
```

- `mkdir` (make directory)

```
$ mkdir directory
```

creates a subdirectory called *directory*in the current working directory. You can only create subdirectories in a directory if you have write permission on that directory.

- `rmdir` (remove directory)

```
$ rmdir directory
```

removes the subdirectory *directory* from the current working directory. You can only remove subdirectories if they are completely empty (i.e. of all entries besides the '.' and '..' directories).

- `cp` (copy)

`cp` is used to make copies of files or entire directories. To copy files, use:

```
$ cp source-file(s) destination
```

where *source-file(s)* and *destination* specify the source and destination of the copy respectively. The behaviour of `cp` depends on whether the destination is a file or a directory. If the destination is a file, only one source file is allowed and `cp` makes a new file called *destination* that has the same contents as the source file. If the destination is a directory, many source files can be specified, each of which will be copied into the destination directory. Section 2.6 will discuss efficient specification of source files using wildcard characters.

To copy entire directories (including their contents), use a *recursive* copy:

```
$ cp -rd source-directories destination-directory
```

- `mv` (move/rename)

`mv` is used to rename files/directories and/or move them from one directory into another. Exactly one source and one destination must be specified:

```
$ mv source destination
```

If *destination* is an existing directory, the new name for *source* (whether it be a file or a directory) will be *destination/source*. If *source* and *destination* are both files, *source* is renamed *destination*. N.B.: if *destination* is an existing file it will be destroyed and overwritten by *source* (you can use the `-i` option if you would like to be asked for confirmation before a file is overwritten in thisway.

- `rm` (remove/delete)

```
$ rm target-file(s)
```

removes the specified files. Unlike other operating systems, it is almost impossible to recover a deleted file unless you have a backup (there is no recycle bin!) so use this command with care. If you would like to be asked before files are deleted, use the `-i` option:

```
    $ rm -i myfile↵
  rm: remove 'myfile'?
```

`rm` can also be used to delete directories (along with all of their contents, including any subdirectories they contain). To do this, use the `-r` option. To avoid `rm` from asking any questions or giving errors (e.g. if the file doesn't exist) you used the `-f` (force) option.

Extreme care needs to be taken when using this option - consider what would happen if a system administrator was trying to delete user `will`'s home directory and accidentally typed:

```
$ rm -rf / home/will↵
```

(instead of `rm -rf /home/will`).

- `cat` (catenate/type)

```
$ cat target-file(s)
```

displays the contents of *target-file(s)* on the screen, one after the other. You can also use it to create files from keyboard input as follows (> is the output redirection operator, which will be discussed in the next chapter):

```
    $ cat > hello.txt↵
hello world!↵
[ctrl-d]
$ ls hello.txt↵
hello.txt
$ cat hello.txt↵
hello world!
$
```

- `more` and `less` (catenate with pause)

```
$ more target-file(s)
```

displays the contents of *target-file(s)* on the screen, pausing at the end of each screenful and asking the user to press a key (useful for long files). It also incorporates a searching facility (press '/' and then type a phrase that you want to look for).

You can also use `more` to break up the output of commands that produce more than one screenful of output as follows (`|` is the pipe operator, which will be discussed in the next chapter):

```
$ ls -l | more↵
```

`less` is just like `more`, except that has a few extra features (such as allowing users to scroll backwards and forwards through the displayed file). `less` not a standard utility, however and may not be present on all UNIX systems.

4.11. Making Hard and Soft (symbolic) Links:

Direct (hard) and indirect (soft or symbolic) links from one file or directory to another can be created using the `ln` command.

$ `ln` *filename linkname*

creates another directory entry for *filename* called *linkname* (i.e. *linkname* is a hard link). Both directory entries appear identical (and both now have a link count of 2). If either *filename* or *linkname* is modified, the change will be reflected in the other file (since they are in fact just two different directory entries pointing to the same file).

$ `ln -s` *filename linkname*

creates a shortcut called *linkname* (i.e. *linkname* is a soft link). The shortcut appears as an entry with a special type ('`l`'):

```
    $ ln -s hello.txt bye.txt↵
$ ls -l bye.txt↵
lrwxrwxrwx   1 will finance 13 bye.txt -> hello.txt
$
```

The link count of the source file remains unaffected. Notice that the permission bits on a symbolic link are not used (always appearing as `rwxrwxrwx`). Instead the permissions on the link are determined by the permissions on the target (`hello.txt` in this case).

Note that you can create a symbolic link to a file that doesn't exist, but not a hard link. Another difference between the two is that you can create symbolic links across different physical disk devices or partitions, but hard links are restricted to the same disk partition. Finally, most current UNIX implementations do not allow hard links to point to directories.

4.12. Specifying Multiple Filenames :

Multiple filenames can be specified using special pattern-matching characters. The rules are:

- '?' matches any single character in that position in the filename.
- '*' matches zero or more characters in the filename. A '*' on its own will match all files. '*.*' matches all files with containing a '.'.
- Characters enclosed in square brackets ('[' and ']') will match any filename that has one of those characters in that position.
- A list of comma separated strings enclosed in curly braces ("{" and "}") will be expanded as a Cartesian product with the surrounding characters.

For example:

1. `???` matches all three-character filenames.
2. `?ell?` matches any five-character filenames with 'ell' in the middle.
3. `he*` matches any filename beginning with 'he'.
4. `[m-z]*[a-l]` matches any filename that begins with a letter from 'm' to 'z' and ends in a letter from 'a' to 'l'.
5. `{/usr,}{/bin,/lib}/file` expands to `/usr/bin/file /usr/lib/file /bin/file` and `/lib/file`.

Note that the UNIX shell performs these expansions (including any filename matching) on a command's arguments *before* the command is executed.

4.12.1. Quotes

As we have seen certain special characters (e.g. '*', '-','{' etc.) are interpreted in a special way by the shell. In order to pass arguments that use these characters to commands directly (i.e. without filename expansion etc.), we need to use special quoting characters.

There are three levels of quoting that you can try:

1. Try insert a '\' in front of the special character.
2. Use double quotes (") around arguments to prevent most expansions.
3. Use single forward quotes (') around arguments to prevent all expansions.

There is a fourth type of quoting in UNIX. Single backward quotes (`) are used to pass the output of some command as an input argument to another. For example:

```
  $ hostname ↵
rose
$ echo this machine is called `hostname` ↵
this machine is called rose
```

4.13. Summary

In this chapter, we learned that:

- Linux is an implementation of UNIX.
- The Linux operating system is written in the C programming language.
- De gustibus et coloribus non disputandum est: there's a Linux for everyone.
- Linux uses GNU tools, a set of freely available standard tools for handling the operating system.

CONCLUSION

In this project I learned a lot of things that in first time and even though not all of things I wanted to do in this project but this is mainly because of the lack of time and knowledge in programming with Delphi programming, but we can say that Delphi database support is very extensive and complete.

I have very high hopes on expanding the capability of this program in near future and from there I will take-off in mastering Delphi to design any project, I will try to take a lot of experience which is very important tool that I will need to take obstacles being faced in the future .

REFERANCE

BOOKS:

IMPORT-EXPORT PROCEDURE AND DOCUMENTS.

MARCOCANTU,"MASTERING DELPHI", SYBEX,

WEBSITES:

WWW.HOWTOIMPORT.COM
WWW.EXPORT911.COM
WWW.FETO.COM
WWW.SYBEX.COM
WWW.KDTOOL.NET

APPENDIX

1. Main menu:

```
unit MainUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, DB, ADODB;

type
  TMainForm = class(TForm)
   MainMenu1: TMainMenu;
   File1: TMenuItem;
   Exit1: TMenuItem;
   Cards1: TMenuItem;
   StockCards1: TMenuItem;
   Companies1: TMenuItem;
   ImportExportDocuments1: TMenuItem;
   ImportDocument1: TMenuItem;
   Exportdocument1: TMenuItem;
   Help1: TMenuItem;
   About1: TMenuItem;
   ADOConnection1: TADOConnection;
   Companies3: TMenuItem;
   AddNewCompany1: TMenuItem;
   UpdateCompanies1: TMenuItem;
   ransportation1: TMenuItem;
   Add1: TMenuItem;
   PaymentTypes1: TMenuItem;
   ADODataSet1: TADODataSet;
   EditPaymentTypes1: TMenuItem;
```

```
    procedure Exit1Click(Sender: TObject);
    procedure StockCards1Click(Sender: TObject);
    procedure Companies1Click(Sender: TObject);
    procedure AddNewCompany1Click(Sender: TObject);
    procedure UpdateCompanies1Click(Sender: TObject);
    procedure ImportDocument1Click(Sender: TObject);
    procedure Exportdocument1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Add1Click(Sender: TObject);
    procedure EditPaymentTypes1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

uses StockUnit, CompanyUnit, ExpImpUnit, TransUnit, PayUnit, AboutUnit;

{$R *.dfm}

procedure TMainForm.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.StockCards1Click(Sender: TObject);
begin
  Application.CreateForm(TStockForm, StockForm);
```

```
 StockForm.BitBtn5.Visible:=false;
 StockForm.BitBtn6.Visible:=false;
 StockForm.BitBtn3.Visible:=FALSE;
 StockForm.DBGrid1.ReadOnly:=TRUE;
end;

procedure TMainForm.Companies1Click(Sender: TObject);
begin
 Application.CreateForm(TStockForm, StockForm);
end;

procedure TMainForm.AddNewCompany1Click(Sender: TObject);
begin
 Application.CreateForm(TCompanyForm, CompanyForm);
 CompanyForm.BitBtn5.Visible:=false;
 CompanyForm.BitBtn6.Visible:=false;
 CompanyForm.BitBtn3.Visible:=FALSE;
 CompanyForm.DBGrid1.ReadOnly:=TRUE;
end;

procedure TMainForm.UpdateCompanies1Click(Sender: TObject);
begin
 Application.CreateForm(TCompanyForm, CompanyForm);
end;

procedure TMainForm.ImportDocument1Click(Sender: TObject);
begin
 Application.CreateForm(TExpImpForm, ExpImpForm);
 ExpImpForm.DocType:=0;
end;

procedure TMainForm.Exportdocument1Click(Sender: TObject);
begin
 Application.CreateForm(TExpImpForm, ExpImpForm);
```

```
  ExpImpForm.DocType:=1;
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
  DBPath : String;
  SrcFile : TextFile;
begin
  AssignFile(SrcFile,ExtractFilePath(Application.ExeName)+'DatabasePath.txt');
  Reset(SrcFile);
  Readln(SrcFile,DBPath);
  ADOConnection1.Close;
  ADOConnection1.ConnectionString:='Provider=Microsoft.Jet.OLEDB.4.0;Data
Source='+DBPath+';Persist Security Info=False';
  ADOConnection1.Open;
  CloseFile(SrcFile);
end;

procedure TMainForm.Add1Click(Sender: TObject);
begin
  Application.CreateForm(TTransForm, TransForm);
  TransForm.Show;
end;

procedure TMainForm.EditPaymentTypes1Click(Sender: TObject);
begin
  Application.CreateForm(TPayForm, PayForm);
  PayForm.Show;
end;

procedure TMainForm.About1Click(Sender: TObject);
begin
  Application.CreateForm(TAboutForm, AboutForm);
  AboutForm.Show;
```

```
end;

end.
```

2. Stock card:

```
unit StockUnit;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, ExtCtrls, DB, ADODB, StdCtrls, Mask, DBCtrls, Buttons, Grids,
 DBGrids;

type
 TStockForm = class(TForm)
  Panel1: TPanel;
  Panel2: TPanel;
  ADODataSet1: TADODataSet;
  DataSource1: TDataSource;
  Label1: TLabel;
  DBEdit1: TDBEdit;
  ADODataSet1CODE: TWideStringField;
  ADODataSet1NAME: TWideStringField;
  ADODataSet1QUATA: TFloatField;
  ADODataSet1SELL_PRICE: TBCDField;
  ADODataSet1BUY_PRICE: TBCDField;
  ADODataSet1NOTES: TWideStringField;
  ADODataSet1BARCODE: TWideStringField;
  DBEdit2: TDBEdit;
  Label2: TLabel;
  DBEdit3: TDBEdit;
  Label3: TLabel;
```

```
    DBEdit4: TDBEdit;
    Label4: TLabel;
    DBEdit5: TDBEdit;
    Label5: TLabel;
    DBEdit6: TDBEdit;
    Label6: TLabel;
    DBEdit7: TDBEdit;
    Label7: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    DBGrid1: TDBGrid;
    ADODataSet2: TADODataSet;
    DataSource2: TDataSource;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  StockForm: TStockForm;
```

```
implementation

uses MainUnit;

{$R *.dfm}

procedure TStockForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action:=cafree;
end;

procedure TStockForm.BitBtn4Click(Sender: TObject);
begin
  Close;
end;

procedure TStockForm.BitBtn1Click(Sender: TObject);
begin
  ADODataSet1.Cancel;
  ADODataSet1.Close;
  ADODataSet1.CommandText:='SELECT * FROM STOCK_CARDS';
  ADODataSet1.Open;
  ADODataSet1.Insert;
  DBEdit1.SetFocus;
  ADODataSet2.Close;
  ADODataSet2.Open;
end;

procedure TStockForm.BitBtn2Click(Sender: TObject);
begin
  try
    MainForm.ADODataSet1.Close;
    MainForm.ADODataSet1.CommandText:='SELECT * FROM STOCK_CARDS
WHERE CODE='''+DBEdit1.Text+'''';
```

```
   MainForm.ADODataSet1.Open;
   if MainForm.ADODataSet1.IsEmpty then
   begin
    ADODataSet1.Edit;
    ADODataSet1.post;
    BitBtn1Click(SENDER);
   end
   else
     MessageDlg('Code is already exists !',mtError,[mBOk],0);
 except on E:Exception do
  begin
    MessageDlg('An Error Occured During Save Opration
!'+#13+E.Message,mtError,[mbOk],0);
  end;
 end;
end;

procedure TStockForm.BitBtn3Click(Sender: TObject);
begin
 if DBEdit1.Text<>'' then
 begin
  if MessageDlg('Are You Sure That You Want To Delete
?!',mtConfirmation,[mbYes,mbNo],0)=mrYes then
  begin
   try
    ADODataSet1.Delete;
    DBEdit1.SetFocus;
   except on E:Exception do
    begin
      MessageDlg('An Error Occured During Delete Opration
!'+#13+E.Message,mtError,[mbOk],0);
    end;
   end;
  end;
 end
 else
  DBEdit1.SetFocus;
end;

procedure TStockForm.FormShow(Sender: TObject);
begin
 BitBtn1Click(SENDER);
end;

procedure TStockForm.BitBtn5Click(Sender: TObject);
var
 Code : String;
begin
 Code:=DBEdit1.Text;
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM STOCK_CARDS WHERE
CODE='''+Code+'''';
 ADODataSet1.Open;
end;

procedure TStockForm.BitBtn6Click(Sender: TObject);
var
 Code : String;
begin
 Code:=DBEdit2.Text;
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM STOCK_CARDS WHERE
NAME='''+Code+'''';
 ADODataSet1.Open;
end;
```

```
end.
```

3. Companies:

```
unit CompanyUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, ADODB, StdCtrls, Buttons, Mask, DBCtrls, ExtCtrls, Grids,
  DBGrids;

type
  TCompanyForm = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    Panel2: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    ADODataSet1: TADODataSet;
    DataSource1: TDataSource;
    ADODataSet1CODE: TWideStringField;
    ADODataSet1NAME: TWideStringField;
    ADODataSet1ADDRESS: TWideStringField;
    ADODataSet1TEL: TWideStringField;
    DataSource2: TDataSource;
    ADODataSet2: TADODataSet;
    DBGrid1: TDBGrid;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  CompanyForm: TCompanyForm;

implementation

uses MainUnit;

{$R *.dfm}

procedure TCompanyForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action:=cafree;
end;
```

```
procedure TCompanyForm.BitBtn1Click(Sender: TObject);
begin
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM COMPANIES';
 ADODataSet1.Open;
 ADODataSet1.Insert;
 DBEdit1.SetFocus;
 ADODataSet2.Close;
 ADODataSet2.Open;
end;

procedure TCompanyForm.BitBtn2Click(Sender: TObject);
begin
 try
   MainForm.ADODataSet1.Close;
   MainForm.ADODataSet1.CommandText:='SELECT * FROM COMPANIES
WHERE CODE='''+DBEdit1.Text+'''';
   MainForm.ADODataSet1.Open;
   if MainForm.ADODataSet1.IsEmpty then
   begin
    ADODataSet1.Edit;
    ADODataSet1.post;
    BitBtn1Click(SENDER);
   end
   else
     MessageDlg('Code is already exists !',mtError,[mBOk],0);
 except on E:Exception do
  begin
    MessageDlg('An Error Occured During Save Opration
!'+#13+E.Message,mtError,[mbOk],0);
  end;
 end;
```

```
end;

procedure TCompanyForm.BitBtn3Click(Sender: TObject);
begin
 if DBEdit1.Text<>'' then
 begin
  if MessageDlg('Are You Sure That You Want To Delete
?!',mtConfirmation,[mbYes,mbNo],0)=mrYes then
  begin
   try
    ADODataSet1.Delete;
    DBEdit1.SetFocus;
   except on E:Exception do
    begin
     MessageDlg('An Error Occured During Delete Opration
!'+#13+E.Message,mtError,[mbOk],0);
    end;
   end;
  end;
 end
 else
   DBEdit1.SetFocus;
end;

procedure TCompanyForm.BitBtn4Click(Sender: TObject);
begin
 Close;
end;

procedure TCompanyForm.BitBtn5Click(Sender: TObject);
var
 Code : String;
begin
 Code:=DBEdit1.Text;
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM COMPANIES WHERE
CODE='''+Code+'''';
 ADODataSet1.Open;
end;

procedure TCompanyForm.BitBtn6Click(Sender: TObject);
var
 Code : String;
begin
 Code:=DBEdit2.Text;
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM COMPANIES WHERE
NAME='''+Code+'''';
 ADODataSet1.Open;
end;

procedure TCompanyForm.FormShow(Sender: TObject);
begin
 BitBtn1Click(Sender);
end;


end.
```

4. Transportation:

```
unit TransUnit;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, ExtCtrls, StdCtrls, Buttons, DB, Grids, DBGrids, ADODB;

type
 TTransForm = class(TForm)
   Panel1: TPanel;
   Panel2: TPanel;
   Panel3: TPanel;
   ADODataSet1: TADODataSet;
   DBGrid1: TDBGrid;
   DataSource1: TDataSource;
   BitBtn1: TBitBtn;
   BitBtn2: TBitBtn;
   Insert: TBitBtn;
   procedure FormClose(Sender: TObject; var Action: TCloseAction);
   procedure BitBtn2Click(Sender: TObject);
   procedure InsertClick(Sender: TObject);
   procedure BitBtn1Click(Sender: TObject);
   procedure FormShow(Sender: TObject);
  private
   { Private declarations }
  public
   { Public declarations }
  end;

var
 TransForm: TTransForm;

implementation

uses MainUnit;

{$R *.dfm}

procedure TTransForm.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
  Action:=cafree;
end;

procedure TTransForm.BitBtn2Click(Sender: TObject);
begin
  Close;
end;

procedure TTransForm.InsertClick(Sender: TObject);
begin
  ADODataSet1.Cancel;
  ADODataSet1.Close;
  ADODataSet1.Open;
  ADODataSet1.Insert;
end;

procedure TTransForm.BitBtn1Click(Sender: TObject);
begin
  try
    ADODataSet1.Edit;
    ADODataSet1.post;
  ADODataSet1.Close;
  ADODataSet1.Open;
  except on E:Exception do
   begin
    MessageDlg('An Error Occured During Save Opration
!'+#13+E.Message,mtError,[mbOk],0);
   end;
  end;
end;

procedure TTransForm.FormShow(Sender: TObject);
begin
```

```
  ADODataSet1.Cancel;
  ADODataSet1.Close;
  ADODataSet1.Open;
end;

end.
```

5. Payment types:

```
unit PayUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, ADODB, StdCtrls, Buttons, Grids, DBGrids, ExtCtrls;

type
  TPayForm = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    DBGrid1: TDBGrid;
    Panel3: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Insert: TBitBtn;
    ADODataSet1: TADODataSet;
    DataSource1: TDataSource;
    procedure InsertClick(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  PayForm: TPayForm;

implementation

{$R *.dfm}

procedure TPayForm.InsertClick(Sender: TObject);
begin
  ADODataSet1.Cancel;
  ADODataSet1.Close;
  ADODataSet1.Open;
  ADODataSet1.Insert;
end;

procedure TPayForm.BitBtn1Click(Sender: TObject);
begin
  try
     ADODataSet1.Edit;
     ADODataSet1.post;
  ADODataSet1.Close;
  ADODataSet1.Open;
  except on E:Exception do
   begin
     MessageDlg('An Error Occured During Save Opration
!'+#13+E.Message,mtError,[mbOk],0);
```

```
   end;
  end;
end;

procedure TPayForm.BitBtn2Click(Sender: TObject);
begin
  Close;
end;

procedure TPayForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  action:=cafree;
end;

procedure TPayForm.FormShow(Sender: TObject);
begin
  ADODataSet1.Cancel;
  ADODataSet1.Close;
  ADODataSet1.Open;
end;

end.
```

6. Import-export:

```
unit ExpImpUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, DB, ADODB, DBCtrls, StdCtrls, Mask, Buttons, Grids,
  DBGrids;
```

```
type
 TExpImpForm = class(TForm)
  Panel1: TPanel;
  Panel2: TPanel;
  Panel3: TPanel;
  ADODataSet1: TADODataSet;
  DataSource1: TDataSource;
  ADODataSet2: TADODataSet;
  DataSource2: TDataSource;
  ADODataSet1DOC_NO: TWideStringField;
  ADODataSet1COMPANY_CODE: TWideStringField;
  ADODataSet1TRANS_CODE: TWideStringField;
  ADODataSet1TOTAL_PRICE: TBCDField;
  ADODataSet1PAYMENT_TYPE: TWideStringField;
  ADODataSet1DOCUMENT_TYPE: TWordField;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  DBEdit1: TDBEdit;
  DBLookupComboBox1: TDBLookupComboBox;
  DBLookupComboBox2: TDBLookupComboBox;
  DBLookupComboBox3: TDBLookupComboBox;
  DBLookupComboBox4: TDBLookupComboBox;
  COMPANIES: TADODataSet;
  DataSource3: TDataSource;
  DataSource4: TDataSource;
  TRANS: TADODataSet;
  DataSource5: TDataSource;
  PAYMENTS: TADODataSet;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  BitBtn3: TBitBtn;
  BitBtn4: TBitBtn;
  DBGrid1: TDBGrid;
  ADODataSet2ID: TAutoIncField;
  ADODataSet2DOC_NO: TWideStringField;
  ADODataSet2STOCK_CODE: TWideStringField;
  ADODataSet2QUANTITY: TFloatField;
  ADODataSet2PRICE: TBCDField;
  ADODataSet3: TADODataSet;
  ADODataSet2STOCKNAME: TStringField;
  ADODataSet2STOCKQUATA: TFloatField;
  ADODataSet2STOCKCODE: TStringField;
  ADODataSet2STOCKPRICE: TCurrencyField;
  ADODataSet2DOCUMENT_TYPE: TWordField;
  ADOCommand1: TADOCommand;
  BitBtn5: TBitBtn;
  BitBtn6: TBitBtn;
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure BitBtn1Click(Sender: TObject);
  procedure BitBtn4Click(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure ADODataSet2CalcFields(DataSet: TDataSet);
  procedure ADODataSet2QUANTITYChange(Sender: TField);
  procedure ADODataSet2BeforePost(DataSet: TDataSet);
  procedure DBGrid1Enter(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure BitBtn3Click(Sender: TObject);
  procedure BitBtn5Click(Sender: TObject);
  procedure BitBtn6Click(Sender: TObject);
private
  { Private declarations }
public
  DocType : Byte;
  { Public declarations }
end;
```

```
var
 ExpImpForm: TExpImpForm;

implementation

uses MainUnit;

{$R *.dfm}

procedure TExpImpForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 Action:=cafree;
end;

procedure TExpImpForm.BitBtn1Click(Sender: TObject);
begin
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM DOCUMENT_HEADER';
 ADODataSet1.Open;
 ADODataSet1.Insert;
 ADODataSet2.Close;
 TRANS.Open;
 COMPANIES.Open;
 PAYMENTS.Open;
 Panel1.Enabled:=TRUE;
 DBEdit1.SetFocus;
end;

procedure TExpImpForm.BitBtn4Click(Sender: TObject);
begin
 CLOSE;
end;
```

```
procedure TExpImpForm.BitBtn2Click(Sender: TObject);
begin
 try
   MainForm.ADODataSet1.Close;
   MainForm.ADODataSet1.CommandText:='SELECT * FROM
DOCUMENT_HEADER WHERE DOC_NO='''+DBEdit1.Text+''' AND
DOCUMENT_TYPE='+IntToStr(DocType);
   MainForm.ADODataSet1.Open;
   if MainForm.ADODataSet1.IsEmpty then
   begin
   ADODataSet1.Edit;
   ADODataSet1.FieldByName('DOCUMENT_TYPE').Value:=DocType;
   ADODataSet2.Edit;
   ADODataSet2.post;
   ADODataSet2.First;
   ADODataSet1TOTAL_PRICE.Value:=0;
   while not ADODataSet2.Eof do
   begin

ADODataSet1TOTAL_PRICE.Value:=ADODataSet1TOTAL_PRICE.Value+ADOD
ataSet2PRICE.Value;
    ADODataSet2.Next;
   end;
   ADODataSet1.post;
   ADODataSet2.UpdateBatch(arAll);
   BitBtn1Click(SENDER);
   END
   ELSE
    MessageDlg('Document Number is already exists !',mtError,[mBOk],0);
  except on E:Exception do
   begin
    MessageDlg('An Error Occured During Save Opration
!'+#13+E.Message,mtError,[mbOk],0);
```

```
  end;
 end;
end;

procedure TExpImpForm.ADODataSet2CalcFields(DataSet: TDataSet);
begin
//
ADODataSet2.FieldByName('TOTAL_PRICE').Value:=ADODataSet2.FieldByName
('QUANTITY').Value*ADODataSet2.FieldByName('STOCKPRICE').Value;
end;

procedure TExpImpForm.ADODataSet2QUANTITYChange(Sender: TField);
begin
 IF (ADODataSet2.FieldByName('QUANTITY').Value mod
ADODataSet2.FieldByName('STOCKQUATA').Value)<>0 THEN
 begin
  MessageDlg('Quantity Has Been Exceeded the Quata Limit !',mtError,[mbOk],0);
  ADODataSet2.FieldByName('QUANTITY').Value:=0;
 end
 else

ADODataSet2.FieldByName('PRICE').Value:=ADODataSet2.FieldByName('QUAN
TITY').Value*ADODataSet2.FieldByName('STOCKPRICE').Value;
end;

procedure TExpImpForm.ADODataSet2BeforePost(DataSet: TDataSet);
begin

ADODataSet2.FieldByName('DOC_NO').Value:=ADODataSet1.FieldByName('DOC
_NO').Value;
 ADODataSet2.FieldByName('DOCUMENT_TYPE').Value:=DocType;
end;

procedure TExpImpForm.DBGrid1Enter(Sender: TObject);
```

```
begin
 ADODataSet2.Cancel;
 ADODataSet2.Close;
 ADODataSet2.CommandText:='select * from DOCUMENT_DETAIL WHERE
DOC_NO=:DOC_NO  AND DOCUMENT_TYPE='+IntToStr(DocType);
 ADODataSet2.Parameters[0].Value:=DBEdit1.Text;
 ADODataSet2.Open;
 ADODataSet2.Insert;
 Panel1.Enabled:=FALSE;
end;

procedure TExpImpForm.FormShow(Sender: TObject);
begin
 BitBtn1Click(Sender);
end;

procedure TExpImpForm.BitBtn3Click(Sender: TObject);
begin
 if DBEdit1.Text<>'' then
 begin
  if MessageDlg('Are You Sure That You Want To Delete
?!',mtConfirmation,[mbYes,mbNo],0)=mrYes then
  begin
   try
    ADOCommand1.CommandText:='DELETE FROM DOCUMENT_HEADER
WHERE DOC_NO='''+DBEdit1.Text+''' AND
DOCUMENT_TYPE='+IntToStr(DocType);
    ADOCommand1.Execute;
    ADOCommand1.CommandText:='DELETE FROM DOCUMENT_DETAIL
WHERE DOC_NO='''+DBEdit1.Text+''' AND
DOCUMENT_TYPE='+IntToStr(DocType);
    ADOCommand1.Execute;
    BitBtn1Click(SENDER);
   except on E:Exception do
```

```
    begin
      MessageDlg('An Error Occured During Delete Opration
 !'+#13+E.Message,mtError,[mbOk],0);
     end;
    end;
   end;
  end
  else
   DBEdit1.SetFocus;
 end;

procedure TExpImpForm.BitBtn5Click(Sender: TObject);
var
 Code : String;
begin
 Code:=DBEdit1.Text;
 ADODataSet1.Cancel;
 ADODataSet1.Close;
 ADODataSet1.CommandText:='SELECT * FROM DOCUMENT_HEADER
WHERE DOC_NO='''+Code+''' AND DOCUMENT_TYPE='+IntToStr(DocType);
 ADODataSet1.Open;
 ADODataSet2.Close;
 ADODataSet2.CommandText:='SELECT * FROM DOCUMENT_DETAIL
WHERE DOC_NO='''+Code+''' AND DOCUMENT_TYPE='+IntToStr(DocType);
 ADODataSet2.Open;
end;

procedure TExpImpForm.BitBtn6Click(Sender: TObject);
var
 Code : String;
begin
 Code:=DBLookupComboBox1.Text;
 ADODataSet1.Cancel;
 ADODataSet1.Close;
```

```
  ADODataSet1.CommandText:='SELECT * FROM DOCUMENT_HEADER
WHERE COMPANY_CODE='''+Code+''' AND
DOCUMENT_TYPE='+IntToStr(DocType);
  ADODataSet1.Open;

  ADODataSet2.Close;
  ADODataSet2.CommandText:='SELECT * FROM DOCUMENT_DETAIL
WHERE DOC_NO='''+ADODataSet1.FieldByName('DOC_NO').AsString+''' AND
DOCUMENT_TYPE='+IntToStr(DocType);
  ADODataSet2.Open;
end;

end.
```

7. Help:

```
unit AboutUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TAboutForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutForm: TAboutForm;

implementation

{$R *.dfm}

procedure TAboutForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  ACTION:=CAFREE;
end;

end.
```