



Working With Net Express

Information in this document is subject to change without notice and does not represent a commitment on the part of Micro Focus. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used only in accordance with the terms and agreement. Unauthorized duplication is a violation of U.S. copyright and other laws, which may result in severe monetary and criminal charges.

Copyright © 2002 Micro Focus International Limited. All rights reserved. Animator®, COBOL Workbench®, EnterpriseLink®, Mainframe Express®, Micro Focus®, Net Express®, REQL® and Revolve® are registered trademarks, and AAI™, Analyzer™, Application to Application Interface™, AddPack™, AppTrack™, AssetMiner™, CCI™, DataConnect™, Dialog System™, EuroSmart™, FixPack™, LEVEL II COBOL™, License Server™, Mainframe Access™, Mainframe Manager™, Micro Focus COBOL™, Object COBOL™, OpenESQL™, Personal COBOL™, Professional COBOL™, Server Express™, SmartFind™, SmartFind Plus™, SmartFix™, SourceConnect™, Toolbox™, VS COBOL™, WebSync™, and Xilerator™ are trademarks of Micro Focus International Limited. Other company or product names mentioned herein may be trademarks or registered trademarks of their respective companies.

No part of this publication may be copied, photocopied, reproduced, transmitted, transcribed, or otherwise reduced to any electronic medium or machine-readable form without prior written consent of Micro Focus International Limited.

Micro Focus Training
9420 Key West Avenue
Rockville, MD 20850
800.318.4283

Working with Net Express
Catalog No.: TR-NX100-U0600R1-S

Preface

Course Information

Description

In this hands-on course, you will gain a working knowledge and degree of proficiency using the Net Express Integrated Development Environment for the development and maintenance of COBOL applications.

Prerequisites

To get the maximum benefit from this course, you must have experience in COBOL programming and an understanding of Windows/ PC GUI interfaces.

Audience

The audience for this course is COBOL application developers.

Objectives

Upon successful completion of this course, you will be able to use the Net Express IDE to:

- Create Net Express projects to support COBOL development and maintenance applications.

- Perform editing of COBOL programs using operations such as find and replace, find in files, block selections, cut, copy and paste, and COBOL copybook manipulations.
- Perform structured browsing and analysis of project components.
- Compile COBOL programs.
- Perform interactive source level debugging of your COBOL programs using the Animator.
- BUILD entire COBOL projects for DEBUG and RELEASE versions of a COBOL application.
- Edit supported data files using Net Express' Data File Editor and convert data file formats.

Duration

The duration of this course is three days.

Course Outline

Module 1 Introduction to Net Express

- What is Net Express?
- Developing/Maintaining COBOL Systems
- Web Development Support
- GUI Development Support
- Additional Support Tools

Module 2 Integrated Development Environment

- Introduction
- The Net Express Desktop
- Net Express Desktop Guided Tour
- IDE Windows Interface
- Multiple Document Interface (MDI)

- Docking Windows
- Available Windows
- Selecting Functions
- Documentation
- Help Systems

Module 3 Net Express Quick Start

- Overview of using the Net Express Tools in the Net Express Development Lifecycle.

Module 4 Working with Net Express Projects

- Net Express Projects
- Definition and Use of Projects
- Project Types
- Steps for Creating Projects
- Project Build Types
- Maintaining Projects
- Opening an Existing Project
- Creating a Project from an Existing Application
- Adding Components to the Source Pool View

Module 5 Create the Test Environment - External Files

- Setting Up External Files
- Assigning Files within the Program
- Assigning Files using Environment Variables

Module 6 Editing COBOL Source

- Generic Debug Build
- The COBOL Editor
- The Editor Window
- Navigating and Scrolling within a File

- Saving Source Files
- Edit Menu Functions
- Selecting Text Blocks and Working with Selected Text
- Finding Text
- Working with COBOL Copy Files
- Using Bookmarks
- Editing Toolbar Functions

Module 7 Compiling and Building COBOL Programs

- Understanding the Compilation Process
- How to Compile
- Controlling the Compiler
- Project Build Settings
- Viewing and Correcting Compiler Errors

Module 8 Testing your Application

- Unit Testing and the Development Process
- Animation Options
- Examining Data Values
- Breakpoints
- Restart
- DateWarp
- Do Statement
- Integration Testing
- View Menu
- Options, Animate Menu

Module 9 Visualize and Analyze Programs

- Program Analysis in the Development Process
- Accessing Data-Item Analysis: Quick Browse

- Procedure Find (Process Analysis)
- COBOL Reports
- Search: Locate COBOL Section
-

Module 10 Building Applications for Release

- Compile and Link using Generic Release Build
- Creating Executable Files
- Package Application Components into EXEs and DLLs
- Run-Time Systems
- Choosing a Run-Time System
- Selecting a Run-Time System
- Using the Link Wizard

Module 11 Data Tools

- ANSI vs. ASCII
- Data Files
- Viewing the Details of a Data File
- Inserting, Deleting, and Repeating Records
- Editing a File
- Editing an Individual Record
- Formatted Editing using Record Layouts
- Creating a New Data File
- Converting Files and Fixing Indexes

Module 12 OpenESQL Assistant

- ODBC and OpenESQL
- SQL in COBOL Programs
- Connecting to Data Sources (Databases)
- SQL Compiler Directive

- Using the OpenESQL Assistant
- Selecting a Table and Query
- Selecting Columns
- Specifying Search Criteria
- Running a Query
- Table Joins
- Adding Embedded SQL to a COBOL Program

Module 13 Using SQL for DB2

- Set up a DB2 application on the PC
- Develop and test DB2 applications
- Pre-test embedded SQL code interactively
- Test the pre-tested SQL code in the program

Module 14 UNIX Option

- Installing SCP and SAMBA
- Connecting to the UNIX Server
- Setting up Publishing to UNIX
- Remote Debugging
- Copying UNIX Applications to the PC

Module 15 COM, DCOM

- Create a COBOL component
- Invoke COBOL component from Visual Basic and COBOL
- Convert legacy COBOL code into components
- Use component automation with Microsoft software

Module 16 Using Services with COBOL

- Use the Mapping Wizard
- Deployment options for Web services, EJBs, and COM interfaces

- Start an enterprise server
- Deploy a Web service
- WSDL client generation
- Run client applications

Module 17 Using Java with COBOL

- Using COBOL and Java together
- Deployment process for an EJB service interface
- Use an EJB service interface

Module 18 Using XML with COBOL

- Understand XML.
- Net Express tool, CBL2XML.
- COBOL syntax extensions to support XML
-

How to Use this Book

Class Guide

This book contains information to supplement class lectures and demonstrations. You may wish to take notes in this book while the instructor is lecturing.

Reference

You can refer back to this book, the guided walkthroughs, and the practical exercises to reinforce your learning and application after class.

Practical Exercises

Exercises are available throughout the course to provide practice using the skills necessary to apply what you've learned in class back in your workplace.

Refer to the steps of the guided walkthroughs for greater detail and assistance while working with the exercises.

About Micro Focus

Company Overview

Micro Focus is the industry leader in COBOL development solutions spanning traditional maintenance and program understanding to business rule mining, Web-enablement and user-interface transformation.

With 70,000 licensed users at more than 7,000 sites around the world, Micro Focus offers unsurpassed breadth of platform support, performance and scalability, and the most comprehensive suite of development and integration environments to help customers succeed in taking full advantage of the power of their legacy systems. Founded in 1976, Micro Focus is a global company that employs more than 450 people worldwide, with principal offices in the United Kingdom and United States.

How to Contact Us

Feedback On The Course

If you have any feedback or suggestions about this course, contact us at:

Micro Focus Training
9420 Key West Avenue
Rockville, MD 20850
800.318.4283
E-Mail: training@microfocus.com

World Wide Web

The world wide web address is: www.microfocus.com

Net Express Table of Contents

Preface	i
Course Information	i
Course Outline	ii
How to Use this Book.....	vii
About Micro Focus	vii
How to Contact Us	viii
Module 1 - Introduction to Net Express.....	1
Overview	1
Objectives	1
What is Net Express?	2
Net Express Tasks	2
Developing/Maintaining COBOL Systems	2
Net Express Integrated Development Environment (IDE)	2
Using Net Express for Development.....	3
Steps for each Project.....	3
Web Development Support.....	4

Table of Contents

Form Designer	4
Internet Application Wizard	4
Solo Web Server.....	5
GUI Development Support.....	5
Dialog System	5
Dialog System Advantages	5
Additional Support Tools.....	6
Data Tools.....	6
Open ESQL Assistant	6
SQL for DB2.....	7
Source Code Control System (SCCS)	7
UNIX Option	7
Expanding COBOL's Horizon	7
Distributed Computing	8
Java-based Technologies	8
Java Development Infrastructures	9
Services	9
Web Services	10
COM	11
EJBs	12
Using Net Express to Extend COBOL Applications	12
Accessing COBOL with Web Services.....	12
Accessing COBOL with a COM Object	13
Accessing COBOL through an EJB.....	13
Accessing COBOL with JavaBeans	14

Accessing COBOL with XML.....	14
Using Server-side (CGI) Programs	14
Module Summary.....	15
Module 2 - Integrated Development Environment.....	1
Overview	1
Objectives	1
Net Express Desktop	2
Title Bar	2
MenuBar.....	3
ToolBar.....	3
Workspace	3
Output Window.....	4
StatusBar.....	4
Hover Hints.....	4
Net Express IDE Windows Interface	4
Multiple Document Interface (MDI)	5
Docking and Hiding Windows	5
Docked Windows	5
Working with Dockable Windows.....	6
Hiding Windows	6
Net Express Desktop Windows	7
Selecting Net Express Functions.....	8
MenuBar.....	9
Toolbar	10

Table of Contents

Shortcut Keys.....	10
Context Menus.....	11
Net Express Documentation and Help	12
Net Express Interface Help	13
Net Express Editor Settings.....	14
Editor Options: Profiles	14
Editor Options: Block/Clipboard Tab.....	15
Editor Options: Remaining Tabs	16
Microsoft/Micro Focus Shortcut Key Assignments	16
Customizing the Keyboard	17
Select your own Hot Key Combinations	18
Remove an Assigned Key.....	19
Customizing the Net Express Toolbar	19
Add a New Button	19
Remove an Existing Button.....	20
Change the Position of a Button	20
Reset the Toolbar to the Installed Defaults.....	20
Setting Environment Variables	21
Adding External Tools.....	22
Adding External Tools.....	22
Module Summary.....	23
 Module 3 - Net Express Quick Start.....	1
Overview.....	1
Objectives.....	1

What is the Net Express Development Lifecycle?	2
Using Net Express Throughout the Lifecycle.....	2
Organize Components in a Net Express Project.....	2
Create the Test Environment	3
Edit Application Source	3
Compile Source.....	3
Debug and Test.....	4
QA Code.....	4
Build/Deploy Application.....	4
Summary.....	6

Module 4 - Working with Net Express Projects	1
Overview	1
Objectives	1
Definition and Use of Projects	1
Benefits of a Net Express Project	2
Project Types	3
Steps for Creating An Empty Project	4
The Project Window.....	7
The Build Tab.....	8
Project Tree View.....	8
Source Pool View.....	9
Project Build Types	10
Maintaining Projects.....	11
Adding Components to Existing Projects	11

Removing Components from Existing Projects	12
Removing Components from the Current Build Type Only.....	13
Removing Selected Components from All Build Types	14
Add Components from the Source Pool View to the Project Tree View	14
Changing the Executable Type When Adding Components.....	16
Opening an Existing Project	16
Creating a Project from an Existing Application	17
Add Files to Project.....	18
Adding Components to the Source Pool View.....	20
Module Summary.....	21

Module 5 - Create the Test Environment.....	1
Overview.....	1
Objectives	1
Setting Up External Files	2
Data File Assignment on the PC.....	2
Compiler Directives.....	3
Direct Allocation via the SELECT Statement.....	3
Dynamic Allocation within WORKING-STORAGE	3
Assigning Files using Environment Variables.....	3
Assign External Files via Command Prompt or Command File.....	4
Assign External Files via Net Express IDE	4
Output Report Files.....	5
Module Summary.....	5

Module 6 - Editing COBOL Source.....	1
Overview	1
Objectives	1
The COBOL Editor.....	2
Loading Files into the Editor.....	2
Open Recently Used Files	3
Create a New File with the Editor	4
The Editor Window – COBOL Source	4
Prefix Area	4
Source Area	5
Source Area - Hover Details	5
Margins.....	6
Autofix	6
Navigating and Scrolling within a File	6
Saving Source Files	8
Edit Menu Functions	8
Selecting Text Blocks and Working with Selected Text.....	9
Using Line Marking	10
Marking Lines of Text.....	10
Marking Columns of Text	10
Working with Lines or Columns of Marked Text	11
Finding Text	12
Text Find and Replace	12
Using the Text Find and Replace Window.....	13

Table of Contents

Using the All Occurrences Option.....	14
Locate	15
Quick Browse.....	16
Find in Files.....	17
Working with COBOL Copyfiles.....	18
Saving Edited Copyfiles	20
Creating Copyfiles from In-line Source	20
Updating Project Dependencies	21
Using Bookmarks	23
Line Numbering	25
Editing Toolbar Functions.....	25
Module Summary.....	26

Module 7 - Compiling COBOL Programs.....	1
Overview.....	1
Objectives	1
Understanding the Compilation Process	2
The Compilation Process.....	2
Files Created by Compiling	2
Compiling .INT or .GNT Code.....	4
How to Compile	5
Compiling a Single Program	5
Compiling Multiple Programs	5
Project Tree View.....	5
Rebuild	6

Controlling the Compiler	6
Setting Project Level Compiler Directives.....	7
Setting Directives for a Program within a Project	7
Project Build Settings – Single or Multiple Programs	8
Build Settings for .CBL/.CPY Components: General.....	8
Build Settings for .CBL Components: Compile.....	9
Build Settings for .INT/.GNT Components	10
Setting Directives Outside a Project	11
Viewing and Correcting Compiler Errors	11
Syntax Error Help	13
Options for Setting Compiler Directives.....	13
Module Summary.....	14

Module 8 - Testing Your Application.....	1
Overview	1
Objectives	1
Testing and the Development Lifecycle	2
Prerequisites to Animation	2
Animation Options	3
Starting the Animator	3
Execution Controls.....	5
Step: Single-line Step.....	5
Step All: Multi-line Step	6
Run.....	6
Run Thru/Run Return with Execute a Perform/CALL Range	7

Table of Contents

Run Thru	7
Run Return.....	8
Run to Cursor.....	9
Skip Statement.....	9
Skip Return	9
Skip to Cursor	10
Examining Data Values	10
The Examine List Window	11
Other Options in the Examine List Window	12
Watch List Window	12
Icons on the Watch List window support the following functions:	13
Working with Table Variables.....	13
Breakpoints.....	13
Normal Breakpoints	14
Conditional Breakpoints	15
Do Breakpoints	15
Program Breakpoints	16
Break on Data Change from the Watch List	16
Controlling Breakpoints	16
Locate and Cycle through Breakpoints.....	16
Times, Break every n Times a Line is executed	17
Clearing/Disabling Breakpoints	17
Restart	18
DateWarp	19
Do Statement.....	19

Integration Testing	20
View Menu Animator Options	21
Animator Sub-Windows	21
Options, Animate Menu	22
Module Summary.....	23
Module 9 - Analyze Source Programs.....	1
Overview	1
Objectives	1
Program Analysis in the Development Process.....	2
Quick Browse – Procedure and Data-Item Analysis.....	3
Data-Item Analysis	3
Procedure Analysis	5
Browse - Section/Paragraph, Data, Calls, and Programs / Entry points.....	5
Line Type Indicators	7
COBOL Reports.....	7
Dead Data	8
Unreferenced Data.....	9
Undeclared Procedures	9
Copyfile Structure.....	9
Unexecuted Procedures.....	9
Program Statistics	10
Module Summary.....	10
Module 10 - Building Applications for Release.....	1

Table of Contents

Overview	1
Objectives	1
Compile and Link using Generic Release Build	2
Creating Executable Files.....	2
.OBJ Modules.....	2
.EXE Modules	3
.DLL Modules	3
Types of Build	3
Packaging Application Components.....	4
Decide on the Application Packaging.....	5
Packaging an .EXE	5
Packaging a .DLL.....	6
Completed Release Build Example	6
Run-Time Systems	7
Static-Linked Run-Time System	8
Shared Run-Time System.....	8
Support Modules.....	9
Run-Time System Advantages/Disadvantages	9
Completely Modular Application.....	10
Modular/Some Elements Statically Linked.....	10
Using the Net Express Run-Time System	11
Using the Link Wizard.....	13
Invoking the Link Wizard.....	13
Setting Advanced Linker Options	15
Additional Advanced Linker Options	16

Packaging Applications using .INT and .GNT Formats	17
Advantages of using Micro Focus .INT and .GNT executables	17
Disadvantages of using Micro Focus .INT and .GNT executables	17
Libraries.....	18
Create Applications for Distribution using.INTs and .GNTs.....	18
Creating a Micro Focus Library File	20
Create an .EXE Trigger Program for your Application	20
Module Summary.....	21

Module 11 - Data Tools.....	1
Overview	1
Objectives	1
ANSI vs. ASCII.....	2
Data Files.....	3
Header Records	3
Data File Extensions	4
Adding a Data File to your Project	5
Opening a File in the Data File Editor	5
Opening an Indexed File in the Data File Editor.....	6
Opening a Non-Indexed File in the Data File Editor.....	6
The Data File Editor Display.....	7
Viewing the Details of a Data File	7
PC File Tab Information.....	7
Keys Tab Information	8
General Tab Information.....	8

Table of Contents

File Maintenance – Records	8
Inserting, Deleting, and Repeating Records	8
Insert Record Before/After	9
Insert Indexed Record.....	9
Repeat Record.....	9
Delete Record	9
Initialize Record.....	10
Undo Record Edit.....	10
Multi Repeat Records.....	10
Multi Delete Records.....	10
File Maintenance - Editing an Individual Record	10
Editing in a Different Key Order	11
Editing Hexadecimal Data.....	11
Searching.....	12
Formatted Editing using Record Layouts.....	13
Creating a Record Layout File	13
Multiple Record Types	13
Saving the Record Layout.....	14
Editing a Data File: Formatted Editing	14
Editing/Viewing the Record Layout	15
Creating New Data Files	15
Create a Non-Indexed File	15
Creating an Indexed File	15
Converting Files and Fixing Indexes.....	16
Fixing File Indexes	17

Module Summary.....	18
 Module 12 - OpenESQL Assistant 1	
Overview	1
Objectives	1
ODBC Interface	2
ODBC Drivers	2
SQL in Net Express COBOL Programs	2
SQLCA	3
Host Variables	3
Connecting to Data Sources (Databases).....	3
Methods to Connect to a Database.....	4
SQL Compiler Directive.....	4
OpenESQL Assistant.....	5
OpenESQL Assistant Functionality	6
Starting the OpenESQL Assistant.....	7
Connect to Data Source	8
Selecting a Table and Building a Query	8
Running a query	10
Query Details.....	11
Specifying Search Criteria.....	11
Sort	13
Auxiliary Code	15
Table Joins	15
Adding Embedded SQL to a COBOL Program	17

Table of Contents

Appendix A - Registering and Connecting a Database	19
Registering a Database.....	19
Connecting to a Database.....	20
Appendix B – Coding SQL Statements.....	22
Embedded SQL Statements	22
SQLCA.....	22
Host Variables.....	23
Declaring Host Variables.....	24
Null Values	25
Data Types.....	26
Integer Data Types.....	27
Character Data Types	28
Cursors	29
Positioned UPDATE and DELETE Statements	31
Update (Positioned)	31
Delete (Positioned).....	31
Searched UPDATE and DELETE	32
Module Summary.....	34
Module 13 Using SQL Option for DB2.....	1
Overview.....	1
Objectives	1
What is SQL Option for DB2?.....	2
Terminology	3
Steps to Use SQL Option for DB2	3

Prepare DB2 Subsystem	4
Download the Source Files	4
JCL to execute GOLFJOIN	5
Prepare DB2 Subsystem: Download DSNTIAUL Files	5
Unload Table Data on Host using the DSNTIAUL Utility.....	5
Download the DSNTIAUL Files	5
XDB Link	6
Accessing the Host	6
Table Names on the Host	6
Module Summary.....	7
Module 14 - UNIX Option.....	1
Overview	1
Objectives	1
Net Express UNIX Tools	2
Portability Issues	2
Installing SCP and SAMBA.....	3
Configuration Information.....	3
Server Capabilities	5
Filename Mapping.....	5
Pre-build and Post-build Commands	6
Copyfile Handling	6
System Copyfiles.....	7
Connecting to the UNIX Server.....	7
Logging on to your Server and Starting Server Express or OCDS	7

Setting up Publishing to UNIX	9
Remote Debugging.....	10
Copying UNIX Applications to the PC.....	11
Using Import to Create your Project.....	12
Module Summary.....	13

Module 15 - Introduction to COM and DCOM 1

Overview.....	1
Objectives	1
COM	1
COM Example.....	2
Loading a COBOL Component Application	2
Registering the COBOL Component.....	3
Using the Component from Visual Basic	3
Using the Component from Dialog System and COBOL	5
More about the Stock Application	6
Automating Word and Excel from COBOL	7
Finding Available Word and Excel Methods	8
DCOM.....	8
Microsoft Transaction Server.....	10
What is MTS?	10
What problems should I look out for using MTS?.....	10
Module Summary.....	11

Module 16 - Using Services with COBOL	1
Overview	1
Objectives	1
Services Review	2
Web Services Review	2
Using Net Express for Services	2
Interface Mapping Toolkit.....	2
Creating a Web Service.....	3
Mapping the Fields	3
Using the Mapping Wizard	4
The Web Service wsCalc.....	7
The Interface Fields	7
New Files Created	8
New Menu Items.....	8
Defining Your Own Interface Mapping	8
Using Preset COBOL Values	11
Introduction to Deploying Services	13
Micro Focus Enterprise Server.....	13
Using Enterprise Server.....	14
Deploying Applications as Services	14
Managing Deployed Services.....	14
Exposing Services to Remote Clients	14
Executing in a Stable Environment	15
Getting Started with Enterprise Server	15

Table of Contents

Installing Enterprise Server	15
Starting Enterprise Server.....	15
Configuring the Enterprise Server	17
Deploying Services with Enterprise Server.....	20
COBOL Web Services	20
COBOL/J2EE Applications.....	20
Transactional COBOL Applications	20
Deploying a Web Service	21
New Files Created.....	22
Consuming Client Applications	24
Net Express and Client Generation	24
Generating a Client Using Mapping	24
Generating a Client using WSDL	27
Module Summary.....	30

Module 17 - Using Java with COBOL.....	1
Overview.....	1
Objectives	1
Using COBOL and Java Together.....	2
Calling COBOL from Java.....	2
Calling Java from COBOL.....	2
Accessing COBOL with an EJB Service Interface.....	2
Mapping the EJB and COBOL Program.....	3
The COBOL Program	3
Creating an EJB Service Interface	5

Deploying the Mapping	5
New Files Created.....	6
Deploying to the Application Server	7
Resource Adapters	7
Deploying with an Unmanaged Connection.....	8
The Client Program.....	8
Summary.....	10

Module 18 - Using XML with COBOL.....	1
Overview	1
Objectives	1
About XML	2
XML and HTML	2
XML Syntax.....	2
XML Tags and Elements.....	2
Example	3
XML Declaration.....	3
Root Element	4
XML Attributes.....	4
Comments in XML.....	5
Using Special Characters.....	5
XML Document Structure	5
Nesting	5
Well-formed Documents	5
Valid XML Documents.....	6

Table of Contents

Using Namespaces	6
Net Express and XML.....	7
XML-enabling a COBOL application	7
CBL2XML Wizard and Utility	8
Generating an XML Schema and Mapping Records	8
XML Syntax Extensions.....	13
COBOL Verbs	14
Generating an XML Instance Document.....	15
Using XML PARSE and XML GENERATE.....	16
Module Summary.....	18

Module 19 - Using Consolidated Tracing Facility	1
Overview	1
Objectives	1
About CTF	2
Benefits	2
CTF Architecture.....	3
CTF Static Configuration Method	4
Setting the MFTRACE_CONFIG Environment Variable	4
CTF Configuration File.....	5
CTF Configuration File Generator.....	5
CTF Dynamic Configuration Method	6
CTF Emitters	7
CTF TEXTFILE Emitter.....	8
CTF BINFILE Emitter	10

CTF WINEVENT Emitter.....	11
CTF IDEDBG Emitter	12
CTF Enabled Product Components.....	14
Module Summary.....	15

Table of Contents

Module 1

Introduction to Net Express

Overview

Net Express is an integrated, project-based, graphical COBOL application development environment that allows you to create, maintain, and support PC-based applications, GUI-client/server applications, and web-based applications.

Objectives

At the end of this module you will be able to:

- Discuss the tasks Net Express helps you to perform.
- Describe the facilities Net Express IDE provides.
- Understand the tasks per project or per programming task when developing using Net Express.
- Understand the way Net Express options can be employed in a distributed computing environment.
- Describe the client and server roles in a distributed computing environment.
- List the technologies supported by Net Express for exposing COBOL applications as services.
- Describe the technologies that Net Express supports for accessing COBOL programs.

What is Net Express?

Net Express is an Application Development Environment. All the software necessary for complete lifecycle coverage is integrated into a graphical environment including development and testing tools. Integration of tools with the organization's software can also be accomplished. This allows developers to multi-task and use graphical development facilities across a variety of project requirements.

Net Express Tasks

Net Express helps you to perform these tasks

- Organize and manage application components
- Create, edit and convert data files
- Create, modify and maintain application source code
- Syntax check application source code
- Unit, integration and regression test application source code
- QA source code and applications
- Compile, link or build application executables.

Developing/Maintaining COBOL Systems

Net Express Integrated Development Environment (IDE)

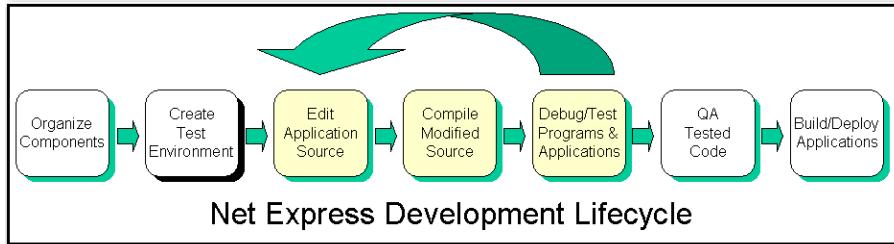
The Net Express IDE provides COBOL developers with facilities to:

- Edit and compile COBOL programs.
- Interactively debug COBOL programs using the Animator, an interactive source level debugger.
- Perform structured browsing and analysis of COBOL programs and other project components.
- Use mouse clicks to access product functions based on application component type.

- Build applications for testing or distribution using simplified, easy to access automated tools.
- Create, edit and convert data files.

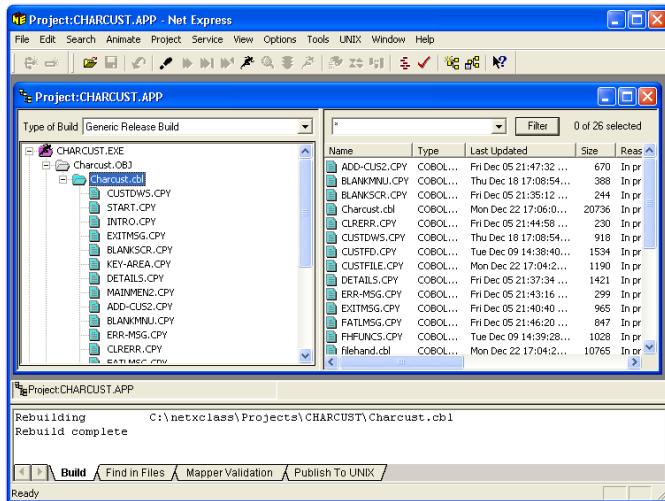
Using Net Express for Development

The step for developing and deploying an application with Net Express are summarized in the Net Express development Lifecycle shown below.



Steps for each Project

Net Express Projects are used to organize source components related to a system, an application, or a program. For each project, the application files must be acquired and organized on the PC. The Net Express Project Window is used to manage and maintain application components, throughout the development lifecycle steps, on the desktop.

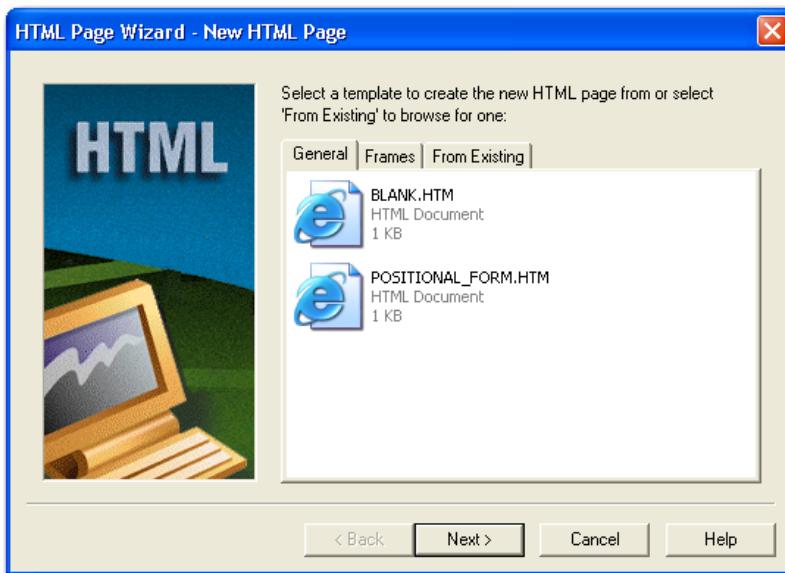


Web Development Support

Net Express provides tools to create, develop, debug and build Internet and Intranet applications using COBOL, Embedded HTML, HTML, ActiveX, and JavaScript. Tools include the Form Designer, Form Express and Solo Web Server.

Form Designer

The Form Designer is used to create user interfaces in either HTML or ActiveX layout forms. It generates COBOL CGIs to handle I/O of the forms (including data conversion and validation). The Event Editor enables events to be connected to methods or properties of other controls or forms using JavaScript.

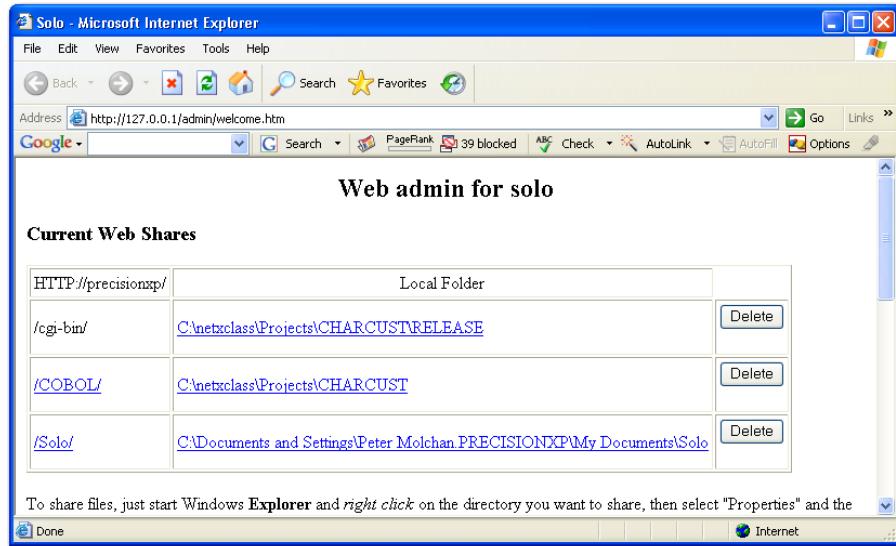


Internet Application Wizard

Use Internet Application Wizard to create web pages from the Linkage Section of an existing COBOL program. This enables applications to take user I/O via the web instead of screen I/O.

Solo Web Server

The Solo Web Server allows debugging of Internet applications on the development machine and eliminates the need for developers to know how to set up or run a web server.



GUI Development Support

Dialog System

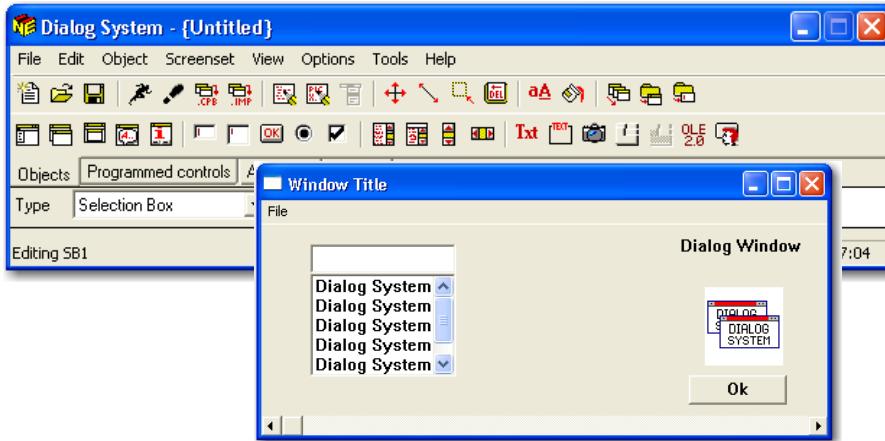
Net Express supports the development of Windows GUI-client/server applications through Dialog System, a tool designed to be used in conjunction with COBOL.

Dialog System Advantages

Dialog System GUI interfaces can be prototyped or customized (normally) without impacting the application program which eliminates the need to re-compile programs when the interface changes.

Dialog System separates the user interface from the application's business logic isolating screen, keyboard, and mouse events from the COBOL

program, reducing its complexity and size considerably and, more importantly, leaving the business logic intact and in COBOL where it has always been.



Dialog System considerably reduces the retraining required for COBOL developers to develop Windows GUI applications allowing them to be productive in a matter of days, not weeks or months.

Additional Support Tools

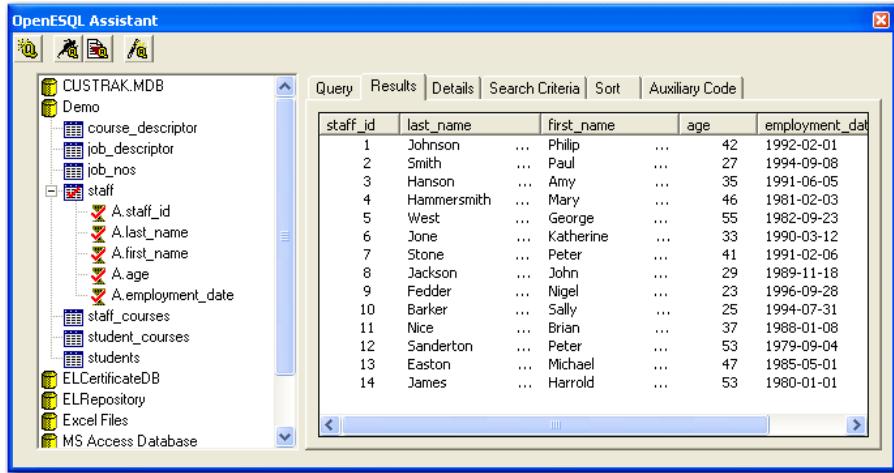
There are additional support tools fully integrated into the Net Express IDE. These tools are discussed below.

Data Tools

These tools allow developers to create data files, edit data files and convert data files to other formats.

Open ESQL Assistant

This tool allows developers to create and test SQL queries using mouse clicks rather than hand-coding embedded SQL. Having built the query, the OpenESQL Assistant allows developers to include embedded SQL in a COBOL program to connect, query and update an SQL database.



SQL for DB2

DB2 RDBMS emulation includes a complete development environment with link to host DB2 databases.

Source Code Control System (SCCS)

A source code control system (SCCS) allows users to manage and control access to source components in a multi-user environment. Each time you open Net Express, it checks to see if you have a compatible SCCS installed. If detected, support is provided by means of a Source Control menu on the IDE menu bar.

UNIX Option

The UNIX Option supports deployment of Net Express applications to a UNIX server and provides tools to support the complete development of UNIX applications using Net Express.

Expanding COBOL's Horizon

Many organizations have a large percentage of their business logic in existing COBOL applications. With Net Express, these organizations can take advantage of the Web, distributed computing, and other new technologies in their business strategies.

Net Express provides features to extend these existing COBOL applications and use them as components or services for access by other client applications written in various programming languages such as Java, C/C++, VB, .NET, C#, JavaScript, etc.

Distributed Computing

In a distributed computing environment two applications work together, one as the client application and one as the server. The client application makes requests by invoking a server application to process the request or to provide a service to the client. The server application is also known as a service.

These client and server applications may be on different machines, different platforms and even written in different languages. The machines may be connected by any kind of communications layer over any kind of network.

The server software is deployed under an application server, which is an application host that listens for requests coming in from clients (in COM, Windows itself acts as the application server). Once the server software is deployed, new clients can be written to use it.

Application servers often provide additional features such as security and scalability, providing valuable tools that the application designer can use to their advantage.

Java-based Technologies

The Java Platform provides support for Enterprise JavaBeans, servlets, portlets (following the Java Portlet specification), JavaServer Pages and several web service technologies. The Java Platform combines the EJB component architecture with these other enterprise technologies on the Java platform for seamless development and deployment of server side applications providing the scalable architecture for executing business logic in a distributed computing environment. The Java Platform specification provides a wide range of services available to applications that are deployed utilizing its technology.

Two major component models in the Java environment are JavaBeans and Enterprise JavaBeans. These are described below.

JavaBeans - JavaBeans is a component model for developing reusable software components for the Java platform. Although these components are officially called Beans, they are commonly referred to as JavaBeans.

Enterprise JavaBeans - Enterprise JavaBeans (EJB) is an extension of the JavaBeans component model that provides for the needs of transactional applications. This API specification allows you to develop and deploy enterprise-level applications that are distributed, object-oriented, scalable, transactional and secure. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and secure. These components are often referred to as EJBs.

Java Development Infrastructures

The cornerstone of all Java technologies is the standard definition and implementations of the infrastructures for developing and deploying applications. Currently, there are three basic infrastructures to which Java Platform conforming applications can be deployed.

Java Platform, Micro Edition (J2ME) - a highly optimized Java runtime environment. The J2ME technology specifically addresses the vast consumer space, and covers a wide range of devices including pagers and smart cards.

Java Platform, Standard Edition (J2SE) - a compiler, runtime system, APIs and other tools for writing, deploying, and running applets and applications developed in the Java programming language.

Java Platform, Enterprise Edition (J2EE) - a superset of J2SE and provides an infrastructure designed for developing and deploying enterprise applications.

Services

Services are applications that process a client's requests. During the creation of their responses they typically access enterprise resources such

as databases. Below are three interface types that services created with Net Express can expose.

- Web services
- COM
- Enterprise JavaBeans (EJB)

Web Services

Web services is a set of standards and mechanisms that enables software components to be deployed and then invoked across the Internet or an intranet. Requests and responses passed between client and service are encoded in XML. This means that the client application need not know anything about the language or deployment mechanism of the service it is invoking. Only the name of the desired service and its parameter structure needs to be known by the client.

Here are some of the benefits of using Web services.

- **Accelerate delivery of applications** – Web service components may be created from existing, proven code or accessed from a third party vendor.
- **Reduce development costs** – By using existing, proven code, development costs are greatly reduced.
- **Reusable** – Web services are designed as components that can be shared and reused by various applications.
- **Accessible by many clients** – Client programs can be written using various technologies such as Java, C++, .NET ,etc.
- **Widely available** – Web services are available on the web, thus making them available to everyone.

The following are some of the standard web protocols used by Web services.

- **HTTP - HyperText Transfer Protocol** is the World Wide Web standard for communication over the internet. It is a set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) via

the web. Relative to the TCP/IP suite of protocols, which are the basis for information exchange on the Internet, HTTP is an application protocol.

- **XML - eXtensible Markup Language** provides the ability to identify or markup data in a text document using descriptive, user-defined tags. These tags describe the content and use of the data and are generally human-readable. XML creates structure to allow data to be exchanged or stored, but does not manipulate the data. Applications written in languages such as HTML manipulate the data.
XML documents follow the rules of a *Document Type Definition* (DTD) file or *XML Schema*.
- **SOAP – Simple Object Access Protocol** is a standard format in XML for sending data between a client and a Web service. The service's WSDL file specifies the format of the SOAP messages it expects and will return.
- **WSDL – Web Services Description Language** is a standard format in XML for describing the input data that a Web service expects from its clients and the output data it will return to its clients. In deploying a Web service you must create and deploy its WSDL file.
- **UDDI - Universal Description, Discovery and Integration** is a standard format in XML for describing the name, location on the Web, and functions of a Web service. A UDDI registry provides a directory service where you can register and search for web services.

COM

Common Object Model (COM) is the Microsoft component model for Windows platforms. Services may be built using COM technology. COM enables an application to expose functionality as a service that is deployed and registered with Windows for other applications to use. A certain amount of Windows programming expertise is needed to code the interface between client and service (known as a COM object).

Recent versions of Windows support remote access to COM objects and provide services such as security and scalability, (as application servers do

in other environments). This improved version of COM is called COM+, but is common practice to simply call it COM.

EJBs

As stated earlier, EJBs are components of enterprise-level applications. Through the use of the Java Connector architecture, a Net Express COBOL service is represented in J2EE through an EJB interface – and your enterprise applications can then easily access this COBOL service.

Using Net Express to Extend COBOL Applications

Net Express offers tools for developing distributed systems. These tools help to create and deploy service interfaces for your COBOL applications so that client software written in various non-COBOL technologies can invoke them.

The technologies supported are listed below.

- Web services
- COM
- EJB
- JavaBeans
- XML
- CGI

Accessing COBOL with Web Services

Net Express can be used to create, consume and deploy a Web service entirely in COBOL. Net Express provides an Interface Mapping Toolkit to map the COBOL program to a Web services interface and deploy the service so a client application can then access the COBOL program as a Web service.

The Web service client support is capable of generating a COBOL client program that can access COBOL Web services created with the Interface Mapping Toolkit as well as Web services created with third party Web service development tools.



Accessing COBOL with a COM Object

Net Express creates interoperability between Microsoft .NET and COBOL programs by providing tools to build a COM object directly from an existing COBOL linkage section. Then, a client application on Windows can invoke the COM object to directly access the COBOL program.

Net Express can also be used to create COBOL applications as COM objects by using the Class and Method Wizards.

Accessing COBOL through an EJB

Use the Interface Mapping Toolkit to map your COBOL program to an external EJB interface. A client application can then access the COBOL application through an EJB.

A J2EE application server (for example, IBM WebSphere or BEA WebLogic) is required to run the EJB, in addition to Enterprise Server which runs the COBOL. EJBs are a feature of the Java specifications, and your clients will need to be written using Java-type facilities (for example, JSP or Java Swing). Java expertise is needed to code the interface between client and EJB.

Accessing COBOL with JavaBeans

JavaBeans is another Java technology that can interact with a Net Express COBOL application.

Deploy JavaBeans under a Java Virtual Machine, a run-time environment for Java, which is downloadable from the Web. The specification that all Java Virtual Machines conform to is J2SE, and deploying a JavaBean under a Java Virtual Machine is often referred to as deploying it under J2SE.

You will need Java expertise to code the interface between client and JavaBean. A client must call a JavaBean directly, using some remote procedure call mechanism.

JavaBeans don't run under an application server and so such things as scalability and security are not handled automatically. Also the JavaBeans are not kept readily available for independent clients to invoke.

You create JavaBeans using the Class and Method Wizards. The Wizards automatically generate a Java wrapper for a JavaBean class which is written using Net Express COBOL. The wrapper contains a corresponding Java method interface for each COBOL method which is contained within the class.

Accessing COBOL with XML

Net Express supports XML schemas enabling you to pass data between XML documents and COBOL applications.

Net Express also provides a series of XML syntax extensions that you can code into your COBOL program manually, or generate automatically using the CBL2XML Wizard or command-line utility. The syntax extensions enable your COBOL program to perform input and output on an XML instance document rather than using a traditional file such as an indexed file.

Using Server-side (CGI) Programs

For Web applications, an older technology that you can use is the server-side program, which runs under control of a Web page server, such as IIS, and can be considered an extension of it. Server-side programs are most

often known as CGI programs, or simply CGIs, after the most common standard, the Common Gateway Interface.

The CGI program does not have a separate client application, but it sends HTML forms itself to Web browsers to provide the user interface. CGIs aren't regarded as components or services, because they are not kept deployed ready for any client to invoke.

For CGIs, you need a Web page server, such as IIS, and depending on how you have built your application you might also need Application Server for Net Express to run the COBOL application.

Module Summary

This module served as an introduction to the Net Express Development Environment. At this point, you should be familiar with the following:

- Net Express is an integrated development environment that provides facilities to develop or extend COBOL applications to the web or as client/server applications.
- Net Express assists the developer with the entire application development lifecycle including project creation, modification, testing and building.
- Form Designer, Form Express and Solo Web Server are tools to assist with web development.
- Dialog System assist with developing Windows GUI-client/server applications.
- Distributed computing environments take advantage of the separation of client and server applications to provide features such as security and scalability.
- Net Express interfaces to various new technologies to expose existing or newly created COBOL application logic for use by client applications via the web or other distributed computing environments.

Module 2

Integrated Development Environment

Overview

Before exploring how to use Net Express for developing applications you need to become familiar with the Net Express Integrated Development Environment (IDE). The Net Express IDE is a dynamic and very flexible environment that can be tailored for optimum productivity. This module introduces you to the desktop: windows, menus, buttons, and other facilities of the IDE.

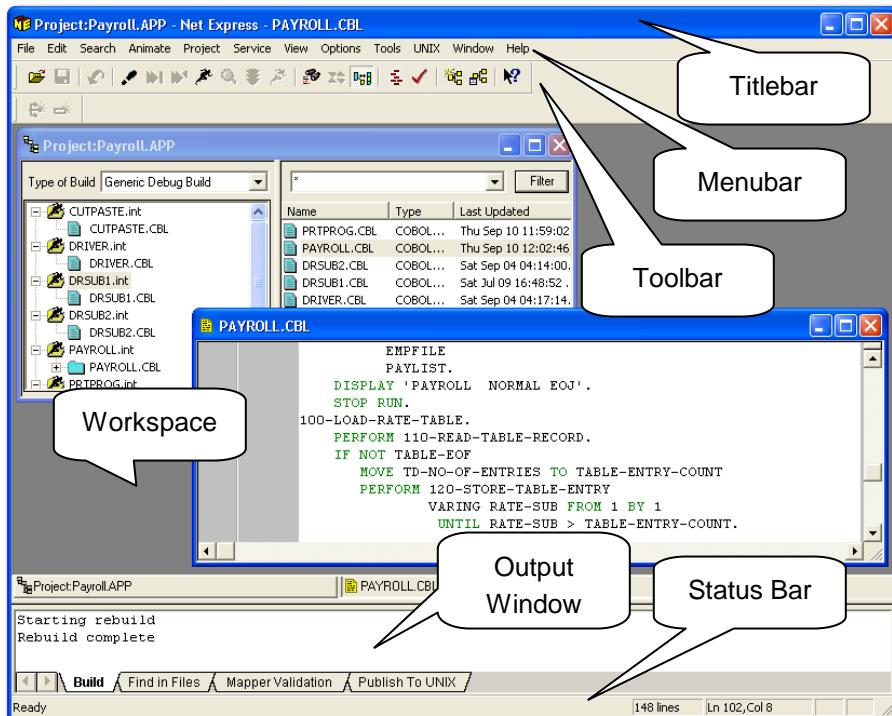
Objectives

After completing this module you will be able to:

- Identify the windows, icons, and functions that make up the Net Express desktop interface.
- Use the Multiple Document Interface to arrange your desktop, including dockable windows.
- Invoke tools and functions using the menu bar, tool bar, function keys, and context menus.
- Use the on-line Help facility to obtain assistance.
- Customize the Net Express IDE: Microsoft/Micro Focus Shortcut Key Assignments, Net Express toolbar, Net Express shortcut keys.

Net Express Desktop

The Net Express IDE Desktop provides developers a flexible environment with windows, menus and buttons that can be tailored for optimum productivity.



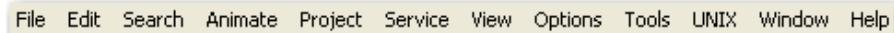
Title Bar

Use the title bar to move, minimize, maximize, restore, and close the Desktop. When a project component, such as a COBOL program or copybook, is loaded, the project and filename of the component is displayed.



Menu Bar

The menu bar provides access to Net Express major functions.



Tool Bar

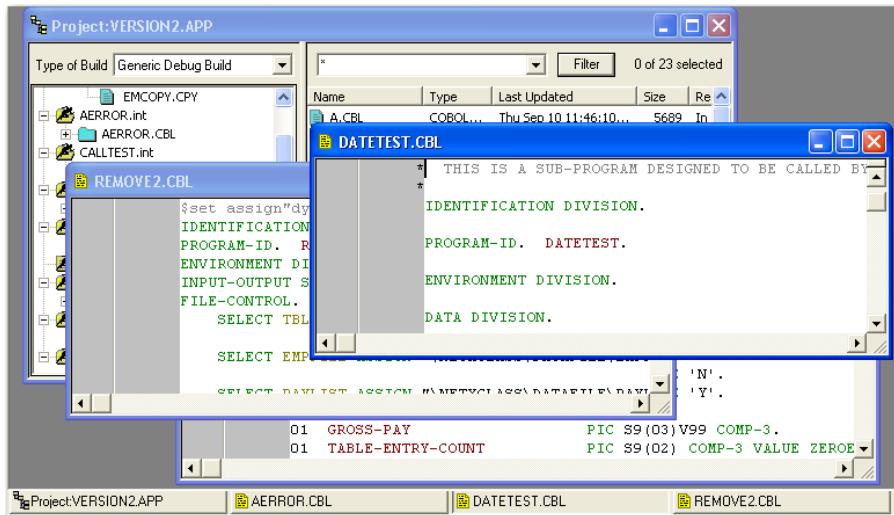
The toolbar provides alternate access to often-used functions using a single mouse click.



Workspace

The Net Express workspace typically displays one or more windows when a project is open. For example:

- A Project Window displaying lists of executable components and source components
 - When editing COBOL programs an edit window displays in the workspace for each program
 - When testing a program an Animator window displays in the workspace.



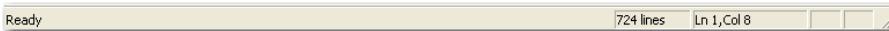
Output Window

The scrollable Output Window displays messages for the Net Express Build, Search results and other functions.



Status Bar

The Status Bar displays information about the desktop and the current function (e.g. displays number of lines and current line/column in the edit session shown above).



Hover Hints

When the mouse hovers over a button, the purpose of that button displays.



Net Express IDE Windows Interface

The Net Express IDE utilizes the latest Windows technology listed below:

- Multiple Document Interface (MDI)
- Docking windows
- Windowpanes to display various functions within Net Express
- Context menus (right mouse click) to maximize the user's access to the different product functions.

Multiple Document Interface (MDI)

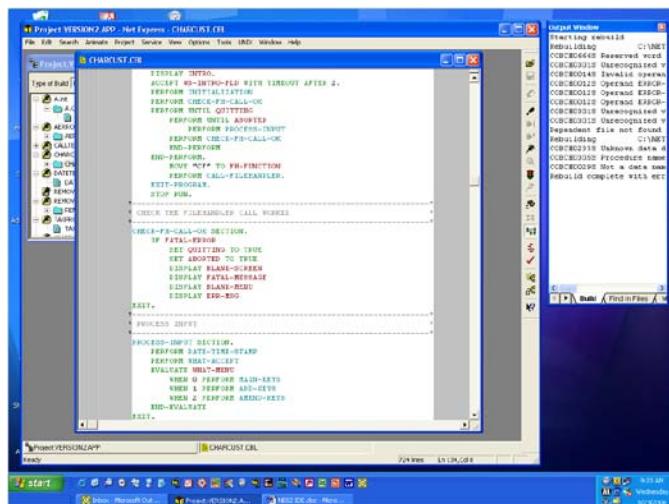
Net Express uses a Multiple Document Interface to display multiple windows within the workspace or client area needed by the functions you perform in Net Express. The size of the window display is limited by the size of the workspace; this means it is not possible to overlay the Net Express desktop with lower level windows.

Docking and Hiding Windows

Some windows are displayed as part of the desktop. These windows include tool bars and those that display specific information about the functions being performed. Understanding the purpose and organization of these windows helps you to control the desktop display and provide the maximum viewing area in the workspace.

Docked Windows

Docked windows (by default) are affixed to the Net Express desktop. You can un-dock and move these windows elsewhere on the desktop or anywhere convenient on the screen. In the example below, the Tools Menu has been moved to the right side of the desktop and the output window moved off the desktop. Settings are preserved from session to session.



Working with Dockable Windows

There are two ways to detach docked windows.

- Move the mouse to an unused area of the window (inside or the border), then hold down the left button and drag the window.
- Right-click on the window and de-select **Allow Docking**. Place the mouse on the window's title bar and move the window.

The window floats above the Net Express desktop and can be moved freely to any location. The window can only be docked if Allow Docking is checked. If Allow Docking remains checked, moving it to the left, right, top or bottom of the Net Express desktop will re-dock the window.

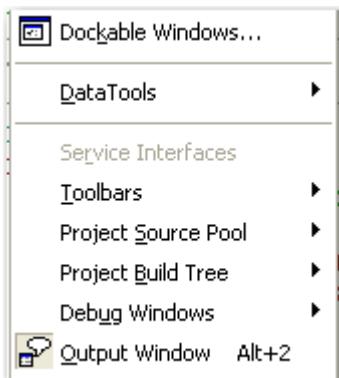


Hiding Windows

When windows are not required by the function being performed, you can hide them to get a larger workspace. Once hidden, these windows automatically redisplay when the user requests or performs a function requiring that window.

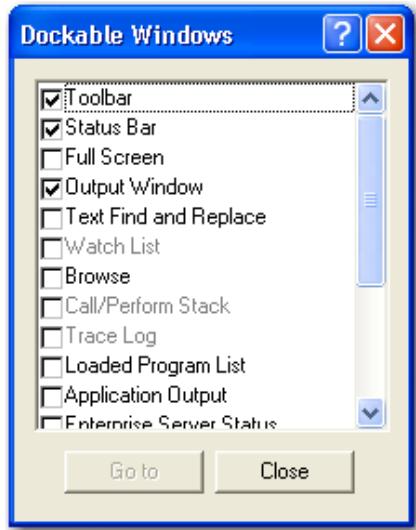
Follow these steps to redisplay windows that do not return automatically.

1. Select **View**, choose window.
2. Or select **View, Dockable**, choose window.



Net Express Desktop Windows

Select **View, Dockable Windows** for a list of all windows that may be displayed on the desktop. Use checkboxes to select/deselect windows.



Use this Window...	To...
Toolbar	Control the display of the toolbar.
Status Bar	Toggle the display of the status bar on/off. The status bar line always appears attached to the bottom of the desktop.
Full Screen	Display a full screen view of the current open window.
Output Window	Display messages resulting from Build, Find-in-Files, Unix Publishing and Source Control operations.
Text Find and Replace	Find/replace text strings when editing/animating programs.

Watch List	Display a list of variables and their values when animating.
Browse	Browse and perform structured analysis.
Call/Perform Stack	Display the nesting level of execution (by paragraph name) of a program when executing.
Trace Log	Debug an object-oriented application to log application messages.
Loaded Program List	Display a list of all loaded programs when animating.
Application Output	Display screen input/output when animating.
Enterprise Server Status	Select and Refresh the Enterprise Server.
OO COBOL	Display a toolbar containing tools associated with Object COBOL development.
Map Screen	Window used to view output screens.
Mapper	Tool Menu used for Mapping functions.
OpenESQL Assistant	Display the OpenESQL Assistant window used to select and connect to a data base source, develop and test queries, and embed SQL coding into COBOL source programs.
Data File Toolbar	Display the Data File Editor toolbar.
Data File Find and Replace	Find/replace text strings when editing data files.

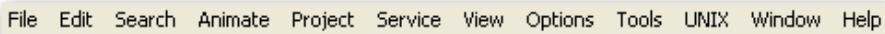
Selecting Net Express Functions

Use these standard Windows techniques to access functions in Net Express.

- Menu Bar
- Toolbar
- Assigned Shortcut Keys
- Context Menus

Menu Bar

Menu Bar options are described in the table below.

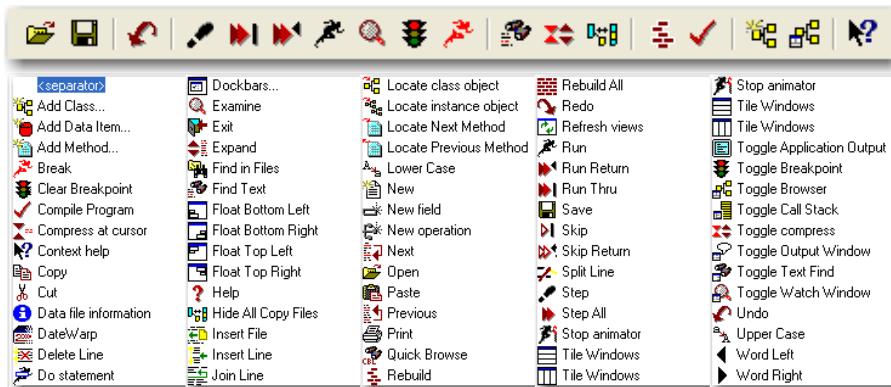


Use this Function...	To...
File	Open existing projects/files, create new project/files, insert files, close opened projects/files, import HTML forms, view copyfiles, and print window contents. The file menu maintains a list of the most recently opened projects/files for easy access.
Edit	Access Line, block and other edit functions for working with COBOL source.
Search	Perform Find/Replace functions; also used to perform structured browsing and analysis of COBOL programs and projects.
Animate	Access Testing functions for animating a COBOL program. Set Breakpoints and control the speed and flow of execution. Examine and set data values.
Project	Change Net Express project settings and control parameters.
SourceControl	Access a source code control system (Optional window when a Source Control option is installed).
View	Alter and control the current windows, and toggle the display of windows on and off.
Options	Set configuration options for Net Express functions.
Tools	Access additional tools from Net Express.
UNIX	Access UNIX tools such as publishing to a UNIX system and creating character-based Dialog System

	interactive applications for UNIX (Optional menu displays when the UNIX option is installed).
Window	Use standard windows interface options to manage multiple windows.
Help	Access the Net Express Help system.

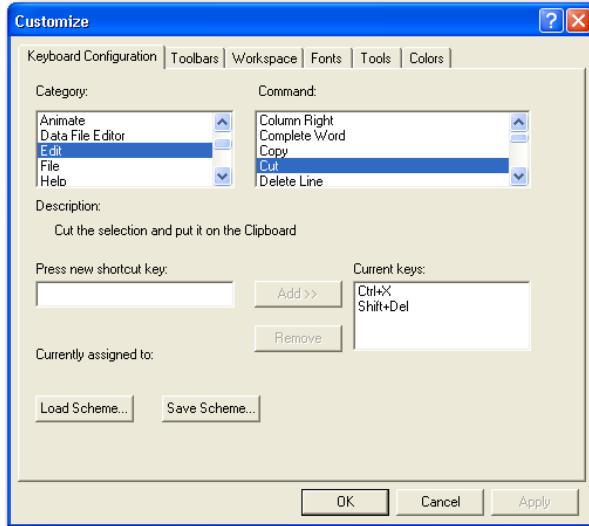
Toolbar

The Toolbar provides you with easy to use, single-click access to an array of often-used functions. You can customize the default icons and configuration of the toolbar for the way you work. This will be covered in a section of this module to follow.



Shortcut Keys

Many functions available through the menu bar and toolbar may be accessed by pressing pre-assigned shortcut keys. Net Express supports both Microsoft and Micro Focus standard shortcut-key assignments. You can also assign your own shortcut keys.



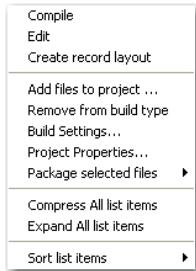
Select **Options, Customize IDE** to customize these keys. **Commands** and their **Current key** shortcuts are listed within the customization **Category**. This will be covered in a section of this module to follow.

Context Menus

Context menus are used extensively in Net Express to allow you quick access to appropriate functions without having to access the menu bar, toolbar or shortcut keys. Context menus are displayed when you right click the mouse on an object. The choices displayed in the pop-up window are sensitive to what the mouse pointer was on (i.e.: its context) when selected.

Context menus are the quickest and most effective way to invoke functions within Net Express and, therefore, should be the preferred method used.

The context menu below is for a COBOL component in the Project Window.



Net Express Documentation and Help

Net Express documentation and help is accessed from the Help option in the main menu. Standard Net Express documentation is installed on your system with the Net Express software. White papers, the Net Express knowledge base and other valuable resources are available on-line using your Internet Browser.

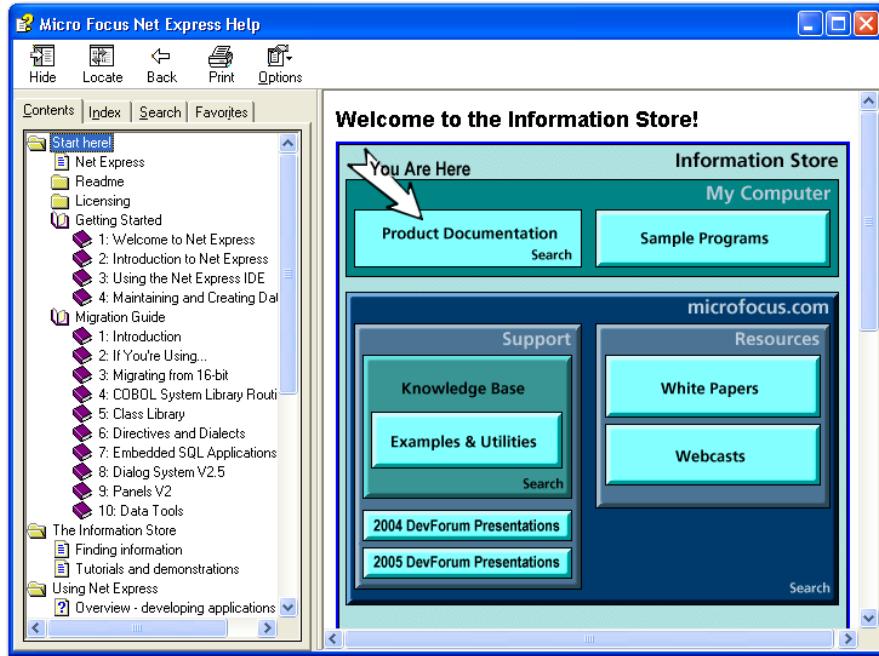
Note that invoking the HELP function within Net Express is specific to the functions being performed in Net Express. For example, if you are running Net Express and want HELP on a Dialog Function pressing HELP from Net Express will not display the associated Dialog System help files. You must request Dialog System Help from within Dialog System and not Net Express.

Help options are described in the table below.



Use this menu option...	To...
Help Topics	Access help using standard Windows help facility including Contents, Index, and Find.
Keyboard	Lists all the current hot keys.
Support	Information on contacting the SupportLine, product updates and SupportLine on the web
About Net Express	Display version information including the update level for each system module.

Via Help Topics you can access getting started overview documentation, review complete documentation for specific functions, and connect on-line to Micro Focus for additional support documentation and Webcasts.



Net Express Interface Help

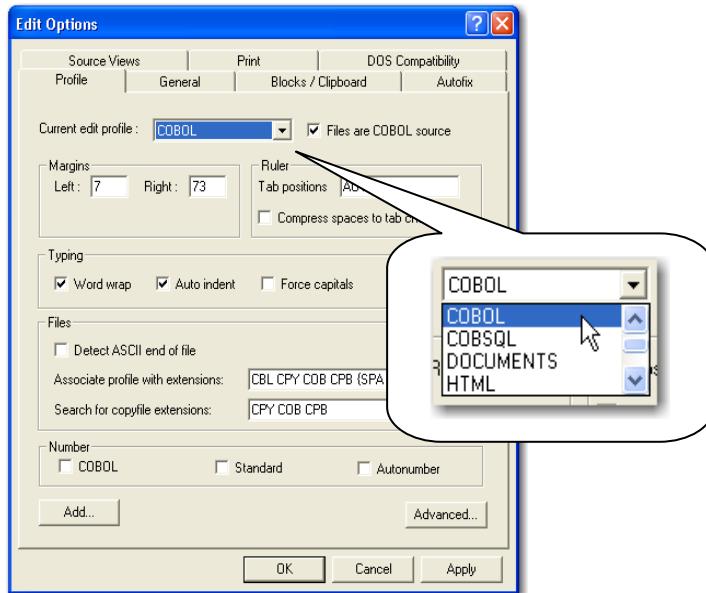
Net Express provides several levels of help directly within the Desktop. Two examples are provided below.

Tool Tips - To obtain a brief description for Toolbar buttons, place the mouse over the object. A tool tip appears at the mouse pointer defining the object the mouse is over. You can also look in the lower left corner of the Status Bar to see the definition.

Button, menu, and window help - To obtain help about a button, menu choice or window within the Desktop click on **Context Help**. Move the mouse pointer to and click on the object of interest. Information about that object is displayed.

Net Express Editor Settings

You can customize the Editor to perform more effectively based on your work preferences and habits. To review the default settings select **Options**, **Edit** to display the Edit Options window.



Editor Options: Profiles

The Profile tab shows the basic configuration for file types loaded into the Editor (e.g., COBOL margins are set to 7 through 73). Editor profiles have been defined for:

- C/C++ source
- COBOL source
- COBSQL source
- Documents
- HTML source
- System

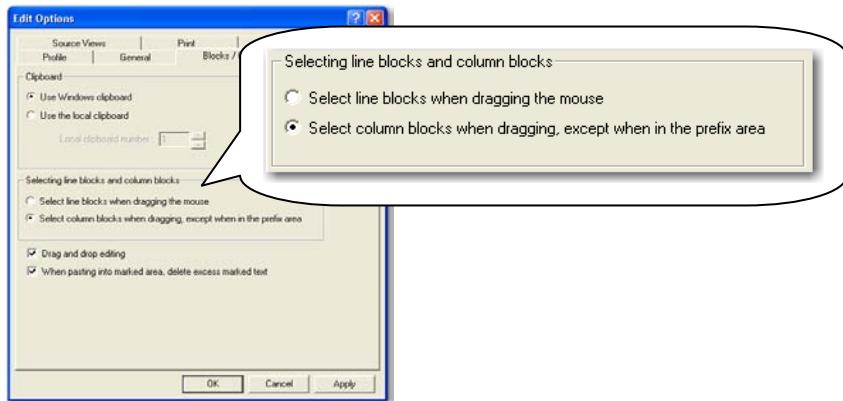
The COBOL Profile

The defaults shown below are active when a COBOL file-type is loaded into the Editor: **Note:** all profiles have similar default settings.

- The rules specified by the COBOL profile (for programs and copybooks) are associated with files loaded into the Editor having file extensions of CBL, CPY, COB, CPB and Spaces. Additional file types can be added.
- Margins are set at column 7 and 73. When editing source, the cursor is positioned on column 8, the beginning of COBOL Margin A.
- Word wrap is turned on. When you type into column 72 the Editor automatically moves to the next line.
- Auto Indent is turned on. When the cursor moves to a new line it automatically positions under the first character of the line above, preserving indentation of your COBOL program source.
- Force Capitals is turned off. This is important for mainframe emulation environments. When turned on the system automatically converts any lowercase characters to uppercase.

Editor Options: Block/Clipboard Tab

The Edit Options Blocks / Clipboard tab determines how the mouse is used to select text. By default the Editor is set to mark line blocks only. COBOL developers typically need to mark column blocks as well. Change this to allow more flexibility when using the COBOL Editor.



To change to column blocks, click **Select column blocks when dragging, except when in prefix area**. Press **OK**.

Editor Options: Remaining Tabs

The remaining Editor Profile tabs are listed and described below.

General—Used to set Editor behavior for margins, deleting and backspacing, and file processing.

Autofix—Fixes spelling errors (e.g., PERFROM→ PERFORM).

Source Views—Used to set high-level performance objectives for source view windows. See **Help**, **Index**, **Option**, Customizing miscellaneous items of behavior for more information.

Print—Used to set printer options.

Command Line based Compatibility—Used when developing command line based applications to change the character set from ANSI to OEM.

Microsoft/Micro Focus Shortcut Key Assignments

Net Express supports both Microsoft and Micro Focus shortcut keys. The following tables list the shortcut functions and additional Micro Focus shortcut support.

Microsoft/Micro Focus Shortcut Key Assignments		
Function	Microsoft Assignment Common Functions	Micro Focus Assignment
Page up/page down	Page Up, Page Down	Same
Word left/word right	Ctrl + Left arrow Ctrl + Right arrow	Same
Start of line	Home	Same
End of line	End	Same
Top of file	Ctrl + Home	Same

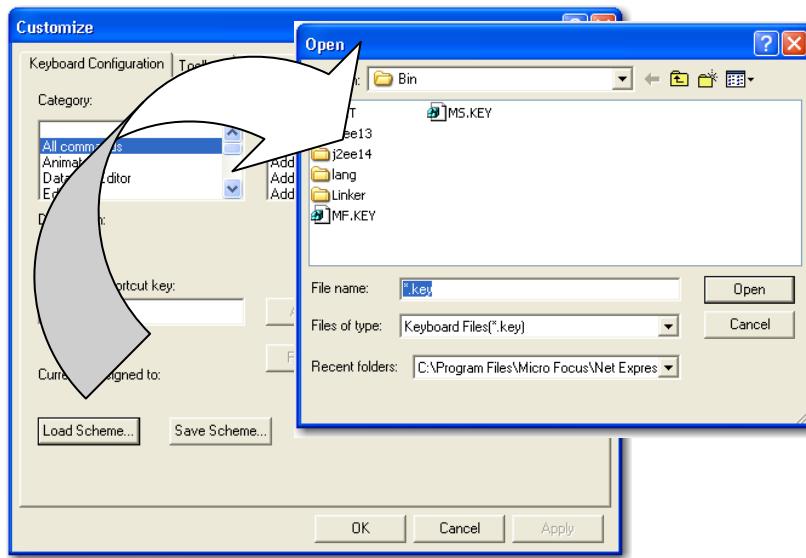
End of file	Ctrl + End	Same
Undo	Ctrl + Z	Alt + Backspace
Redo	Ctrl + Y	Alt + Shift + Backspace
Scroll up	Alt + Up arrow	Ctrl + Num. keypad 8
Scroll down	Alt + Down arrow	Ctrl + Num. keypad 2
Delete word left	Alt + Left arrow	Alt + Num. keypad 4
Delete word right	Alt + Right arrow	Alt + Num. Keypad 6

Micro Focus Specific Line Edit Functions	
Function	Micro Focus Assignment
Insert line	F3
Delete line	F4
Repeat line	F5
Restore deleted line(s)	F6
Split line	Ctrl + F5
Join line	Ctrl + F6

Customizing the Keyboard

To enable the additional functionality available through the Micro Focus shortcuts, follow the steps below.

1. Select **Options, Customize IDE, Keyboard Configuration Tab.**
2. Press **Load Scheme** to display the Open scheme dialog box.
3. From the Open scheme dialog box select MF.KEY to load the Micro Focus shortcut key scheme or select MS.KEY to load the Microsoft shortcut key scheme.



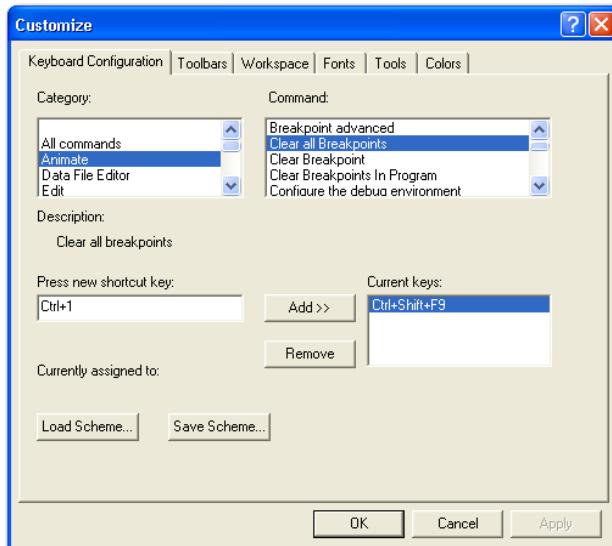
4. Press OPEN. Press OK.

Select your own Hot Key Combinations

1. Select the category of keys to configure from the Category list box.
2. Click on a function in the Command list. Currently assigned hot keys, if assigned, appear in the Current Keys list box. Review the description to ensure you are changing the correct shortcut key for the desired function. If the keys are already assigned to some other function, the current assignment appears under Currently assigned to:
3. Press the new keystrokes you want to assign in the Press New Shortcut Key entry field.
4. Press **Add>>**. **Note:** If the keys are currently assigned to some other function then you will override that assignment.
5. Press **Apply**, press **OK**.

Remove an Assigned Key

1. Click on the key to remove in the list.
2. Press the **Remove** button.
3. Press **Apply**, press **OK**.



Customizing the Net Express Toolbar

The Net Express Toolbar can be tailored for quick selection of Net Express functions. Select **Options, Customize IDE, Toolbars**.

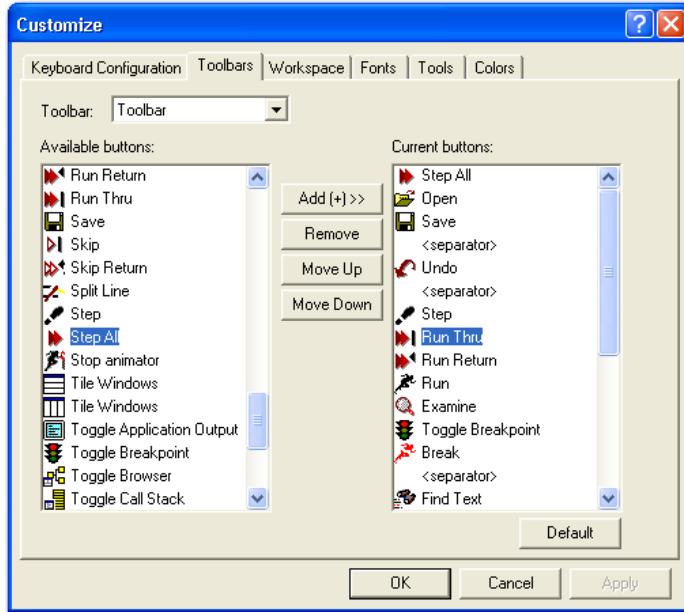


Add a New Button

Follow the steps below to add a new button to the toolbar:

1. In the **Available Buttons** list click on the desired button.
2. Highlight a button in the Current Buttons list. The new button will be inserted prior the highlighted button.

3. Press **Add>>**. Press **Apply**. Press **OK**. The Toolbar has changed.
4. The example below adds the Step All button to the toolbar.



Remove an Existing Button

Select the button name to be removed in the Current Buttons list box; press **Remove**.

Change the Position of a Button

Select the button name to move in the Current Buttons list box. Press **Move Up/Down**.

Reset the Toolbar to the Installed Defaults

Press **Default**. When finished configuring the Toolbar press **OK**

Setting Environment Variables

Environment variables can be used to specify files and/or file locations that are required for execution. Some examples include:

COBCPY - Locates copyfiles

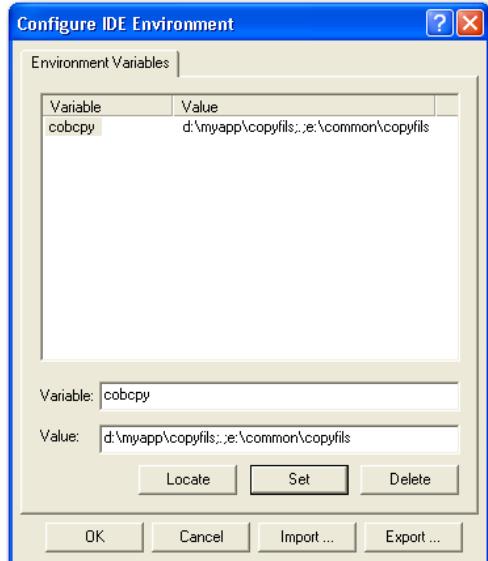
COBDATA - Locates data files

COBDIR - Locates COBOL system software

COBIDY - Locates Animator information (.idy) files.

An option to setting these variables in batch files is to set them in Project Properties.

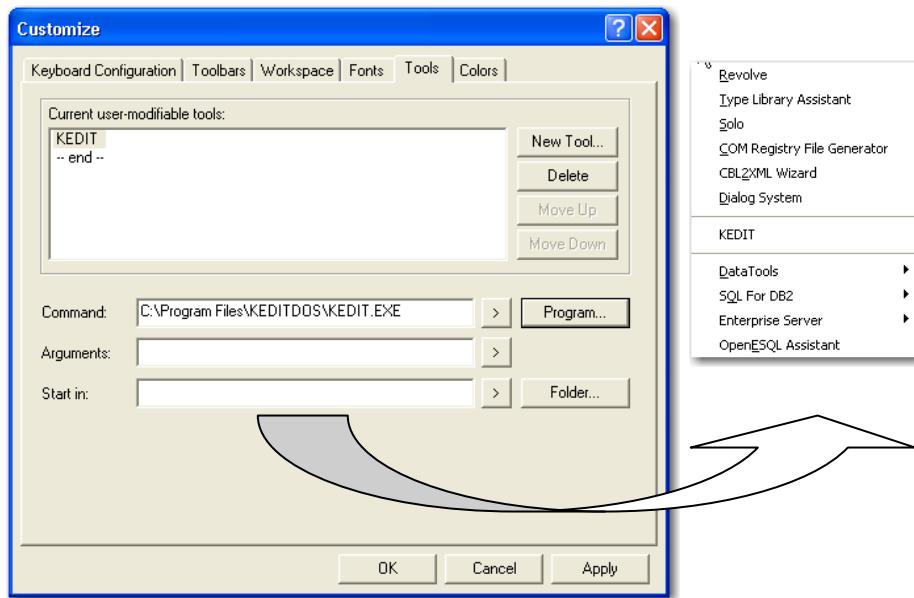
1. Select **Project, Properties**.
2. Click the **IDE** button in the Environment Section.
3. Supply the desired parameters in the **Variable** and **Value** fields.
4. Click **Set** to add the variable. Click **OK** when finished.



Adding External Tools

Any number of additional Windows-based tools and software packages can be added into the Net Express menu system accessed from the Tool Menu. Follow the steps below to add an external tool:

1. Click **Options, Customize IDE**.
2. Select the **Tools** tab.
3. Click **New Tool** and key in the Tool Menu text desired.
4. Use the buttons at the bottom of the tab to set the tool parameters.
5. When finished, click **OK**. The tool added can now be started from the Tools Menu.



Module Summary

The Net Express Integrated Development Environment (IDE) utilizes the latest Window technologies.

Net Express features dockable windows that may be docked or detached and moved around.

You may access Net Express functions using the menu bar, toolbar, shortcut keys, or context menus.

Net Express provides online documentation and help to assist with utilizing the product features.

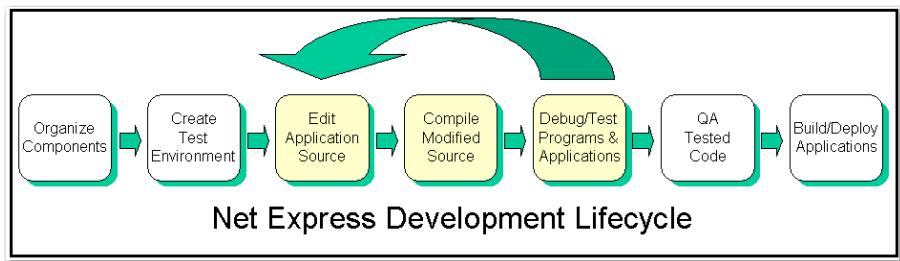
Net Express provides an easy-to-use and configurable interface allowing the user to select their preferred look and feel for the Net Express Desktop.

Module 3

Net Express Quick Start

Overview

The purpose of this module is to give you a high level overview of using the Net Express Tools in the Net Express Development Lifecycle.



Objectives

At the end of this module you will be able to:

- Discuss the tasks Net Express helps you to perform.
- Understand project tasks as they relate to Net Express Development Lifecycle.
- Identify the tools available in Net Express.

What is the Net Express Development Lifecycle?

The Net Express Development Lifecycle is a project-oriented group of related processes.

Task	Description
Organize Components	Gather all base source code and data files required to build the application
Create the Test Environment	Set up the Net Express project and set project parameters
Edit Application Source	Make required modifications to application source
Compile Source	Compile source for testing and debugging
Debug and Test	Clean up code and apply test plans to ensure quality of code
QA Code	Enforce quality standards of application code
Build/Deploy Application	Produce executable code and package application

Using Net Express Throughout the Lifecycle

The following section will walk you through the process of building and deploying a very simple application using Net Express tools. After you open Net Express, follow all the steps very carefully.

Organize Components in a Net Express Project

Gathering necessary files:

1. Click File, New.
2. Select Project, Click OK.
3. With Empty Project selected, supply the following:
Project Name: QSTART
Folder: x:\netxclass\projects\QSTART.

4. Click Create.
5. Click Yes to create a directory/folder.
6. You now have an empty project. Right click in the left pane of the project window and select Add files to project.
7. Select the program x:\netxclass\IDEPROGS\QTAXPROG.cbl.
8. Click Add and be sure to click Yes to copy the file into your folder. The project window now has your program listed.

Create the Test Environment

Identify and setup datafiles for the project:

1. Right click on QTAXPROG.cbl and click Edit.
2. Note that the SELECT statements list two input files:
TBLFLE.DAT
EMPFLE.DAT.
3. Close the Editor.
4. Using Windows Explorer, copy the files from x:\netxclass\Datafile to your project folder.

Edit Application Source

Make required changes to application source:

5. Right click on QTAXPROG.cbl and click Edit.
6. Page down to 0000-MAINLINE.
7. Make the following change:
from PERFORM 040-DISPLAY 25 TIMES.
to PERFORM 040-DISPLAY 2 TIMES.

Compile Source

With required program changes made, compile the program:

8. Click the Checkmark icon in the Net Express toolbar.

9. Verify that the program compiled clean by looking at the messages in the output pane at the bottom of the window. You should see a message that says 'Rebuild complete'

Debug and Test

Test and debug the recently compiled application source.

10. With the Edit session still open, select Animate, Start Animating from the main menu. You will be positioned at the first line of executable code.
11. Click the Step icon (footprint) from the toll menu 2 times. You will see that a new output window opens.
12. Continue to click the step icon until you clear the OPEN INPUT file commands. The output window now displays a line "TAX PROGRAM LOADED'. If you hover your mouse over the file names in the INPUT statements (TBLFILE, EMPFILE) you should see a good 0/0 status.
13. Keep stepping through the entire program until you hit stop run. Occasionally check the output window to see the progress of program execution. At program end, the output window will display 'TAX PROGRAM TERMINATED SUCCESSFULLY'.
14. When you hit stop run, click OK.
15. Click Animate, Stop Animating.
16. Close the Animate/Edit window.

QA Code

QA requires a 10% ratio of comments to code.

17. Click Search, Show COBOL Reports, Program statistics.
18. Verify that this code will pass the QA requirement.
19. Close the Program Statistics window.

Build/Deploy Application

With project testing and QA complete, the application is ready to be packaged. This part is a bit more complicated. Don't worry if it does not

work. This is your first experience building an application. More chances follow in the course later.

20. Using the dropdown in the left pane of the project window, select Generic Release Build.
21. Right-click on QTAXPROG.cbl; select Package Selected Files, Executable File.
22. In the Define New Executable.File window, it will read QTAXPROG.exe.
23. Toggle off Include this package file in all build types (i.e., don't add it to the Debug Build). Press Create.
24. Right-click on QTAXPROG.cbl; select Package Selected Files, Dynamic Link Library. Again, toggle off Include this package file in all build types. Press Create.
25. Set the link parameters. Right-click QTAXPROG.exe; select Build Settings... On the Link tab, select Static, character, verbose linker output. (Don't close the window yet).
26. Click on QTAXPROG.dll in the Project Build Settings Window. On the Link tab, select Static, verbose linker output. Press Close. The application will be shipped as statically linked, with QTAXPROG.exe as the driver, and a separate QTAXPROG.dll.
27. Select Project, Rebuild All.
28. Using Windows Explorer, open the Release folder in your project folder. Verify the creation of QTAXPROG.exe and QTAXPROG.dll files in this folder.
29. Double click on QTAXPROG.exe. If all went well, the program runs. Watch the output screen closely as it will close when the program finishes.

Summary

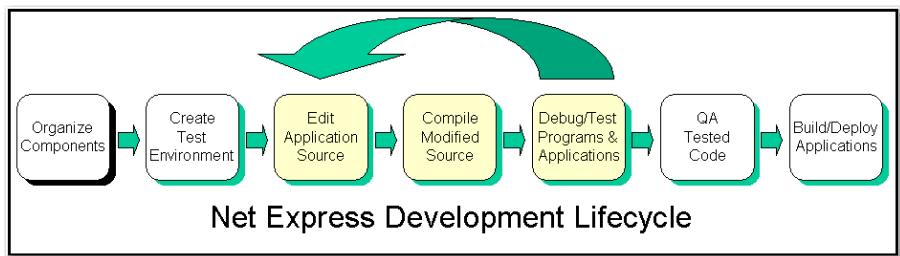
In this module you have had an overview of using the Net Express tools to set up, test and deploy an application. While this was a very simple set of steps, you have had exposure to all the major phases in the Net Express Application Development Lifecycle. This will serve as a foundation to the learning experiences in the modules ahead that explore development in much more depth.

Module 4

Working with Net Express Projects

Overview

Net Express projects are used to organize source components related to a system, an application, or a program. Once you have organized your components in a Net Express project, you are ready to move on to developing and testing.



Objectives

After completing this module, you will be able to:

- Define the different types of Net Express projects.
- Follow the required steps to create projects (new projects with no existing base components or projects based upon an existing base).
- Identify and understand the use of the panes in the project window.
- Maintain existing projects (add and remove files).

Definition and Use of Projects

Net Express provides a project-based framework within which you can organize work for ease-of-use and access. A *project* is a user-defined collection of source components and Build (compile) commands used to create a system, an application or program. There are two types of Builds: a debug Build for debugging, and a Release Build with production links.

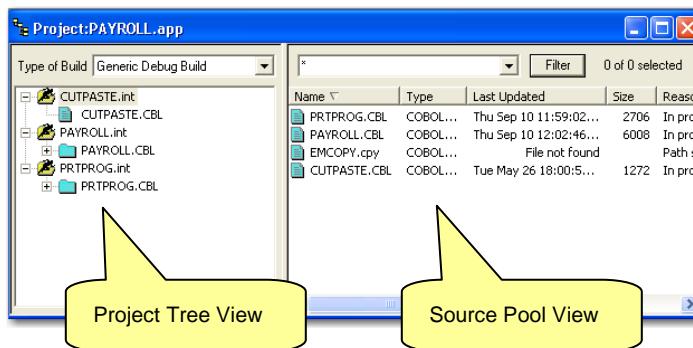
Benefits of a Net Express Project

A project provides an intuitive interface showing the relationships between components of a system, an application or a single program.

Project Tree View—A tree view of executable project components used when performing a Build. Separate Project Tree Views are provided for the Debug and Release versions of the system, application or program. This view shows the relationship of each component to the other components in the project within Build type.

Source Pool View—Shows source components, which may or may not be represented in the Project Tree View. Components not appropriate for the Project Tree View (i.e., not executable such as help files, icon files, bitmap resource files, etc.) are included in the Source Pool View to allow easy access and packaging of the distributable system, application or program.

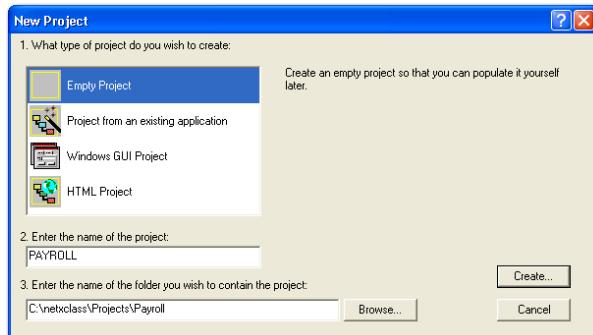
From either view, project components, regardless of the type, can be accessed easily using the mouse to perform appropriate tasks based on the component type selected. For example, COBOL source programs can be edited and compiled. Executables can be run or Animated.



Builds can be performed on individual components, a group of components or all project components. Incremental project Builds save time by compiling and linking only those components that have changed since the last Build. You can easily switch between available Build types.

Net Express supports multiuser environments where different developers can work on separate modules within the same application. This maintains consistency in the Build. **Note:** Each user can have their own project with shared components or different components from the same system or application, but Net Express projects are single-user and not multi-user.

Project Types



A project type is selected when creating a new Net Express project. There are four project types:

Empty Project—Create the framework for a project. Once a project has been created, source components can be created or, if existing already, added at a later time.

Project from an Existing Application—Create a project that invokes a wizard prompting you to add source components to the project at the time it is created. Additional source components can be added at anytime.

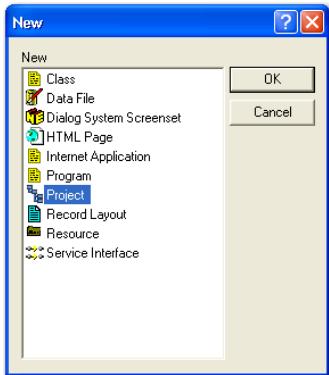
Windows GUI Project—Create a Windows GUI project using Dialog System.

HTML Project—Create an Internet/Intranet project.

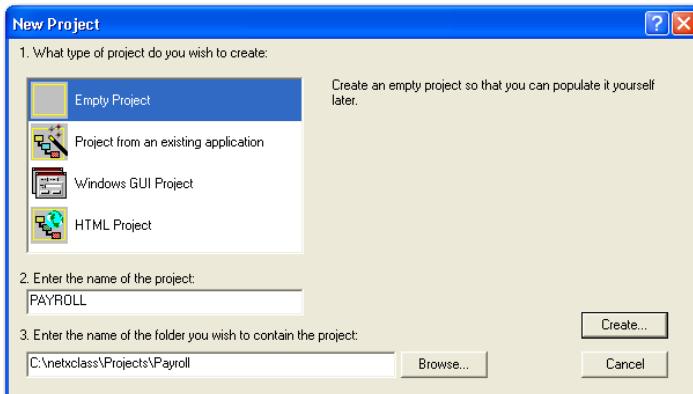
Steps for Creating An Empty Project

To create a project for COBOL applications using the Project Wizard follow these steps.

1. Select **File, New**.
2. On the New Project window, select **Project**.



3. Click **OK**. The New Project wizard is invoked and the opening window displays.



4. Choose **Empty Project** to create the framework for your COBOL application project.

5. Enter **a name for your project** in the Enter Name field. The Project Wizard creates a project file (filename .APP) in the specified project folder.

2. Enter the name of the project:

PAYROLL

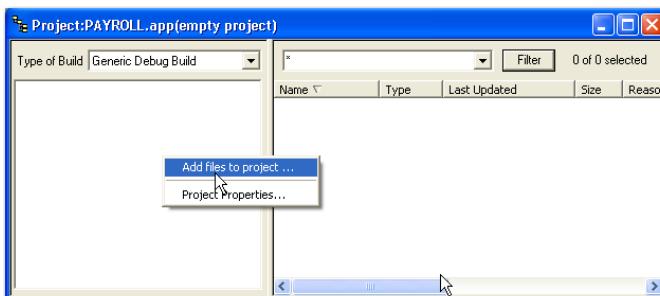
6. Enter **the project folder** where the new project will be stored or use the Browse button to select the appropriate folder.

3. Enter the name of the folder you wish to contain the project:

C:\netxclass\Projects\Payroll

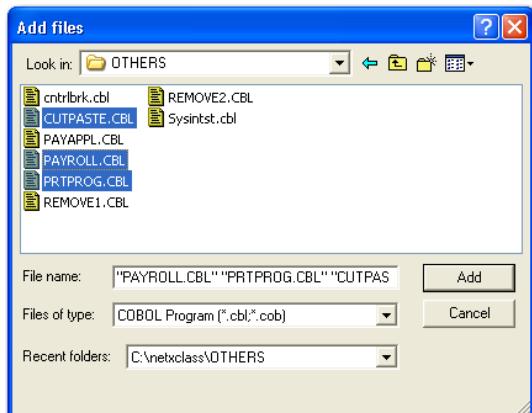
Browse...

7. Click **Create**. You will not be prompted to add components to this project. Components belonging to this project may be newly created within the project or if existing, added at a later time.
8. Respond to any additional window prompts as required. When complete, the project window displays.
9. Right-click in the **left-hand view** (Project Tree View) and select **Add Files to Project** to add source files and resolve their paths.

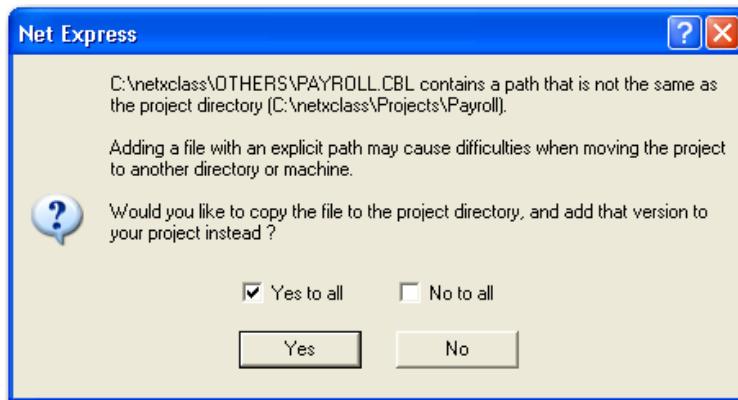


10. In the Add Files window, navigate to **the folder that contains your files** (e.g., \netxclass\ideprogs). To aid navigation, there is a drop-down list of recently accessed directories.

11. Select the desired file; press **Add**. You can choose multiple files using standard Windows methods (Ctrl key for individual files or the Shift key for contiguous files).



12. When adding files to projects, Net Express advises you if the selected components are not resident in the current Project Folder. A selection screen is displayed to specify the way you want to access source code.



If you select **YES** or toggle **YES to all** and select **YES**, Net Express copies the components into the current project folder. When you edit those components in Net Express you will be editing a copy. The original version remains intact.

If you select **No** or toggle **No to all** and select **No**, Net Express creates a pointer to the original source component(s) located in the originally selected folder. All changes in Net Express are made to the original component in the original folder.

Note: In class exercises you will choose Yes because that choice means you are modifying copies only of the original source files. Then, if you need to refresh a project you can easily do so. The first program added to the project is set to be the Main Program (the first program to load) when testing. You can modify this setting later.

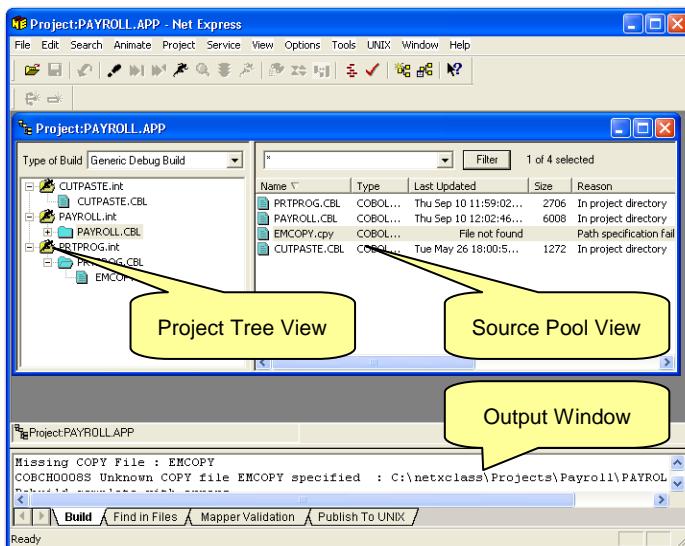
The Project Window

The Net Express Project Window is designed to allow users to quickly assess the components contained within a project and see their relationship to other components within the project. The project window is split into two panes:

- **Project Tree View** (left side) listing the executables and their components.
- **Source Pool View** (right side) listing all of the source components within the project.

Below the Project window, the Output window provides the following tabs:

- **Build** - messages displayed when adding or rebuilding project files (start, completion, errors).
- **Find in Files** - display area for results of a Search.
- **Mapper Validation** – error messages from the Interface Mapper.
- **Publish to UNIX** – informational messages when publishing to UNIX.
- **View, Output Window** – toggles between showing and hiding the Output window.

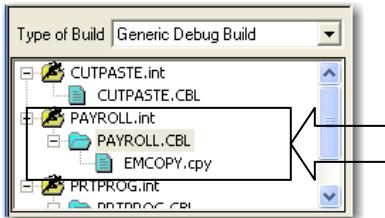


The Build Tab

Each component, as it is added to the project, is scanned for dependent copybooks. For example, in the PAYROLL project above, PAYROLL.CBL references a copybook, EMCOPY.cpy, which was not automatically added to the project when it was created and cannot be found by the project. You would receive a message File not found, or Path Specification Failure for the copybook EMCOPY in the Source Pool View. More specifics details are provided in the Output Window. In the example above, the paths for the missing elements must be resolved before a successful project Build for these components can be completed.

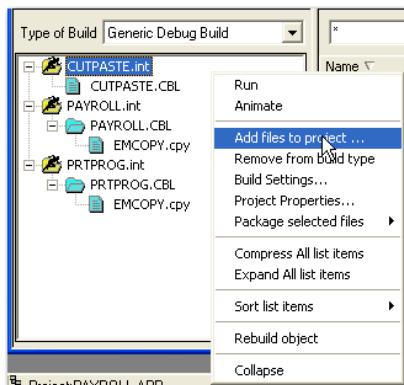
Project Tree View

Project components are shown using a tree structure. Using the Project Tree View you can easily see dependencies within a project.



For example, the EMCOPY.cpy copybook has been loaded into the project and PAYROLL.CBL has been compiled as a Generic Debug Build. After a successful, compile, the Build creates an executable, PAYROLL.INT, for testing in the Animator. The extension .int indicates this is an intermediate (interpreted) compile. Each executable file is shown in the Project Tree View whether it exists or will be created in the future.

The number of trees displayed in the Project Tree View represents the number of executable files to be created when the project is built. Right-click on the Project Tree View to add or delete files, change executable types, open files for editing or debugging, set options for compiling and linking and building the project.



Source Pool View

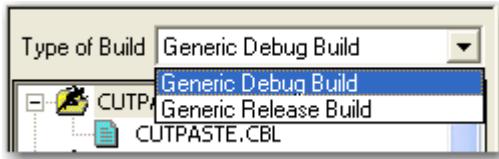
The Source Pool View displays all source files included in the project whether or not the current Build type requires them. You might include files in the Source Pool View that will not appear in the Project Tree View such as help, documentation or source files that you copy from.

Name	Type	Last Updated	Size	Reason
PRTPROG.CBL	COBOL...	Thu Sep 10 11:59:02...	2706	In project directory
PAYROLL.CBL	COBOL...	Thu Sep 10 12:02:46...	6008	In project directory
EMCOPY.cpy	COBOL...	File not found		Path specification fail
CUTPASTE.CBL	COBOL...	Tue May 26 18:00:5...	1272	In project directory

The Source Pool View provides file information for all the components in the project: Name, Type, Last Updated, Size, Reason (why it is there or if it is missing), Location (path). You can also set a filter using file wildcards (*) to tailor the list being viewed.

Project Build Types

The term Build refers to the process in which components are compiled and executables are created. Net Express has two pre-defined Build types.



Generic Debug Build—Used to create executables for the Animator to interactively debug COBOL programs at the source code level. When Type of Build is set to Generic Debug Build, source files in the project are compiled to intermediate code (.INT), by default. When animation of completed modules is no longer necessary, the default can be changed or to generated (optimized) code (.GNT) or object (.OBJ) code. Resulting executables are placed in a pre-defined folder within your project folder. By default a folder named \DEBUG is created under the project folder for these executables. You can change the name of the folder.

Generic Release Build—Creates stand-alone executables for distribution to the target environment. When Type of Build is set to Generic Release Build, source files in the project are compiled to industry standard object code. Based on the defined project tree view, which shows the relationship of each executable component to another, the .OBJ files are linked into system executable (.EXE and .DLL) files. The resulting executables are placed in a pre-defined folder of your project folder. By default a folder named \RELEASE is automatically created under your project folder. You can change the name of the folder.

Maintaining Projects

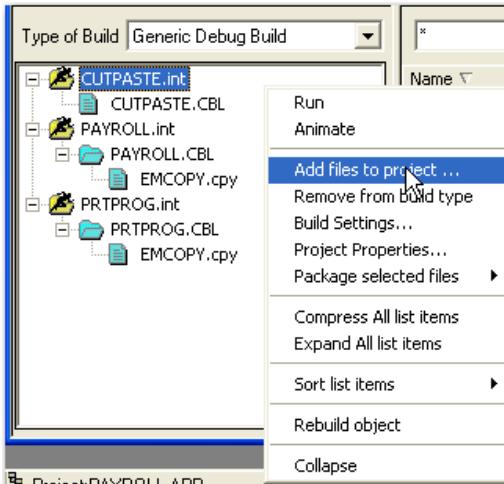
Once a project has been created it may be necessary to add components that were identified during the project-create phase as being required but not found (e.g., EMCOPY.CPY in the Exercise). Over time you will also want to add new components and/or remove obsolete components from the existing project. New components can be added and obsolete components removed whenever necessary.

Adding Components to Existing Projects

Before adding a component you need to decide how the component will be added. You can add components as an executable to the current Build type or to all Build types or add components to the Source Pool View only.

Adding components to the Project Tree View adds the component name to both the Project Tree View and Source Pool View and creates an executable for the component when a project Build is performed.

To add executable components, follow these steps.



1. Within the Project Tree View, right-click the **Project Tree View** to display the Project Tree View context menu.
2. Select **Add files to project**. Or select **Project, Add files to project**. Once you select Add files to project, the Open dialog box displays.

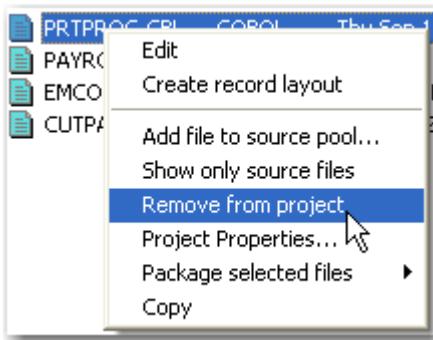
3. Use the same procedures used when creating the initial project to locate and select the component(s) to be added to the Project Tree View.

Removing Components from Existing Projects

Over time obsolete components must be removed from the project. Before removing a component decide exactly what is to be removed and from where (remove the component from the entire project or remove the component from the current Build type only). The method selected to remove components determines if the components are removed from the entire project (all build types and the source pool), or from the current build type only. Removing components from a project does not delete the corresponding source file on the disk. If required, use Windows Explorer to physically remove these files from the disk.

Follow these steps to remove components from the project.

1. Right-click on **the source component** in the Source Pool View. The Source Pool View context menu displays.
2. Select **Remove from project**.



3. Confirm the removal of the selected item from the project.
4. Press **Delete**. If more than one component is selected and you don't want to confirm each component's removal, uncheck the **Show this dialog for each item in the list**. All references to the component(s) in

the Project Tree View and Source Pool View for all Build types are removed.

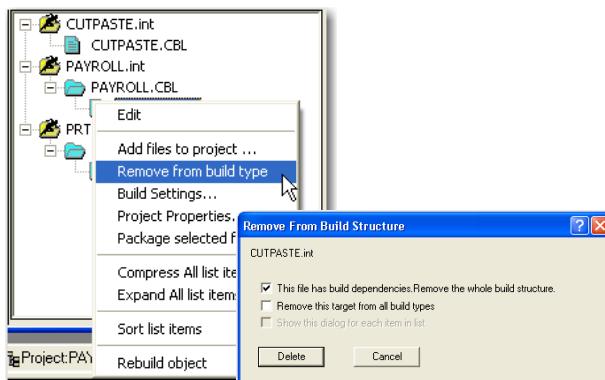
When using this method it is possible to remove all references to the selected file(s) from the project and all build types, however, it may still be necessary to remove additional related components individually. For example, removing a .CBL source file will remove all references to the file in both the Project Tree and Source Pool views. However, the executable associated with that component in the Project Tree View is not removed and must be removed separately.

Removing Components from the Current Build Type Only

Removing components from the current Build type only removes the component from the Project Tree View for the selected Build type and leaves it in the Source Pool View. The user can select to leave or remove the component from all other Build types.

Follow these steps to remove components from the current Build type.

1. In the Project Tree View, select the components to be removed using standard Windows methods for selection.
2. Right-click on **the highlighted source component(s)** to remove. The Project Tree View context menu displays.
3. Select **Remove from Build type**.

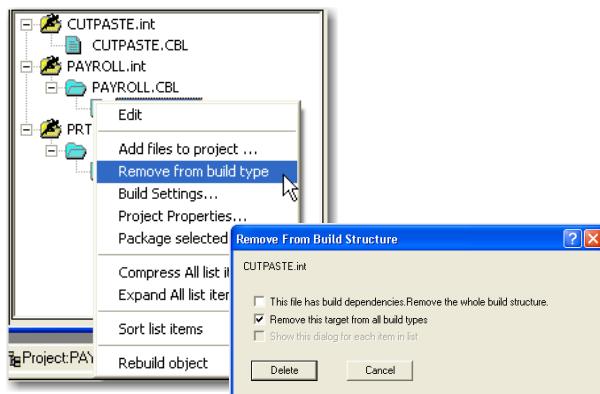


4. To confirm the removal of the selected item from current Build type only, press **Delete**.

Removing Selected Components from All Build Types

To remove selected components from all Build types follow these steps.

1. In the Project Tree View, select the components to be removed using standard Windows methods for selection.
2. Right-click on the highlighted source component(s) to remove. The Project Tree View context menu displays.
3. Select **Remove from Build type**.
4. Check **Remove this target from all Build types**.



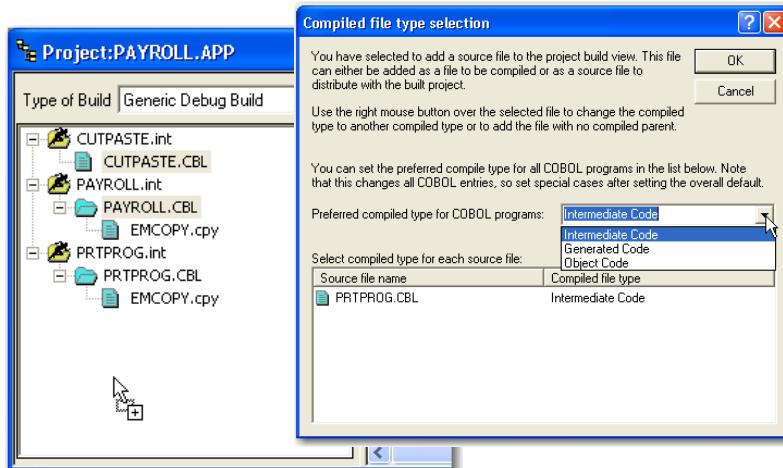
5. Press **Delete**. All references to the component(s) in the Project Tree View for the selected Build type are removed. If you selected to remove from all Build types it has been removed from the Project Tree View for all Project Build types. The component remains in the Source Pool View and can be re-added at a later time.

Add Components from the Source Pool View to the Project Tree View

When creating a new program it is helpful to add it to the Source Pool View for the project so that you can easily access it for editing purposes. However, you would not want it to appear in the Project Tree View until it is ready to compile. Once coding is complete you will want to add it to the Project Tree View so that it can be compiled and debugged.

Follow these steps to add components from the Source Pool View to the Project Tree View.

1. In the Source Pool View, select the component(s) to be added to the Project Tree View using standard Windows methods for selection.
2. In the Source Pool View, hold down the left button on a selected object to be added to the **Project Tree View**.
3. Drag the selected object(s) slowly towards an area of the Project Tree View that does not contain an entry already. It may be necessary to collapse the tree view so white space is displayed on which to drop the dragged objects. The mouse shape changes to appear as multiple pages (plus sign) when the pointer is placed in an acceptable location.

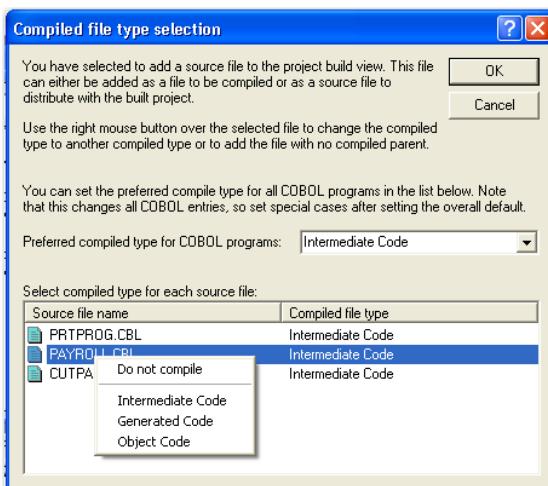


4. Release the left button.

5. When adding source files from the Source Pool View to the Project Tree View, specify the executable Build type to be added to the Project Tree View. To add .INT as the executable type press **OK**. By default, Net Express assumes you want to add an .INT for compiling and debugging.

Changing the Executable Type When Adding Components

1. Right-click on the component to change. The Compile Type Context Menu displays.
2. Select the desired executable type to add. **Note:** Generated Code refers to compile optimization – the code runs faster. It is discussed later in the course.



Opening an Existing Project

Project details are maintained in a file that is located in the project folder named at project creation. The name of this file is *ProjectName.APP*. Files with the .APP extension are project files. There are two ways to open existing projects. The method used depends on whether you have accessed the project recently.

Here are the steps to open a project that you have worked with recently.

1. Select **Files, Recent Projects**.

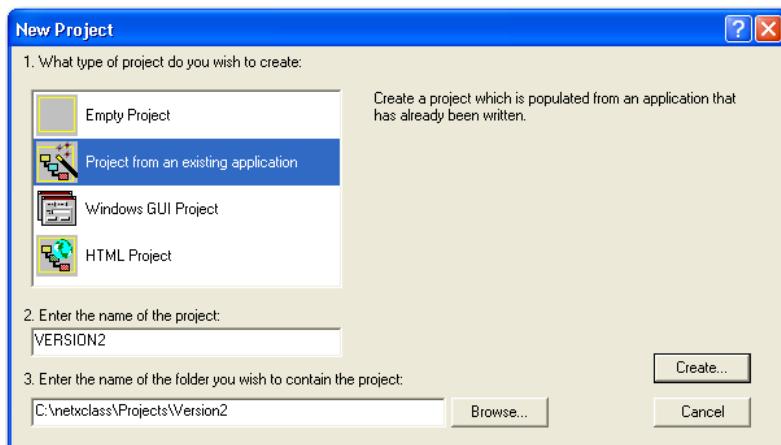
2. Select **the project** you want to open. The Recent Projects list contains the names of the last four projects (default) you have accessed. To increase the number of projects displayed on the Recent Projects pulldown select **Options > Customize > Workspace** window and change the default setting.

If the project you wish to open does not appear on the Recent Projects list, open it using these steps.

1. Select **File, Open**.
2. Using standard Windows methods locate **the drive and path of your project folder**, and search for **project files (*.app)**.
3. Select **the project file (*.app)** from the list of files in the project folder.
4. Click **Open**.

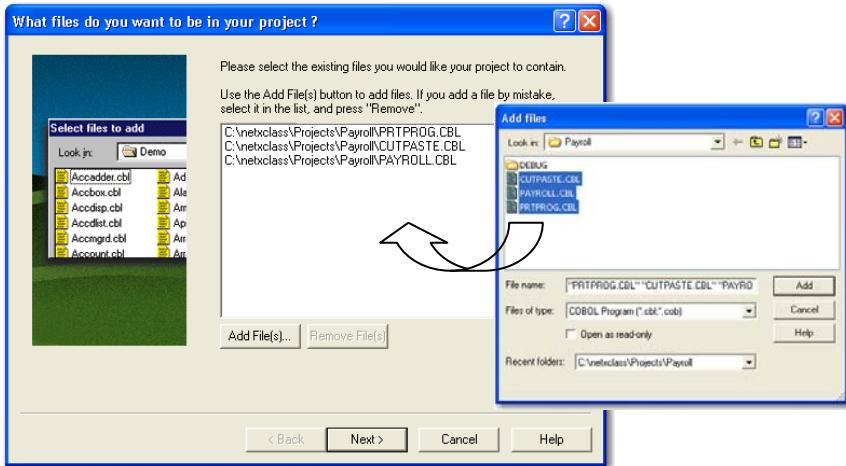
Creating a Project from an Existing Application

Selecting **Project from an existing application** in the New Project window invokes the New Project Wizard. The New Project Wizard creates a project file (filename.APP) in the specified project folder, prompts you to add existing components to the project and to identify the Main Program for this project.



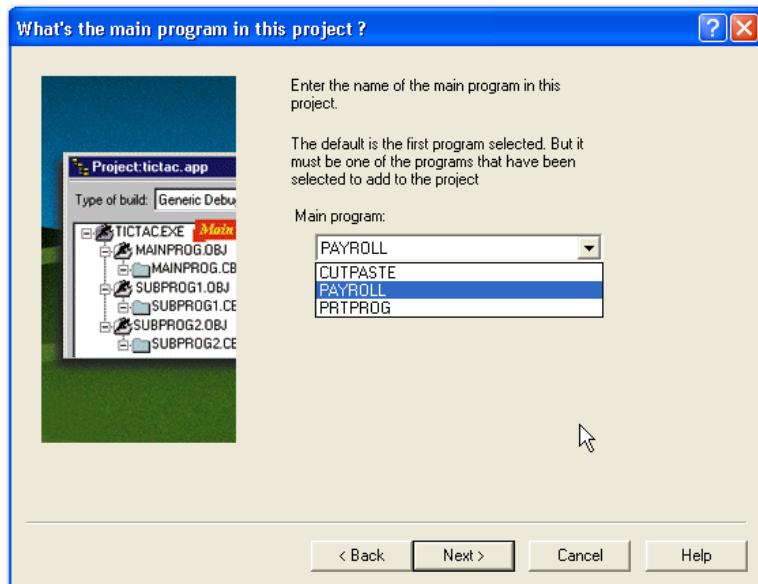
Add Files to Project

Net Express creates a folder for your project (if required) and displays the **What files do you want to be in your project?** window. Use this window to locate existing components you want to include in this project.



1. Press **Add Files** to open the Add Files Dialog Box.
2. Select the **drive and path** where the components exist you wish to add to this project.
3. Change the **Files of Type** as necessary to correspond to the file types you wish to add to the project (e.g., .CPY, .CBL).
4. Using standard Windows methods, select one or more components within the selected folder to add to your project. Holding the **Shift** key down allows you to select a range by clicking on the first and last item in the range. Holding the **Ctrl** key down allows you to select interspersed components by clicking on each one.

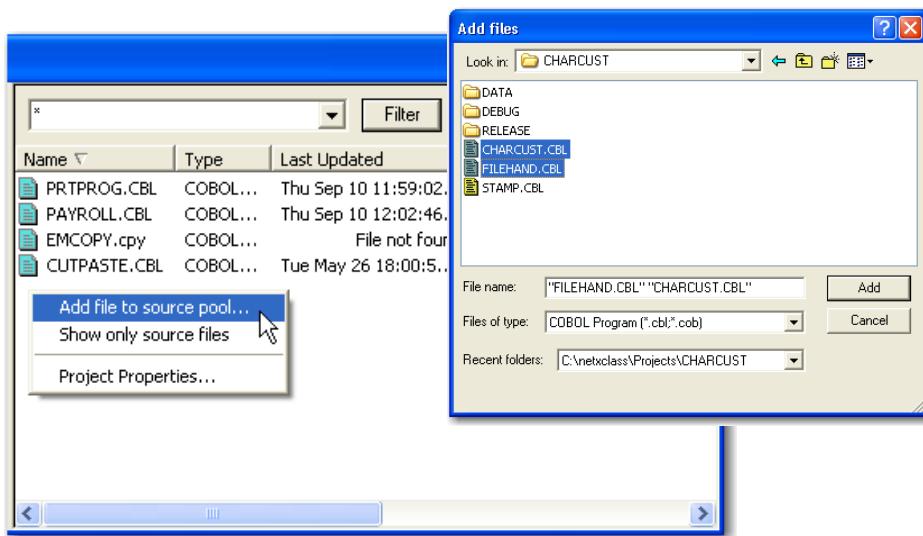
5. Click **Add**. The file list on the Add Files window is updated to reflect the components added to this project. You may continue to add component types and/or components located in other folders by repeating this same process.
6. When you have identified all of the components you wish to add to this project click **Next**.
7. On the **What's the main program in this project?** dialog box, select the program from the list that is the main program for this project (e.g., the Main program might be the driver that calls the other programs in the application).



8. Press **Next**.
9. Press **Finish**.

Adding Components to the Source Pool View

Over the life of a Project it will be necessary to add components to the project. You may also want to add documentation files. Adding components to the Source Pool View adds components to the Source Pool View only. When a project Build is performed an executable will not be created for any component that is not in the Project Tree View.



1. Right-click In the Source Pool View. The Source Pool View context menu displays.
2. Select **Add files to source pool** The Add dialog box displays.
3. Select **Files of type**. All types OK e.g., COBOL copybook (*.CPY)
4. Select **the folder where the files are located**.
5. Select **files to add**. Use the **Shift** and **Ctrl** keys to select ranges and multiples.
6. Press **Add**.

Module Summary

Net Express projects organize source components related to a system, an application, or a program.

Net Express provides various types of projects. Choose **Empty Project** to create a project file only. Use **Project From Existing Application** to select existing components at the time the project is created, **HTML Project** to start a new **HTML** based project, and **Windows GUI** to create a new Dialog System project.

Add and remove files within the project as necessary.

The Project Window provides a project tree view and a source pool view.

Two types of builds are supported by default, Debug Build or a Release Build.

4.1 Creating Projects from an Existing Application

In this Exercise you create the Calltest project. Use the New Project Wizard to create the CALLTEST Application Project using the information below.

1. Select File, New, Project. Project type: Project From An Existing Application and supply the following project parameters:

Project name: **CALLTEST**

Folder name: Supply the following path for the new folder

x:\NETXCLASS\PROJECTS\CALLTEST

Note: x:\ is the drive where class files are located. Please ask your instructor.

You will receive a message that the location does not exist. Press **YES** to create the folder.

2. Add the following files to the CALLTEST project from the folder x:\NETXCLASS\DEPROGS: **CALLTEST.CBL, DATETEST.CBL**.
3. Press **Next**.
4. Select **CALLTEST.CBL** as the Main Program for the project.
5. Select **Finish** to complete the process and create the project. Respond **Yes** to the copy message.
6. After verifying that the programs have been added to the Project Tree View and the Source Pool View, close **the project window**.

4.2 Creating an Empty Project

In this exercise you create the Payroll project.

1. Use the New Project Wizard to create the PAYROLL application project using the information below.

Project type: **Empty Project**

Project name: **Payroll**

Folder name: Supply the following path for the new folder
x:\NETXCLASS\PROJECTS\PAYROLL

2. Right click in the Project Tree View and add the following files to the Payroll project.

From Folder	Files
x:\NETXCLASS\OTHERS	PAYROLL.CBL PRTPROG.CBL CUTPASTE.CBL
x:\NETXCLASS\IDEPROGS	DRIVER .CBL DRSUB1.CBL DRSUB2.CBL

3. After verifying that the programs have been added to the Project Tree View and the Source Pool View, close **the project window**.

4.3 Create and Maintain a Project from an Existing Project

1. Create a new project, VERSION2, using the **From An Existing Project** type. Create it in the folder x:\NETXCLASS\PROJECTS\VERSION2. Press **Create**.
2. On the **What files do you want to be in your project?** window, press **Add Files**.
3. Look in x:\NETXCLASS\PROJECTS\CALLTEST. Select **CALLTEST.CBL, DATETEST.CBL**. Press **Add**, then press **Next**.
4. Choose **Calltest** as the main program. Press **Next**, then **Finish**.
5. Use the **Add Files To Project** method to add files to the Version2 project. From x:\NETXCLASS\OTHERS, add **REMOVE1.CBL** and **REMOVE2.CBL** to the VERSION2 project. Copy the files to the project folder.
6. From x:\NETXCLASS\IDEPROGS, add **A.CBL, AERROR.CBL, TAXPROG.CBL, and VSAMDEMO.CBL**. Copy the files to the project folder.
7. Add **CHARCUST.CBL** from x:\NETXCLASS\PROJECTS\CHARCUST. Copy the file to the project folder.
8. Add program documentation file **A.TXT**. Add **files of type Plain Text File (*.TXT)** located in the x:\NETXCLASS\IDEPROGS folder to your VERSION2 project's Source Pool View. **Hint:** In the Files of type field, press the letter P. When prompted, add as a plain text file.
9. Using the Source Pool View, remove **REMOVE1.CBL** from the Project. Is it also removed from the Project Tree View?
10. Look through the Source Pool to determine the status of dependent copybook files. Are any copybooks marked **File Not Found**?
11. Right-click on **EMCOPY.CPY** in the Source Pool View. Select **Add file to Source Pool**. EMCOPY.CPY is in x:\NETXCLASS\OTHERS; files of type is COBOL copybook (*.cpy). Add it to the Source Pool View.

12. Select **A.CBL** in the Project Tree View. From the Project menu bar choice select **Update Dependencies**. What is the status of the EMCOPY.CPY in the Source Pool View?

4.4 Resolving COPYBOOK locations using an Environment Variable (Optional)

The COBCPY environment variable specifies the path to a copybook folder. The environment variables are set in **Project, Properties** under the Environment: IDE button. You can set up a concatenated search path. **Note:** **Project, Properties** are dealt with later in the course. If closed, reopen the VERSION2 project.

1. Rebuild **Charcust.int**. It produces a list of copybook errors: Dependent file not found.
2. Right-click **Charcust.int**. From the context menu, select **Project Properties** or select **Project Properties** from the Project pulldown of the menubar.
3. Press the **Environment IDE** button.
4. For Variable enter **COBCPY**.
5. For Value enter the folder (or concatenated list of directories) where the copybooks are located. In this case, the folder is **\NETXCLASS\PROJECTS\CHARCUST**.
6. Press **Set**; press **OK** to close the Configure IDE Environment window.
7. Press **OK** to close the Project Properties window.
8. Rebuild **Charcust.int**. The copybook missing errors are resolved.

4.5 Open an Existing Project

In this exercise, you open an existing project that is not included on the Most Recent Projects list.

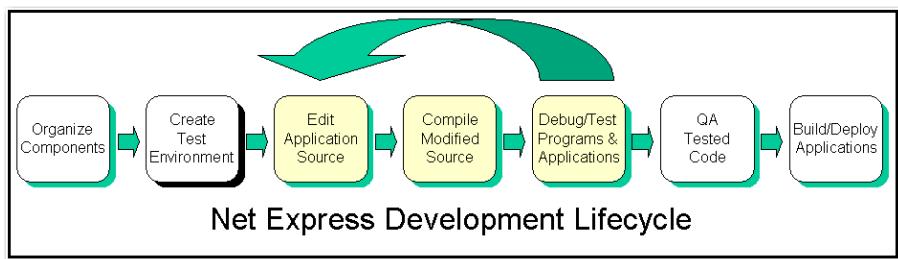
1. Select **File, Open** to open the CHARCUST application project: **Hint:** application projects have the extension .APP. The project is located in `x:\NETXCLASS\PROJECTS\CHARCUST`.

Module 5

Create the Test Environment

Overview

Creating the test environment involves preparing and defining data files that are external to your programs. You have a number of choices depending on the amount of flexibility you want to provide for testing and eventual implementation. This module is focused on defining files that already exist. Data tools, which are specifically designed for building, modifying and maintaining data files, are covered in a later module.



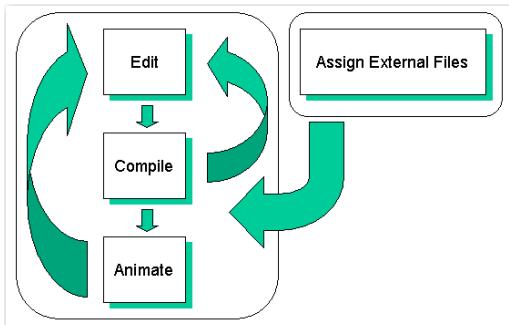
Objectives

At the end of this module you will be able to:

- Describe differences between the three different methods of assigning input and output data files to COBOL programs.
- Choose appropriate methods for assigning input and output data files to COBOL programs.
- Code environment variables to assign files.

Setting Up External Files

Your COBOL input and output data files must be assigned to physical PC filenames before attempting I/O against them within the Animator using environment variables or internal assignment statements.



Data File Assignment on the PC

```

SELECT OLD-PARTS-MF-IN   ASSIGN TO OLDPARTS.
SELECT CUR-ACT-FILE-IN  ASSIGN TO CURTRANS.
SELECT PROD-LINE-FILE-IN ASSIGN TO PRODFILE.
SELECT SORT-WORK-FILE   ASSIGN TO SORTFILE.
SELECT NEW-PARTS-MF-OUT  ASSIGN TO NEWPARTS.
SELECT REPORT-FILE-OUT  ASSIGN TO REPORTER.
  
```



Net Express resolves the physical file assignment of a typical COBOL SELECT clause such as SELECT INFILE ASSIGN TO MYMAST via the following three methods:

- Direct allocation through SELECT filename (hard-coding)
- Dynamic allocation within Working-Storage
- External mapping through environment variables

Both direct allocation and dynamic allocation assign the filenames within the program source (internally). This is the default if you compile your program with any of the PC COBOL dialects (e.g., ANSI'85 or Micro Focus default). External mapping assigns the filenames outside the program source.

Compiler Directives

The table below lists the directives used when referencing external data files.

Directive	Application
ASSIGN(DYNAMIC)	Direct and Dynamic file allocation
ASSIGN(INTERNAL)	External mapping through environment variables

Direct Allocation via the SELECT Statement

The physical filename and location are assigned in quotes as part of the Select statement. For example:

```
SELECT MYMAST ASSIGN TO "x:\DATA\ACCTMAST.DAT".
```

Dynamic Allocation within WORKING-STORAGE

The physical filename and location are assigned dynamically as a WORKING-STORAGE item.

```
SELECT MYMAST ASSIGN TO MASTER.  
WORKING-STORAGE SECTION.  
01 MASTER PIC X(21) VALUE "x:\DATA\ACCTMAST.DAT".
```

This provides more flexibility in that files can be specified in the picture clause and, if required, swapped in the source logic. For example:

```
IF USER = 'A' MOVE 'x:\DATA\CCTNEW.DAT' TO MASTER.  
OPEN INPUT MYMAST.
```

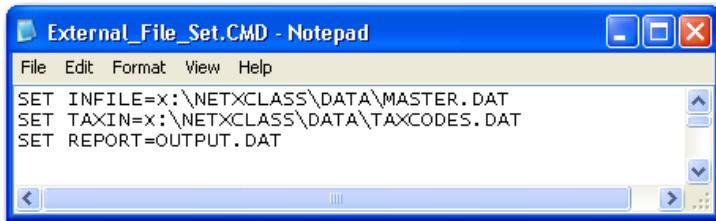
Assigning Files using Environment Variables

The logical filename from each SELECT clause can be set as an environment variable. Files can be set at a command prompt under Windows, batched in a command file or set within the Net Express IDE depending on the needs of your project.

```
SELECT CUSTFILE ASSIGN TO INFILE  
SELECT TAXFILE ASSIGN TO TAXIN  
SELECT OUTFILE ASSIGN TO REPORT
```

Assign External Files via Command Prompt or Command File

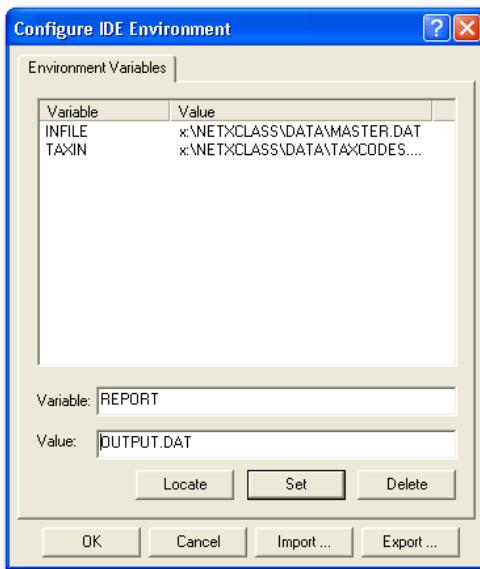
For the above SELECT statements, the environment variables can set as follows under Windows. Note that paths must be supplied if the data file is not in the project default directory.



```
External_File_Set.CMD - Notepad
File Edit Format View Help
SET INFILE=x:\NETXCLASS\DATA\MASTER.DAT
SET TAXIN=x:\NETXCLASS\DATA\TAXCODES.DAT
SET REPORT=OUTPUT.DAT
```

Assign External Files via Net Express IDE

Follow the steps below to set the external file variables within the project.



1. Right-click in **the Project window** and then select **Project Properties**.
2. Press the **IDE button**.
3. Enter the **ASSIGN name** in the VARIABLE field.

4. Add the **physical file name**, which may include the path, in the VALUE field.
5. Press **Set**.
6. Press **OK** when all the ASSIGN names are resolved to close the properties window.
7. Select **Project, Rebuild All** (since you have changed the directives).

Output Report Files

When creating an output report file, you can specify that you want the output to be in ASCII format (even if the program is compiled with CHARSET EBCDIC). For example:

```
REPORT=x:\TESTDATA\OUTPUT.DAT[PRINTER]
```

Module Summary

There are three methods to assign data filenames to your program: direct allocation, dynamic allocation, and external mapping.

Direct allocation and dynamic allocation assign the filenames within the program source.

External mapping assigns the filename outside the program in Windows or through the Net Express IDE.

5.1 File Assignment for CALLTEST

1. Define **environment variables** for the data files for CALLTEST.CBL. The data files are located in the folder x:\NETXCLASS\DATAFILE and named CUSTFILE.DAT and CALLFILE.DAT.
2. Define **two more entries** for the two output files, ERRFILE.DAT and CALLRPT.DAT. These output files do not need to physically exist prior to executing the program. Opening them for OUTPUT within the program creates them automatically.
3. Compile **the program** using the directive Assign(Dynamic) which is the default and clean-up any errors.
4. Test - Start the Animator (select Animate, Start Animating from the main menu). After the Animator starts, use the Step icon to move through the program only until you get past the file open statements. Ensure files open successfully by checking the file status codes after each OPEN. Do this by examining (right-click over filename, choose Examine) the filename in the OPEN statement. All should have last status of 0. Close the Animator.

5.2 PC-Target File Assignment (Optional)

1. Create a **project** for **MSTFILUP**. Add MSTFILUP from \IDEPROGS.
Modify MSTFILUP to use fixed file assignment.

2. Copy the input files to the \MSTFILUP directory.

3. The program is currently coded to use fixed file assignment.

```
SELECT OLD-PARTS-MF-IN    ASSIGN TO 'MST3DATA.ASC'.
SELECT CUR-ACT-FILE-IN    ASSIGN TO 'ACT3DATA.ASC'.
SELECT PROD-LINE-FILE-IN  ASSIGN TO 'PRD3DATA.ASC'.
SELECT SORT-WORK-FILE     ASSIGN TO 'SORTFILE.ASC'.
SELECT NEW-PARTS-MF-OUT   ASSIGN TO 'NEWPARTS.ASC'.
SELECT REPORT-FILE-OUT    ASSIGN TO 'REPORTER.ASC'.
```

4. Compile **the program** using the directive Assign(Dynamic) which is the default.

5. Test. Start the Animator (select Animate, Start Animating from the main menu). After the Animator starts, use the Step icon to move through the program only until you get past the file open statements. Ensure files open successfully by checking the file status codes after each OPEN. Do this by examining (right-click over filename, choose Examine) the filename in the OPEN statement. All should have last status of 0. Close the Animator.

6. Modify MSTFILUP to use DYNAMIC File Assignment (Working-Storage) instead of FIXED file assignment. This requires the use of logical ASSIGN filenames. For example (Note: To make life easier, the code is already in the program commented out):

```
SELECT OLD-PARTS-MF-IN    ASSIGN TO MST3DATA.
SELECT CUR-ACT-FILE-IN    ASSIGN TO ACT3DATA.
SELECT PROD-LINE-FILE-IN  ASSIGN TO PRD3DATA.
SELECT SORT-WORK-FILE     ASSIGN TO SORTFILE.
SELECT NEW-PARTS-MF-OUT   ASSIGN TO NEWPARTS.
SELECT REPORT-FILE-OUT    ASSIGN TO REPORTER.
```

The logical names are then resolved in Working storage with variables.

For example,

```
05      MST3DATA PIC X (12) VALUE "MST3DATA.ASC".
```

Compile **the program** again.

7. Step past the **OPEN statement**.
8. Modify MSTFILUP to use external file assignment. This also requires the use of logical ASSIGN filenames, but no Working-Storage fields are required.
9. Compile **the program** using the ASSIGN(EXTERNAL) directive.
10. Animate MSTFILUP. It should ABEND on a file not found -- the files have not been assigned externally.
11. Right-click on **MSTFILUP.CBL**. Select **Project Properties**.
12. Under Project Directives, add **ASSIGN(EXTERNAL)** just before the semi-colon.
13. Press the **IDE** button. Enter variable/value pairs for:

MST3DATA	MST3DATA.DAT
ACT3DATA	ACT3DATA.DAT
PRD3DATA	PRD3DATA.DAT

14. Press **OK**. Close the **Project Properties window**.
15. Re-compile the program and animate it. It should find the files.

5.3 Providing File Access for SYSIN/SYSOUT

Introduction

The program Sysintst.cbl accepts a single line of SYSIN (the current date, in Gregorian date format). Its DISPLAY statements should be mapped onto SYSOUT (recall that the SYSIN and SYSOUT files have no extensions). For this exercise, SYSIN has been created in \NETXCLASS\DATAFILE. It contains a date in Gregorian format which looks like 06-27-98. SYSINTST reformats that date, and adds the century.

Details

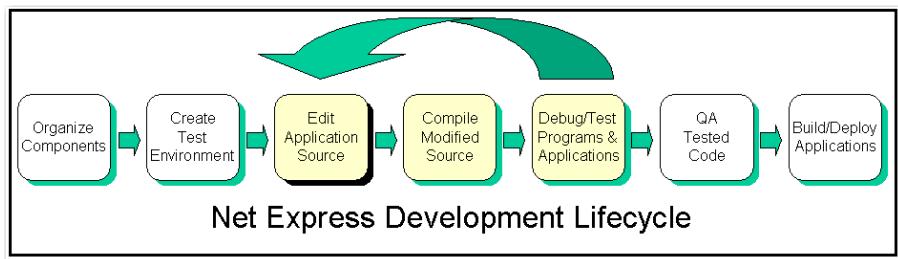
1. Create a **SYSINTST project** in a new SYSINTST directory.
2. Add **Sysintst.cbl** to the project from \NETXCLASS\OTHERS.
3. Add the SYSIN file from \NETXCLASS\DATAFILE to the source pool; this copies SYSIN to the project directory. **Hint:** Search using *.* wildcards.
4. Modify the project directives. Select **Project, Properties**. Add INDD and OUTDD just before the semi-colon.
5. Compile **SYSINTST**.
6. Animate **SYSINTST**. Step through the statements, and examine as the SYSIN-DATE-IN gets reformatted and moved to SYSOUT-DATE-OUT. When the program ends close the program (in order to unlock the SYSOUT file).
7. Select **File, Open** and open **SYSOUT** and view the contents. Close the SYSOUT file when finished. Or type in a command window **TYPE SYSOUT**.
8. Close the **editor**.
9. You can also try the program using NOINDD and NOOUTDD. The program expects input from the Application Output console and returns the result to the console.

Module 6

Editing COBOL Source

Overview

After organizing your application source components into projects, use the editor to create new components and modify existing ones.



Objectives

At the end of this module you will be able to:

- Load files into the editor
- Customize the editor
- Navigate through a file
- Work with and manage copybooks
- Find and replace text
- Block columns and lines of text: copy, cut, and paste
- Use bookmarks to speed file navigation.

The COBOL Editor

The Net Express Editor takes full advantage of drop-down menus, pop-up context menus, mouse functionality, push buttons, and other GUI productivity aids. The Editor provides extensive editing facilities with features directed specifically at editing COBOL source programs. The Editor is designed to recognize COBOL coding rules such as margins and reserved COBOL words and is tightly coupled with the compiling and testing tools, which enhances productivity.

Listed below are some of the features supported by the Editor:

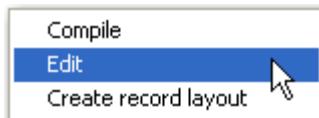
- Use Microsoft and Micro Focus shortcut key assignments for added productivity
- Navigate quickly with COBOL keyword highlighting
- Find and replace text, COBOL variables, and keywords
- Auto correct common spelling mistakes
- Easily work with COBOL copyfiles to view, modify, and create them
- Run In-line compiles with error tags
- Customize many of the Editor's facilities and performance features.

Loading Files into the Editor

There are several methods used to load existing files into the Editor. The method you use will depend on what you are doing and whether or not the file is included in the current project.

Load Project Files from the Project View or Source Pool

Follow the steps below to load project files into the Editor from the Project View or Source Pool.



1. Locate the COBOL source file (file types .CBL).
2. Right-click the component to load.

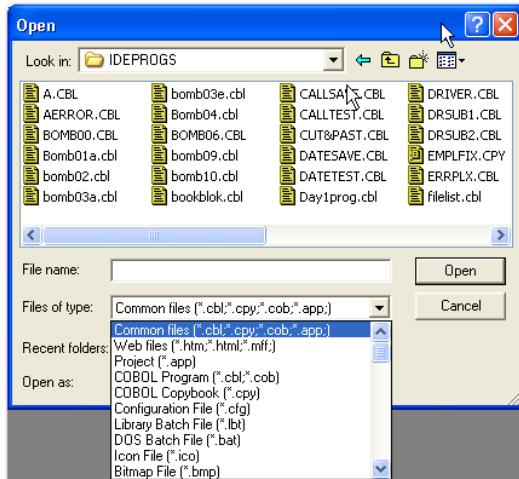
3. Select **Edit** from the context menu.

or

1. Double-click the component. Note: The result here depends on the components file type (.CBL or .CPY invokes the Editor, .INT invokes the Animator).

Load Non-Project Files

You can also edit files that are not in the current project (e.g., you may want to copy and paste code from one program to another). Follow these steps to open non-project files. **Warning:** The current project's properties do not apply when a non-project file is opened.



1. Select **File, Open**.
2. If necessary, change the **Files of Type selection** to the extension for the file type you want to open in the Editor.
3. In the Open dialog box, select the **folder where the file resides**.
4. Double Click a file or Select the file(s) to open; press **Open**.

Open Recently Used Files

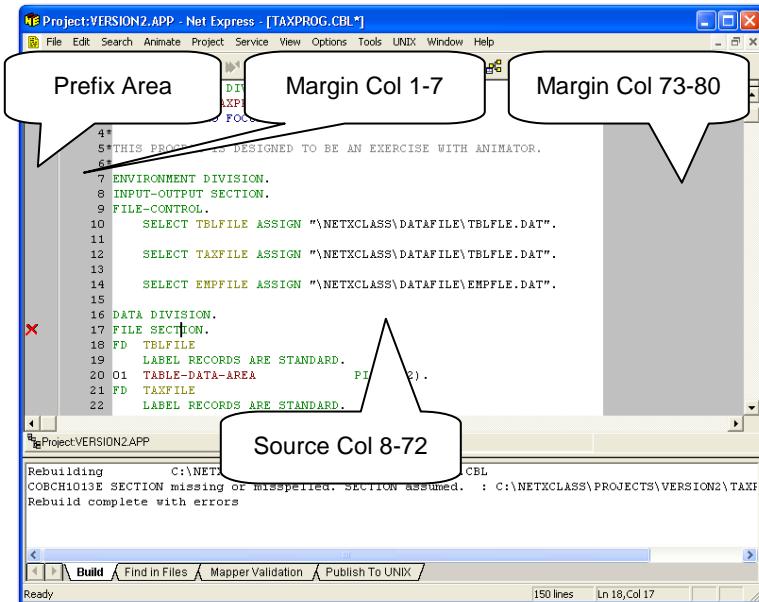
Use **File, Recent Files** to open one of the last four files accessed from the Recent Files list. The number of files in the list is configurable.

Create a New File with the Editor

To create an empty file in the Editor, select **File, New, Program**.

The Editor Window – COBOL Source

When opening a file for edit, the Editor displays the selected component in the client area of the Net Express Desktop. The window may or may not have been maximized depending on your last use of the Editor. For optimum view of the COBOL source you may want to undock the output window.



Prefix Area

Appears to the left of the Prefix Border Line. This area is used to display icons resulting from different operations. For example, an X icon marks a line with a compiler error; a magnifying glass marks a line with references to items that were examined. You can turn off/on display of the Prefix Area when editing COBOL source files. If displayed, the Prefix Border Line appears as a vertical shaded line on the left side of the edit window. To hide the Prefix Area, right-click on any line in the Editor and remove the

checkbox next to Prefix. To re-display the Prefix Area, right click on any line in the Editor and select Prefix Area.

Source Area

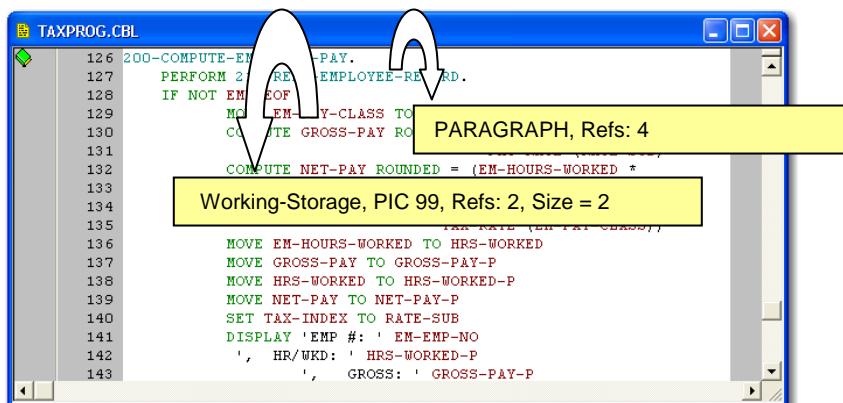
Following the rules of COBOL, the source area is divided as follows:

Column	Description
1-6	COBOL line numbers. Appears grayed.
7	Continuation and comments. Appears grayed.
8-72	COBOL source code. Appears white with colored text. Color defaults are set to enhance COBOL understanding. For example: procedures are displayed as green, variables are dark red. Defaults can be changed or reset from the main menu by selecting Options, Customize IDE, Colors .
73-80	Appears grayed.

Source Area - Hover Details

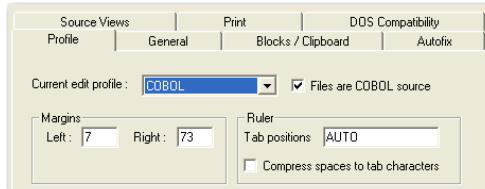
The Editor provides on-demand analysis details about COBOL source.

When the mouse is used to hover over a COBOL item such as a procedure or variable, information is displayed to aid in quick understanding of where an item is defined, its definition and size, how many times it is used, etc.



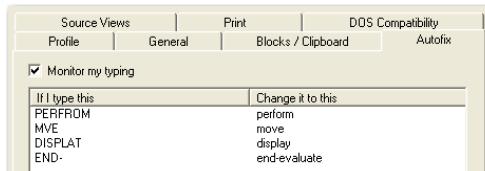
Margins

Margins differ based on the type of component loaded into the Editor. This allows you to easily use the editor to edit files other than COBOL source files. To set or modify file margins use **Options, Edit, Profile**.



Autofix

Autofix is an Editor option that can help prevent common typo errors and speed-up coding operations via text shortcuts. In the example below, two common typos for 'PERFORM' and 'DISPLAY' are set to Autofix. 'End-' is listed as COBOL shorthand for 'end-evaluate'. To set the keyword list, use the **Add, Change, Delete** buttons under **Options, Edit, Autofix**.



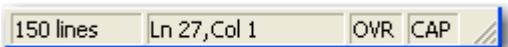
Navigating and Scrolling within a File

Standard Windows methods are used to navigate through the source document loaded into the Editor.

Up, Down arrows	Move the cursor one line up or one line down.
Left, right arrows	Move the cursor one character to the left or right.
Ctrl + Home/End	Beginning/end of file.
Page Up/Down	Move the cursor one page up/down.
Home (Recursive)	Depends on where the cursor is. From the middle of a line, press Home to:

	<p>Move to the home position, column 8, of the line.</p> <p>Move to the beginning of the line, column 1.</p> <p>Move to the beginning of the first line displayed on the screen.</p> <p>Move to the beginning of the file.</p>
End (Recursive)	<p>If the cursor is in column 1 to 6, successively hitting the End key will:</p> <p>Move to the continuation and comment column, column 7.</p> <p>Move to the beginning of COBOL margin A, column 8.</p> <p>Move to the beginning of comment area, column 73.</p> <p>Move to the end of the maximum_line length (column 256).</p>
Tab	Move the cursor 4 characters to the right.
Shift + Tab	Move the cursor 4 characters to the left.
Ctrl + Right/Left Arrow	Move the cursor to the start of the next/previous word on the line.
Insert	Toggles the typing mode between INSERT and OVERTYPE mode. When OVERTYPE mode is selected OVR appears on the status line.
Caps Lock	Toggles the mode between CAPS ON and CAPS OFF.
Scroll Bars	<p>Using the mouse, vertical and horizontal scrolling can be achieved using the scroll bars of the Editor window. For example:</p> <p>Place the mouse on the Slider and hold down. Drag the Slider to the desired location. Click above/below the Slider to move one page up/down. Use the mouse on the Up/Down scroll bar arrow to move one line.</p>

Note: The Net Express Desktop Status Line displays the current cursor row/column, total number of lines, and status of the Insert and Caps Lock keys are also displayed.



Saving Source Files

- | | |
|----------------|--|
| File, Close | Closes the file in the current edit window. If the file has been modified and not saved you will be prompted as to whether or not you want to save the file before closing the current window. Responding No results in the loss of all modifications you've made since the last save was performed. |
| File, Save All | Saves all files in all open windows that have been modified since the last save operation. |
| File, Save | Saves all modified files in the file loaded in the current window. If this window includes copyfiles that you modified, they will be saved to disk. |
| File, Save as | Saves the current file with a new name. |

Edit Menu Functions

These following functions are available from the Edit menu. If a choice is not appropriate for the function being performed it appears disabled (grayed out) on the Edit menu. For example, Data Tools are not required when editing COBOL source.

Undo—Reverses previously performed edit functions. Multiple undo operations may be performed.

Redo—Reverses previous undone edit functions that were reversed using the Undo function. Multiple redo operations may be performed.

Cut—Removes selected text to the clipboard for pasting in another location of the same source, or another source file.

Copy—Copies selected text to the clipboard for pasting in another location of the same source, or another source file.

Paste—Inserts previously cut or copied text into the current source module at the cursor location.

Delete—Deletes selected text from the edited source. Deleted text is not placed into the clipboard and cannot be pasted.

Suggest Word—List syntactically correct COBOL code depending on cursor location.

Select All—Selects the entire source file text without using the mouse.

Deselect All—Removes the selection from anywhere within the source file without using the mouse.

Class Wizard—Add Methods or Data Items

Data Tools—Manage records in a data file.

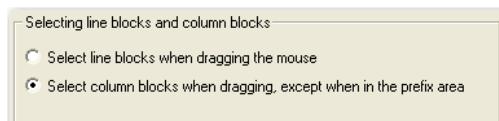
Selecting Text Blocks and Working with Selected Text

Select text so that you can delete it, move it or copy it to another location in the same source file or to another source file. There are two methods for selecting text:

Line blocks—The entire line is selected. By default, the Editor marks lines, not columns.

Column blocks—Where columns within a line (or several lines) are selected.

With column marking selected it is possible to select columns within a line (or several lines) or entire lines. We recommend you change the default from line to column marking for this functional support. Use **Options, Edit Blocks/Clipboard** to change the default from line blocks to column blocks.



Using Line Marking

Selected lines are marked in their entirety. When a copy is performed the entire line is copied to the clipboard. When a cut is performed the entire line is removed leaving no unwanted white space. When a paste is performed the text lines are inserted above the current cursor line.

Marking Lines of Text

Follow these steps to mark lines of text using the mouse (when column marking is selected).

1. Place the **mouse pointer in the Prefix Area** (to the left of the line).
2. Press **the left mouse button** on the first line to select, hold the left mouse button down.
3. Drag the mouse to the **last line to select**.
4. Release the **left mouse button**.

To mark lines of text using the keyboard, follow these steps.

1. Place the cursor in **column 1 of the first line** to select.
2. Hold down the **Shift** key.
3. Press the **Up** or **Down** arrows to select as many lines as necessary.

Marking Columns of Text

Follow the steps below to mark columns of text using the mouse.

```
TAXPROG.CBL
1  IDENTIFICATION DIVISION.
2  PROGRAM-ID.  TAMPROG.
3  AUTHOR.  MICRO FOCUS.
4*
5*THIS PROGRAM IS DESIGNED TO BE AN EXERCISE WITH ANIMATOR.
6*
7  ENVIRONMENT DIVISION.
8  INPUT-OUTPUT SECTION.
9  FILE-CONTROL.
10  SELECT TBLFILE ASSIGN "\NETXCLASS\DATAFILE\TBLFILE.DAT".
11
12  SELECT TAXFILE ASSIGN "\NETXCLASS\DATAFILE\TBLFILE.DAT".
13
14  SELECT EMPFILE ASSIGN "\NETXCLASS\DATAFILE\EMPFILE.DAT".
15
16  DATA DIVISION.
```

1. Place the mouse pointer **on the column** to begin marking.
2. Press the **left mouse button and hold it down**.

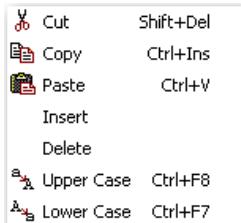
3. Drag the **mouse horizontally and/or vertically** to mark the desired columns.
4. Release **the left mouse button**.

Follow these steps to mark columns of text using the keyboard.

1. Place the cursor on the line and column to begin marking.
2. Hold down the **Shift key**.
3. Press the **Left or Right, Up or Down** arrow keys to extend the selection to the ending line and column of text.

Working with Lines or Columns of Marked Text

Point anywhere within the highlighted area of marked text and right-click to display the context menu.



CUT	Cuts the marked text to the clipboard. Cutting column text will cause any text to the right of the selected column to shift left.
COPY	Copies the marked text to the clipboard.
PASTE	Paste previously cut/copied text from the clipboard. The position of the cursor is used as the beginning point for pasting. Any text already on the line will be pushed to the right of the pasted text. When pasting text that was marked with column marking, any characters to the right of the cursor are pushed to the right toward the COBOL right margin and the cut columns inserted. If the insertion causes characters to move beyond the COBOL right margin a warning message appears.

INSERT	Inserts blank lines (same number as are marked) prior to the marked lines.
DELETE	Deletes marked text to the deleted line stack, not the clipboard.
Upper Case	Toggle selected case to Upper Case
Lower Case	Toggle selected case to Lower Case

Note: See the Appendix regarding the MF keystroke scheme for hot keys and added functionality.

Finding Text

The Search menu and context menus provide access to several useful functions when editing COBOL source files.

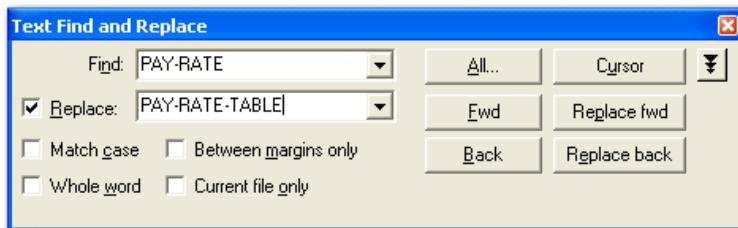
Find Text—Performs text find operations and, if selected, replaces a text string with another text string.

View Related Text—Locate and Quick Browse functions to enhance COBOL understanding.

Find Text in Files—Performs text finds across multiple projects or any selected files.

Text Find and Replace

The Editor supports COBOL text Text Find and Replace operations using the Text Find and Replace window. To invoke this window, from the Menu bar, select **Search, Find Text**, from the keyboard **Alt + S, F** or the binocular icon located on the toolbar.



Using the Text Find and Replace Window

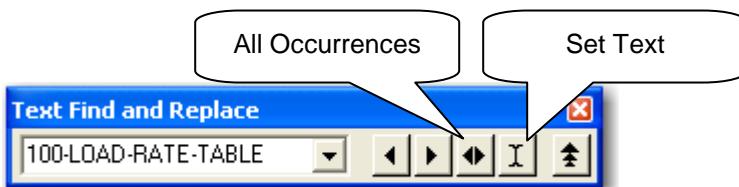
The Text Find and Replace dockable window allows you to enter a text string to search on. Optionally, a text string can be supplied to replace the search string. The controls allow you to modify the operation of find and replace.

1. Enter the **search string** in the Find entry field. If you have previously searched on this item you can press the down arrow in the Find field and select it from the list.
2. To replace text, select the **Replace checkbox** then enter the string to replace with in the Replace entry field.
3. Check any of the following options to maximize your search.

Match case	Perform a case sensitive search.
Whole word	Search for a whole word (not embedded strings).
Between margins only	Limit the search to specified margins. Position the cursor within the margin area to search.
Current file only	Limit the search to only the current file the cursor is in at the moment.
Fwd and Back	Search forward and back in the file from the current cursor location. Found items are outlined in red.
Replace Fwd and Back	Locate the next/previous occurrence of the search string, replace it with the replace string, and move to the next occurrence of the search string. Note: When performing the first Replace operation the search string is located and outlined in red. It is not replaced until you press the Replace button a second time.
All	Locates (highlights) (optionally replacing)

	all occurrences of the search string.
Cursor	Use when the cursor is on the text string to find in the source file. The word the cursor is on is placed into the find entry field.
Show Toolbar View 	Toggle between the Text Find and Replace window and the small toolbar widow.

4. The Toolbar View is a convenient floating window. Use the set text icon to search for text at the current cursor position. Press the left arrow to locate the next occurrence or press right arrow to locate the next occurrence from the cursor position to the end of the program. To find all occurrences press Find all occurrences of specified text use the all occurrences icon.



5. Select **View, Clear, Text finds** to remove the find tags and highlighting from all lines (optional).

Using the All Occurrences Option

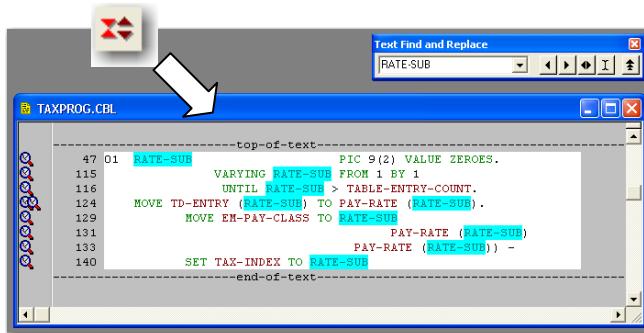
When working with source programs that have not been compiled the Text Find and Replace **All Occurrences** option allows you to locate and view all occurrences of the find text string. Once the program has been compiled you may use Text Find and Replace, or Browse, to perform a similar function.

To find all occurrences of a text string, follow these steps.

1. Enter the **text string to find** in the Find entry field.
2. Select the **All** button from the Find Replace Window or the **All occurrences icon** of the toolbar to locate and highlight all occurrences. You will be positioned on the first occurrence of that search string in the

source listing. The text string is highlighted on the line and a magnifier icon is placed in the prefix area to indicate that a line is included in the search result.

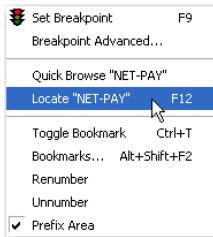
3. To view each occurrence in the expanded source mode, press **Forward** or **the right arrow** to locate the next occurrence (or **Back**, **left arrow**). To view all of the occurrences of the search string, use the Toggle Compress highlighted lines button on the toolbar. A window showing only those lines containing the search string displays.



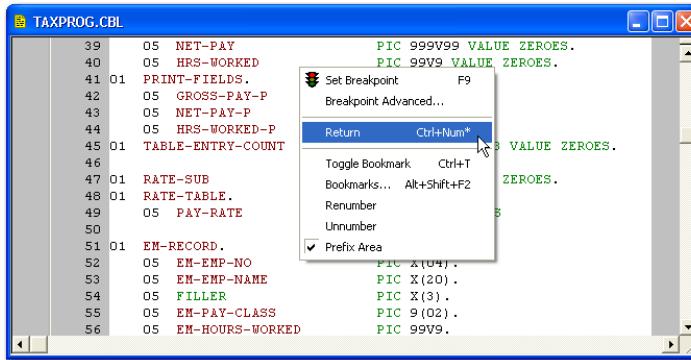
When using the compressed view you do not have the whole picture of how that search item is used in the source. To fully understand how the item is used, position the cursor anywhere on the line and press the Toggle Compress highlighted lines button again to expand at that point.

Locate

While editing, it may be necessary to quickly view the definition of a variable, paragraph, file, etc., to either view or make changes. Right click on a COBOL item and select **Locate** from the context menu.



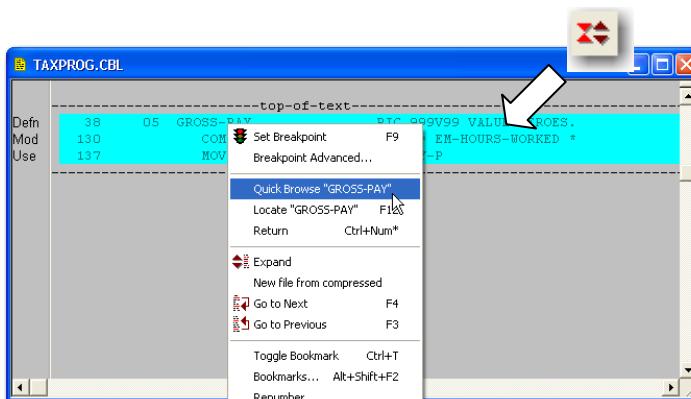
To return to the place where the Locate was executed, Right click anywhere in the edit palette to redisplay the context menu. Select **Return**.



For quick navigation in a program select **Search, Locate COBOL Section**. Select one of the following: Procedure Division, Working-Storage Section, Linkage Section, File Section or Screen Section.

Quick Browse

The Browse option line-highlights all references to a COBOL item, COBOL verb, etc. Right click on the item to research (variable, paragraph, verb) and select **Quick Browse**. The editor repositions at the definition of the item and highlights all further references. As shown below, the Browse can be compressed/ expanded as needed.

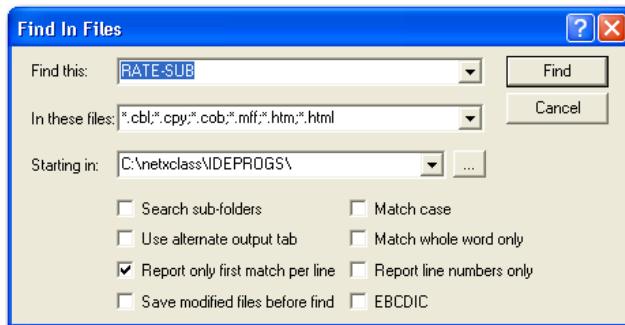


Find in Files

Use **Search, Find in Files** to search for a text string across a project, a directory and its subdirectories, or an entire drive. Use Find in Files for impact tracing of datanames, use of copybooks, and other cross-program analysis. Results are displayed on the Find in Files tab of the Output Window.

Follow these steps to search for a text string using Find in Files.

1. Enter the **search string** in the Find this entry field. Wild cards are not needed since you can (by default) search for part of a string. If you use



the space bar then the spaces will be part of the search (ER- only searches for text beginning with ER-).

2. Enter the file types to search for. By default the search is restricted to source files most important to Net Express (COBOL, copybooks, HTML files, etc.), but you can enter other file types as necessary or use wildcards (such as *.* or name.*).
3. In the Starting in field, type the root for the search. By default, sub-directories are searched. Click the Browse button (...), to the right of the Starting in box, to browse the directory structure and select the appropriate starting folder.

You may wish to compare results between different searches (for example. ER- vs. EM-). Check the **Alternate Output tab** to display a second tabbed

The screenshot shows a COBOL editor interface. At the top, there's a title bar for 'TAXPROG.CBL'. Below it is a code editor window containing the following COBOL source code:

```
138      MOVE HRS-WORKED TO HRS-WORKED-P
139      MOVE NET-PAY TO NET-PAY-P
140      SET TAX-INDEX TO RATE-SUB
141      DISPLAY 'EMP #: ' EM-EMP-NO
142      ', HR/MKD: ' HRS-WORKED-P
143      ', GROSS: ' GROSS-PAY-P
144      ', TAX RATE: ' TAX-RATE (TAX-INDEX)
145      ', NET PAY = ' NET-PAY-P.
146 210-READ-EMPLOYEE-RECORD.
147  READ EMPFILE INTO EM-RECORD
148  AT END
149  MOVE 'Y' TO EMP-EOF-SU.
```

Below the code editor is a results window titled 'TAXPROG.CBL' showing search results:

```
C:\netxclass\IDEPROGS\TAXPROG.CBL(132, 57): 131
C:\netxclass\IDEPROGS\TAXPROG.CBL(134, 52): 133
C:\netxclass\IDEPROGS\TAXPROG.CBL(141, 36): 140
59 occurrence(s) in 8 file(s), 36 file(s) scanned
```

At the bottom of the interface, there are tabs: 'Build', 'Find in Files', 'Find in Files 2', 'Mapper Validation', and 'Publish To UNIX'. The 'Find in Files 2' tab is currently selected.

window with the results of the second search. No intelligence, such as searching for datanames, is used; it is simply a text string search.

For example, search for both “ER-“ and “EM-“ within IDEPROGS. Check **Use Alternate Output tab** to obtain two tabs at the bottom of the result window. The result is two lists of occurrences.

If you double-click on a result line, the source file will be opened in the Editor to the line of interest. Multiple windows can be opened so that you can compare the uses.

Working with COBOL Copyfiles

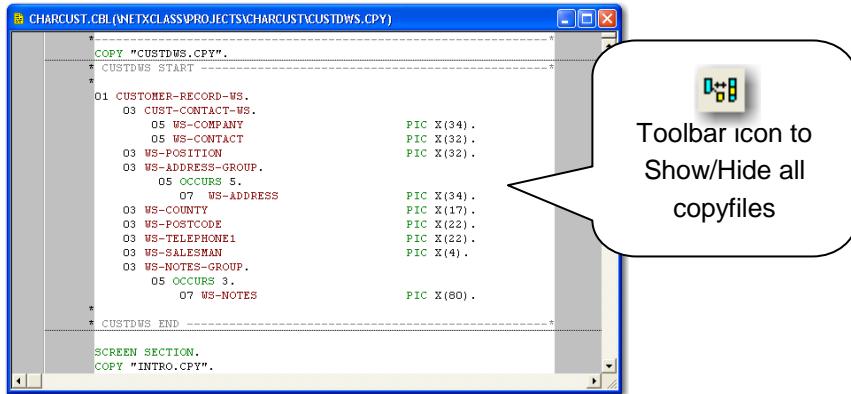
Embedded COBOL copyfiles can be viewed without the need to open additional windows in the Editor. Copyfiles can also be modified/saved within the same Editor window. Select **File, Copyfile** to access the Copyfile options that are described below.

Show—shows the contents of the copyfile immediately below the copy statement (when the cursor is positioned on a copy statement).

Hide—Hides the expanded copyfile source (when the cursor is positioned on a copy statement, or within the body of a displayed copyfile).

Save—Saves modified copyfiles only. Source files containing the copy statements, even if modified, are not saved.

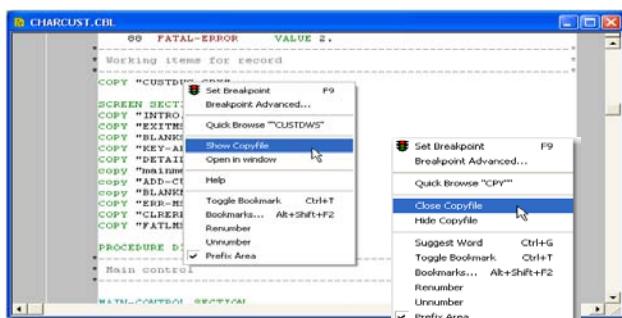
Close—Closes the view of the copyfile. Prompts to save if changed.



When copyfiles are shown within the Editor, they are locked so other users access them as read only. **File, Copyfile, Close** allows you to remove the lock on a copyfile so that other users may open it for editing. When working in a shared project with many users, close the copyfile if you do not need a lock on it. By default copyfiles are collapsed when you open a COBOL source file that has not been compiled. Once compiled, copyfiles are expanded in the code.

View, Hide All Opened Copyfiles—A toggle button on the toolbar can be used to open and close all copyfiles in the program.

Another way to quickly access the copyfile options are via a context menu within the edit session. Right click on a copy statement to display the context menu. Options displayed (Show, Close, Hide) will depend on the current show status of the copyfile.



Saving Edited Copyfiles

Once you have expanded the copyfile, you can view the source code within the Editor or modify the code as necessary providing you have update access to the copyfile.

If desired, you may also open the copyfile in a separate Editor window. In addition to using **File, Open** you can right-click on the copy statement and select **Open in window** from the context menu.

To save an edited copyfile, select:

File, Save—Saves all modified files, including copyfiles, in the active Editor window.

File, Save All—Saves all modified files, including copyfiles, in any open Editor window.

Creating Copyfiles from In-line Source

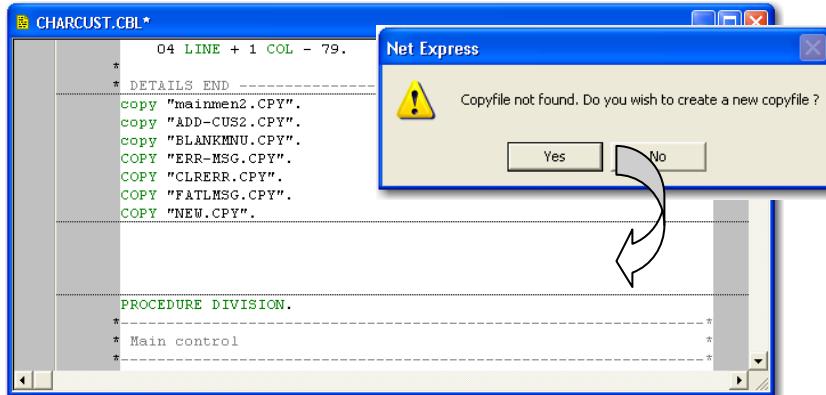
There are two ways to create copyfiles from inline COBOL source while editing.

1. Mark the lines of text (not columns) to be removed from the in-line COBOL source and used to create a copyfile.
2. Cut the marked text from the program using the context menu **Cut**.
3. Select **File, New, Program**. Press **OK**. An empty Editor window is opened for you to paste your new copyfile code into.
4. Paste the cut text into the empty Editor window.
5. Select **File, Save As** From the Save As dialog box, enter the name of the new copyfile. Be sure to assign it the proper copyfile extension. Press **Save**.

In this method, all the work is done within the current window.

1. Mark the lines of text (not columns) to be removed from the in-line source and used to create a copyfile. Cut the marked text to the clipboard.

2. After cutting the code, insert a new COPY statement in your program. Position the cursor on the new copy statement line; select **File**, **Copyfile**, **Show**.
3. Since you are requesting to open a copyfile that does not exist, the Copyfile not found message displays. Respond **YES**. The Editor opens an area for you to add your copyfile code.



4. Paste the code from the clipboard between the two horizontal divider lines showing the beginning and end of the copyfile. **Note:** If the copyfile divider lines do not show you can turn them on from **Options**, **Edit**, **General** tab; set **Show divider lines between copyfiles**.
5. Save the new copyfile before moving the cursor away from this copyfile. **Note:** The title bar shows the location of the new copyfile. Verify that the location is correct before saving the file.
6. Select **File**, **Copyfile**, **Save copyfile name**.

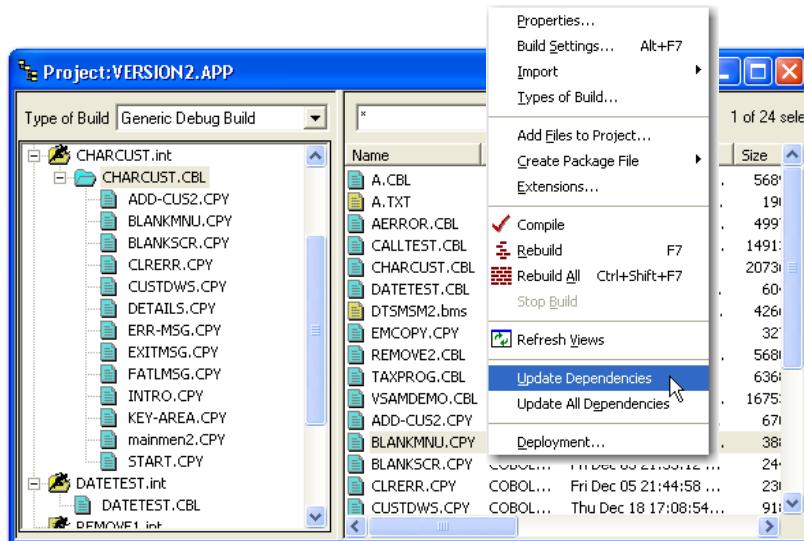
Updating Project Dependencies

After adding copyfiles to your COBOL source programs, return to the Project Tree View and update the project to include any new dependencies you have created. Follow the steps below:

1. Return to the Project Tree View. The Project Tree View shows a plus sign in front of COBOL source programs that contain embedded

copyfiles. To view the embedded copyfile dependencies for a program, click on the plus sign. The Project Tree View now displays all of the dependent files for the selected program. New copyfiles embedded within the COBOL source are *not* reflected in the dependent list.

2. To update the dependencies to reflect new copyfiles in the COBOL program source, select **Project, Update Dependencies**. This updates the dependencies for only the program(s) selected in the Project Tree View pane. Select **Project, Update All Dependencies** to update all dependencies in the project. Use caution when choosing to update all dependencies within a project. Since each source must be searched for embedded dependencies this may take considerable time in large projects.



Note: Updating the project dependencies will ensure that your view in the Project Window of all project components is correct and up-to-date. It is possible, however, to compile programs successfully (as long as dependent copybooks can be found) without updating the dependencies. However, to insure an up-to-date project you should always update dependencies after any activity that may affect them.

Using Bookmarks

Set Bookmarks on lines of source so that you can easily jump to marked lines when necessary. Bookmarks can be set using Search, Toggle Bookmark or the source line context menu. Bookmark icons appear in the prefix area on all bookmarked lines.

```

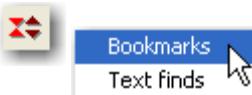
TAXPROG.CBL
 97    CLOSE TBLFILE
 98        EMPFILE.
 99    CLOSE TAXFILE.
100    DISPLAY 'TAX PROGRAM TERMINATED SUCCESSFULLY'.
101    STOP RUN.
102
103 040-DISPLAY.
104    DISPLAY ''.
105
106 050-GET-TAX-INFO.
107    READ TAXFILE INTO TAX-RECORD
108        AT END MOVE HIGH-VALUES TO TAX-RECORD.
109
110 100-LOAD-RATE-TABLE.
111    PERFORM 110-READ-TABLE-RECORD.
112    IF NOT TABLE-EOF
113        MOVE TD-NO-OF-ENTRIES TO TABLE-ENTRY-COUNT
114        PERFORM 120-STORE-TABLE-ENTRY

```

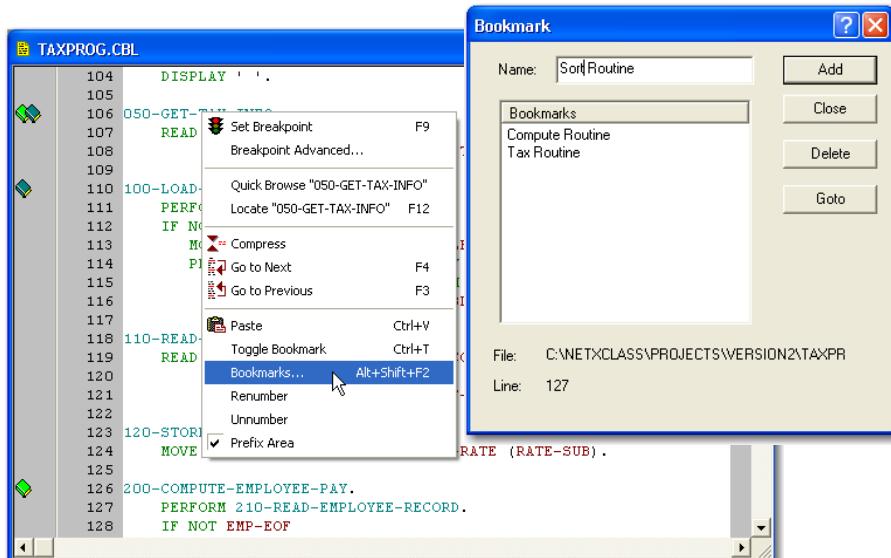
You can set multiple bookmarks. With multiple bookmarks set there are several ways that you can jump from one bookmark to another.

- **Jump to Next/Previous Bookmark.** Select View, Next/Previous, Bookmark, use the shortcut key F3/F4, or (while on the bookmark) use the context menu's Go to Next or Go to Previous.
- **When only bookmarks are present in the source.** From the toolbar, toggle Compress/Expand. The view is compressed to show only those lines containing bookmarks. Position the cursor on the bookmark line you want to jump to, then toggle Compress/Expand; the view is expanded and you are positioned on the selected line.
- **When bookmarks and other line tags are present in the source.** Toggle Compress/Expand. Select the line tag type to compress from the context menu, position the cursor on the bookmark line you want to jump to, then toggle Compress/Expand; the view is expanded and you are positioned on the selected line. **Note:** The scenario above occurs when

the cursor is not positioned on a line having a line tag. If the cursor is positioned on a line with a line tag, the Editor assumes you want to compress that line type and shows the compressed lines. You can also compress/expand line tag types by selecting **View, Compress, Line tag type**.



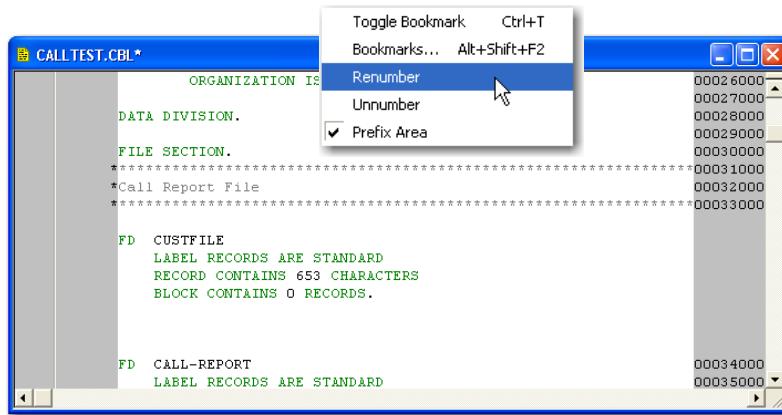
- **Name and Jump to Specific Bookmarks.** Use the Bookmarks option to open a window for naming specific bookmarks. The list of named bookmarks is maintained via the widow pushbuttons. Navigate between named bookmarks by highlighting the desired bookmark in the list and clicking Goto. The bookmark icon is displayed in a different color in the prefix column but are included if bookmark compression is requested.



Line Numbering

The Net Express editor provides the option for storing code with or without line numbers. With the file open, **right click** in the edit area to display the context menu.

- Click **Renumber** to display or renumber the source listing.
- Click **Unnumber** to remove line numbering.



Editing Toolbar Functions

The following is a summary of the toolbar buttons that are useful when editing COBOL source files.

	Open an existing document —Accesses the open file dialog box using a single mouse click. This is the same as doing a File, Open.
	Save the active document —Saves all modified files in the current window. This is the same as doing a File, Save.
	Undo the last action —Undoes the last edit action performed. This may be repeated until the arrow changes from red to gray. When the arrow turns gray it means that there are no more actions to undo. This is the same as doing an Edit, Undo.

	Find the specified text —Opens the Find Text window. This is the same as doing a Search, Find Text.
	Compress/expand —Is very useful in conjunction with other functions within the editor. For example, when doing a Find All operation, all lines containing the text string to find are highlighted. Using this button, you can toggle between viewing just the highlighted lines containing the text or viewing the full source with the highlighted lines embedded in their original location.
	Collapse Copyfiles —Expands or collapses copyfiles in the COBOL source.
	Compile Program —Compiles the program loaded in the current edit window.

Module Summary

The Net Express Editor provides many specific features to assist with editing COBOL programs.

You may change Editor options to allow column blocking in addition to line blocking.

The Editor provides functions such as copy, cut, paste, and find/replace.

You may view and modify (if allowed) embedded copyfiles in the Editor.

A useful book-marking facility is provided.

You may compile the program loaded in the Editor window.

6.1 Find Text, Text Find and Replace, and Find in Files

In this exercise you will find text strings within COBOL source programs in the CALLTEST project. You will also use various methods to replace the text.

1. Open the **CALLTEST project**.
2. Open the CALLTEST.CBL COBOL source program into the Editor.
3. Make a backup copy of the program by saving the current file as **CALLSAVE.CBL**.
4. Close this new program **CALLSAVE.CBL** and reload the original program **CALLTEST.CBL** using the File menu.
5. Using the FIND function, locate the **PROGRAM-COUNTERS definition** in the Working-Storage Section of the program.
6. There are three elementary items within the PROGRAM-COUNTERS group. For each item in the group we must prefix all occurrences with COUNT. Use the following methods.
 - For the elementary item PAGES use the 'ALL' occurrences option to replace all occurrences of PAGES with COUNT-PAGES.
 - For the elementary item RECORDS use the compress option to view and inspect each item before replacing it. Be sure to only change those entries that are required.
 - Use your method of choice to change all occurrences of ERRORS to COUNT-ERRORS.

6.2 Working With Copyfiles

In this exercise you will use various methods to create new copyfiles in the CALLTEST project.

1. If not open, open the **CALLTEST project**.
2. Open the program **CALLTEST** in the editor.
3. Using the FIND functions, locate all occurrences of the variable **FILE-STATUS**. Compress **the finds**.
4. Position your cursor on the definition of the variable and expand the code. The variable FILE-STATUS is an elementary item that contains a number of 88 level definitions. We want to create a copyfile for this variable and use the copyfile in the program instead of the current hard coding.

Create this copyfile. Name it **FILESTAT.CPY**. Verify that you have performed this task correctly. Save CALLTEST.

We need to add a new Select/Assign entry and an FD entry for an ERROR file that we are adding to this program.

5. Copy the existing **CALL-REPORT Select/Assign** and paste **copied code** directly below the CALL-REPORT entry.
6. Change the name of the file to **ERROR-FILE** and the file assignment to be **ERROR.DAT**.
7. Copy **the FD for the existing CALL-REPORT** and its' associated record description, REPORT-REC and paste the **copied code** directly below the CALL-REPORT entry. Change the name of the FD to **ERROR-FILE**.
8. Edit the **Record Contains line** to show that the new record will contain 67 characters.
9. Mark the **01 REPORT-REC line and all of its associated elementary items**.
10. Replace all occurrences of REPORT with **ERROR**.

11. Delete **all but the first three elementary items** within the ERROR-REC group.
12. Insert a **new 05 ERROR-GROUP** immediately after the 01 ERROR-REC statement.
13. Change the existing **05 levels for ERROR-COMPANY and ERROR-CONTACT** to level 10.
14. Change the **ERROR-PRODUCT PIC X(10)** to **ERROR-STATUS PIC X(1)**.
15. Save **CALLTEST**.

Now we want to make a copyfile for the CALL-REC and CUSTOMER-RECORD record descriptions under their FDs.

16. For the entry 01 CUSTOMER-RECORD and its elementary items create a copyfile named **CUST-REC.CPY**. Be sure to verify that you have created it correctly before continuing!
17. For the entry 01 CALL-REC create a copyfile named **CALL-REC.CPY**.

6.3 Write a New Program (Optional)

Write a new program, Salary3.cbl, based on two existing partially completed, programs: Salary1.cbl and Salary2.cbl. Salary3.cbl will produce a payroll report. It will read in a file that contains the following information for each employee.

- Name
- Social security number
- Hourly pay rate
- Hours worked this week

The output report will have some calculations. It will contain the following information:

- Name
- Social security number
- Total weekly pay (you will calculate it)

The output report will also have a final line, produced on EOF, which calculates and reports the average salary for all employees. You will need to calculate this average (using an accumulator and a counter).

1. Create a **Salary project** under the ...\\PROJECTS directory. Add **Salary1.cbl** and **Salary2.cbl** from ...\\IDEPROGS. Clone Salary1.cbl to **Salary3.cbl**. Add **Salary3** to the project.
2. Your new program, Salary3.cbl, has some problems. The SELECT statements are incorrect. Fortunately, the correct statements are in Salary2.cbl. The OUTPUT record layout is incorrect. The correct record layout is in Salary2.cbl. Open a **second window**; edit **Salary2.cbl**. Copy the correct lines into your Salary3.cbl.

The Working-Storage variables in Salary3 are probably usable. But the Procedure Division contains all sorts of extra code, including tables, subscripts, and sorts – you don't need any tables, subscripts, or sorts since this is a simple read/write program.

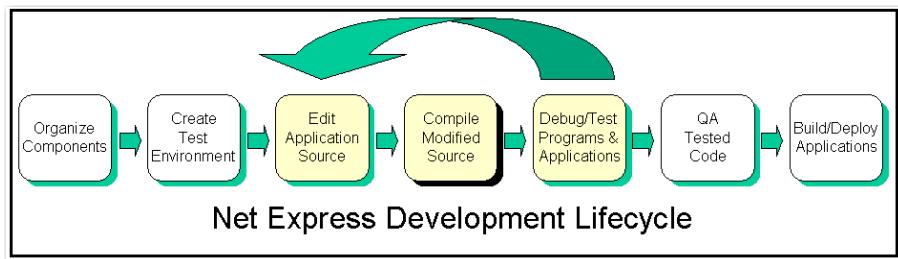
3. Clean up the Procedure Division. **Add the calculations and the MOVEs.** Suggestion: Either put DISPLAYs in your program prior to each WRITE statement so that you can see the report during testing, or change the select statement for the output file so it outputs to the console: SELECT ... ASSIGN CON.
4. Press the **red check button** to compile **Salary3.cbl**. Fix any errors and recompile.
5. Press the **Step or Run button** to animate (test) the program. The correct average salary is \$427.08. (But don't hard code it).

Module 7

Compiling COBOL Programs

Overview

After modifying source, you must compile it. Compiling is the first step to prepare the programs for deployment.



Objectives

At the end of this module you will be able to:

- Describe the Net Express compile process.
- Identify files created by the compile and their purpose.
- Describe the differences between compiling and optimizing.
- Use directives to control the compiler at the component level and the project level.
- Compile your program.
- View and correct compiler errors using the Editor.

Understanding the Compilation Process

Net Express compiles programs either for debugging (General Debug Build) or for deployment (General Release Build). The compiler is integrated with the Editor making it easier and quicker to fix syntax errors. The compile process is controlled by passing parameters (directives) to the compiler via build options.

The Compilation Process

1. Specify directives (parameters) that control the compile process and determine the output files types required (e.g., compile to .INT Intermediate or .GNT Generated or Optimized code?).
2. Select programs to compile.
3. Execute the compile.
4. Fix syntax errors. **Note:** For information on creating executables (General Release Build), see *Module 9 Building Applications for Release*. The balance of this course assumes you are performing a General Debug Build.

Files Created by Compiling

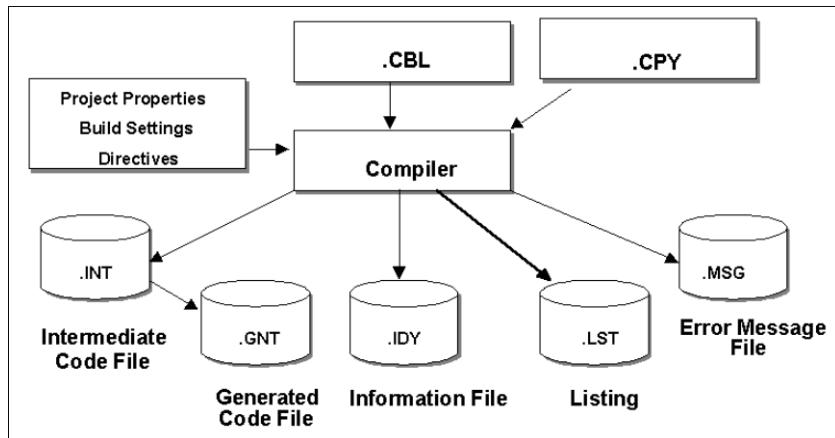
The compile creates a number of files depending on the results of the compile and options that the user requires.

Extension File Purpose

.INT	By default, the Compiler produces the .INT (intermediate code) file used by the Animator when animating COBOL source programs.
.GNT	You can also create a .GNT (generated native code) file, which compiles more slowly but executes quicker than .INT code. Compiling to .GNT code occurs as a second step after compiling to .INT code. Because the .IDY file is created by the first (.INT compile) step, it is available

during animation of .GNT code.

- .IDY The .IDY, cross-reference animation information, file is required by the Animator to debug a program. The .IDY contains information about the program's structure and contents, including debugging information.
- .LST The optional .LST COBOL compile listing contains a source code listing. This file can also contain data/procedure cross-references if desired. **Note:** By default, this option is turned off since it creates a file for every COBOL source program compiled.
- .MSG Compiler errors (informational to severe) are written to the .MSG file. The .MSG file is merged with the edited code so that you can inspect all the coding errors and determine what caused each error. It is automatically erased on a successful compile.



Compiling .INT or .GNT Code

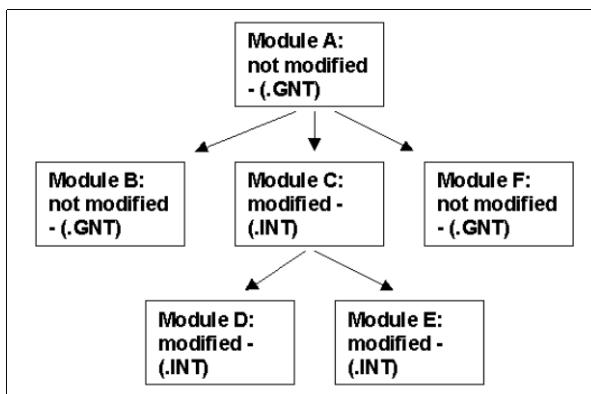
Before a COBOL program can be executed, it must be compiled into an executable module. When working in the Generic Debug Build type, it is possible to create two types of executable modules.

filename.INT modules—INTermediate executable code modules in Micro Focus proprietary format used by Net Express Animator for interactive testing and debugging (e.g., stepping through the code line-by-line).

filename.GNT modules—GeNeraTed native code (i.e.: optimized) modules in Micro Focus proprietary format used in the Animator during interactive testing/debugging of applications, usually when some modules have to be executed, but there is no necessity to debug the code (e.g., a common date routine) or for large programs to execute faster. **Note:** .GNT files can be debugged in the Animator (e.g., Step, Examine, etc.).

.GNT files can be used:

- For common sub-programs that have not been modified but are needed for interactive testing of larger applications or systems (e.g., unmodified driver programs that CALL a mix of modified and unmodified subroutines, for example, end-of-day reports).
- To create output files when no debugging is required.
- When demonstrating a prototype application to users. .GNT files execute faster through better memory use; this has no effect on file reads and writes.



How to Compile

Compiling a Single Program

To compile a single program (.cbl), select **the program** from the Project Tree View and then choose one of these methods.

- Right-click on the **.CBL module** to be compiled and click the **Compile button** in the toolbar.
- Select **Project, Compile pgmname.cbl**.
- Right-click a **program-name** and select **Compile pgmname.cbl**.
- From within the Editor select the **Compile Program button** on the toolbar.

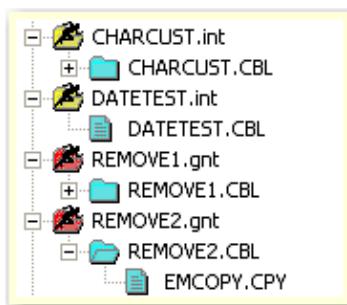
Compiling Multiple Programs

To compile multiple programs you must select (highlight) the programs.

1. Use Ctrl + Left click to select **.cbl names** in the project details.
2. Once you have highlighted them, you can use any of the methods above to compile the programs.

Project Tree View

The Project Tree View displays the executable file type to be created by each COBOL source program when it is compiled. In example below the modules CHARCUST and DATETEST are compiled to .INT for debugging, while two sub-programs REMOVE1 and REMOVE2 are compiled to .GNT for quicker execution since they no longer require debugging.



Rebuild

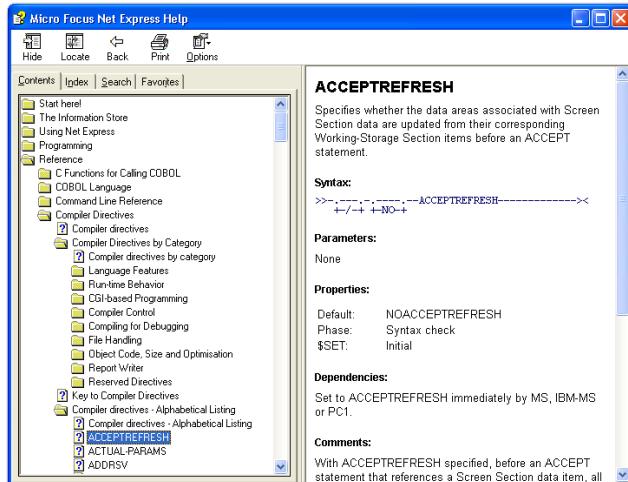
Rebuild All—Use when all COBOL source programs in a project are to be compiled. To rebuild, select **Project, Rebuild All**. Shortcut keys are **Alt+Ctrl+F8**.

Rebuild—Use when you have made changes to one or more components in the project. Rebuild compiles only those components that have been changed since the last Build was performed. Select **Project, Rebuild** or click the **Rebuild** button on the menu bar. Shortcut keys are **Ctrl+Shift+F8**.

Rebuild Object— Use to compile single programs within the project regardless of the status of other modules within the same project. From the Project Tree View, right-click the **.CBL or .INT module** to be compiled, and select **Rebuild object**.

Controlling the Compiler

The Compiler is directed to invoke specific functions when compiling a COBOL source program through Compiler Directives. The compiler supports many COBOL dialects/syntax levels and platforms including Windows and UNIX. The Reference Section of the Net Express Help System lists all directives categorically and alphabetically.

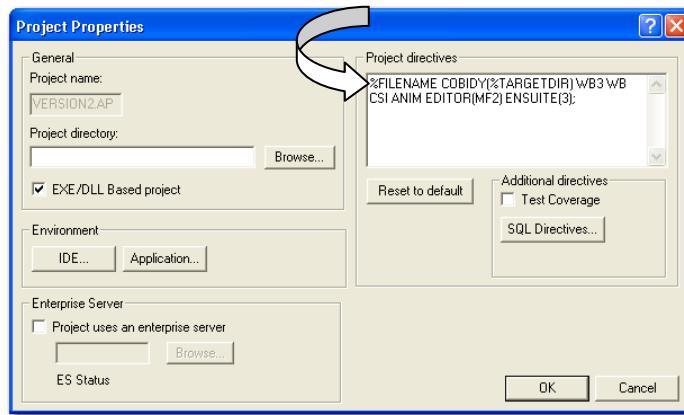


Setting Project Level Compiler Directives

To set compiler directives at the project level, select **Project, Properties** or from the Project Tree View, right-click to invoke the context menu and select **Project, Properties**. The Project Properties Window will be displayed.

The Project Properties window contains settings that apply to the entire project. Changes made to the Project Properties window affect every program within the project unless the properties are specifically overridden elsewhere. Enter compiler directives to be applied to all the project's COBOL source programs in the Project Directives list box.

When entering additional Project Directives in the project directives list box, remember that directives must be separated by a space, there are no spaces within a directive, and a semicolon must follow the last directive.



Setting Directives for a Program within a Project

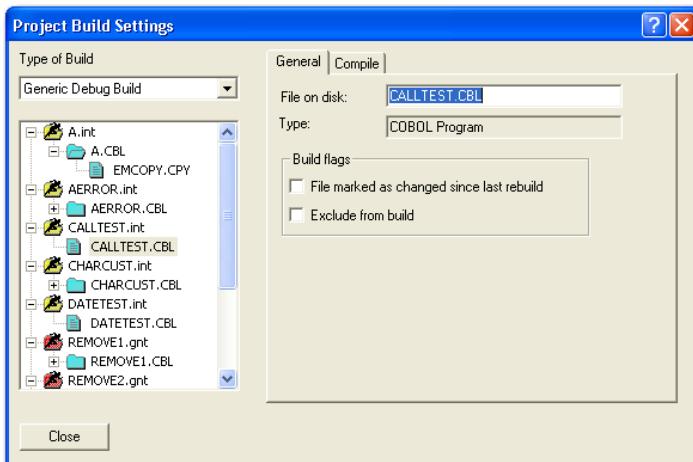
To set additional and/or overriding compiler directives for a selected single program or several selected programs within a project follow these steps.

1. Select **the program(s)**.
2. Select **Project, Build settings** or from the Project Tree View, right-click to invoke the context menu and choose **Build Settings**.

Project Build Settings – Single or Multiple Programs

The Project Build Settings window allows the user to control compiling by setting specific controls for single or multiple programs selected in the project tree view. When a .CBL component type is selected in the Project Tree View, two tabs are displayed, General and Compile. The General tab displays general information (File name on the disk, File Type) concerning the selected module. The directives supplied on the Compile tab control the creation of the .INT and .GNT executables. The most recent selection under ‘Compile to’ will be the default of the re-compile for the module or group of modules. Note: all setting specified on these tabs will override Project level directives and remain unchanged.

Build Settings for .CBL/.CPY Components: General



Build Flags

- **File marked as changed since last rebuild**—Indicates whether the program has been modified since the last Build. When checked, the file was modified after the last Build and requires compiling.
- **Exclude from build**—Allows you to temporarily exclude this component from Project Builds while leaving the component in the Project Tree View.

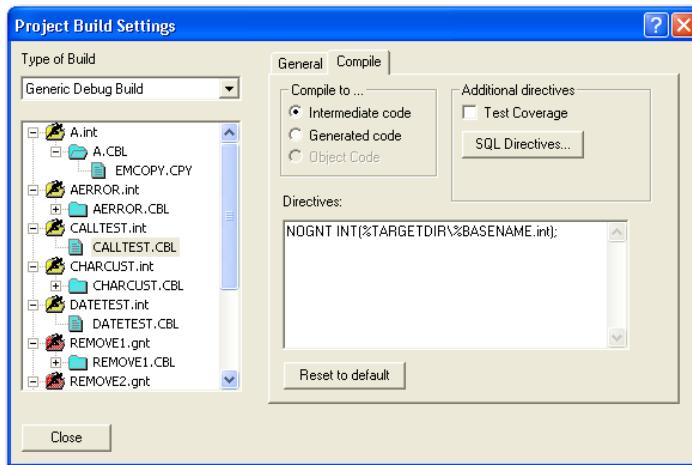
Build Settings for .CBL Components: Compile

The Compile tab displays information concerning the type of executable module that is created when a Project Build is performed and allows the user to enter specific compiler directives for that program. **Compile to** determine the type executable code created when a Project Build is performed.

Intermediate code—creates an .INT module for interactive, source level debugging of your COBOL source programs.

Generated code—creates a .GNT module for execution only.

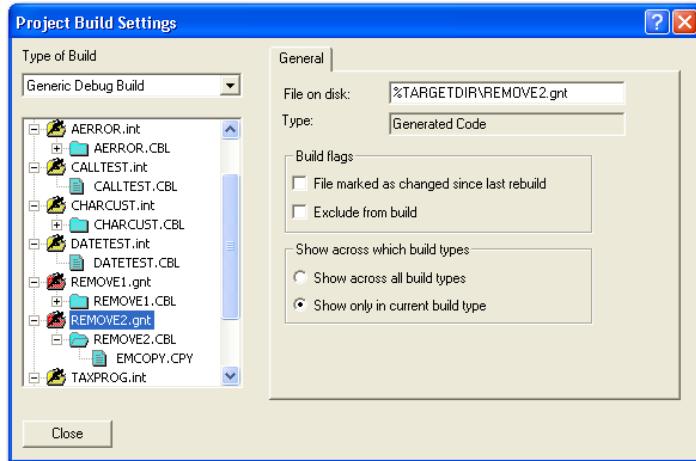
Object code—creates an industry standard .OBJ for linking and creating executables. In this example the object code selection is disabled since this is the Generic Debug Build and not the Generic Release Build settings.



Compiler directives—entered in this window apply only to the program selected in the Project Tree View and to no other program in the project (unless specifically set for that program also). Directives settings entered on this window override directives entered on the Project Properties window.

Build Settings for .INT/.GNT Components

When you select an .INT component type in the Project Tree View, Net Express displays the General tab showing information about the compiled module.



File on disk—Indicates the destination folder and compiled file name on disk. **%Targetdir** is an environment variable indicating the executable module is in the default directory for Generic Debug Builds.

Type—Indicates module is intermediate or generated code.

Build flags

- **File marked as changed since last build**—Indicates whether the program has been modified since the last Build was performed. When checked the file was modified after the last Build and requires compiling.
- **Exclude from build**—Allows you to temporarily exclude this component from Project Builds while leaving the component in the Project Tree View.

Show across which build types

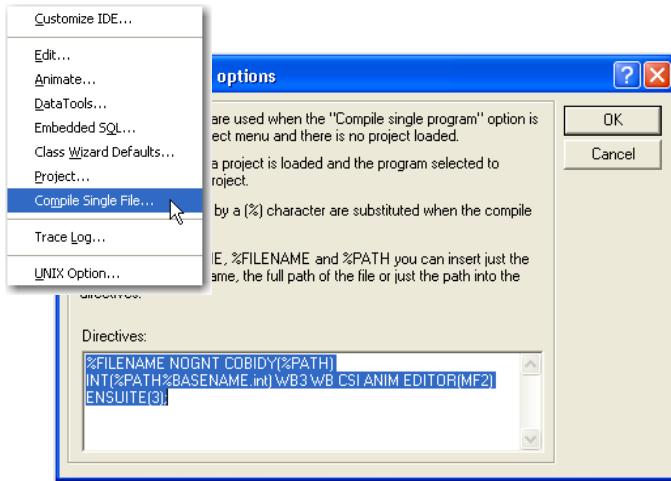
- **Show across all build types**—Indicates that this module type will appear on all Project Build Types.

- **Show only in current build type**—Indicates that this module appears only in the current Project Build Type for this project and not others that may be available for this project.

Note: Different Project Build Types and their purpose will be discussed in a later chapter.

Setting Directives Outside a Project

It is possible to compile programs from the Net Express desktop when no project is open, or to compile a program that is not a component of the current project. To set directives for use when compiling a program, outside of an open project, Select **Options, Compile Single Program**.

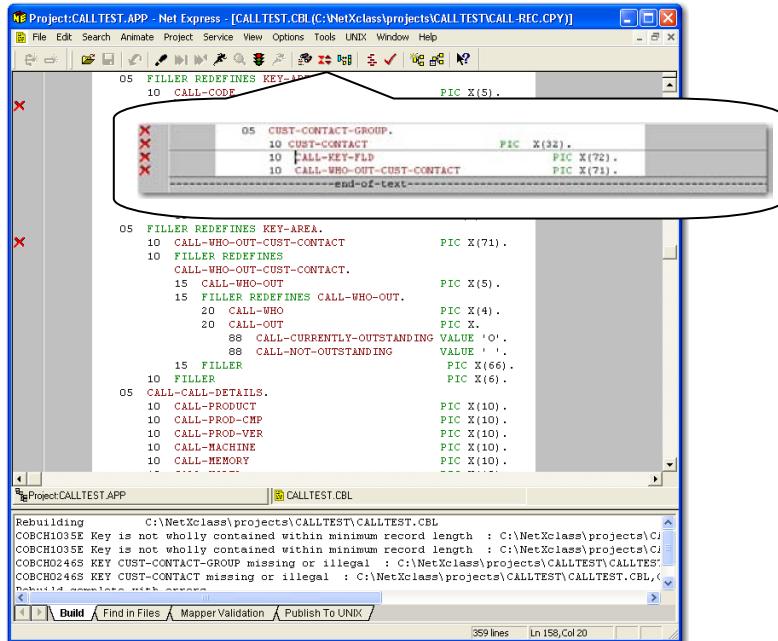


Viewing and Correcting Compiler Errors

After compiling your COBOL programs you will need to fix any compiler errors. Errors display in the Output window.

The Application Output window displays program name, line and column number where the error is located, error message number, severity level of the message and a brief message defining the error encountered. To locate the error in the appropriate component, double-click on the error line.

You are automatically positioned on the error line corresponding to the error that you selected in the Application Output window.

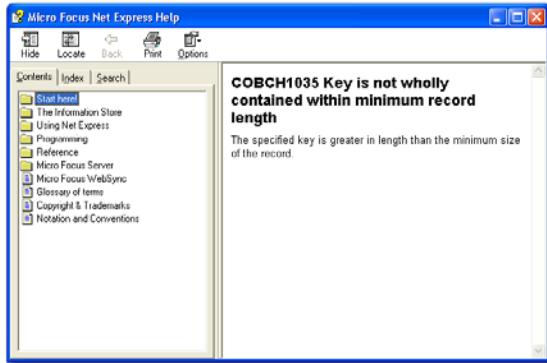


When errors are encountered during a compile each line containing an error is marked with the X icon. You can compress and view only those lines in a program containing an error. Keep the following in mind.

- The error may not be readily apparent. If it is not, use Syntax Error Help to review the details of the error.
- The error may not be on the line that is flagged in error. If no error is evident when looking at the line, position the cursor on the line and expand the code. View the code immediately preceding the error line; it may contain the error.
- The Net Express compiler, like every other COBOL compiler, will reward one of your errors with many additional errors.
- Correcting a few errors, especially the blatant ones, and recompiling the program may reduce the number of errors considerably.

Syntax Error Help

If you need more information, right-click on the error line in the program and select Syntax Error Help. A detailed description of the syntax error displays.



Suggest Word can help you correct spelling errors for data names. To use this facility, position the pointer on the word in question and right-click on the suspect word then select Suggest Word. A short list of words having similar spellings appears at the pointer. Select the correctly spelled word.

Important: If you right-click on what you believe to be a misspelled word and the choice Suggest Word does not appear in the context menu, then the word is probably spelled correctly. There may be another error on the data definition line for that variable and it is flagged as an error.

Options for Setting Compiler Directives

Compiler directives have a system default that can be overridden by the user. You can assign directives at different levels to provide control over the compile process for any of the project's components.

The compile hierarchy is shown in the table below.

Project Level	Compiler directives set at the project level affect all programs in that project when compiled.
Program Level within Project	Compiler directives can be set individually for a single (or multiple) programs within a project. Directives set at the program level override those set at the project level.

Embedded in each Program	The \$SET compiler directive can be placed within the COBOL source program and overrides all other directives settings no matter where they were specified.
COBOL.DIR Side File	COBOL.DIR (any file with the extension of .DIR contains compiler directives) is a high level, system wide directives file normally used to control environmental settings for multiple projects. It can also be placed within the current project directory for project-specific directives. COBOL.DIR can be pointed to using the COBDIR environment variable in Project Properties.
MYDIRECTIVES.DIR Side File	The MYDIRECTIVES.DIR file is be used by a single user to apply directives that are not used by other members of the project team. Include USE (mydirectives.dir) in the Project Properties or in a \$SET statement in your program to use this side file.

Directives are picked up in a definite sequence from Project Level through embedded (\$SET) statements. Since this can add complexity to your application we recommend that you use the Project level and Program Level within Project unless you have special requirements.

Module Summary

Decide on the goal of the compile. Net Express compiles programs either for debugging (General Debug Build) or for moving to the next steps in deployment (General Release Build).

Decide what type of files you want produced by the compile (.INT, .GNT, .IDY, .LST, .MSG).

Use directives to control the compiler.

Compiler errors display in the Output Window and may be corrected in the Edit Window.

7.1 Review Questions

1. How would you specify that you want a compile listing named COBOL.LST for a particular program?
2. What is the difference between a .INT and a .GNT file?
3. If a variable is misspelled, and is flagged by the compiler, what is the easiest way to fix it?
4. How can you easily move from one syntax error in a program to the next?

7.2 Compiling a Project's Programs

1. Open the **CALLTEST project**.
2. Load the **CALLTEST program** into the Editor.
3. Compile **CALLTEST**.
4. Correct **any errors** found in the CALLTEST program.

7.3 Changing Build Settings (Optional)

1. Create a new project named **PAYTEST** in x:\NETXCLASS\PROJECTS\PAYTEST.
2. Add the **ERRPLX.CBL**, **PAYTEST5.CBL**, **PAYERR.CBL** files from x:\NETXCLASS\DEPROGS.
3. **Rebuild All**. Note that copybooks are not found. Resolve them in the Source Pool (the file emcopy.cpy is in x:\NETXCLASS\OTHERS) and **Rebuild All** again. There are errors in some of the programs.
4. From the Project Tree View, right-click on **PAYTEST5.CBL** and select **Build Settings**.
5. Press the **Compile** tab.
6. Add the directive **WARNING(3)** just prior to the semicolon (be sure to leave a space after the previous directive. Warning level controls the compiler's sensitivity to syntax errors. Note: you may need to remove the directive 'NOGNT INT(%TARGETDIR%\%BASENAME.int)' if the compiler flags it as invalid).
7. Generate a listing file by adding the **LIST(COBOL.LST)** directive prior to the semicolon (again, leave a space between directives).
8. Click **Close**.
9. Right-click on **PAYTEST5.CBL** and select **Compile**.
10. Look in the Output window and notice the messages.
11. Edit **COBOL.LST** in x:\NETXCLASS\PROJECTS\PAYTEST.
12. Notice the Options at the top of the listing including **WARNING(3)** and **LIST(COBOL.LST)**. Scroll down a few pages and notice 'I' error messages that are interspersed in the code.

If you have compiler errors or execution errors that require technical support, you may need documentation of the compiler directives used and the errors found.
13. Close the editor on **COBOL.LST**.

14. From the Project Tree View, right-click on **PAYTEST5.CBL** again and select **Build Settings**.
15. Change Warning Level directive from Informational **WARNING(3)** to Error; that is, **WARNING(1)**.
16. Click **Close**.
17. Right-click on **PAYTEST5** in the Project Tree View and select **Compile**.
18. Look in the Output window and notice the messages - no errors. If you look at the COBOL.LST again you will find the errors are gone and that the Options at the top now include **WARNING(1)**.
19. Close the **Editor**.

7.4 Locating Errors

1. From the Project Tree View, right-click on **PAYERR.CBL**, then select **Compile**. Note that you can see the errors in the Output window, but no red X's because the editor is not open.
2. Hide the **Output window**. Add the directive **WARNING(1)** to the build settings. Right-click on **PAYERR.CBL** and select **Edit**. Notice that you can't immediately see the errors in the editor (unless they inadvertently occur on the first screen; and ours don't).
3. Click the **Compress button**; there they are; click the **Compress button** again to expand the code. Notice you are positioned on the first error. There is another way to quickly get to the first error.
4. Close **the editor**.
5. Right-click on **PAYERR.CBL** and select **Edit**. You still can't immediately see the errors in the editor.
6. Select **View, Next, Syntax Error** to move to the first error. Or press **F8** (if MF key scheme is loaded). Another way to position on an error is to double-click on **the error details** in the Output Window. This works even if the file is NOT loaded in the editor. Select **View, Output window** to see the Output window. Note that the error message displays at the bottom of the Net Express window.
7. Double-click on **one of the error messages** in the Output window. This takes you right to that error. Double-click on a few more of the errors in the Output window.
8. Click the **compile** button in the toolbar. Notice that you can immediately see the errors in the editor. You are positioned on the first one (always positioned on the third physical line on the screen).

Suggest Word and Next. The first error message (233-S Unknown data description qualifier PIX) doesn't have anything to do with an invalid data item name, so Suggest Word will not help.

9. Select **View**, **Next**, **Syntax Error**, or press **F8** (MF Keys, if loaded) and watch the error message at the bottom; still nothing to do with an invalid data item name.
10. Move down to the error: “**operand EL-EMP-NO is not declared**”. Right-click on **the data item** and select **Suggest Word**. Select the correctly spelled data item **XEL-EMP-NO**.
11. Compile **the program** and fix any remaining errors.
12. Close the **Editor**.

7.5 Compile Listing and Cross Reference

1. Add **DAY1PROG.CBL** to the PAYTEST project from **\NETXCLASS\IDEPROGS**. Its copybooks are in **\NETXCLASS\OTHERS**. Note that the copybooks are nested!
2. For program DAY1PROG.CBL, change **the Build Settings** to produce a compile listing. Leave the other ‘listing’ options unchanged.
3. Try the directive, **LIST**. Compile. LIST types the listing to the screen.
4. Try the directive as **LIST()**. LIST() creates progname.LST in the Project directory.
5. Try the LIST() directive with the cross-reference directive, **XREF list() xref**.
6. Use **File, Open** to view or edit the compile listing.
7. Notice the compiler options at the top. Scroll down thru the program; notice the interspersed error messages. Scroll down to the bottom. Notice the error messages, page ejects, the ‘totals’, and the cross-reference listing.

Optional Steps

1. What do the symbols (#, *, ?, etc.) mean?.
2. Scroll down until you see WS2-LCNT (59# 83* 84?) and WS2-NBR (61# 74* 83).
3. Scroll up until you see line 59 and then line 61. The “#” after the line number (i.e.: 59#) indicates where the item is defined.
4. Look at line 83 to see that “*” indicates that WS2-LCNT is ‘modified’. A blank indicator means that WS2-NBR is ‘used’.
5. Look at line 84 to see that the “?” is indicates that WS2-LCNT is ‘tested’.
6. Close the listing file.

7.6 Correcting Compiler Errors

In this exercise you will compile and correct any errors in your PAYROLL and CALLTEST projects.

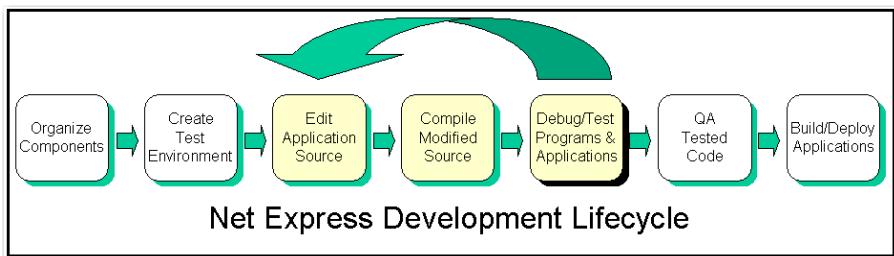
1. Open the **PAYROLL project**.
2. REBUILD the **entire project**
3. Correct any errors found in all programs. You will need to add a couple of report fields to PRTPROG.CBL.
4. If you encounter any program module that is in really bad shape REMOVE it from the project entirely. **Hint:** Use the Source Pool and REBUILD the project.

Module 8

Testing Your Application

Overview

The Net Express Animator is a powerful testing and debugging providing the developer with a visual environment for code execution. The Animator provides functions to control the speed and flow of execution, insert breakpoints, and examine and modify execution data in-stream.



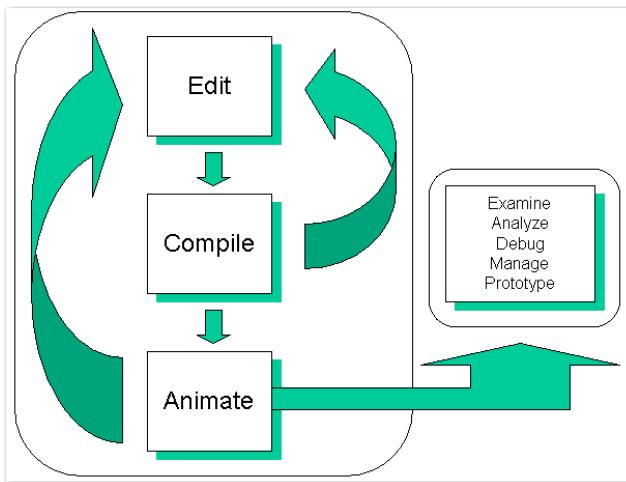
Objectives

At the end of this module you will be able to:

- Visually execute your program source.
- Examine, monitor and alter the contents of a variable, midstream.
- Alter the execution path to experiment with different testing approaches.
- Set a variety of Breakpoints for more productive testing.
- Modify system run date.
- Analyze test data validity.

Testing and the Development Lifecycle

The advanced functionality integrated into Net Express, coupled with the speed of dedicated workstation hardware, allows you to move quickly through edit and syntax check/compile phases of the development lifecycle. This enables you to spend more of your development and maintenance time testing with the Animator. This hastens application understanding and improves code quality.



Prerequisites to Animation

The following are prerequisites for program Animation:

- A clean compile of the program(s) to be Animated in the session
- Intermediate and other internal files, output from the compiler (Object code and an .IDY Animator file)
- Input test data files and file mappings. All external data files must be available to your program prior to Animation. These files must be mapped to your SELECT ... ASSIGN filenames using proper directives and any required environment variables.

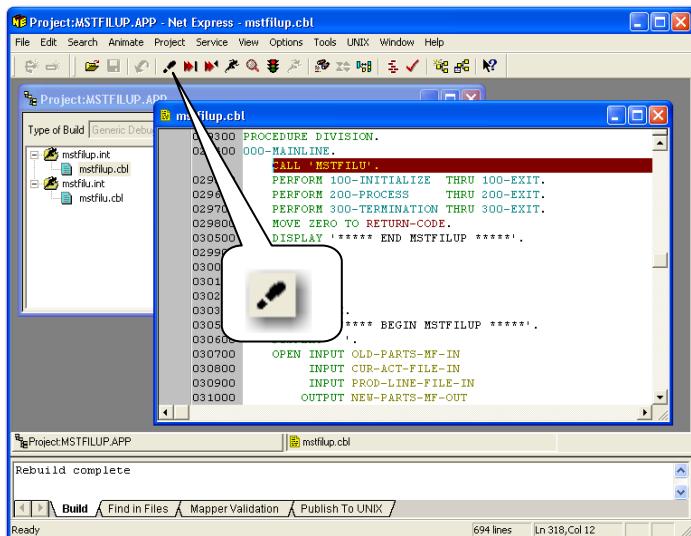
Animation Options

The Net Express Animator provides options to:

- Manage the debugging/testing process (execution control, restarts, breakpoints and others).
- Examine variable content during execution. Set up and examine variable watch lists and monitor in text or hexadecimal display modes.
- Analyze COBOL logic via access to source information tools during Animation.
- Prototype changes to COBOL logic on the fly. Enter and execute COBOL statements, modify and test logic, without modifying the source.

Starting the Animator

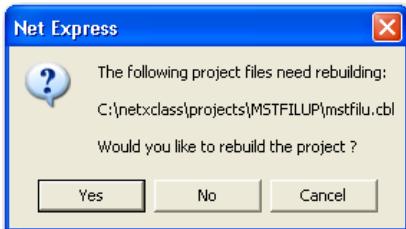
A program can be Animated by selecting it from the list of project programs, specified in Animate, Settings or can be started from the program currently open in an edit session. The Animator will position to the first line of executable code.



Use one of the options shown below to start the Animator:

1. From the Main Menu, Select Animate, Start Aminating
2. Press F11
3. Click the Step icon in the Toolbar menu
4. Right-click on an INT/GNT/EXE in the program tree view and select Animate from the context menu.

If you have other programs in your project that you have not yet compiled, Net Express will prompt you with an option to rebuild the project. Click No if you want to continue with Animating your current selection.



Note: programs don't stop running in the Animator until they logically terminate (STOP-RUN, etc.) or are intentionally closed. Closing the Animator window in midstream execution does not close the Animator. Animation options while a program is active are:

1. Stop Animating – Closes the Animator
2. Restart Animation – Starts execution at the beginning of the program and refreshes the Animator Window
3. Step - Starts execution at the next instruction and refreshes the Animator Window.



Execution Controls

The full set of Animator execution controls are available from the Animate option of the main menu.



Many of the execution controls have icons that can be added to the toolbar. Right click the toolbar, click Configure and add the button(s) to the Current buttons.

Step: Single-line Step

Step executes the current line and positions the cursor on the next sequential instruction. The current line is always highlighted and displayed in reverse video.

Use Step in any of these ways:

- Click the **Step icon**.
- Open the Animate menu and select **Step**.
- Pres F11 (MS scheme) or Ctrl+S (MF scheme).

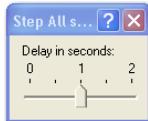
Use single-line Step when you want total flexibility and complete control over logic flow, program statement execution and execution speed. You can also use single-line Step in combination with other Animator techniques

(examine field values, Set/Unset Breakpoints and try out logic before modifying programs, DO statements).

Step All: Multi-line Step

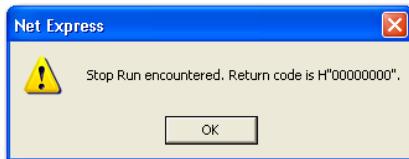
Step All begins executing the current statement and continues to execute each instruction in logical sequence. You can stop by pressing Ctrl+Break. Because Step All moves through the code continually, it is often a good idea to combine it with other techniques that monitor or control program execution i.e., examine and monitor field values or Breakpoints.

To perform a Step All, open the Execute menu and select **Step All**. Use the pop-up slider to control the speed of execution.



Run

Run executes the program at top speed and does not display your source or any Animator visual options (Monitors, lists, etc.) while executing. Run stops only at Breakpoints, GOBACK/STOP RUN and ABEND conditions. Run displays a message box to inform you why execution stopped. Use Run to:



- Execute a complete process ranging from a paragraph to a break point to an entire application.
- Produce output files and screens for inspection and analysis; for example, to demonstrate an application to users.
- Execute subroutines to produce values passed back to a main program (e.g., a data routine).

- Save debug time. You can Run directly to a program error (ABEND condition) or Breakpoints (Combine Run with Breakpoints to execute through logic and get to a line where you begin detailed debugging).

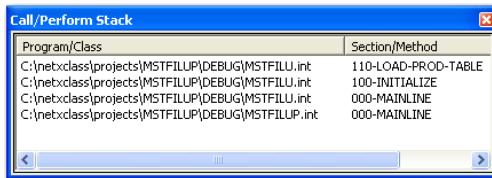
Note: If there is the possibility of your program going into an infinite loop, perform a Step All before a Run.

Perform a Run one of these ways:

- Click the Run push-button.
- Open the Animate menu and select **Run**
- Press **Ctrl + F12**. To prematurely end a Run, press **Ctrl + Pause/Break**.

Run Thru/Run Return with Execute a Perform/CALL Range

Perform/CALL range refers to all COBOL paragraphs, sections and subroutines that are logically contained within the scope of the particular performing/calling statement. Net Express shows the call/perform stack (select View, Dockable Windows, Call/Perform Stack) history of execution flow while executing an application (top line is the most current).



Use Run Thru to quickly execute a section of your program. Run Thru is often used in place of RUN + BREAKPOINTS to:

- Perform initialization logic, such as load tables, open files, etc.
- Execute at run speed through paragraphs/sections that are not crucial to your debugging or are already debugged.

Run Thru

You are positioned on a PERFORM statement. The chain executed by the PERFORM is one you need to execute but not to debug. To speed up the test, perform a Run Thru to execute all of the COBOL statements in a

Perform/CALL range. While executing, your source or any Animator visual options, are not displayed. Run thru stops only at:

- End of current Perform/CALL chain.
- Breakpoints within Perform/CALL range.
- GOBACK/STOP RUN/ABEND conditions (if encountered within the Perform/CALL range).

Perform a Run Thru in one of these ways.

- Click the Run Thru icon.
- Open the Animate menu and select **Run Thru**.
- Press **Ctrl+X or F10**.

Run Return

You have stepped into a routine you need to execute but not debug. To skip the remainder of the chain and speed up the test, perform a Run Return. Run Return executes all remaining COBOL statements in a Perform /Call range.

Run Return stops only at:

- Next Sequential Instruction after end of Perform/CALL chain is reached - (i.e., the NSI past the original Perform/CALL statement).
- Breakpoints within Perform/CALL range.
- GOBACK/STOP RUN/ABEND conditions (if encountered in statements prior to end of Perform chain).

Perform a Run Return in one of these ways.

- Click the **Run Return** push-button.
- Open the **Execute** menu and select **Run Return**.
- Press **Ctrl+C or Shift+F11**.

Note: If you try to Run Return on a PERFORM of a paragraph at the highest logical level in a program (level 1), the Animator displays a warning that the current Perform level = 1 - and will not execute the Perform because there is no higher level to return to.

Run to Cursor

Run to cursor is a one-time breakpoint plus a run. Run to Cursor executes all COBOL statements between the current statement and your cursor focus. While executing Run to Cursor does not display/maintain Animator source-level debugging options.

Run to Cursor stops only at:

- The line your cursor's focus is set on.
- Breakpoints between the current line, and your cursor's focus.
- STOP RUN/GOBACK or ABEND.

Perform a Run to Cursor in one of these ways.

- Click on the line where you want to halt execution.
- Open the Animate menu and select **Run to Cursor**.
- Right-click on a line and select **Run to Cursor**.
- Press **Ctrl+R**.

Use Run to Cursor to quickly execute a group of statements in your program and bypass statements that do not require debugging. Run to Cursor is often used in place of Run+Breakpoints when the statements are only being executed once.

Skip Statement

Skips over and does not execute the current instruction. The execution point is set to the next sequential instruction. If an ABEND stops your execution you can skip over the failing statement. Access the Skip Statement feature as follows:

- Select **Animate, Skip Statement**.

Skip Return

Skips past all remaining statements to the end of the current Perform/CALL chain (similar to Run return). Access the Skip Return feature as follows:

Select Animate, Skip Return.

Skip to Cursor

Skips directly to the location where the cursor focus is re-set without executing any statements. Skip to cursor gives you complete control over the flow of execution. Access the Skip to Cursor features in one of these ways:

- Select **Animate, Skip to Cursor**
- Right-click on a line and select **Skip to Cursor**.

Examining Data Values

During Animation you will often want to inspect or monitor the value of a variable. The selected data values are retained across Animator sessions.



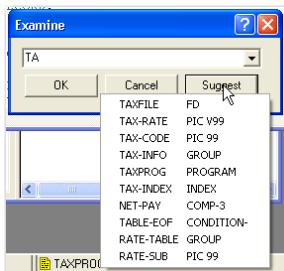
To view the contents of a variable that is on the current screen you may:

- Double-click on the variable name
- Right-click on the variable name and select Examine from the context menu
- Pause the mouse over the variable for a moment (after enabling Options, Animate, Show tool tips to examine data).

To view the contents of a variable not on screen (by keying in the variable name):

Click the Examine pushbutton in the toolbar menu or press Ctrl-E

Type in the variable name or Type in a partial name and select **Suggest** to choose from up to 10 variables that best meet your criteria.



The Examine List Window

The Examine List window shows the current value of a variable and provides an option to alter the values for testing purposes. Follow these steps to change a variable's value.

1. Click the **Change** pushbutton.
2. Type new value(s) in the Change Value window. Hex toggles the view to hex values and back to text. Clear initializes to spaces or zeroes.
3. Press **Apply**.



Follow these steps to create a list of alternative values for a variable.

1. In the Change window, press **More>>**.
2. Type over the current variable value.
3. Press **Add**.
4. Iterate steps 2 and 3 until you have loaded the list.

5. To choose from the alternatives you can press < or > , then press Apply. Show displays the list value when the current value is from the execution of the code.

To monitor a selected variable during Animation, click the Monitor pushbutton in the Examine List window.



Other Options in the Examine List Window

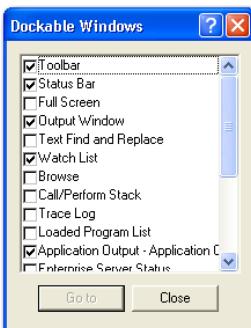
View and/or modify the contents in Hex/Text.

View other variables in the data-item structure.

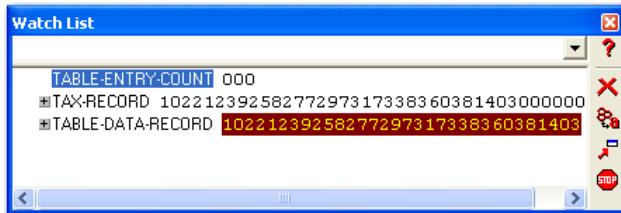
- Press **Parent** to view the group-level of the variable. The small up arrow indicates there is a group level above this variable.
- Press **Zoom In** to view the immediate children of the parent variable. Click on the plus (+) sign.
- Press **Zoom Out** to compress the children back to the immediate parent. Click on the minus (-) sign.

Watch List Window

The Watch List window, aka the Monitor window, allows tracking more than one variable in a single window. The monitor displays the variables and their values. The Watch List window must be first enabled under **View, Dockable Windows**.



To monitor a selected variable during Animation, select **Add to list** to add the variable name to the Watch list.

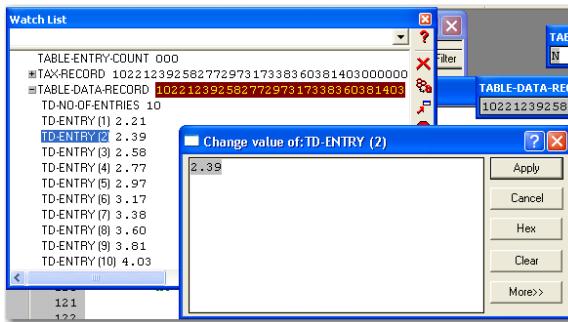


Icons on the Watch List window support the following functions:

- Remove - deletes the variable from the Watch list
- Hex Toggle – toggle between hex and text display
- Monitor – Open a separate monitor window for the selected variable
- Break on Data Change - causes a break in execution when the value of the selected variable changes.

Working with Table Variables

Table variable can be monitored and changed. Click the '+' on the selected table variable to display all the entries. Double click on any table value to change the value.



Breakpoints

Breakpoints are stops in your program that halt Animation on a line before executing the statement on that line. The Prefix area must be toggled on to

see the Breakpoint symbols. (To toggle on the Prefix area, right-click on any line and select Prefix). The four basic types of Breakpoints are:

Normal - Animation **always** halts on the line.

Conditional - Animation only halts on the line if a given condition is true at the time the line is executed.

Do Statement Breakpoints – Execute a special coded statement at the breakpoint.

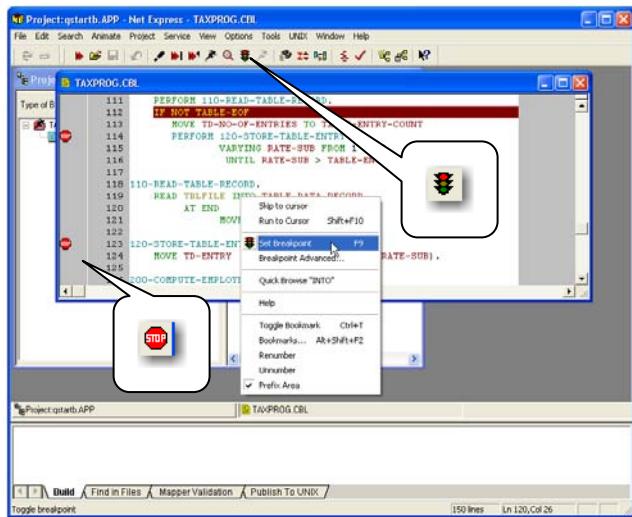
Program-level Breakpoints - Animation stops at the entry-point of a Called program.

Break on Data Change from the Watch List - Animation stops when the data item in the watch list changes value.

Normal Breakpoints

To set a Normal Breakpoint (Stop Sign Icon in Prefix Area) use one of the following options:

1. Double-click **in the prefix area** (the column on the left side of your screen) of the statement you wish to stop on.

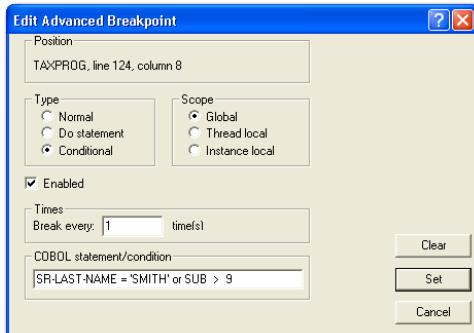


2. Position your cursor mouse pointer on the line you wish to break on; select **Animate**, **Breakpoint**, and toggle Set
3. Position your cursor mouse pointer on the line you wish to break on; click the **Breakpoint Icon** on the toolbar
4. Right-click anywhere in the line (including the prefix area) and select **Breakpoint**.

Conditional Breakpoints

Use Advanced Breakpoints to set Conditional Breakpoints that specify under what conditions you want Animation to stop on the breakpoint line. All are tied to a selected statement (except program breakpoints).

1. Position your pointer on the line on which you want to stop.
2. Right-click; select **Animate**, **Breakpoint**, **Set Advanced**.
3. Press the **Conditional** radio button.
4. Enter the condition that, when true, will cause the break. For example, SR-LAST-NAME = 'SMITH' or SUB > 9. **Note:** Do not enter the IF.
5. Click **Set**.



Do Breakpoints

Use Advanced Breakpoints to Execute a COBOL statement before a line is executed

1. Open the **Set Advanced Breakpoint window**, click **DO**.

2. Type a valid COBOL statement into the Parameter box, click **Set**. **Note:** Execution does not stop on the line where the statement Breakpoint is set. Instead, your COBOL statement is executed before the statement on the Breakpoint line.

Program Breakpoints

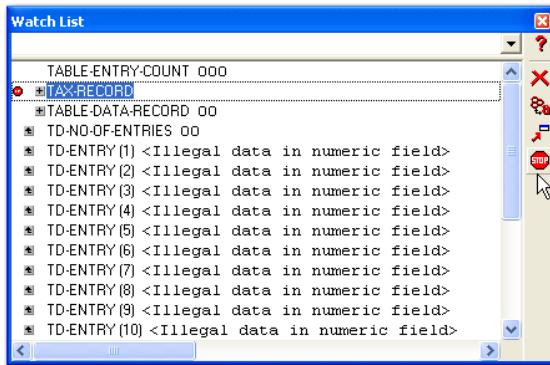
Follow these steps to set a Breakpoint at the entry-point of a called program.

1. Select **Animate, Breakpoint, Program**.
2. Key in the Program Name; click **Set**.

You can enable (check) or disable program breakpoints.

Break on Data Change from the Watch List

This option will halt Animation when a Break on Data Change is set from the Watchlist. From the open Watchlist, click the breakpoint icon.



Controlling Breakpoints

Locate and Cycle through Breakpoints

- Right-click on a line that has a Breakpoint, and select **Compress**.
- Select **View, Compress, Breakpoints**.

```

B TAXPROG.CBL
-----top-of-text-----
12.3 120-STORE-TABLE-ENTRY.
12.7    | PERFORM 210-READ-EMPLOYEE-RECORD.
13.2    | COMPUTE NET-PAY ROUNDED = (EH-HOURS-WORKED *
|   end-of-text-

```

- Using the MF.Key scheme, press **F7** (cycle forward), or **F8** (cycle backwards).
- Select **View, Next, Breakpoint** or **View, Previous, Breakpoint**.
- Select **Animate, Breakpoint, Show All**. All breakpoints within the project are listed.



Times, Break every n Times a Line is executed

Times works with Normal, Do statement, and Conditional breakpoints.

1. Right-click cursor on **line where breakpoint is to occur**.
2. Select **Breakpoint Advanced**.
3. Select **Normal**. Type a number between 001 and 256 in the Times: box.
4. Click **Set**.

Clearing/Disabling Breakpoints

Clearing means removing a breakpoint permanently. But you may want to leave the breakpoints for future testing, while disabling them for an integration test. To clear all local Breakpoints (i.e., those with a breakpoint indicator):

- Select **Animate, Breakpoint, Clear all**.

You can also clear individual breakpoints or clear all from:

- **Animate, Breakpoint, Show all.** Select either the Statement Breakpoints tab or the Program Breakpoint tab. In addition, you can clear, enable, or disable breakpoints throughout the entire project. Enabling/disabling does not affect DO statement breakpoints since they do not halt execution.

To Unset a single Breakpoint, use one of the following options:

- Double-click your mouse on the **Breakpoint symbol** in the left margin column
- Set your cursor mouse pointer on the line to be unset; select **Animate, Breakpoint**, and toggle **Set**
- Position your cursor mouse pointer on the line you wish to unset the break, click the **Breakpoint Icon** on the toolbar
- Right-click on a line, and select **Remove** (or **Disable**) to disable a breakpoint without removing it; disabled breakpoints are retained across sessions).
- Position your mouse cursor pointer on the line you wish to set/unset, click the line, then press **F9**.

Restart

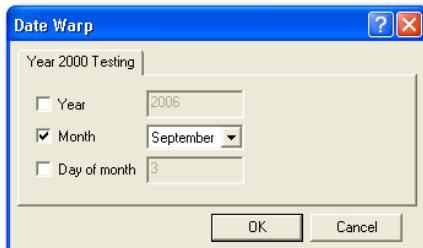
Completely resets execution to the entry-point of your program.

- Closes all opened files.
- Does not back out file changes from current run.
- Re-initializes WORKING STORAGE variables. For example, returning value clauses back to their original state, unvalued variables to compiler defaults, and resets Perform stacks.

To Restart an application select **Animate, Restart Animation**. You can set keys to perform a Restart. See the Appendix for Keyboard Configuration.

DateWarp

DateWarp is used to modify the internal Animator run date. While it was originally intended to enable Y2k testing, the issue of testing programs across date boundaries (end of year, end of month, end of business cycle) remains valid even after Y2k. DateWarp has no effect on the system date of your computer. **Animate, DateWarp** is available once you start Animating a program.



1. Select **Animate, DateWarp**, click on the Year / Month / Day of Month checkboxes and enter the desired date.
2. Execute the program.

Do Statement

The Do Statement facility allows you to enter and execute a COBOL statement during Animation. Use a Do statement to:

- Ease unit testing.
- Perform any paragraph in your program.
- Open and close files.
- Initialize or modify the value of fields not on the current screen.

To access the Do statement facility follow these steps. Note: Do Statement is only supported in .INT execution.



1. Click the cursor on the line of interest.

2. Select **Animate, Do** statement.
3. Type in your statement and click **OK**. The statement is compiled and executed immediately. After the Do statement is executed, you may continue Animating. If you would like to re-execute the code, click on the line of interest and select **Animate, Do statement** again. You can execute the Do statement on any line of code. The code is NOT added to your program. It is just executed in the Animator.

Integration Testing

By combining .INT (intermediate compiled) with .GNT (generated) versions of programs, you can effectively:

- Create data file(s) at top speed orders of magnitude faster.
 - Test interactive applications without viewing source (e.g., DISPLAY/ACCEPT, Dialog System).
1. Compile all programs to .GNT
 2. Right-click on **.CBL** to be generated.
 3. Select **Build Settings...**
 4. Click on the **Compile** tab; select **Generated Code**.
 5. Compile the **program**.
 6. Erase **programname.INT**.
 7. Create a batch file and execute it at the Net Express operating system prompt.

For example, create a batch file in the \debug directory named TIMELIST_BATCH.CMD:

RUN TIMELIST

Execute it at the Net Express operating system prompt.

Notes: The same example when executed with .INTs takes about .05 seconds. The file assignment was done using fixed file assignment syntax (assign xyz). You can also use external file assignment (assign xyz). Certain project and debug functions are available from the command prompt. For

more information, open the Net Express Command Prompt (**Start, Programs, Net Express, Net Express Command Prompt**) and enter **MFNETX /?**.

View Menu Animator Options

The following options of the View menu are very useful when Animating and editing in the Animator on-the-fly.

Align—Move the source view so the current line is the third line from the top of the window.

Where—Position the cursor at the current executable line; very useful when scrolling/debugging large programs.

Last Edit Position—Move the cursor to the last line that was edited.

Hide All Copy Files—Hides all copy files (only the COPY statements show after the program has been compiled).

Full Screen—Expands the Net Express Desktop to full screen. To return, press the image in the center of the floating icon. Warning: Do not close the floating icon.

Animator Sub-Windows

The following are all sub-windows of the Edit/Animate functions.

Toolbar	toggle on/off
Status Bar	toggle on/off
Output	toggle on/off
Find and Replace	toggle on/off
Browse	toggle on/off
Watch List	toggle on/off
Call/Perform stack	toggle on/off
Application Output	toggle on/off

Loaded Program List	toggle on/off
---------------------	---------------

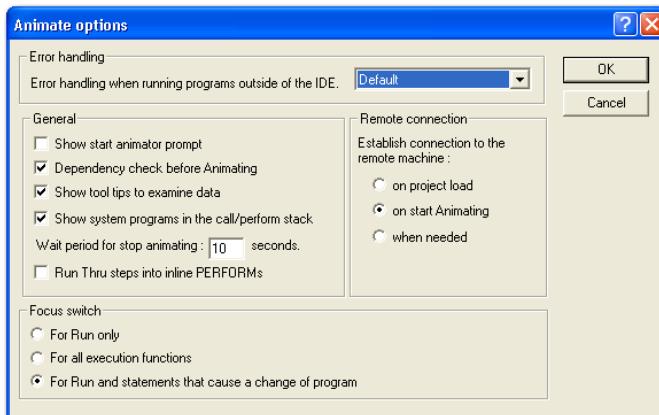
Options, Animate Menu

The following options should be considered for Error handling when executing outside the IDE.

Default—No debug screen shown unless animation is specifically chosen.

Enable Just-in-time animation—If a program ABENDS while executing outside the Animator (e.g., at an operating system prompt), start Net Express, open a project, and load the program for debugging.

Enable Coredump—Transfer the CBLCORE-file to your project directory and start Net Express, select **Animate, Start Animate**, enter CBLCORE. You may watch the variables and the call/perform stack from the moment, when the runtime error occurs. **Note:** to enable the CBLCORE-dump, start the **cblcored.exe** on the PC, where you want to create a CBLCORE.”



Module Summary

The Animator provides options to manage the debug process, examine variable content, and analyze COBOL logic.

You may control execution speed of your program during animation.

Breakpoints are available to stop or pause the execution of your program as necessary during animation.

Exercises and Review Material

Combining Animation Techniques to Solve Problems -

Run/Monitors/Step/Restart

Most real-life debugging is accomplished through a combination of Animation techniques. After you've used Animator for awhile, you'll develop your own style, preferences, and usage practices for applying the facilities and Animation techniques you learned in this chapter to unique problems in your job.

A general set of guidelines or recommendations for unit testing with Animator many people have found useful are the following.

1. Edit/Compile the program
2. Unit test
3. Begin with a program Run ... (unless you suspect the problem lies at or near the beginning of the code, or is in an infinite loop), in which case use Step, or Step all
4. Execute your program to:
 - **EOJ** - Edit/browse any output data files written by your program to verify results with the Data File Editor
 - **Breakpoints** - strategically placed in your program to setup detailed-debugging
 - **An ABEND** - Analyze the type of problem or Run Time Error that occurred (bad data, bad subscript, infinite loop, etc.)
5. Re-Animate your application.
6. Combine Monitors/Stepping/Skipping, etc. to do detail-level-debugging and COBOL logic research and analysis.
7. Once you believe you understand and can solve the problem, simply:
 - Edit the source within the Animator
 - Compile the program; loop back to unit test and verify your fix.

8.1 Review Questions

Answer the following questions about the Net Express Animator.

1. What types of files used by Animator are required prior to testing your program?
2. What is meant by the term Perform Range?
3. What is the difference between Run, Run Through, and Run Return?
4. What is the difference between Skip to Cursor, and Run to Cursor?
5. How do you inspect and/or modify the contents of data items during Animation?
6. How do you view the values in a table?
7. What is the difference between Normal and Conditional Breakpoints?
8. What is a Program Breakpoint?

8.2 Step All, Examine, Do, Locate

1. Open the **CALLTEST Project** and load the COBOL program **CALLTEST**.
2. Use the Step All function to view the execution of this code. There will be several errors in the execution of the code caused by uninitialized data items.
3. Use **EXAMINE** to change the current value to zero for all of these items to allow the program to continue execution.
4. Restart the application, and this time use Do statement to execute the statement INITIALIZE PROGRAM-COUNTERS. This will now allow successful use of the counters. Execute the program to the end.
5. To fix the problem permanently, use the Locate function to find the definition of these items. Add a Value 0 clause to each item to initialize all three counters to zero.
6. Recompile the program CALLTEST and ensure that it executes successfully to STOP RUN.

8. 3 Breakpoint, Locate

Introduction

CALLTEST.CBL's error routine for CUST-STATUS needs to be tested.

1. Place a breakpoint on the line that tests for **VALID-STATUS**. Run to the breakpoint.
2. Locate the definition for **VALID-STATUS**.
3. Change the current value of the appropriate 05 level to an invalid status (the possible values are documented in the code). Return to the statement that caused the breakpoint.
4. Use Step to ensure that the error routine ERROR-ROUTINE-STATUS executes correctly.
5. Unset the breakpoint if continuing to the next exercise.

8.4 Program Breakpoint, Skip to Start

1. The called program, DATETEST, needs to be debugged. Load Calltest.cbl. Use the program breakpoint function to stop the execution when DATETEST is called.
2. Within DATETEST, use the Examine function to change the value of MONTH and test that each of the EVALUATE conditions is correct and that there are no incorrect conditions. Step to GOBACK. Use Skip to Cursor Position (pointing to the first line of the Procedure Division) to restart DATETEST between each test with a different value for MONTH.
3. You can also experiment with **Animator**, **DateWarp** to reset the Animator run date.

8.5 Control-Break Problem

Note: For the rest of the exercises in this module, you will need to create a new project. Name it FIXBOMBS and add all the BOMBnn.cbl programs from \IDEPROGS. All the required input files are statically assigned to \netxclass\DATAFILE. If you have your class exercises in a different place, you will need to modify the program SELECT statements. Note that this is also the folder where output files will be created.

BOMB10 is a control break/report program that reads a single input file containing Student-Record (record type 1)- personal information and Course-Record (record type 2)- course information of the preceding record. After BOMB10 reads a student record, it rolls up course credits for each student and writes output records describing the student's course/credits for a semester. The correct report should read:

S T U D E N T C R E D I T S R E P O R T

STUDENT NAME	COURSES	CREDITS
JOE SMITH	3	9
SAM SHULTZ	2	6
AL JONES	5	13
SALLY SWELL	3	9
HAROLD HARDWORK	3	9

TOTAL STUDENTS PROCESSED IS: 5

BOMB10 currently produces.

S T U D E N T C R E D I T S R E P O R T

STUDENT NAME	COURSES	CREDITS
JOE SMITH		
JOE SMITH	3	9
SAM SHULTZ	2	15

AL JONES	5	28
SALLY SWELL	3	37
TOTAL STUDENTS PROCESSED IS:		6

Look over both reports above. BOMB10's code currently generates the following problems.

- CREDITS get rolled up, instead of being reset for each student.
- The control break logic/report writing logic is incorrect. Joe appears twice!
- The total number of students is incorrectly calculated.

Run to EOJ - View/Analyze Output Results

1. Edit BOMB10. To see the output you have a couple of choices:
 - Add DISPLAY statements before the execution of every WRITE statement in the program.
 - Edit the output file: **File** menu, **Open**, BOMB10.OUT once you have reached STOP RUN.
2. Compile BOMB10.
3. Run the program to EOJ.

Fix the Problem

Isolate the problem areas and modify BOMB10's logic. You will need to add:

- Initialization logic for accumulated credits.
 - Another initial READ to get past the TYPE-1 logic (which is causing the print of 2 JOE SMITH records).
 - Logic to force out the last record (for Harold Hardwork).
1. Compile and retest your fixes.
 2. View BOMB10.OUT in the editor.

8.6 Putting It All Together: Solve Program ABENDs

Introduction

This Exercise uses a number of BOMBnn.CBL programs. These programs contain one or more bugs that cause run-time errors (ABENDs). Solve the BOMBnn program ABENDs. Use the Animation facilities and techniques you've learned in this chapter to research the problem, and to modify and correct the coding mistakes. **Hint:** Set up a separate project for each program. The programs are in ...\\DEPROGS

Later exercises in this section contain detailed walk-thrus and sample solutions for the BOMBnn programs. If you have any difficulty understanding and fixing the programs, **or** if you simply would like to learn perhaps a different approach to solving the ABENDs in the BOMBnn programs you may:

- Skim through the instructions/solution in your book for hints
- Follow the step-by-step approach for each BOMBnn program.

BOMB02.CBL

Problem: Program fails with Run Time Error 163 - illegal characters in numeric field.

Prerequisites: To perform this exercise you need an understanding of COBOL internal data representation (i.e., the difference between GROUP, DISPLAY and COMP-3 fields) and MOVE statements. Data item initialization (**VALUE**) coding.

Recommended Animation facilities and techniques:

- Step All to error condition
- Restart application
- Monitor/Watch the variable(s) that cause the abend
- Step - view the contents of the Monitor/Watch window

BOMB03E.CBL

Problem: Program goes into an infinite loop (actually, a forced abend prevents it being infinite).

Prerequisites: To perform this exercise you need an understanding of COBOL file processing techniques, and end-of-file conditions

Recommended Animation facilities and techniques:

- **Step** - view program logic flow
- Watch List
- Restart application

BOMB06

Problem: Bomb06 has multiple bad data errors and a fall through error. This program is supposed to process all records in the input file. Records with invalid data (e.g., blanks in NUMERIC fields) should be DISPLAYed, and bypassed.

Prerequisites: An understanding of COBOL internal data representation (i.e., the difference between **GROUP**, **DISPLAY** and **COMP-3** fields in **MOVE** statements). The effect **GO TO** has on paragraph **Perform** Chains.

Suggested Animation facilities and techniques:

- **Run** to all error conditions
- Restart
- **Monitor/Watch** the variable(s) that cause the abend
- **Step** - view the contents of the **Monitor/Watch** window

BOMB09

Problem: Table subscript errors.

Prerequisites: An understanding of COBOL internal table handling and numeric truncation

Suggested Animation facilities and techniques:

- **Run** to error condition
- Restart application
- **Monitor/Watch** the variable(s) the subscript that causes the table overflow
- Conditional Breakpoints
- Step

Note: The problem is not completely solved until a list of all the names in last-name alphabetic sequence is displayed.

8.7 Step through a Program and Examine Data-Items

Introduction

The BOMB02 program has a logic error that causes it to abend with a Run-Time Error 163. Your assignment – either use the techniques presented in this chapter and solve the problem (i.e., unassisted) or follow the step-by-step walk-thru.

Combining Step and Watch Lists to Debug Invalid Data Abend

1. Compile **BOMB02**. Load it into the Animator.
2. Single Step through **BOMB02** until you get the **Run Time Error 163**
- Illegal characters in numeric field message.
3. Close the Net Express error window.
4. Your cursor will be on the line that produced the bad data condition.
5. Double-click the **CTR-EMPLOYEES** field. This opens an Examine window on the contents of **CTR-EMPLOYEES**.
? What value does **CTR-EMPLOYEES** contain? View the data in hex (x'3030').
? What is wrong with this data? (+ Hint: If you don't know, follow this example step-by-step through the solution. If you do know, solve the problem and go on to the next Exercise).

Restart, Step and Watch/Monitor lists to analyze the problem

Reminder: As mentioned during the chapter, **Step** is useful for meticulous, comprehensive debugging, because it gives you complete flexibility in the problem-solving techniques you apply during Animation.

1. Restart BOMB02 (Select Animate, Restart application).
(Background/COBOL explanation): **CTR-EMPLOYEES** is an unsigned, PIC 9(3) COMP-3 field that contained non-numeric data, (a two-byte COMP-3 field that contains valid zeros should be represented internally as x'**00 0F** -

not x'30 30), which is why the ADD statement failed. Use **Step** to see how it obtained this value.

2. Before you begin Stepping, double-click CTR-EMPLOYEES, and select Add to list so that you can follow the values as they change. Note that CTR-EMPLOYEES is initialized to Spaces (x'20 20).
3. Step until you reach the statement “MOVE ZEROS TO NUMERIC-WORK-AREAS.”
4. Step once more. Note that the monitor changes and shows x'30 30.
5. Continue to Step until you reach the Run Time Error condition. Notice that the value has not changed since the “MOVE ZEROS...” above.

Fix BOMB02 during Testing using the Integrated Editor

1. Right-click on CTR-EMPLOYEES, and select Locate definition. Notice NUMERIC-WORK-AREAS is the group item for CTR-EMPLOYEES.
(Background/COBOL explanation): The statement MOVE ZEROS TO NUMERIC-WORK-AREAS is invalid because it is a group move, and therefore propagates the USAGE DISPLAY zeros (x'30 30) into COMP-3 elementary fields instead of valid COMP-3 numeric data. To fix this problem, modify BOMB02’s code:
2. Remove the existing MOVE ZEROS TO NUMERIC WORK-AREAS; code MOVE statements to zero out each of the elementary fields in the INIT-ROUTINE paragraph ...or...add a VALUE 0 clause to each of the elementary fields in the NUMERIC-WORK-AREAS group ...or... use INITIALIZE NUMERIC-WORK-AREAS
3. When you have made your changes, recompile, Animate and single step through the program until you are past the original error. Then Run the program to completion and exit Animator.

8.8 Using Step

Introduction

The **BOMB03E** program has a logic error that causes it to go into an infinite loop. Use the techniques presented in this chapter and solve the problem **or** follow the step-by-step walk-thru.

Combine Step with Watch/Monitor lists

1. Compile **BOMB03E** and load it into the Animator.
2. Select **View, Call/Perform Stack**. You are positioned on the PERFORM INIT-ROUTINE statement. Press the Run Thru icon. You are positioned on the PERFORM PROCESS-EACH-RECORD statement.
3. Press **Run Thru**. The program loops and takes the forced abend. So, the problem is in PROCESS-EACH RECORD. Notice in the Call/Perform Stack that the chain of execution leading to the abend was as follows: TOP-LEVEL, PROCESS-EACH-RECORD, then ABEND-ROUTINE.
4. Close the **Call/Perform Stack**.
5. Restart Animation (Animate, Restart Animation).
6. Double-click:
ER-FIRST-NAME, and Add to list
ER-LAST-NAME, and Add to list
ER-SOC-SEC-NUM, and Add to list
7. Step through the program (if you have set up your MF.SET for the keyboard, use Ctrl + S. Note the values in the Monitor/Watch List. Do they change? Step allows you to see that the infinite loop occurs in the PROCESS-EACH-RECORD paragraph.
8. Close the Monitor/Watch List.

Use Examine and Find/Compress in Animator

Introduction

Now that you know that the infinite loop occurs in the **PROCESS-EACH-RECORD** paragraph, you need to figure out why. Combine the Examine, Find, and Compress techniques (from the Edit chapter) to help analyze what you've discovered so far.

Quick Browse – A New Technique

1. Right-click the paragraph label **PROCESS-EACH-RECORD** and select **Quick Browse, and Compress**.
2. From the compressed view, you can see that PROCESS-EACH-RECORD executes until END-OF-DATA occurs. We need to find out why the END-OF-DATA condition is never reached. Quick Browse is discussed further in the Analyzing chapter.

Use Quick Browse to Analyze the Logic Flow

1. Right-click on **END-OF-DATA** and select **Quick Browse**. Notice that END-OF-DATA is an 88-level that tests true when a 'Y' is moved to its elementary-level field.
2. To find the elementary level field - using **Quick Browse** Do a Quick Browse on the group level: SW-END-OF-DATA. Press the **compress** icon.

Several things become apparent from this view. **SW-END-OF-DATA** is the field that must contain a "Y" for **PROCESS-EACH-RECORD** to break out of the Perform loop ...and... **SW-END-OF-DATA** is modified by the **AT END** condition. So we suspect that the **AT END** condition never occurs.

1. Let's investigate this. Position your mouse-pointer over the AT END MOVE 'Y' TO SW-END-OF-DATA line in the compressed view. Right-click and select Expand. The READ...AT END occurs in the READ-ROUTINE paragraph. Let's find out what performs READ-ROUTINE.

Use Quick Browse to Visualize a Paragraph's Logic

1. Right-click the **READ-ROUTINE paragraph label** and select **Quick Browse**.
2. Use compress and expand to see that READ-ROUTINE is only executed from INIT-ROUTINE.
3. Use the same technique to analyze INIT-ROUTINE. Note that INIT-ROUTINE is only executed from TOP-LEVEL. So we have found that PROCESS-EACH-RECORD loops because the READ-ROUTINE paragraph is only executed once - so we never get to an AT END condition.

Fix BOMB03E

1. Add PERFORM READ-ROUTINE at the end of the PROCESS-EACH-RECORD paragraph.
2. Re-compile BOMB03E.
3. Monitor (double-click) and select Add to list:
ER-FIRST-NAME
ER-LAST-NAME
ER-SOC-SEC-NUM
4. Step all to BOMB03E's STOP RUN. Close your Animator session.

8.9 Solving Multiple Errors

Introduction

BOMB06 is a reporting program that reads a single input file, does some calculations, and writes an output report file passed to a print utility. Unfortunately, the programmer working on some updates to BOMB06 recently got re-org'd into another division, and you've been asked to finish the work, which is in the early stages of Unit-Test. You've been told that there could be incomplete logic, and several outstanding bugs.

Work on this exercise in one of two ways.

- Go through the step-by-step following the Walkthrus.
- Simply use the Animator techniques presented in this chapter.

Prepare BOMB06 for Animation/RUN to first program error

1. Compile BOMB06. **Load** and **Run** BOMB06. Explanation: BOMB06's first error is a 163 Illegal character in numeric field. The COMPUTE statement in which BOMB06 ABENDs refers to **ER-HOURS-WORKED** and **ER-PAY-RATE**.

Analyze Illegal Character Problems with Examine

1. (Double Click) – **Examine in HEX** - the contents of both these fields - x"20" – ASCII spaces. How did blanks get into numeric fields?
2. Right-click on **ER-PAY-RATE**, and **Locate** its definition. Scroll up in the source, and you will see that these fields are part of the input file.
Assuming that the program is supposed to edit out bad input data, you will need to add an IF statement to the logic.
3. Right-click and select **Return**.

Edit BOMB06 - Compile and Step all to next error

1. Change the COMPUTE statement program logic to read:
IF ER-HOURS-WORKED NUMERIC AND ER-PAY-RATE NUMERIC
THEN
COMPUTE WEEKLY-PAY = ER-HOURS-WORKED * ER-PAY-RATE

```
ELSE  
MOVE ZERO TO WEEKLY-PAY.
```

2. Re-compile **BOMB06**.
3. **Examine** and **Add to list** ER-HOURS-WORKED and ER-PAY-RATE.
4. **Step all** through BOMB06 until the Run Time Error. Explanation:
Another 163 Run Time Error, a MOVE of ER-SOC-SEC-NUM to OR-SOC-SEC-NUM.

Find/Analyze and fix the Data Exception Error

1. **Right-click** over **OR-SOC-SEC-NUM**, and **Locate** its definition. Note that it has a PIC 9(9) clause.
2. Right-click and select **Return**
3. Double-click **ER-SOC-SEC-NUM** to **Examine** its contents. More blanks - not acceptable in a numeric MOVE.
4. Change the MOVE statement, and add another IF test to ensure that **ER-SOC-SEC-NUM** is numeric - if not, move zeros to **OR-SOC-SEC-NUM**

Re-compile/setup the next unit test - Step all to error

1. Re-compile BOMB06.
2. **Step all**. This time the program fails with a Run Time Error 141 File already open. Background/COBOL explanation: As we have already seen several times in this course, errors such as this are often caused by program logic (sequence) problems and, especially, fall-through.

Use Call/Perform Stack to Follow the Logic

1. Select **View, Call/Perform Stack**
2. Re-start **BOMB06**. The Call/Perform Stack shows that the statement to be executed is in TOP-LEVEL.
3. Watch the Call/Perform Stack window carefully. Press the **Step icon**. TOP-LEVEL has passed control to INIT-ROUTINE.

4. Select **Animate, Step All**. What happens in the Call/Perform Stack?

The program flows from TOP-LEVEL to PROCESS-EACH-RECORD, but instead of returning to TOP-LEVEL it falls through to INIT-ROUTINE.

Carefully read all statements for an understanding of the branching logic.

Look for possible fall-through situations. Look carefully at the READ-ROUTINE paragraph. Can you see that:

READ-ROUTINE is not Performed, but instead is arrived at via GO TO (**GO TO READ-ROUTINE**)?

Before end-of-file is reached, the IF NOT END-OF-DATA condition branches back up into PROCESS-EACH-RECORD with another GO TO?

But - upon end-of-file the IF NOT END-OF-DATA condition tests false - so execution simply falls into INIT-ROUTINE?

Fix the Sequence Error, Re-compile, Run to EOJ

1. Close the **Backtrack menu** and solve this error, by changing this statement.

GO TO READ-ROUTINE (in PROCESS-EACH-RECORD)

...to...

PERFORM READ-ROUTINE

2. Comment out the **GO TO** in **READ-ROUTINE** (don't forget the punctuation!).
3. Compile and Run the program to successful completion.
4. Close **the Animator**. Stop Animating.

8.10 Using Breakpoints

Introduction

BOMB09 is a program that produces a sorted external file that is used in various reporting and decision-support applications. Business analysts have questioned BOMB09's output for some time (you will too - when you see the output). But the immediate concern is that BOMB09 was pulled from production over the weekend when it failed on a subscript out of range error. BOMB09 actually contains several bugs. Let's investigate.

Details

1. Compile (if not yet compiled), then load **BOMB09** into the Animator.
2. Run **BOMB09**. BOMB09's first problem occurs in the PROCESS-EACH-RECORD paragraph. It is a 153 Subscript out range error. The statement BOMB09 dies on references ER-FIRST-NAME - and, TB-FIRST-NAME (SUB).

Examine Subscript Error Fields

1. Examine **TB-FIRST-NAME**; it is set to occurrence (0). Right-click **TB-FIRST-NAME** and locate its definition. How many times does it occur? (100).
2. If you **Examine** the contents of **SUB** you will see zeros. This is a problem. How did zeros get into a table subscript? Let's look at the use of SUB throughout the program.
3. Select **Search, Quick Browse** for SUB. Press the **Compress** icon. (**Note**) There are quite a few references to SUB throughout your program. But the very first reference to SUB is a MOVE +0 TO SUB statement.
4. Right-click this line, and click **Expand**. This shows that the MOVE +0 occurs before the Perform of PROCESS-EACH-RECORD.
5. Right-click on **PROCESS-EACH-RECORD** and select **Locate** so that you can see the PROCESS-EACH-RECORD paragraph.

What is the first COBOL statement in the paragraph? (**Add +1 to SUB**). Since the **153** error occurs in this paragraph, it doesn't look like MOVE +0 is the problem - or any other obvious COBOL logic exposed by **Quick Browse**. In fact, we've ruled out this category of problem. Let's use a combination of techniques to continue analyzing SUB as it is used throughout BOMB09.

Use Step/Examine/Restart/Watch lists to understand what's happening to SUB

Examine the contents of SUB as it executes step-by-step.

1. **Restart** the Application.
2. **Step** SLOWLY line-by-line through the code until PERFORM PROCESS-EACH-RECORD is the current line.
3. **Examine** SUB in Hex (so you can see the signed data). Add **SUB** to list.
4. Continue to **Step** through BOMB09 line-by-line, watching the contents of **SUB** change until **SUB = +5**. (**Analysis**) After each successful READ, **SUB** is incremented by the **ADD** statement in PROCESS-EACH-RECORD. Animating step-by-step through a problem like this *could take awhile*. Recall that to execute up to a given point in your program use Run combined with Breakpoints., and see if that gets us any closer to the problem.
5. Close the Monitor/Watch List.
6. Restart the Application.

Run through 50 iterations of PROCESS-EACH-RECORD without stopping (Advanced Breakpoints)

Setting a Conditional Breakpoint - Since the subscript problem occurs when after SUB equals zero, set a Conditional Breakpoint on the ADD line.

1. Scroll down to the PROCESS-EACH-RECORD paragraph.
2. Right-click your mouse in the **ADD +1 to SUB**. Line.

3. Select **Breakpoint Advanced...**
4. Type **50** in the Every box
5. Click **Set** - note the Breakpoint STOP sign.

Recall that we did a **Locate** on TB-FIRST-NAME. We saw it OCCURS 100 times.

6. Run BOMB09 - When the ADD line is executed 50 times, the Breakpoint takes effect.

Set a Conditional Breakpoint on the ADD line.

1. Scroll down to the PROCESS-EACH-RECORD paragraph.
2. Right-click in the ADD +1 to SUB. line
3. Remove the (every...) breakpoint.
4. Select **Breakpoint Advanced...**, and select **Conditional**.
5. Type **SUB > +98** in the COBOL Statement box
6. Click **Set** - note the Conditional STOP icon.
7. Run BOMB09 - at the next Breakpoint return to your source where SUB is now at +99. So far, so good.

Combine Step/Locate/Editing to fix BOMB09's 1st problem

1. Step until SUB increments from 99+ to 00+. I think we've isolated the problem.
2. Right-click **SUB**, and Locate the definition of **SUB**. SUB is defined as a two-byte signed-numeric field. When +1 is added to +99 it simply rolls over to +00.
3. Change SUB's picture clause to: PIC S9(4). Re-compile BOMB09.
4. Run the program. Did it finish successfully? Hint: see the next Exercise.

8.11 Analyze & Solve Subscript-out-of-bounds Errors

Even though you've fixed one error, BOMB09 contains several more bugs.
Use **Examine** and **Locate** on TB-FIRST-NAME (subscript out of range) and
on SUB (+101) to solve the next Subscript out of range error.

Close the **Animator**.

8.12a Invalid Output Errors

The code in BOMB09 produces two levels of invalid output data. To view the first level of invalid output data follow these steps.

1. Select **File, Open**. Open **BOMB09.OUT** in ...\\Datafile (hint: use *.* as Files of type...) This is the output report from executing BOMB09 to STOP RUN. Once the file is loaded, you will see that only one data record has been written to this file (**unless you've already fixed this problem during your analysis of BOMB09's previous errors**). Recall from the first BOMB09 bug that there are over 100 input records.
2. Close the file.

8.12b Optional Challenge

1. Reopen the project (select File, Recent Projects).
2. Animate BOMB09.
3. Find and fix the problem. Hint: Another subscript problem -- what should the limit be for the number of records written to the output file?

8.13 Using Run to Create an Output File

Background: uses for program RUN

By running your program, you execute the code at maximum speed. You do not see your COBOL statements as Run executes, nor do you see the results of monitors. Run is typically used in debugging to:

- Produce output files that are then browsed and analyzed
- To proceed directly to an ABEND condition
- To execute a large section of code in order to set up detailed Animation using other debugging techniques (such as Breakpoints)

This exercise does not require you to do any debugging. Simply note the use of RUN in debugging situations where you need to produce output files.

Load, Compile, and Run BOMB10

1. Load **BOMB10** into the Animator and compile it.
2. Click the **Run** push-button on the bottom toolbar. This will execute BOMB10, which runs to STOP RUN successfully (but, as we'll soon see, does not produce the correct output). Close the Animator, Stop Animating.

8.14 Run thru and Run return

1. Animate **BOMB10** - your cursor will be positioned on the PERFORM 100-INITIALIZATION statement. If you didn't need to see and debug any of the code in this paragraph, you could run through it. **Run through** and **Run return** allow you to execute quickly through large paragraphs and sections of COBOL logic, while maintaining control over your Animate session. **Run through** executes (in Run mode) all of the remaining statements in a Perform chain (which includes all paragraphs and subparagraphs of the "current paragraph").
2. Click the **Run through** icon. This will execute all of the statements in 100-INITIALIZATION, and position your cursor on the next sequential instruction after the Perform. To verify that all of the code in 100-INITIALIZATION was executed, double-click (Examine) SR-NAME (it contains values) and STUDENT-FILE (it has been opened).
3. Click the **Run through** icon again. This will execute all of the statements 200-PROCESS-RECORDS. To verify that all of the code in 200-PROCESS-RECORDS was executed, double-click END-OF-DATA (all records have been Read and the end-of-file condition has been set).
4. Click the **Run through** icon one more time. This will execute all of the statements 300-WRAP-UP. To verify that all of the code in 300-WRAP-UP was executed, double-click STUDENT-FILE (closed), and GRAND-TOTAL-LINE (need to scroll to the end of your source).
5. Select **Animate, Restart application**. **Run return** executes (in Run mode) all of the remaining statements inside a Perform to the end of the Perform range and then returns control to the next sequential instruction on the statement directly below the original Perform.
6. **Step** on the PERFORM 100-INITIALIZATION line. This will take you to the OPEN INPUT STUDENT-FILE line in 100-INITIALIZATION.
7. Click the **Run return** icon. This will execute all of the remaining statements in the 100-INITIALIZATION Perform range - including the Performs of 211-PAGE-CHANGE-RTN and 230-READ-A-RECORD. Notice where your cursor is. To verify that all of the code in 100-

INITIALIZATION was executed, double-click SR-NAME (it contains values) and STUDENT-FILE (it has been opened).

8. **Step** once while on the PERFORM 200-PROCESS-RECORDS line. This will take you to the `IF SR-RECORD-TYPE IS EQUAL TO '1'` statement in 200-PROCESS-RECORDS.
9. Click the **Run return** icon again. This will execute all of the statements 200-PROCESS-RECORDS. To verify that all of the code in 200-PROCESS-RECORDS was executed, double-click END-OF-DATA (all records have been Read and the end-of-file condition has been set).
10. **Step** once while on the PERFORM 300-WRAP-UP line. This will take you to the MOVE CTR-STUDENTS TO GTL-STUDENT-LINE statement in 300-WRAP-UP.
11. Click the **Run return** icon one last time - this will execute all of the statements in 300-WRAP-UP and position your cursor on the GOBACK instruction. To verify that all of the code in 300-WRAP-UP was executed, double-click STUDENT-FILE (closed), and GRAND-TOTAL-LINE (need to scroll to the end of your source).

8.15 Skipping Code

Skip statement, **Skip return**, and **Skip to cursor** all allow you to skip past paragraphs and sections of COBOL logic without executing any of the statements in the code. This allows you great flexibility and control over your Animator session. Of the Skip options, **Skip to cursor** can be the most useful and simplest to visualize.

Skip to Cursor: Mechanics

1. Animate **BOMB10**. Your cursor will be positioned on the PERFORM 100-INITIALIZATION statement.
2. **Click** your mouse pointer in the PERFORM 200-PROCESS-RECORDS line.
3. Select **Animate, Skip to cursor**. This will skip over all of the statements in 100-INITIALIZATION, and position your cursor on the line where your cursor was positioned.
4. To verify that none of the code in 100-INITIALIZATION was executed, double-click (Examine) SR-NAME (contains blanks), and STUDENT-FILE; (it is still closed).
5. **Step** once while on the PERFORM 200-PROCESS-RECORDS line. This will take you to the IF SR-RECORD-TYPE IS EQUAL TO '1' statement in 200-PROCESS-RECORDS. If you need to...Scroll down a bit in the window, so you can see the whole IF statement.

Using Skip to Cursor to Analyze an IF statement

1. **Double-click** on **SR-RECORD-TYPE** and click **Hex**. It contains binary zeroes (the file was not opened/read).
2. Step once while on the IF statement. Your cursor is now positioned on the ELSE statement because the compare tested false.
3. Assuming we wanted to travel down the "true" path of this If statement instead of the ELSE path, **click** in the PERFORM 210-PROCESS-1-RECORDS line in the true path.

4. Select **Animate**, **Skip to cursor**. If we wanted to, we could now debug logic on the “true” path of the IF statement. However, if we wanted to debug the “false path” we could do the following:
5. Click in the PERFORM 220-PROCESS-2-RECORDS. Select Animate, Skip to cursor.
6. Change the value in **SR-RECORD-TYPE** to **2** (using the variable Monitor); **Step**.
7. Through the use of **Skip to cursor**, we have an easy, flexible, yet powerful means of exploring and iterating through complex logic; useful in Unit Testing changes.
8. Close the Animator. Stop Animating.

8.16 Execute at an Operating System Prompt

Introduction

Executing at an operating system prompt can be a useful technique for building job streams, especially when testing with your team in a LAN environment.

External file assignment is also demonstrated in this Exercise. External file assignment is especially useful when you need to test your program with a number of versions of an input file (e.g., an October version, a November version, and a December version).

1. Open the BOMB10 project.
2. Change the Project Properties' Project directives to include ASSIGN(INTERNAL).
3. In Bomb10.cbl, change the ASSIGN names to be logical (i.e., ASSIGN BOMB10IN, ASSIGN BOMB10OT).
4. Re-compile the program.
5. Using an editor (e.g., Notepad) create the following in the bomb10\debug directory. Note: external file assignment.

```
SET BOMB10IN=\netxclass\datafile\BOMB10.DAT  
SET BOMB10OT=\netxclass\datafile\BOMB10.OUT  
RUN BOMB10
```

6. Save as BOMB.CMD.
7. Be sure all needed files (Bomb10.int, Bomb10.idy, Bomb10.dat) are located within the bomb10\debug directory. In your own environment you may want to qualify the path to the data file (e.g., c:\mydir\bomb10.asc).
8. Open the Net Express Command Prompt (from the Start menu). Execute the batch file as follows:

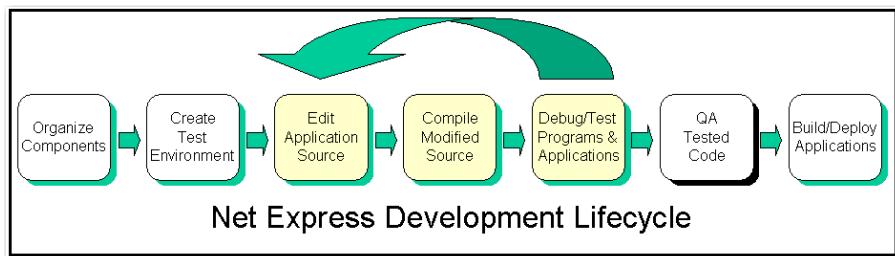
- C:\NETXCLASS\PROJECTS\BOMB10\DEBUG\DEBUG>bomb
- C:\NETXCLASS\PROJECTS\BOMB10\DEBUG>SET
BOMB10IN=BOMB10.DAT
- C:\NETXCLASS\PROJECTS\BOMB10\DEBUG>SET
BOMB10OT=BOMB10.OUT
- C:\NETXCLASS\PROJECTS\BOMB10\DEBUG>RUN BOMB10.

Module 9

Analyze Source Programs

Overview

In addition to using the Animator to debug programs, you can use a desk-checking capability to visualize and analyze programs without actually executing them. The Search menu provides a gateway to a variety of reports and cross-reference tools that are useful for testing and QA efforts.



Objectives

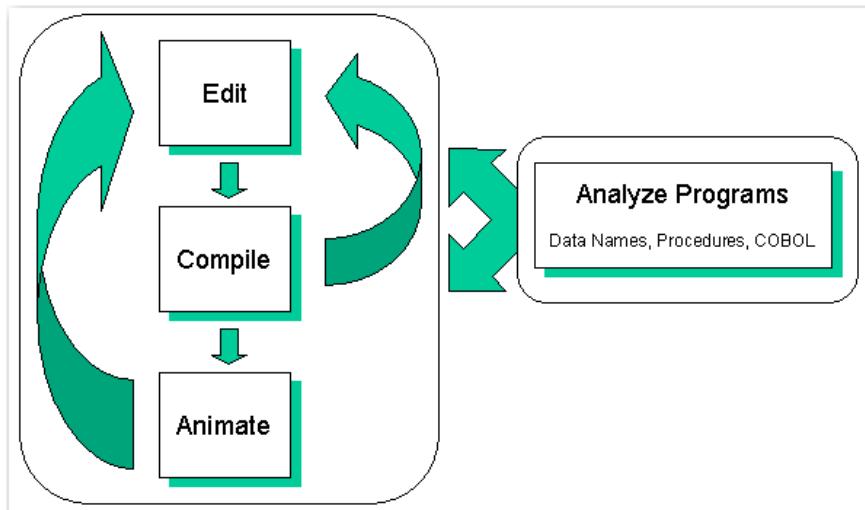
At the end of this module you will be able to:

- Generate lists of references with usage of paragraphs and sections, data, and CALLs in your programs.
- Analyze the use of a data-item throughout your program.
- Analyze the forward-impact of changing a data-item in your source.

- Create statistical summary reports highlighting aspects of program source (program size and complexity, quality assurance, structure, comment ratio).
- Use the summary reports to better understand your program's use of COBOL keywords, copybook structure and use, I/O operations, entry/exit points, dead data, unexecuted procedures.

Program Analysis in the Development Process

Program analysis, visualizing and understanding a program, generally occurs at three places in the application lifecycle.



- **Before making modifications to a program**, visualize the structure of the program within an application. Identify the use of data items in program flow within the application.
- **After making modifications to a program, but before unit test**, verify the changes and assess the quality and maintainability of the changes.
- **During QA and unit testing**, trace errors, such as bad data, to their source. Assist in making corrective modifications.

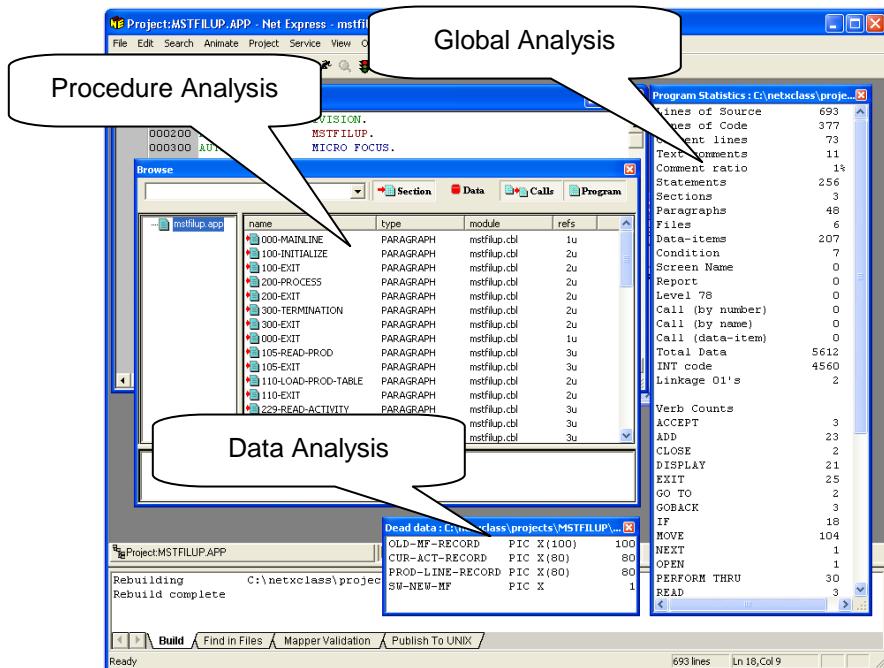
Because program analysis is integrated into the Editor/Animator, all three options are easy. **Note:** You do not need to have a clean compile to utilize these program analysis functions.

Program analysis functionality is integrated into the Search menu.

- Find functions
- Show COBOL reports
- Browse
- Quick Browse

There are three basic program analysis features.

- Data-item analysis
- Procedure analysis (paragraph/section branching)
- Global program information and statistical analysis



Quick Browse – Procedure and Data-Item Analysis

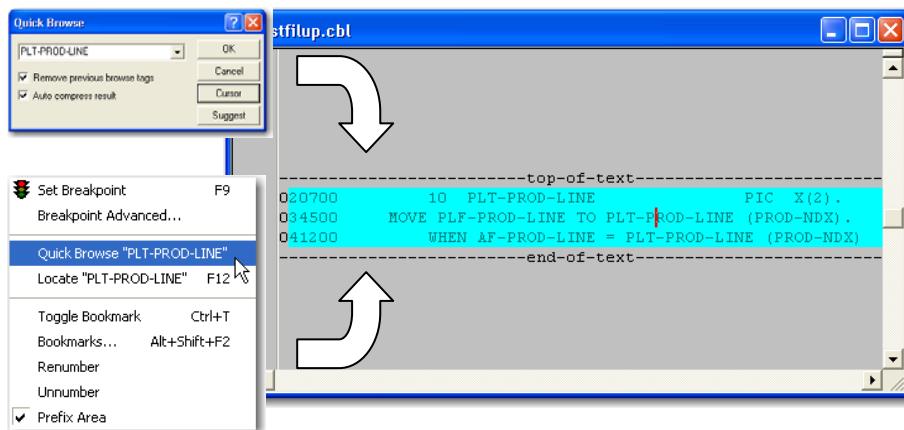
Data-Item Analysis

Once you have compiled your program, the Search menu changes and allows you to search for COBOL item information and explore your program's use of data-items showing:

- Where data-items modified, tested, referenced, etc.
- Which fields are cross-referenced by other fields
- What procedures may need to be analyzed if changes are made to data-items (impact analysis).

To search for a data-item on the current page, use the context menu and right-click on data-item, select **Quick Browse** from context menu or move cursor to data-item and press **Ctrl+F**.

To search for a data-item that is not on the current page, access the Search menu. Select **Search**, **Quick Browse** or **Ctrl+Q**. Supply data name and click OK.



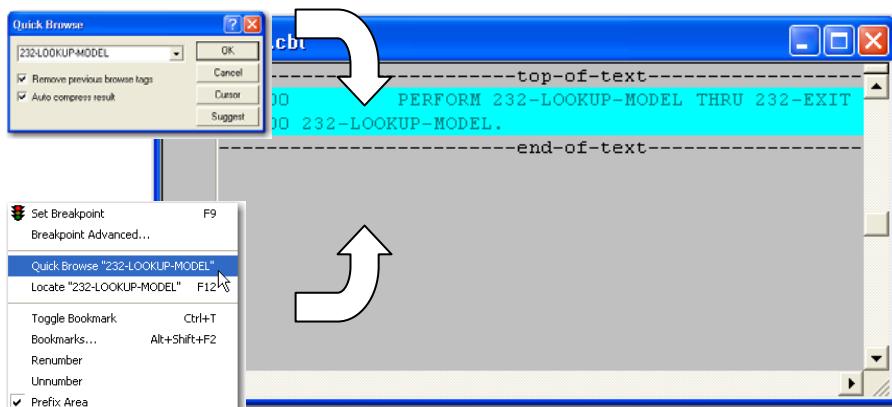
All lines containing the chosen data item are highlighted. You can compress these lines. If you use **Ctrl+Q** you can choose to have your Quick Browse results automatically compressed. Use Suggest if you're not sure of the spelling.

You can do variable-to-variable Impact Analysis by right-clicking any other data-item and selecting **QUICK BROWSE** at any time. To clear a Quick Browse, select **View**, **Clear**, the name of the item.

Procedure Analysis

Once you have compiled your program, the Search menu changes and allows you to search for COBOL item information and explore your program's use of procedures showing:

- How paragraphs and sections are nested.
- How paragraphs and sections are linked together by PERFORMs, GO Tos.
- Where the external program CALLs are.



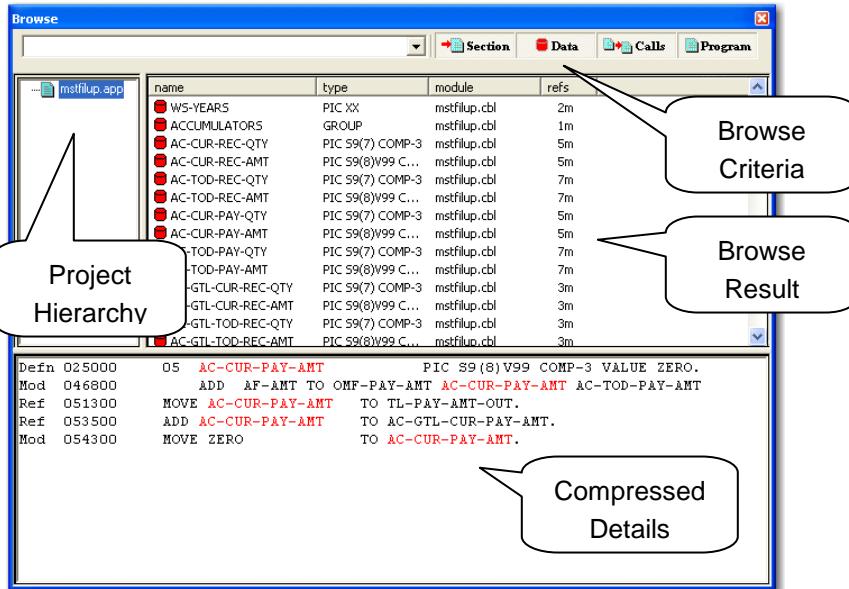
To search for a procedure on the current page, use the context menu and right-click on the procedure, select **Quick Browse** from context menu or move cursor to data-item and press **Ctrl+F**.

To search for a procedure that is not on the current page, access the Search menu. Select **Search**, **Quick Browse** or **Ctrl+Q**. Supply the procedure name or use Label to obtain a list of procedures. Click OK.

All lines containing the chosen procedure are highlighted. You can compress these lines. If you use **Ctrl+Q** you can choose to have your Quick Browse results automatically compressed.

Browse - Section/Paragraph, Data, Calls, and Programs / Entry points

Browse provides another view of the use of procedures and data with additional options to view calls and program entry points. To invoke Browse, select **Search, Browse**. The Browse window contains:



- List/Unlist push buttons for the four search-type criteria: Section or Paragraph, Data, Calls, and Programs/Entry points.
- An entry field (* = wildcard) to tailor results when generating desired lists.
- A project level hierarchy pane.
- A browse result pane displaying Name, Type (for data items, PIC clause is displayed), module, number of references (1 for definition, the rest for modified, used or tested) and a flag indicating if the item usage is (m)odified, (u)sed, or merely (d)efined in the module.
- A detail pane displaying a compressed view of references to the item.

Right-click for a context menu that provides options to Sort by Name, Goto Definition and Highlight references in the expanded source code.



There are some useful options available when using the Browse function. For instance, left-click on an item in the result window pane to see a compressed view of all references in the display panel and obtain a cross-reference into the expanded source code.

Line Type Indicators

Whether using Quick Browse or Browse, each highlighted line has indicators to show how the data item is used. The indicators are listed below.

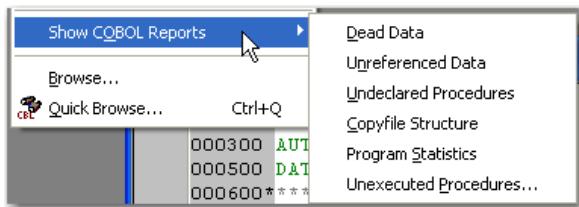
Data Items	
Mod	Contents of data-item storage are modified by this instruction
Defn	Line where the data-item is declared
Test	Data-item is compared (usually IF or PERFORM... UNTIL)
Use	Data-item is referenced but not modified or tested.
Procedures & Programs	
From	Line where the procedure is executed from.
Defn	Line where the procedure is defined.
Call	Line where a subprogram is called or an entry point in a subprogram..

COBOL Reports

Statistics on your program's components, including the number of:

- Files
- Paragraphs, unreferenced or unexecuted paragraphs
- Subroutine calls
- Conditional statements (IF, PERFORM UNTIL, etc.)
- COBOL verbs
- Calculations and arithmetic operations
- Unreferenced data items in the Procedure Division

Select **Search, Show COBOL Reports**. The following are examples reported from the same program.



Dead Data

List of all data items (the area of memory) unused in the program. Not only is the data name never referenced, but the bytes are never addressed by group-levels, redefines, or in any other way.

Dead data : C:\netxclass\projects\MSTFILUP\...		
OLD-MF-RECORD	PIC X(100)	100
CUR-ACT-RECORD	PIC X(80)	80
PROD-LINE-RECORD	PIC X(80)	80
SW-NEW-MF	PIC X	1

Unreferenced Data

List of all data items not referenced in procedure division. This list includes dead data and data items not referenced explicitly in the program.

Unreferenced data : C:\netxclass\projects\MSTFILUP\...		
OLD-MF-RECORD	PIC X(100)	100
CUR-ACT-RECORD	PIC X(80)	80
PROD-LINE-RECORD	PIC X(80)	80
OMF-DAY	PIC XX	2
PRODUCT-LINE-TABLE	GROUP	250
MONTH-TABLE-DEFINITION	GROUP	60
MONTH-DATA	GROUP	60
MONTH-TABLE	GROUP	60
WS-HOLD-AREAS	GROUP	122
WS-DAY	PIC XX	2
MF-YR	PIC XX	2
MF-MON	PIC XX	2
MF-DAY	PIC XX	2
ACCUMULATORS	GROUP	94
REPORT-CONTROLS	GROUP	45
SWITCHES	GROUP	7
SW-NEW-MF	PIC X	1
NEW-REC-WAITING	Condition	

Undeclared Procedures

List of all procedures referenced by the program but not defined (e.g.: PERFORM XYZ when XYZ does not exist). The Compiler flags them.

Copyfile Structure

An indented list of the copy files used in the program. The program itself is treated as the top-level copybook.

Unexecuted Procedures

Paragraphs or sections that are never executed in your program (by PERFORM or GOTO). They may be fall-thrus.

Undeclared procedures : C:\netxclass\...	
No undeclared procedures	

Program Statistics

A list of helpful statistics about your COBOL program.

Lines of Source	694	Verb Counts		I-O	17
Lines of Code	378	ACCEPT	3	Program Exits	3
Comment lines	73	ADD	23	Arithmetic	23
Text comments	11	CALL	1	ANSI'85 END-..	0
Comment ratio	1%	CLOSE	2	Maximum Nesting	3
Statements	257	DISPLAY	21	Overlapping Performs	0
Sections	3	EXIT	25	Program Volume	7376
Paragraphs	48	GO TO	2		
Files	6	GOBACK	3		
Data-items	207	IF	18		
Condition	7	MOVE	104		
Screen Name	0	NEXT	1		
Report	0	OPEN	1		
Level 78	0	PERFORM THRU	30		
Call (by number)	0	READ	3		
Call (by name)	1	RELEASE	1		
Call (data-item)	0	RETURN	1		
Total Data	5612	SEARCH	2		
INT code	4567	SET	4		
Linkage 01's	2	SORT	1		
		WRITE	11		

Module Summary

The Program Analysis tools of Net Express provide you with a variety of techniques for:

- Program visualization.
- Data-item and code research.
- Application understanding.

The prerequisite to obtaining program analysis functionality is to have compiled your source. There are three basic types of analysis.

- Data-item
- Procedure (paragraphs/sections and how they are branched to/chained)
- Global program information and statistics

Data-item and procedure analysis is done via Quick Browse and Browse. Global program information/statistics are available from Show COBOL Reports.

You can use program analysis functions during:

- The analysis cycle of a maintenance or production support project to quickly learn and/or visualize the logic in a program.
- While you edit your source to understand the impact of changes you make to data-items or procedures.
- During unit testing to work productively and effectively with certain types of ABEND and logic error situations.

9.1 Review Questions

Define and describe the following COBOL Program Analysis terms and concepts.

1. How do you analyze Data-Items? When would you use Data Item Analysis in COBOL program maintenance, production support and development?
2. What could you use the COBOL metrics (maximum nesting, overlapping PERFORMs, program volume) for? To help answer this question, open your MSTFILUP project. Edit: MSTFILUP.CBL.
3. Try the options. From which Editor/Animator menu(s) can you find an option to:
 - List all paragraphs that are never referenced in your program.
 - Generate Program statistics.
 - List the number of sections and paragraphs in a program.
 - Scroll directly to the start of PROCEDURE DIVISION.
 - Open a window that lists all COPY files used in a program.
 - List the maximum level of IF nesting IF statements.
 - List all of the COBOL verbs used.

9. 2 Move Processing To A Subroutine

1. The section, FORMAT-REPORT-DATE, is to be removed from CALLTEST.CBL and become a sub-program, DATETEST.CBL.
2. Use the COBOL item find option to get information about the section, FORMAT-REPORT-DATE.
3. Locate the statement, PERFORM FORMAT-REPORT-DATE, including the section name, and cut it to the clipboard.
4. Mark all code from the section FORMAT-REPORT-DATE, including the section name, and cut it to the clipboard.
5. A skeleton program, DATETEST.CBL, has been provided for you. Open this program for editing and paste the text from the clipboard to the Procedure Division. (The Linkage Section has already been set up for you).
6. Return to the CALLTEST.CBL program. Use **Locate** to find the definitions of the data used by FORMAT-REPORT-SECTION (now in the DATETEST.CBL program). Copy this data (including the 01 level item) to the Working-Storage Section of the DATETEST.CBL program.
7. Save DATETEST.CBL and close that edit window to return to the CALLTEST.CBL program edit window.
8. Save CALLTEST.CBL.
9. Select **Project, Rebuild All** to compile both CALLTEST.CBL and DATETEST.CBL.
10. Correct any compile errors until you have successfully compiled both CALLTEST.CBL and DATETEST.CBL

9.3 Combining Data Analysis and Edit

1. Recompile CALLTEST and use **SHOW... DEADDATA**, to find all the data in the Working Storage section of CALLTEST that is no longer needed. Delete ONLY those lines that define this dead data and save CALLTEST.CBL.
2. Recompile the program, CALLTEST, and fix any compiler errors.
3. Compile DATETEST; fix any compiler errors.
4. Animate CALLTEST.
5. Use the program breakpoint function to stop the execution when program DATETEST is called.
6. Within DATETEST, use the Examine function to change the value of MONTH and test that each of the EVALUATE conditions is correct and that there are no incorrect conditions. Step to GOBACK. Use Skip to Cursor Position (pointing to the first line of the Procedure Division) to restart DATETEST between each test with a different value for MONTH.

9.4 Data Analysis: Data Item FIND (Optional)

1. Open the **FIXBOMBS** project; animate **BOMB09.CBL**.
2. Use Quick Browse to **find** the variable SUB. Answer these questions.
On how many lines is SUB found? _____ (See Message Line).
If the definition of SUB was expanded from PIC 9(2) to PIC 9(5), list some of the variables affected.

3. Open the **MSTFILUP** project. Edit **MSTFILUP.CBL**.
4. Use **Show COBOL Reports menu** to answer these questions.
How many data items are never referenced? _____
How many data items are deaddata ? _____
How many paragraphs are never referenced? _____

5. Open the Program statistics window and answer these questions.
How many lines of source are there? _____
How many lines of code are there? _____
How many Program exits (GOBACKs) are there? _____
Is this program well-documented (Comment ratio)? _____
What is its Program Volume? _____ Does the program contain SECTIONS: _____ SORT: _____ SEARCH: _____

6. Open the **Finance** project. Edit **FINANCE.CBL**. Use the COBOL Locate menu to quickly position your cursor at the start of the WORKING-STORAGE SECTION; PROCEDURE DIVISION.

7. Open the Program statistics window and answer these questions.
How many lines of source are there? _____
How many lines of code? _____
Which program is more complex, FINANCE.CBL or MSTFILUP.CBL?

9.5 Data-Flow Analysis

Introduction, Part 1

Due to corporate restructuring, the Payroll application needs to be modified. In the current application, an employee's gross-pay is calculated by multiplying hours-worked by a pay-rate. The pay-rate is currently declared as a 9V99 field in the data file, TBLFILE.DAT. You need to increase the size of the field to 999V99.

TBLFILE.DAT currently exists as a fixed-length sequential 32-byte file in a single record that is read into Working-Storage and used to populate an internal COBOL table. The new file contains one record per/entry. Each record is 5 bytes long and contains only the pay-rate in the format PIC 999V99. We expect that there may be up to 300 different pay-rates. Read the below for a description of WHAT to do. For the HOW TO, do the steps below the numbered bullet points.

Details, Part 1

Prepare the program for analysis.

1. Create a PAYAPPL project in a new \PROEJCTS\PAYAPPL directory.
Add PAYAPPL.CBL and its copybook (EMCOPY) from
\NETXCLASS\OTHERS.
2. Compile the program and load it into the Editor.

Perform Impact Analysis on modifications to the old pay-rate. Find all TBLFILE references (PAY-RATE exists in TBLFILE).

1. Scroll to the top-of-file.
2. Position your cursor on the TBLFILE identifier (on line 6 after the SELECT verb).
3. Right-click and select **Quick Browse**.
4. **Compress** your view of the source. This shows you all lines in your program that reference TBLFILE.

Modify references to the TBLFILE FD. Change the declarations to accommodate the file's new format.

1. Position your mouse on the **FD statement for TBLFILE**.
2. Right-click and select **Expand**. Recall that the file has changed size from 32 to 5 bytes. Change the RECORD CONTAINS line and PIC clause accordingly.

Determine the impact of changes to TBLFILE and the pay-rate fields.

1. Compress the program source to view all TBLFILE references. Note the READ statement that shows that TBLFILE is read into TABLE-DATA-RECORD (you're going to have to modify TABLE-DATA-RECORD).
2. Position your cursor over the **data-item TABLE-DATA-RECORD**.
3. Right-click and select **Locate**. This positions your cursor over TABLE-DATA-RECORD's declaration.

Introduction, Part 2

The existing TBLFILE has a two-byte area (TD-NO-OF-ENTRIES) that contained the total number of entries, and 10 three byte (PIC 9V99) entries (TD-ENTRY) in a single record. This single record file format has been replaced with (up to 300) five byte pay-rate records. If you were going to modify this element, you would need to remove the TD-NO-OF-ENTRIES field and change the definition on TD-ENTRY. But it's usually best to analyze what impact changes will have on programs before doing the modifications.

The following represents an impact analysis process using Animator and the techniques presented in this chapter.

Details, Part 2

Tag both of these **fields/lines** for future impact analysis.

1. Position your cursor on TD-NO-OF-ENTRIES.
2. Use **Ctrl+T** or right-click and select **Toggle Bookmark**.

3. Repeat for TD-ENTRY. Do a complete impact analysis on each of these fields before you change/delete them. Start with TD-NO-OF-ENTRIES. Note that when doing impact analysis, it is best to “drill down” and tag lines with fields to be changed until you get to the lowest level of cross-reference. This is especially important when you are deleting field entries.

Find and analyze all fields that reference TD-NO-ENTRIES.

1. Position your cursor on the TD-NO-OF-ENTRIES field.
2. Right-click, select **Quick Browse**, and compress your view of the source. Note when you attempt to compress a tagged line, you will be prompted for what to compress, the item of interest or Bookmarks. Select TD-NO-OF-ENTRIES. Notice that TD-NO-OF-ENTRIES is MOVED to TABLE-ENTRY-COUNT. It appears from the program logic that TABLE-ENTRY-COUNT stores the original number of table entries in the pay-rate file, probably for future “limit processing”.

Analyze TABLE-ENTRY-COUNT.

1. Position your cursor over the data-item TABLE-ENTRY-COUNT.
2. Right-click and select **Quick Browse**. Notice the PIC clause for TABLE-ENTRY-COUNT. Recall that originally there were 10 occurrences of pay-rate, and that it is anticipated that there may be up to 300 in the new pay-rate file.
3. Change the PIC clause from PIC 9(02) to PIC 9(03) in order to set aside enough digits.

Keeping track of changes.

1. Compress to see that in addition to the reference to TD-NO-OF-ENTRIES you found two other data-items that are impacted by changes to TABLE-ENTRY-COUNT: RATE-SUB and EM-PAY-CLASS. Both of these data-items will have to be studied.
2. Tag these lines to ensure that you remember to change them.
3. Position your cursor on the RATE-SUB field.

4. Right-click and select **Locate**.
5. Toggle on a bookmark on the definition.
6. Compress the finds of RATE-SUB (to see more clearly).
7. Repeat for EM-PAY-CLASS.

Optional

Modifications to data-items dependent on your primary change. RATE-SUB is used as a table subscript for TD-ENTRY. Note also that RATE-SUB's size is defined as PIC 9(02). This Picture clause is too small for the new 300-entry table. Recall from 2.8 that EM-PAY-CLASS is also impacted by changes to TABLE-ENTRY-COUNT.

Using data-flow item Find/Compress and drilldown, locate and tag as many new data items impacted by RATE-SUB and EM-PAY-CLASS as you can find throughout the source.

9.6 Extending the Analysis

Looking at the compressed view of TD-NO-OF-ENTRIES, we see that, besides the original declaration (Defn->) there is only one (Use->) of the field in this program. So, all we need to do is figure out the impact of MOVE TD-NO-OF-ENTRIES TO TABLE-ENTRY-COUNT.

Place your cursor on the MOVE line, and Expand the source. Recall that TBLFILE was originally a one record file, containing all the pay-rates. Since the new file design consists of multiple five byte records, how would you modify the original logic?

Where would you put the logic? **Note:** Do not try to actually code/test/run the solution. Simply determine the areas of interest and add comments to the code.

9.7 Challenge: Do You Have What IT Takes?

The MSTFILUP program has been executing in production for some time. The users complain that the program generates an awful lot of date errors – they think something is wrong with the program, not with the data (and, to tell the truth, they're right).

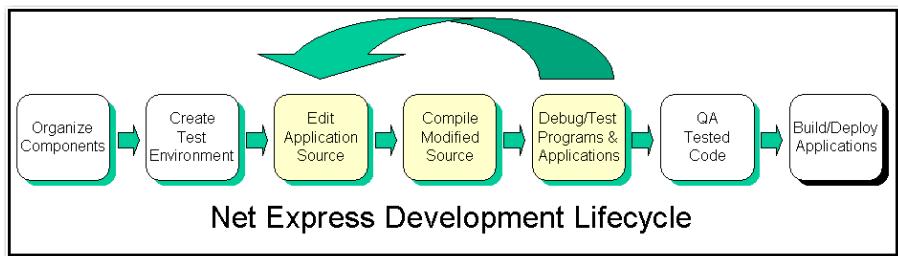
Fix the logic in MSTFILUP so that it no longer generates any date errors. (Do NOT solve the problem by commenting out the DISPLAY *** DATE ERROR... line). The correct result is 0 date errors. (If your fix results in just 3 date errors you're on the right track – keep going). Note: Use all your new skills – breakpoints, Quick Browse, expand / compress, etc.

Module 10

Building Applications for Release

Overview

Once you have edited, compiled, and tested your application, it is ready for release to production. There are options to choose from based on the size and complexity of the application.



Objectives

At the end of this module you will be able to:

- Choose a run-time system i.e., static-linked, shared (dynamic linking), run-time support modules and compare standalone and modular applications. Set advanced Linker options.
- Create and Package application components into EXEs and DLLs i.e., choose components for the distributed application, decide on the packaging of the application, use of libraries.
- Use the Link Wizard.

Compile and Link using Generic Release Build

After completing the development, maintenance and interactive source-level debugging cycle of COBOL applications using the Generic Debug Build functions (edit, compile, animate), you will want to deliver the application in "built" form to the target environment for execution. Net Express simplifies this process with its project-based environment. Most of the same principles used for Generic Debug Build apply to the Generic Release Build.

The following elements are required to produce the application release build:

- Object Code files
- Run-time support modules
- Application executable programs (.EXEs and .DLLs).

Creating Executable Files

Net Express creates .OBJ, .EXE and .DLL files used for deploying new releases with the selected run-time system (resident code mapped into the address space of a process, called on to provide services to the running program).

.OBJ Modules

The compiler translates COBOL source files into industry-standard object code files (.OBJ). An .OBJ file contains no calls to the operating system, does not access any system devices and cannot be executed directly. Rather, an .OBJ calls a run-time system, which performs these tasks for the program. The object code file must be linked to the run-time system to produce an executable file that can be run by the operating system. Not unlike a mainframe link step, when a Net Express program is linked, it is joined with any other programs that it calls, with any run-time support modules that it uses, and finally with the run-time system that it needs. These .OBJ intermediate files are used by the Generic Release Build during the link process to create .EXEs and .DLLs.

.EXE Modules

System executable files, (.EXE) are used for the main program in applications, for standalone applications, and for mixed language applications. They are executable modules that can be run in a stand-alone environment. .EXE files are deployed to the target environments for execution.

.DLL Modules

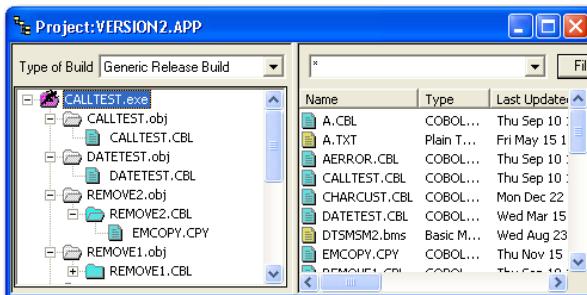
Dynamic link library (.DLL) modules contain executables, which have been linked together as a .DLL rather than as an .EXE. Linking to a .DLL format offers advantages for memory management in a Windows environment.

They are shipped to the target machines along with at least one .EXE. DLLs use memory efficiently, and enable an application to be modular in construction.

Note: Both .EXE and .DLL consist of native machine code and are optimized for speed. They can include object code from programs written in a mixture of languages, and can be shared between processes.

Types of Build

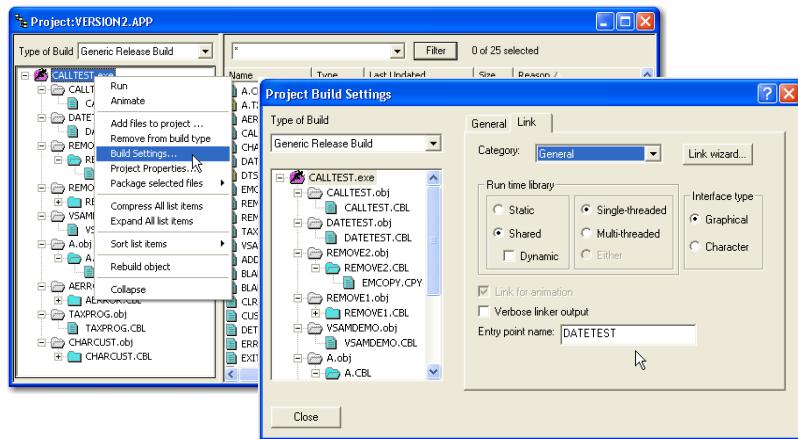
A release build, by default compiles and links the source files into one system executable file using the shared run-time system. The compiler options and build settings will vary, depending on the type of application



being produced. For example, you may want to group two sub-programs into a .DLL file, and change the run-time system option.

The **Types of Build** pull-down selection box on the Project window allows you to switch between the Generic Debug (default) and Generic Release build types. Follow these steps to switch from Generic Debug Build to Generic Release Build.

1. Select the **Types of Build** pull down in the Project Tree View pane.
2. Select **Generic Release Build**. The Project Tree View displays a tree view of the hierarchy of the built application. When accessed the first time, the display assumes that all modules are to be linked to form a single .EXE driven by the program you selected as the main program when the project was created. You can change the main program by specifying the Entry point name field on the Link tab of the Build Settings dialog box.



Packaging Application Components

Package application components into .EXE and .DLL modules for your application before performing a Build or Rebuild All. Review all components in the Project Tree View and remove (or exclude from Build) all the components that are not required by the distributed application.

1. From the Project Tree View pane, right-click on **the (highest level) component** to remove.

2. Select **Remove From Build Type** to remove selected components from the Project Tree View pane leaving them in the source pool pane.

Removed components may be reinstated to the Project Tree View pane at a later time. To remove components from the project from the Source Pool pane, right-click on the component to remove.

3. Select **Remove From Project**. Selecting Remove from project removes the selected components from the project altogether. Removed components may be added back to the project at any time.

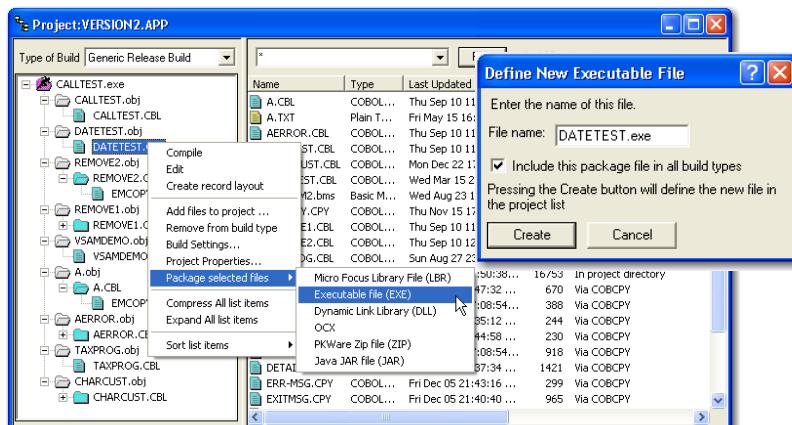
Decide on the Application Packaging

Before beginning you must decide on the exact packaging of the application. What components will be packaged together and delivered as an .EXE? What components will be packaged together and delivered as a .DLL? Having made these decisions, continue by packaging these components into a separate .EXE plus DLLs.

Packaging an .EXE

To package components into an .EXE follow these steps:

1. Select all components that are to be packaged together to form a single .EXE. Select as many application components as necessary. Right click on any selected component.

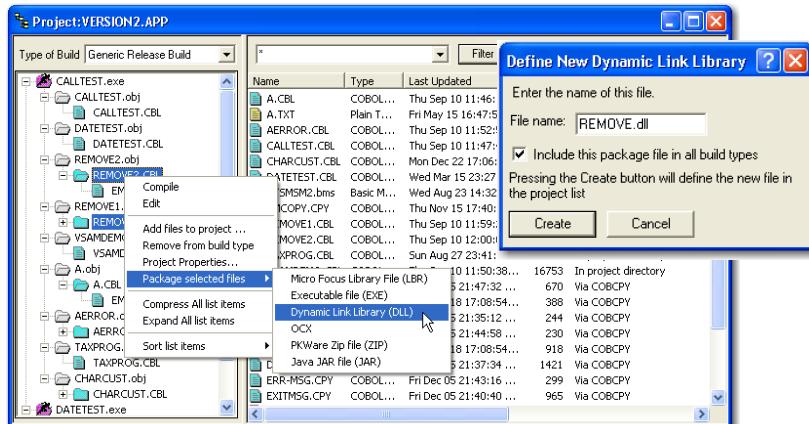


2. Select **Package Selected Files, Executable file.**
3. In the Define new Executable file dialog box, **enter the .EXE file name.**
When a project Build or Rebuild All is performed, the selected files are packaged together in a single .EXE.

Packaging a .DLL

To package components and create a .DLL follow these steps.

1. Select **all components** necessary to form a single .DLL.
2. Right-click on any selected component.
3. Select **Package Selected Files, Dynamic Link Library (DLL).**
4. In the Define new Dynamic link library dialog box enter the **.DLL file name.** When a project Build or Rebuild All is performed, the selected files are packaged together in a single .DLL.



Completed Release Build Example

After you have completed the process of selecting components and assigning an executable type of .EXE or .DLL, the Project Tree View will display the distribution project's hierarchy.

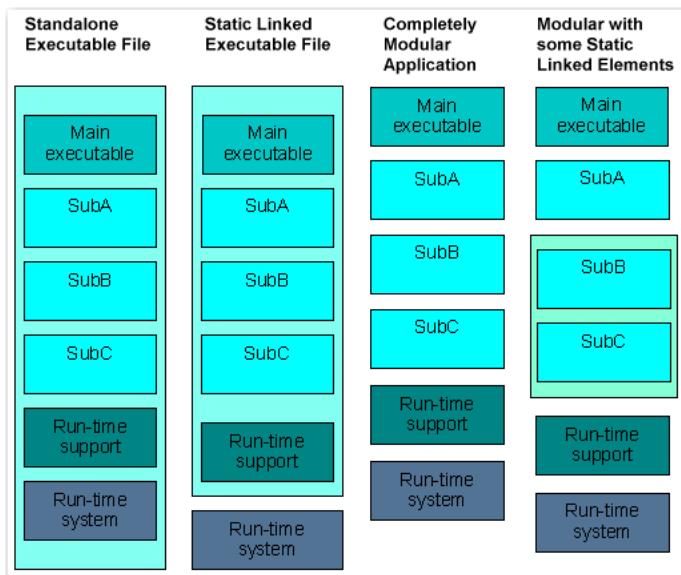
Using the examples shown above, the following occurs:

- DATETEST are compiled to .OBJ and linked to form the stand-alone executable DATETEST.EXE.
- REMOVE1 and REMOVE2 are compiled to .OBJ and linked to form REMOVE.DLL dynamic link library.

After performing a Build or Rebuild All the resulting .OBJS, .DLLs and .EXE are placed in the default directory defined for the Generic Release Build type. In this case it is: \NETXCLASS\PROJECTS\VERSION2\RELEASE.

Run-Time Systems

A run-time system is the interface between a program and the operating system. The Net Express run-time system provides everything needed by a COBOL program to run. The run-time components support advanced features of the COBOL language such as the extended ACCEPT/DISPLAY syntax.



There are two Net Express run-times: the static linked run-time system and the shared run-time system. Choosing between the static-linked run-time

system and the shared run-time system depends on whether the application is to be standalone or modular.

A standalone application consists of the static run-time system and necessary support files built into a single executable file. For example, a small utility or an application for selling to an end user would not need to share the run-time system. This is the simplest method of packaging, requiring only one executable file for your users. **Note:** You cannot build object-oriented applications as standalone applications.

A shared or modular application consists of more than one executable file. The executable files are loaded into memory when called by a program. In the corporate environment it would be advantageous to share the run-time system conserving space and memory.

Note that you may need to customize either linking choice of special support modules are required by your application.

Static-Linked Run-Time System

When using the Static-Linked option, the run-time files are included in the final executable file together with the program modules. Run-time calls are resolved within each executable file, so shipping a separate copy of the run-time system with your application is unnecessary. Programs linked with the static linked run-time system are larger than those linked with the shared run-time system. The linked subprograms are all loaded into memory at the same time as the program that calls them.

It may also be necessary to use the static run-time system with a statically linked .DLL file. For example, if the calling program were written in another language, all of the COBOL programs would be linked into one .DLL file, with the run-time system provided as part of the .DLL.

Shared Run-Time System

When using the shared run-time option (aka dynamic linking), the run-time files are not packaged with the executable file. External calls are made to the shared run-time system components, via pointers in the executable, which are loaded as required when a program is run. Programs are only

loaded into memory when called and are available to be shared with other COBOL applications.

Dynamic linking enables programs running in separate sessions to share the same loaded run-time system components, reducing memory usage.

Support Modules

Support modules contain routines to support advanced features of the COBOL language. For example, if your program makes use of the extended ACCEPT/DISPLAY syntax, it needs to call an additional module (ADIS) not typically packaged with the run-time systems. You can build these support modules into the application's executable files, through either run-time system.

Run-Time System Advantages/Disadvantages

Standalone and modular applications each have advantages and disadvantages.

Standalone		
	Advantage	Disadvantage
	Easy to deliver to users.	Can use a lot of memory. All programs are loaded at the same time. Note: minimally must deliver the following with the app - PRODFILE from the application server licensing, aslmpclocate.exe, and aslmpcsilent.exe
	Executes quickly. All programs are loaded into memory at the same time.	Difficult to maintain. All code is in one executable file.
		Cannot be an object-oriented application.
Modular		
	Advantage	Disadvantage
	Easy to maintain. All called programs remain in separate executable files.	Executes slower than a comparable standalone application. Each called program has to be loaded into memory.

Completely Modular Application

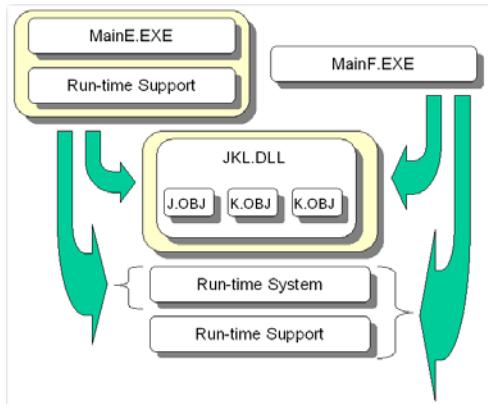
- All the programs are linked dynamically then linked to the shared run-time system.
- When you ship, supply all application executables, the run-time support executables, and the shared run-time system executable.

Example: Assume the main executable is linked from a file main.obj, and that the application uses ADIS (ACCEPT/DISPLAY) and the Callable File Handler. Ship:

Application files	main.exe, suba.dll, subb.dll, and subc.dll
Run-time support modules	adis.dll and extfh.dll
Shared run-time system	cblrtss.dll
	Various other files

Modular/Some Elements Statically Linked

In the illustration below, Subprograms J, K and L are statically linked to create a dynamic link library JKL.dll. The file JKL.dll is linked to the applications dynamically. The .EXE files have been linked in different ways to the shared run-time system.

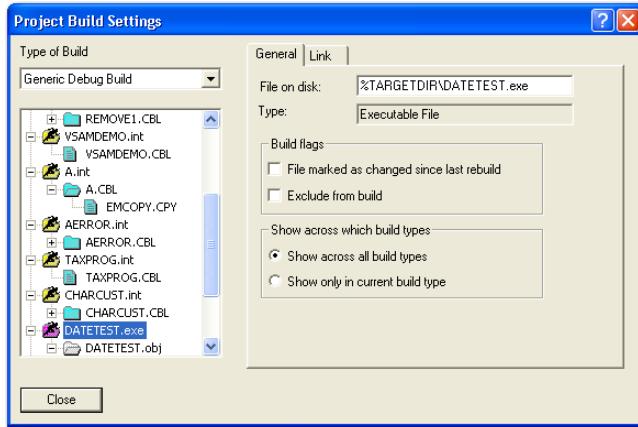


When you ship, supply all of the required executable files for the applications. This could include the executable run-time support modules and/or the executable shared run-time system.

Using the Net Express Run-Time System

Follow the steps below to set Net Express run-time build settings:

1. Open the **Build Settings window**. Right-click on an **executable name** in the Project Tree View pane. Select **Build Settings**. The Project Build Settings window displays for your project. This window will be used to select each executable and assign the appropriate run-time system.



2. Set the **General tab options**. The General tab allows you to set general options for the component selected in the Project Tree View displayed. When an executable is selected:
 - Select **File marked as changed since last rebuild** if you want to force a file to be rebuilt the next time the project is rebuilt.
 - Select **Exclude from build** if you do not want the file to be rebuilt when the project is rebuilt, even if it has changed. You may use this as an option to removing the file from the build type.
 - The **Show across which build types**, allow you to set whether this module appears in all build types or only this one build type. Select **show across all build types** if this component is to be included in all project build types.
 - Select **Show only in current build type** if this component should only appear in the current builds type.

3. Set the **Link tab Options**. The Link tab's General category selection allows you to assign the required run-time system to the selected executable. Select the required Run-time Library for your application.

When Static is selected:

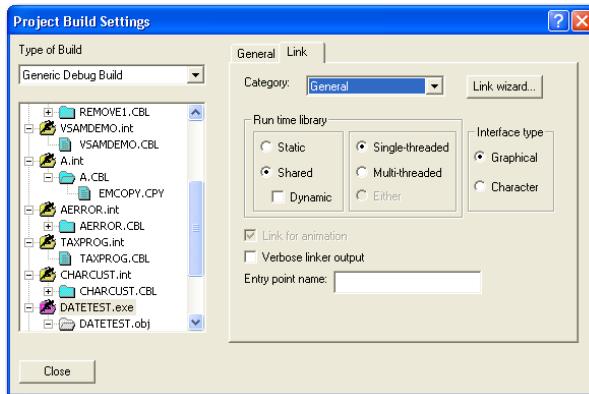
- Link for animation builds in code required to animate the executable.
- Verbose linker output: the linker displays all messages during the link process.

When Shared is selected:

- Dynamic, dynamically binds your executable to a run-time system. (For more information, refer to the Dynamic Binding of the Run-time in the Appendix.)

Select the interface type. Graphical if your application uses a graphical user interface, or a character user interface and you want to use the default COBOL text window. Select Character if your application uses a character user interface and you want it to run as a WIN32 application.

Enter the base name (the filename without its path or extension) of the .OBJ file.



4. Rebuild the Project. Having packaged the application components into .EXE and .DLL files and selecting the required Run-time System support, all that remains is to perform a Build or Rebuild All. From the

menu bar select **Project, Build** or from the menu bar select **Project, Rebuild All**. Review any messages appearing in the Application Output window to insure that the project was built without errors.

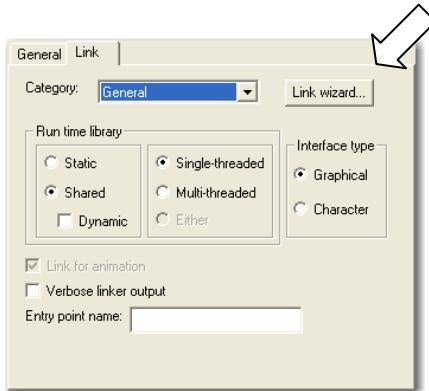
Using the Link Wizard

The Link Wizard assists you in optimizing the way your application is linked by changing settings on the Link tab for you. The Link Wizard is especially helpful when:

- Your application programs use Object COBOL syntax.
- Your program is a CGI server-side program.
- Your application is multi-threaded.

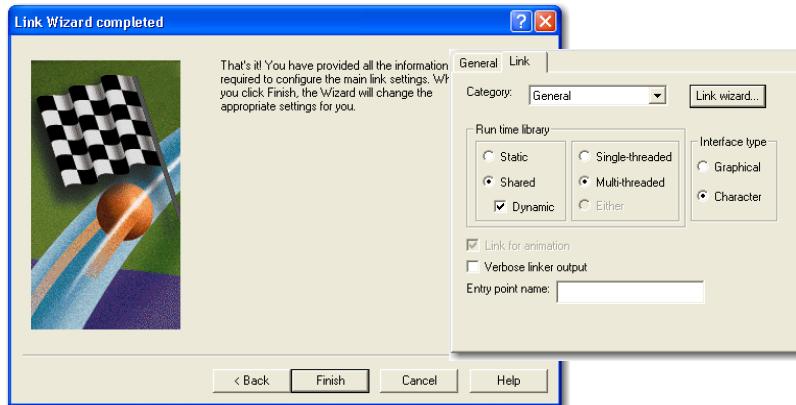
Invoking the Link Wizard

1. Right-click on **the .exe** in the Project window; select **Build Settings...**
2. Press the Link tab in the Project Build Settings window. The Link Wizard will guide you through setting up the link by walking through a number of screens.



3. The first screen is an overview screen; press **next**.
4. If the program does not use Object COBOL syntax press **next**, otherwise toggle Object COBOL on and press **next**.

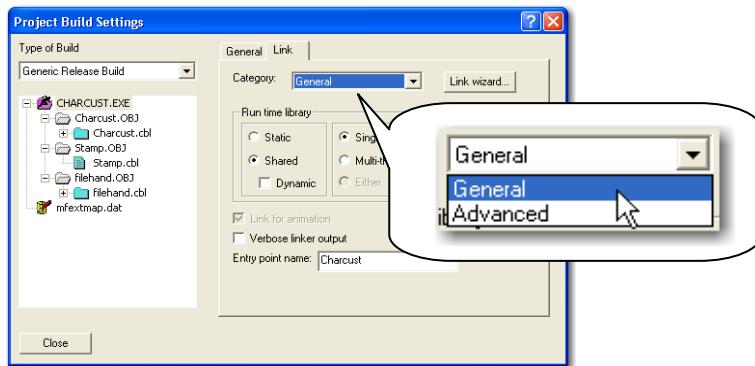
5. If the program is a server to a Web application using CGI, toggle it on, otherwise, toggle it off. Press **Next**.
6. If the program has a graphical user_interface (i.e., Dialog System), toggle it on, otherwise toggle character user interface on. Press **Next**.
7. For a standalone executable (with the run-time system built in), toggle on standalone executable. For a shared executable (with a separate run-time system), toggle it off. Press **Next**.
8. If the program is multi-threaded, toggle it on. Otherwise, toggle single-threaded. Press **Next**.
9. If you wish to accept the settings, press **Finish**. The Link window will display the resulting settings. If you wish to change the settings, you may find you need to remove and reload the files in the project window. This is because the settings for the executable are retained, suppressing the display of all the Wizard's panels.



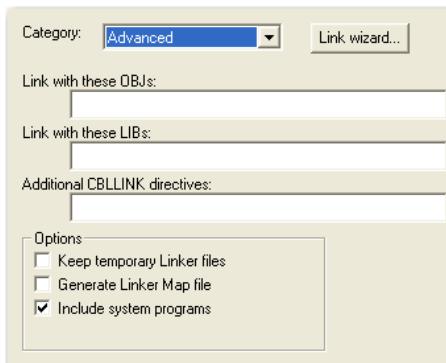
Setting Advanced Linker Options

Most applications can be linked using the parameters contained on the Link tab's General category selection window. In some applications it may be necessary to specify additional link commands using the Link tab's **Category:** Advanced selection.

Enter the names of additional .OBJ modules that are not part of the project that are to be linked with this executable in the Link with these OBJ's entry field.



Enter the names of additional .LIB modules that are not linked by default and required by your application in the Link with these LIB's entry field. Enter any additional linker directives in the Additional CBLLINK directives entry field. (CBLLINK is the link utility. Typically it is invoked from within a Net Express project).



Additional Advanced Linker Options

Additional options may also be specified.

- Select **Keep temporary Linker files** when you need more files as an EXE or DLL from the link process. (e.g. If your DLL needs to be called from a non-COBOL executable or the COBOL-CALL from an EXE/DLL has a CALL-CONVENTION 8 or 74 (also directive LITLINK), you will need to provide a LIB file together with your DLL.) The other files like LNK, and DEF can be used to resolve some linking problems.
- Select **Generate Linker Map file** to create the .MAP file which contains a timestamp, a preferred loading address, a list of public entry points used in the linked object files, and a fixup (load offset) table. (Not selected by default.)
- Select **Include System Programs** if you want any run-time system support modules needed by your application to be automatically linked with it. Run-time support modules provide support for:
 - User-defined classes in SPECIAL NAMES paragraphs
 - CGI programs
 - External files and data
 - Enhanced ACCEPT and DISPLAY syntax
 - Intrinsic functions
 - PC_Print Library routines.
- Select **Bind to current Release** of the runtime only if you are using dynamic binding for the shared run-time system and you do not want this .exe to bind to an older version of Net Express that may be on the user's system. If not selected, you must link any support modules needed by adding them to your project. If your application requires run-time support for functions other than listed above, you must link the appropriate modules by adding them to your project. (Selected by default.)
- **Link Level - Lite:** Static link with the lite run-time system. The lite run-time system is smaller; it provides full support for many COBOL applications consisting of .obj files only. (Not for use with Micro Focus)

library (.lbr) files, dynamic calling of .int and .gnt files, run-time system configuration using COBCONFIG, shared memory allocation, or DD_name file mapping). For information on COBCONFIG, see Configuration Files in the on-line help Index.

- **Link Level – Base:** Static linked with the base run-time system (the default run-time system). It provides all support needed for linking .OBJ files only. Do not use it you need support for Micro Focus library (.lbr) files or dynamic calling of .int or .gnt files.
- **Link Level - Full:** Static link with the full run-time system. The full run-time is slightly larger; it provides full support for all COBOL applications, including support for Micro Focus library (.lbr) files and dynamic calling of .int and .gnt files. **Note:** To debug statically linked system executable files or dynamic link libraries, select the Link for Animation checkbox (General in the Category list).

Packaging Applications using .INT and .GNT Formats

While we recommend that you create your applications by compiling to industry standard object code and linking the resulting modules, it is possible, as in previous versions of Micro Focus Workbench, to deliver applications consisting of .INT and .GNT files.

Advantages provided by using Micro Focus .INT and .GNT executables

- .INT and .GNT files are relatively compact
- Linking of .INT and .GNT files is not required
- Maintenance of the application is simplified.

Disadvantages of using Micro Focus .INT and .GNT executables

- .INT and .GNT files are not shareable as are .EXE's and .DLL's
- .INT files are interpreted and slow when executing
- Does not follow the industry standard methods for creating executables.

Libraries

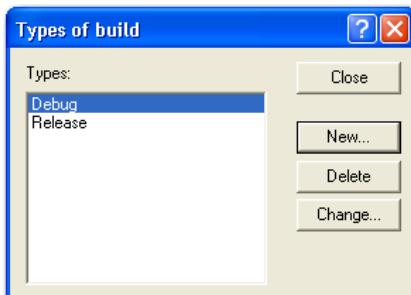
A Micro Focus Library File is a collection of files stored together in a single file with a .LBR extension. Using .LBR files offers several advantages:

- .LBR files may contain any type of file including .INT's, GNT's, data files (e.g., password files), etc. required by your application (Note: .LBR files are read only).
- Packaging many files into one file reduces the number of files that need to be shipped with your application.
- When changes occur there is no need to re-link the application. You need only to ship the new version of the .LBR file or a single .INT or .GNT.

Create Applications for Distribution using .INTs and .GNTs

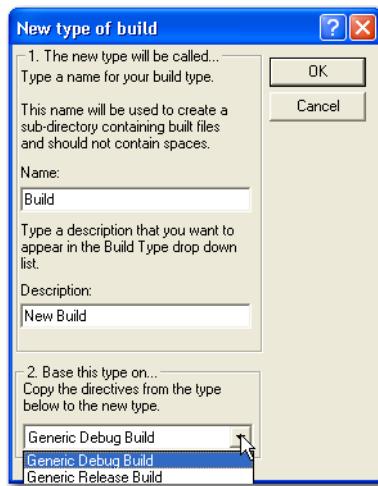
Create a new build type for your project; i.e., open the project for the new Build Type first. The new Build Type will apply to the opened project only.

1. Select **Project, Types of Build** from the menu bar. The Types of Build window is displayed showing all valid build types currently defined to Net Express.

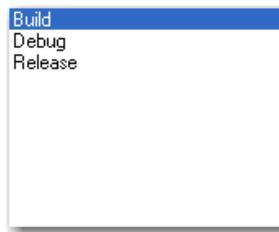


2. Select **New**. The New Type of build window is displayed.
3. Enter a Build Type Name that will appear in the Types of Build list window. The name entered here will be used as the default directory name where resulting executables are placed when a **Build** or **Rebuild All** is performed.

4. Enter a Build Type Description that appears in the Build Type pull-down on the Project Tree View pane.
5. Select the Base this type from the pull-down of current valid Build Types. (The Generic Debug Build was selected because we will be creating .INT and .GNT executables.) This copies the existing Build Directives to your new build type.



1. Once you create the new Build Type, return to the Project Window and select the new Build Type.

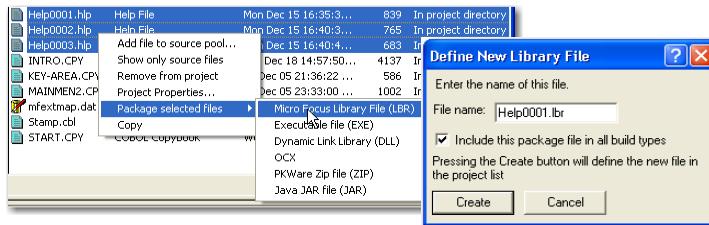


2. From the Project Tree View pane follow standard procedures to remove or exclude any unwanted components from the Build Type.

3. Select the COBOL components and access the Build Settings Compile tab to select either .INT or .GNT for the compile to type.
4. Package selected components into a Micro Focus Library File (.LBR).

Creating a Micro Focus Library File

6. In either the Project Tree or Source Pool pane select all components that are to be packaged together.
7. Right-click on the selection(s) and select **Package selected files**
8. Click **Micro Focus Library File (LBR)** and the **Define New Library File** window displays.



9. Enter the name of the Micro Focus Library File.
10. Select **Create**.
11. The Project Tree View pane now displays the created Micro Focus Library File details.

Create an .EXE Trigger Program for your Application

Creating an application that consists of a mixture of .INT and .GNT files, requires the creation of a trigger program. Trigger programs are always linked to the shared run-time system to create a system executable file. The trigger program is the first program executed and it is responsible for linking in the parts of the run-time system and run-time support modules required by the application. The trigger program must:

- Load the run-time system
- Load the required run-time support modules
- Call your first application program.

Module Summary

Net Express creates .OBJ, .EXE, and .DLL files.

Select the type of build that works for you and your application.

Choose the appropriate run-time system.

Use the Link Wizard to optimise the way your application is linked.

10.1 Packaging and Building an Application

In this exercise you will create an executable using the Timelist project.

Note: If you are working with a new release of Net Express, you will be prompted to rebuild the project when you open it. Allow Net Express to rebuild the project.

1. Open the **Timelist project**. To gain some insight into the whole process, select all the files in the Project Tree View and remove them from all Build Types.
2. Be sure the Project window's Type of Build is set to Generic Debug Build. Add the two COBOL programs back from the source pool as intermediate code.
3. In the Project Window's Type of Build, select **Generic Release Build**. If you reloaded the Generic Debug Build files from the source pool you will notice that a Generic Release Build involves different files. Add the programs to the Generic Release Build, too.
4. Right-click on **Timelist.cbl**; select **Package Selected Files, Executable File**.
5. In the **Define New Executable File** window, it will read Timelist.exe.
6. Toggle off **Include this package file in all build types** (i.e., don't add it to the Debug Build). Press **Create**.
7. Right-click on Timesort.cbl; select **Package Selected Files, Dynamic Link Library**. Again, toggle off **Include this package file in all build types**. Press **Create**.
8. Set the link parameters. Right-click **Timelist.exe**; select **Build Settings...** On the Link tab, select **Static, character, verbose linker output**. (Don't close the window yet).
9. Click on **Timesort.dll** in the Project Build Settings Window. On the Link tab, select **Static, verbose linker output**. Press **Close**. The application will be shipped as statically linked, with timelist.exe as the driver, and a separate timesort.dll.

10. Select **Project, Rebuild All**. The .exe is placed in /netxclass/timelist/release. Execute it at the operating system prompt:
`\NETXCLASS\projects\timelist\release > timelist`

It is not necessary to use the Net Express operating system prompt since the module can execute standalone. You can also use Explorer to find it, then double-click to execute.

Results

Size of shipped application:

In bytes, how large is **timelist.exe**?

In bytes, how large is **timesort.dll**?

Speed of execution:

How many hundredths of a second does **timelist.exe** take to execute?

Using General Debug Build, compile timelist.cbl to a .gnt (Build Settings, Compile, Generated Code). How many hundredths of a second does **timelist.gnt** take to execute?

10.2 Create Release Build using Shared RTS

Introduction

The application CHARCUST allows users to access customer records in an indexed sequential file (CUSTOMER.DAT) to add new records, retrieve existing records, update records, access on-line help for the application. A supplied sub-routine, FILEHAND.CBL, is an I/O module for CUSTOMER.DAT. STAMP.CBL is called to convert the current date for DD/MM/YY to words, using intrinsic functions. HELP.GNT / .OBJ is called to display various help screens within the application. The project Charcust has been debugged and tested and needs to be packaged to be released.

Note: If you are working with a new release of Net Express, you will be prompted to rebuild the project when you open it. Allow Net Express to rebuild the project.

Details

1. Load the project CHARCUST (in x:\NETXCLASS\PROJECTS\CHARCUST). Ensure that the Generic Debug Build is active and run the program CHARCUST. (Try "Big" for Company).
2. Note: The location of Customer.dat is specified by mfextmap.dat, an 80-byte line sequential file. Verify that it points to Customer.dat.
3. Change the Type of Build to **Generic Release Build**, and rebuild the project.
4. By default this will package the application in one .EXE file and link with the shared Run-time System.
5. Test that the application will work outside of the IDE by:
Opening a Net Express Command Prompt.
Changing directory to x:\Netxclass\Projects\Charcust\release.
Typing **CHARCUST** to run the application.

6. The application will run, however at this stage any required Run-time support modules are being picked up from the Net Express system directories. In order to ship this application to an end-user it will be necessary to supply those additional Run-time support modules.
7. It is necessary to package some of the application programs into .DLL files and a Micro Focus Library (.LBR) file, rather than having everything linked together into one large .EXE.

Create a .EXE file

1. Select the .obj files (*Charcust* only) in the Project window.
2. Right-click the file(s) in the Project window and select **Package selected files**: select Executable file (EXE).
3. Type the name of the new .EXE file, followed by **Create**. You may receive a warning because Charcust.exe already exists; overwrite it.

Create a .DLL file

1. Select the files in the Project window (Stamp.obj and filehand.obj).
2. Right-click the file(s) in the Project window and select **Package selected files**.
3. Select Dynamic Link Library (DLL).
4. Type the name of the new .DLL file (filehand.dll), followed by **Create**.

Change the Run-time System

1. Right-click the charcust.exe file in the Project window and select **Build Settings**.
2. Select the **Link** tab followed by the **Static** radio button.
3. Close the **Project Build Settings** window.
4. To verify execution is using your files, erase all the .obj and .int files in the ...charcust\release directory. Re-run the application at the Command Prompt.

10.3 Using the Link Wizard

Introduction

In this exercise you will create the timelist executable using the Link Wizard for both Static and Shared links.

Details

1. Open the **Timelist project** and select **General Release Build** as Type of Build. Select **all the files in the project pane** and delete them; then add the programs back from the source pool.
2. Create an .exe. Select **Timelist.cbl**. With **Ctrl** held down, also select **Timesort.cbl**.
3. Right-click on Timelist.cbl. Select **Package Selected Files, Executable File**. In the Define New Executable File window, it will display Timelist.exe.
4. Toggle off **Include this package file in all build types** (i.e., don't add it to the Debug Build). Press **Create**.
5. Set the link parameters. Right-click **Timelist.exe**; select **Build Settings...** Press the Link tab. Press the **Link Wizard** button.
6. Toggle the options:
 - The program does not use Object COBOL syntax
 - This is not a server-side program
 - This program has a character interface
 - Yes, I would like this program to be a standalone executable
 - No, it is single-threadedPress **Finish**. Review that the settings are correct. Press **Close**.

7. Select **Project, Rebuild All**. The .exe is placed in ...\\timelist\\release.
8. Navigate in Explorer to ...\\timelist\\release. Double-click on **timelist.exe** to execute it.

9. Repeat Steps 5-8 except use the Wizard to build a Shared executable. Unless you use a Dynamic link you will need to copy Cblrtss.dll (the shared run-time) to your project. It is in the Net Express\Base\Bin directory.

10.4 Create a Driver for an INT/GNT Build Type

Using the procedure outlined above, create an IntsandGnts Build Type and use it for this example.

1. Create a new project, Driver. Add Driver.cbl from ...\\IDEPROGS.
2. Add Drsub1.cbl and Drsub2.cbl from ...\\IDEPROGS.
3. Select Drsub1.int and Drsub2.int (not .cbl); right click and select **Package selected files → Micro Focus Library**. Define the New Library File DRSUB1.lbr.
4. Right-click on Driver.int; select Package selected files → Executable file. This is the trigger file.
5. Rebuild all.
6. Right-click on Driver.exe; select Build Settings. Press the **Link** tab, select the **Shared / Dynamic run-time library**. Also, select Interface type: **Character**.
7. Rebuild all. Using Explorer, navigate to ...\\Driver\\IntsandGnts; double-click on Driver.exe.

10.5 Create a Micro Focus Library File

The help files (Help0001.hlp, Help0002.hlp & Help0003.hlp) for the Charcust application (x:\NETXCLASS\PROJECTS\CHARCUST) are to be packaged in a library file.

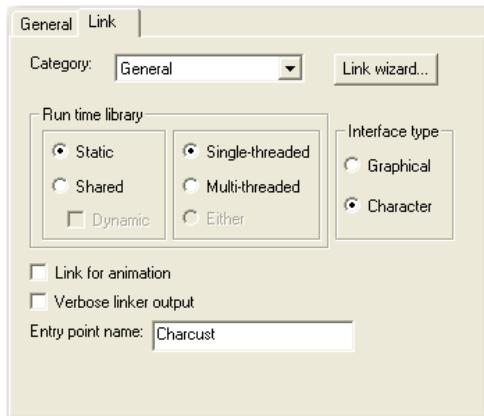
1. Load the CHARCUST project. Be sure that the Generic Release Build is active.
2. Select the three help files in the Source Pool, and package as CUSTHELP.LBR.
3. Drag help.GNT from the Source Pool View to the Project Tree View .
4. Rebuild the project and ensure that the application runs correctly.
5. Test that the application will work outside of the IDE by:
 - Opening a Net Express Command Prompt.
 - Changing directory to x:\NETXCLASS\PROJECTS\CHARCUST\RELEASE.
 - Typing CHARCUST to run the application. **Note:** Although the files are selected within the source pool, the .LBR file will be added into the Project View when packaged.

10.6 Create a New Build Type

In order to package the CHARCUST application as a standalone .EXE file, the build settings need to be changed. The link options need to be changed to link with the Static Run-time system etc. To make the task of switching between packaging with the Shared or Static Run-time system easier, a new Build Type can be created.

1. Load the CHARCUST project
(x:\NETXCLASS\PROJECTS\CHARCUST).
2. Add a new build type: STATIC; base this new build type on the Generic Release Build.
3. Add the description for the new build type: Generic Static Release Build.
4. Change to the Generic Static Release Build. All the .OBJ files will be linked into one .EXE file.

The Build Settings need to be changed to ensure that the Static Run-time system is used when rebuilding the project. The General link options should be changed to the following.



10.7 Build Settings for Generic Static Release Build

Main Program: CHARCUST

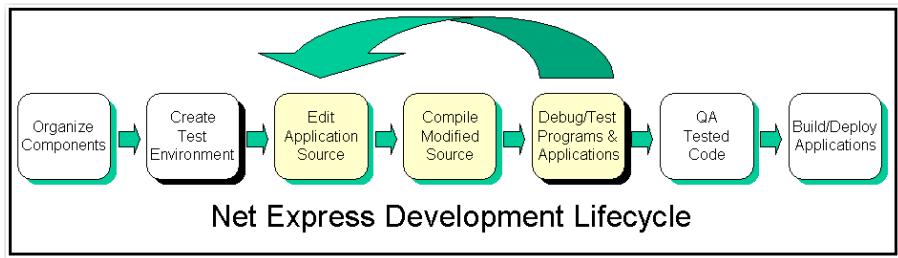
1. Load the CHARCUST project
(x:\NETXCLASS\PROJECTS\CHARCUST). Be sure that the Generic Static Release Build is active.
2. In the Project View window remove the CHARCUST.EXE if present.
Note: Ensure the file is removed from this build type only, and do not remove dependencies.
3. Package all the .OBJ files as CHARCUST2.EXE, with charcust as the entrypoint. **Note:** Ensure the .EXE is created in this build type only.
4. Package the help files (.HLP) in a Micro Focus Library File (.LBR), as before. You can include Help.GNT in the library. Name the library help.lbr.
5. Rebuild the project and ensure that the application runs correctly.
6. Test that the application will work outside of the IDE by:
 - Opening a Command Prompt.
 - Changing directory to x:\Netxclass\Projects\Charcust\static (or whatever you called the new Build type).
 - Typing CHARCUST2 to run the application.

Module 11

Data Tools

Overview

A critical part of creating the test environment involves the preparation of data files. The Net Express Data Tools provide the ability to create files and to modify them before and during debugging and testing. Net Express Data Tools support a variety of data file organizations and conversion between organization types and character sets.



Objectives

At the end of this module you will be able to:

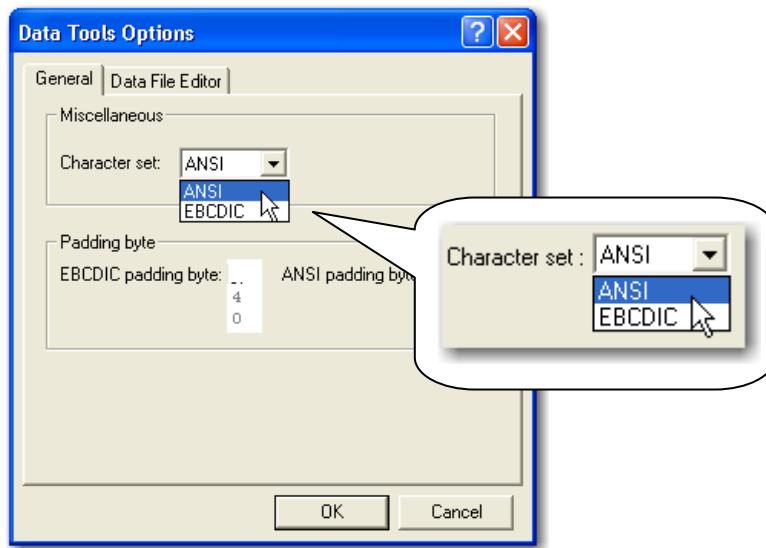
- Use Net Express features for editing, searching, and maintaining test data.
- Create record layout files for formatted editing of data files.
- Edit data in unformatted and formatted modes.
- Create new test data files, including indexed files.

ANSI vs. ASCII

Net Express always uses the 7-bit ANSI (American National Standard Code for Information Interchange) character set, which is based on the industry standard character set. DOS applications, many third-party file transfer utilities, and many previous Micro Focus COBOL systems, save files (including all source files) using extended ASCII. The first half of the extended ASCII character set is identical to the first half of the ANSI character set. Characters in the second half of the extended ASCII character set, however, are determined by the DOS code page in use at the time the file was created, and do not necessarily display correctly when viewed using the ANSI character set.

Files on the PC that use extended ASCII (e.g., source files from Micro Focus Workbench or Net Express v1.0) can be converted to ANSI before using them in Net Express. The MFOEMTOA command-line utility is provided for this purpose.

Mainframe data files use the 8-bit EBCDIC (Extended Binary Coded Decimal Interchange Code) character set. The default character set can be toggled between ANSI and EBCDIC under **Options, Data Tools**.



Data Files

The following types of files, stored on a server or PC, can be edited with the Data File Editor:

Type	Record Length	Features
Sequential	fixed and variable-length records	usual data file organization, length delimited
Indexed	fixed and variable-length records	indexed (e.g. VSAM), length delimited
Relative	fixed and variable-length records	delimited by x'0D0A'
Line sequential		printer files; delimited by x'0D0A' – a carriage-return/line feed.

Header Records

Variable-length sequential and all indexed files have a header record that provides file organization and record length information to the Data File Editor.

File Header record - 128 bytes			
Header	Variable length record - Record 1	pad	0D0A
Header	Variable length record - Record 2		0D0A
Header	Variable length record - Record 3	pad	0D0A

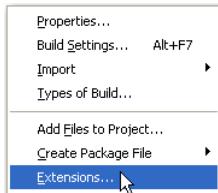
In order to edit a file that doesn't have a header (e.g., fixed-length sequential and line sequential files), you must provide file definition information (i.e., file organization and record length). The Data File Editor will prompt you for this information. You can determine file organization and record length by looking at the SELECT and FD statements in your COBOL programs.

Data File Extensions

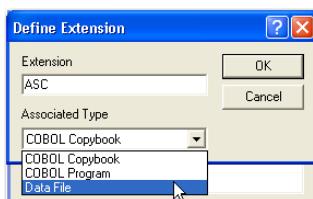
By default, Net Express recognizes data files with an extension of .DAT as data files. Data files with other extensions are considered unrecognized in the Source Pool View. This is not a concern during program execution because a SELECT statement will open any file type specified (e.g. some exercises in this course use an extension of .ASC to help differentiate the character set of the data).

To use the Net Express Data File Editor, however, the new file type must be defined for the project. Follow the steps below:

1. Select Extensions from the Projects menu.



2. Select Add from the User Defined Extensions window.
3. Supply the new file extension (no Dot) and use the drop down to specify Data File.



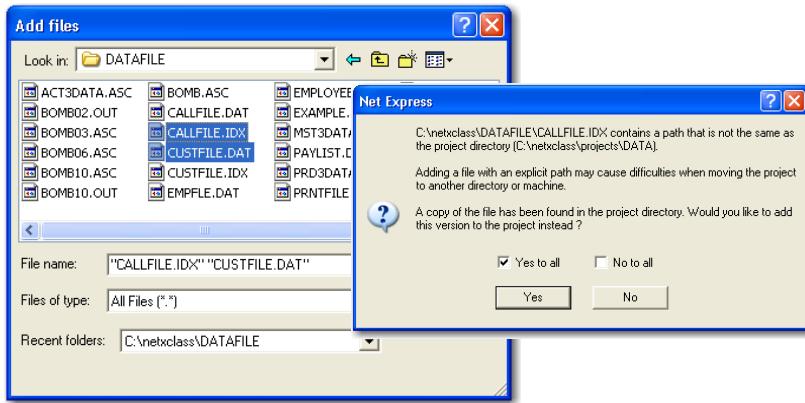
4. The file type will be added in the User Defined Extensions window. Files with that extension will now be treated as valid data files.

User Defined Extensions				
Extension	Associated Type			
ASC	Data File			
MST3DATA.ASC	Data File		Mon Feb 27 15:19:3...	1400
EXAMPLE.ASC	Data File		Mon Feb 27 11:12:2...	656
EMPLOYEE.ASC	Data File		Tue Jul 30 13:25:42 ...	4800
BOMB.ASC	Data File		Mon Feb 27 11:12:4...	704

Adding a Data File to your Project

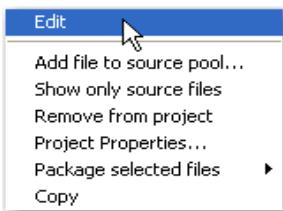
Data files can be added to a project by copying the files from a source directory to the current project directory or can be copied from the source directory when being added as a new file in the Source Pool View.

1. Select **Project, Add Files to Project**. Switch to the source folder.
2. Select **Files of Type: all (*.*)**.
3. Click on **filename.dat** (and **filename.idx**, if indexed). Press **Add**.
4. Click **Yes** to copy the files to the project.



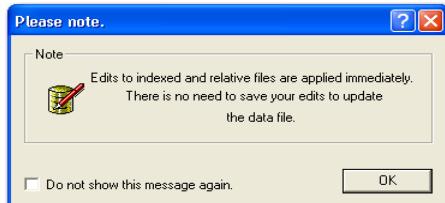
Opening a File in the Data File Editor

You can open a data file in the Data File Editor by double clicking on the filename listed in the Source Pool View or by right-clicking and selecting **Edit**.



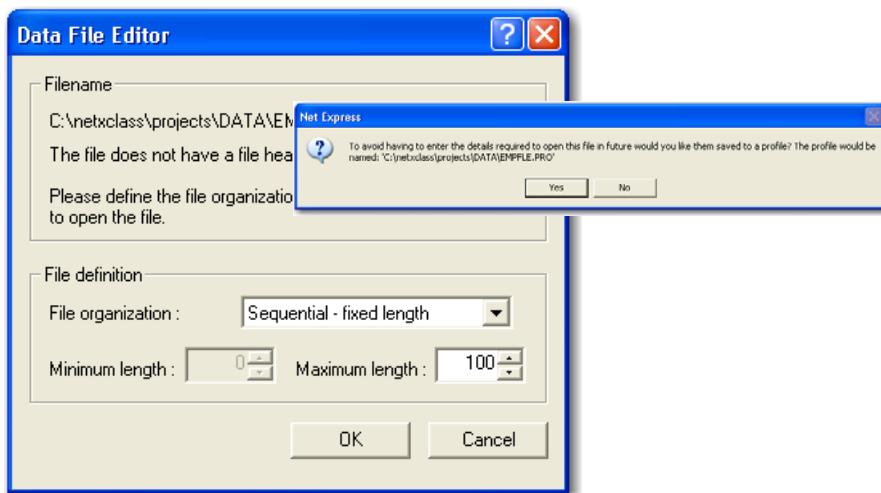
Opening an Indexed File in the Data File Editor

If you have selected an indexed file, a notification is displayed indicating updates will be applied immediately to the file. Optionally, the notification can be turned off for future use.



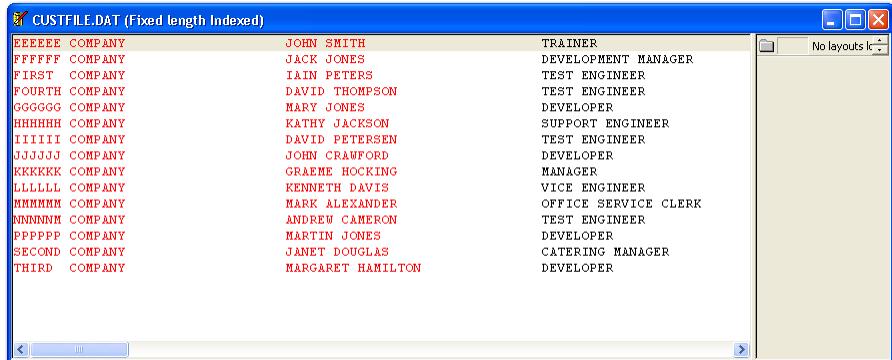
Opening a Non-Indexed File in the Data File Editor

For fixed-length sequential and line sequential files the Data File Editor will prompt you for file organization and record length. Variable-length files have a header record. The Data File Editor will offer to create a Profile file (filename.PRO) that will store the file organization and record length. Because the filename is the same as the data file it will be associated with the data file; if you respond Yes then you will not have to enter this information again.



The Data File Editor Display

The Data File Editor displays data file contents as individual records. For indexed files, primary key fields are shown in red, alternate keys are blue. If applicable, record layouts are listed in the right pane of the window.



EFFFFF COMPANY	JOHN SMITH	TRAINER
FFFFFF COMPANY	JACK JONES	DEVELOPMENT MANAGER
FIRST COMPANY	IAIN PETERS	TEST ENGINEER
FOURTH COMPANY	DAVID THOMPSON	TEST ENGINEER
GGGGGG COMPANY	MARY JONES	DEVELOPER
HHHHHH COMPANY	KATHY JACKSON	SUPPORT ENGINEER
IIIIII COMPANY	DAVID PETERSEN	TEST ENGINEER
JJJJJJ COMPANY	JOHN CRAWFORD	DEVELOPER
KKKKKK COMPANY	GRAEME HOCKING	MANAGER
LLLLLL COMPANY	KENNETH DAVIS	VICE ENGINEER
HHHHHH COMPANY	MARK ALEXANDER	OFFICE SERVICE CLERK
NNNNNN COMPANY	ANDREW CAMERON	TEST ENGINEER
PPPPPP COMPANY	MARTIN JONES	DEVELOPER
SECOND COMPANY	JANET DOUGLAS	CATERING MANAGER
THIRD COMPANY	MARGARET HAMILTON	DEVELOPER

Viewing the Details of a Data File

Right-click anywhere in the data view window and select **File Information**.



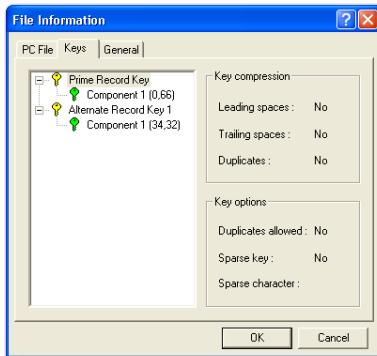
The file information displayed on the tabs is defined below.

PC File Tab Information

- File format
- Organization
- Record format and length
- Compression (if any)
- Created date-timestamp
- Size

Keys Tab Information

Displays primary and alternate keys, with offsets from 0 and field length.
Also shows key compression and option stats.



General Tab Information

- Character set (ANSI or EBCDIC)
- Backup file name and location

File Maintenance – Records

Inserting, Deleting, and Repeating Records

You can insert, repeat, and delete records in either of these ways.

- Right-click on a record and select the option you want from the context menu
- Click on a record and select **Edit, DataTools**; select an option.



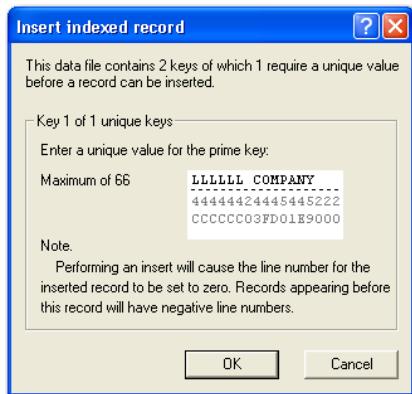
Insert Record Before/After

Inserts a blank (initialized) record before or after the current record.

Warning: When editing unformatted files (editing without a Record Layout) the Data File Editor does not know where numeric or COMP- fields are located so the record is initialized to spaces.

Insert Indexed Record

An indexed file requires a unique primary key. You will be prompted to enter a key value.

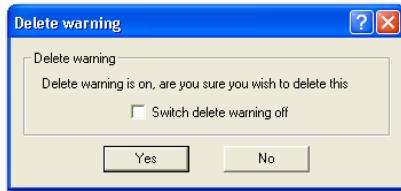


Repeat Record

Repeats the current record. For unformatted files this is a handy way to obtain a correctly formatted record (rather than using Insert Record). Indexed files will prompt for a unique key.

Delete Record

Deletes the current record. By default, a delete warning will be displayed that can optionally be switched off. You cannot undo a delete.



Initialize Record

Unless you are editing formatted files, that is, using a Record Layout this will clear the record to spaces.

Undo Record Edit

If you change the contents of a single record and DO NOT MOVE OFF THE RECORD, you can undo the changes. Warning: Once you move off the record the changes cannot be undone.

Multi Repeat Records

Repeat the selected record multiple times. You will be prompted for the number of repetitions. This option does not apply to indexed files that require unique keys.



Multi Delete Records

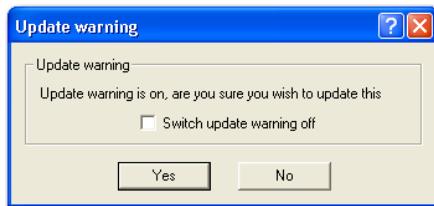
Delete multiple records below the selected record. You will be prompted for the number of records to delete.

File Maintenance - Editing an Individual Record

Before you begin to change data within an existing record, remember the purpose of the Insert key. It controls whether your changes will overtype existing characters, or will be inserted in front of them. By default, the Data File Editor starts out in Overtype mode. If you want to change this, click the Data File Editor tab of the **Options, Data Tools** menu and uncheck Always

overtype in unformatted mode. **Warning:** Typing in unformatted Insert mode will alter your data's field positions.

Click on a record and start typing to change a record. The first time you start to type you will receive a warning message (which you can turn off). For indexed and relative files, the record is saved each time you move off it.



Editing in a Different Key Order

When an indexed file is opened, a dockable Data File toolbar is displayed. The default key order for an indexed file is the Prime Key. If the file has alternate keys, you can reset the record sequence by selecting the key from a drop down list.



Editing Hexadecimal Data

Hex mode is most appropriate when data includes COMP fields or other unprintable characters. Right-click on a record. Choose **Show Hex**. The record is displayed in hex. You can also use **View, DataTools, Hex (Alt + V, D, H)**. You can edit either the text or the hex values.

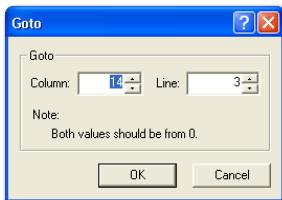
	JOHN SMITH	JACK JONES
FFFFEE COMPANY	FFFFEE COMPANY	FFFFEE COMPANY
FFFFFF COMPANY	FFFFFF COMPANY	FFFFFF COMPANY
FIRST COMPANY	IAIN PETERS	DAVID THOMPSON
FOURTH COMPANY	DAVID THOMPSON	MARY JONES
GGGGGG COMPANY	MARY JONES	KATHY JACKSON
HHHHHH COMPANY	KATHY JACKSON	
IIIIII COMPANY	DAVID PETERSEN	DAVID PETERSEN
JJJJJJ COMPANY	JOHN CRAWFORD	KENNETH DAVIS
LLLLLL COMPANY	KENNETH DAVIS	MARY ALEXANDER
MMMMMH COMPANY	MARY ALEXANDER	ANDREW CAMERON
NNNNNN COMPANY	ANDREW CAMERON	MARTIN JONES
PPPPPP COMPANY	MARTIN JONES	JANET DOUGLAS
SECOND COMPANY	JANET DOUGLAS	MARGARET HAMILTON
THIRD COMPANY	MARGARET HAMILTON	
IIIIIZZI COMPANY	DAVID PETERSEN	TEST ENGINEER

Searching

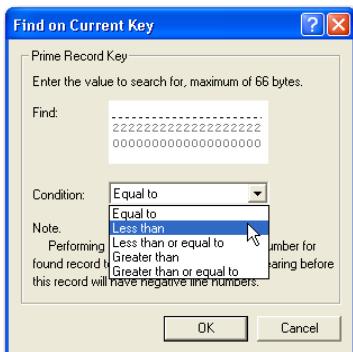
You can use the Search, Data Tools menu for searching data files. The **Data File Find and Replace** option works like the source file Search. Data file finds are done one at a time and a Replace All button is available.



The **Goto** option lets you specify a Line (record number) and Column number to jump to. Note: Both these values are offsets from 0. (Line numbers can be negative for indexed files after an insert or a search).



The **Find on Indexed Key** option will take you directly to the record containing the key specified. The search is case sensitive. The search looks for the leftmost character of your find string starting in the leftmost byte of the key field. You can also search for conditions where the searched for key is greater/equal or greater or less/equal or less than the entered search value.



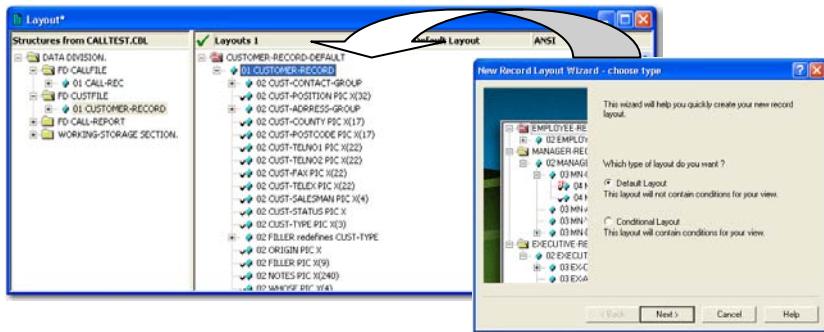
Formatted Editing using Record Layouts

The Data File Editor uses a files Record Layout to display field names, formats, and data values. Some files contain COMP- data or multiple record types (a default plus one or more conditional record layouts). Record Layouts make editing these files easier.

Creating a Record Layout File

To create a Record Layout you need an .IDY file from a General Debug compile of a program that contains the record layout.

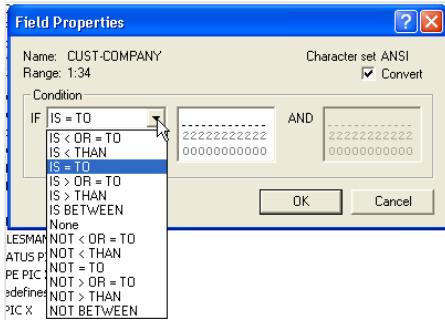
1. Select **File, New, Record Layout**.
2. Navigate to the folder containing the IDY file (the project DEBUG folder) select a .IDY file, and click **Open**.
3. Right-click on the group level that describes your default record layout; choose **New Record Layout**.
4. Choose **Default Layout** and click **Next**; click **Finish**.



Multiple Record Types

1. If the data file contains more than one record layout, right-click on the next group level that describes the conditional record layout. Select **New Record Layout**.
2. The New Record Layout Wizard pops up with Conditional Layout selected. Press **Next**. Press **Finish**.

3. Fully open (+) the conditional Record Layout. From the list of elementary items, double-click the field that serves as the switch for using this layout. The Field Properties window pops up.



4. From the Condition drop-down, select the condition for this switch (e.g., IS = TO). Enter the value that, when true, indicates this layout describes the data. Press **OK**.

Saving the Record Layout

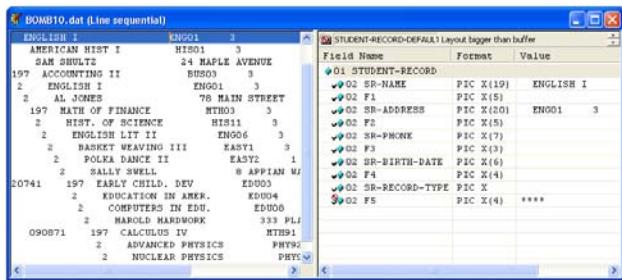
Select **File, Save**; save the file (filename.str) into your project directory (.str indicates a structure file).

Editing a Data File: Formatted Editing

You can now do formatted editing in the right-hand pane of the edit window. This is the easiest and safest way to work with fields for example changing values. It is safer because you can only make changes that are consistent with the definition of the field. For example, you cannot:

- Type non-numeric characters in a numeric field
- Type too many characters

To edit a data item, click on its Value and start typing. The context menu is available in both panes. If you insert or initialize a record it will be formatted correctly (e.g., numeric fields will be initialized to zeroes).



Editing/Viewing the Record Layout

You can add the STR file to the project. Once loaded, you can double-click on it to edit it. You can also open it by using the File, Open menu. With a multiple record-type file, you can delete as with the single record-type file. You can also change the properties by expanding until you see the record type entry. Right-click on the record type entry and select Properties.

Creating New Data Files

Create a Non-Indexed File

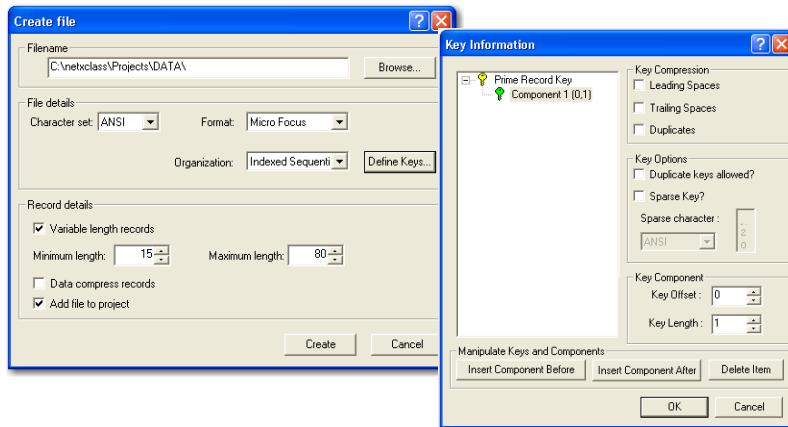
Select **File, New, Data File**. You will be prompted for the information to create the new data file. Initially the new file is empty. Right-click in the edit window and choose **Insert Record After**. Thereafter Repeat Records and the other options are available.

Creating an Indexed File

1. Select **File, New, Data File**. You will be prompted for the name, organization, and record length for the new data file.
2. Choose Indexed as the file organization.
3. After defining the record length, define the keys: press Define Keys.
4. Press Insert Key. For Key Information, enter the starting position (from 0) and key length. **Note:** Net Express supports split keys consisting of components.

5. If the primary key is complete, press **OK**. This returns you to the previous window.
6. To create an alternate index, press Define Keys again. Press Insert Key After.
7. Again, enter the Key offset and length. If complete, press **OK**.
8. When you have completed defining keys, press Create. The Data File editor will open on the new, empty file.

Right-click and select Insert Indexed Record. You will need to supply primary key and alternate key values.



Converting Files and Fixing Indexes

Data File Convert allows you to convert between data file formats. To convert a file select **Tools**, **DataTools**, **Convert**. The window is divided into three parts: Input File, Character Set Conversion, and New File.

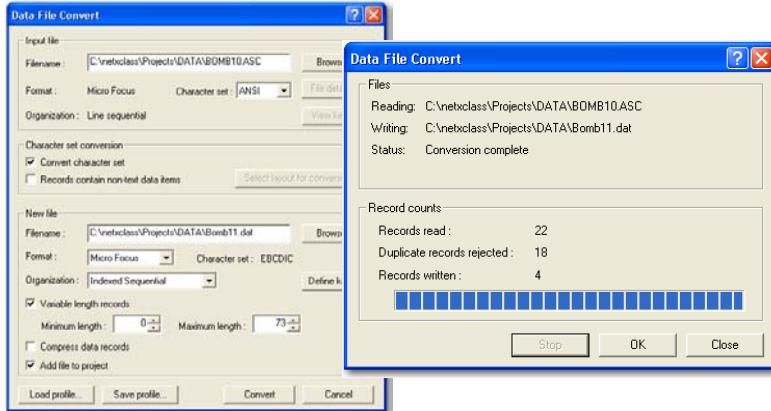
Input File--Use Browse to locate the file input to the conversion. Select its character set (ANSI/EBCDIC). If indexed you may View Keys.

Character Set Conversion--Check Convert character set if you are converting a file from EBCDIC to ANSI or ANSI to EBCDIC. If you check Convert then you can check that the records contain non-text (e.g., COMP)

data items. Non-text data items require a layout for correct conversion – the conversion proceeds byte-by-byte and will convert COMP data incorrectly without a Record Layout.

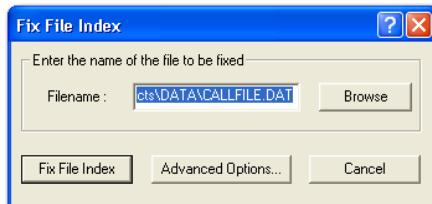
New File--Provide a new Filename. When converting to an indexed file, it is suggested that a .dat extension be supplied. The *filename.idx* index will be created for you. If the file is indexed you need to define keys.

Press **Convert**. The Conversion process will execute and provide a report window with the results. Close the Convert window. Examine the results in the Data File Editor. For conversions that you will repeat, you can save the conversion parameters as a Profile (.PRO). The next time you need to convert the file, select **Tools, DataTools, Convert, Load Profile**.



Fixing File Indexes

To fix a corrupt index, select **Tools, Data Tools, Fix File Index**. You only need the data part of the file. For missing or corrupt index, use advanced options to define the keys.



Module Summary

The Net Express Data File Editor is an important part of the Net Express Development Lifecycle – both in Creating the Test Environment and for setting special conditions in Debugging and Testing.

The Data File Editor is used to create/edit/maintain sequential, indexed, relative, or line sequential data files.

The Data File Editor simplifies the process of viewing the details of a data file.

The Data File Editor provides search/replace capabilities.

11.1 Creating a Record Layout File

In this exercise, you will create a Record Layout for a data file that contains multiple record types.

1. Create a new Project called **Bomb10**. Add **Bomb10.cbl** from \DEPROGS. Add **Bomb10.asc** from \DATAFILE.
2. Use the **Extension** option from the **Project** menu to add .asc file types, as data files, to the project.
3. Using **General Debug Build**, compile Bomb10.cbl to create the .idy file in the \DEBUG directory.
4. Select **File, New, Record Layout**. Navigate to \DEBUG, choose **bomb10.idy**, and click **Open**.
5. Right-click on Student-Record (the default record layout) and choose **New Record Layout**. Choose **Default Layout** and click **Next**; click **Finish**.
6. Since the data file contains more than one record layout, right-click on Course-Record (the conditional record layout). Select **New Record Layout**.
7. The New Record Layout Wizard pops up with Conditional Layout selected.
8. Press **Next**. Press **Finish**.
9. Fully open Course-Record. Double-click the switch field, RECORD-TYPE, The Field Properties window pops up.
10. From the Condition drop-down, select **IS = TO**, then enter **2** as the value. Press **OK**.
11. Select **File, Save**; save the file as bomb10.str in the project directory; close the window.
12. Right-click bomb10.asc; select Edit. The file is sequential – fixed length, and the maximum length is 74. The file will open in formatted mode. As you move through the file, the record layout will change according to the value in RECORD-TYPE.

11.2 Create a New Project with Data Files

1. Create a new project DATATEST in the folder
x:\NETXCLASS\PROJECTS\DATATEST.
2. Add the following files to the project making sure that the files are copied to your project directory:

File Type	File Name	File Folder
COBOL files	DFILE1.CBL DFILE2.CBL	\NETXCLASS\IDEPROGS \NETXCLASS\IDEPROGS
Plain Text file	DFILE1.TXT	\NETXCLASS\DATAFILE
Data file	DFILE2.DAT	\NETXCLASS\DATAFILE

3. Look at the TXT file in the editor, you will see it contains some record data. We will be using this to create a data file.
4. Build the project to ensure that the programs have compiled cleanly.

11.3 Convert Text Data to produce an Indexed File

To convert DFILE1.TXT to DFILE1.DAT:

1. Select **Tools, Data Tools, Convert.**
2. Do NOT fill in the filename; instead, press **Browse**.
3. Locate x:\NETXCLASS\PROJECTS\DATATEST. Use Files of Type: All files (*.*).
4. In the File Details window, pull down File Organization and select **Line Sequential**.
5. Set the Maximum record length to 39. (Line sequential files are always considered variable length). Press **OK**.
6. For the new file name, enter DFILE1.DAT. The format is Micro Focus (the default).
7. Select Organization: Indexed Sequential
8. Turn off Variable length records.
9. Max length should be 39
10. Click **Define Keys.**
11. Click **Insert key.**
12. In the Key Component window, set the primary key offset to: 0
13. Set the key length to: 5
14. Click **OK.**
15. Click **Convert.**
16. When conversion is completed successfully click **cancel** to close Data File Convert.
17. Double-click this file to edit it (notice the key highlighting).
18. Use the context menu to add another few records to this file.

11.4 Creating and Using Record Layouts

1. Select **File, New, Record layout** and click **OK**.
2. In the Debug directory open the DFILE1.IDY file.
3. Right-click on the data item DATA-FILE-ONE and click on **New Record Layout**.
4. Make sure that Default Layout is selected; press **Next**, press **Finish**.
5. Select File, Save As; save the file DFILE1.STR into the directory above Debug (i.e., Datatest).
6. Close the Record Layout window.
7. Check that you are still editing the data file DFILE1.DAT. Select File, DataTools, Load Record Layouts and open the file DFILE1.STR. Notice how the view of the file has now changed.
8. Add a few new records to this file using the default record layout (not padding bytes). Edit some of the records, both in record view mode and structure view mode.
9. Close DFILE1.DAT.

11.5 Creating and Using Conditional Record Layouts

1. Select **File, New, Record layout** and click **OK**.
2. In the Debug directory among the .idy files, open the DFILE2.IDY file.
3. Right-click on the data item ORDER-RECORD and click on **New Record Layout**.
4. Make sure that Default Layout is selected and click on **Next**; press **Finish**.
5. Right-click on the data item PAYMENT-RECORD and click on New Record Layout. PAYMENT-RECORD is a conditional record layout. Whenever Record-type = P then use this record layout.
6. Make sure that Conditional Layout is selected and click on **Next**, press **Finish**.
7. In the right-hand pane expand the PAYMENT-RECORD until you find RECORD-TYPE.
8. Right-click on the field RECORD-TYPE and click on Properties.
9. Pull down the Condition list box; select IS = TO with the value P and click OK. Note that this value is case sensitive – use upper case (x'50').
10. Back in the left-hand pane, right-click on CANCELLATION-RECORD; press New Record Layout.
11. Make sure that Conditional Layout is selected and click on **Next**. Click on **Finish**.
12. Define the RECORD-TYPE switch's condition as IS = TO C
13. Save the DFILE2.STR file.
14. Close the Record Layout window.
15. Double-click the data file DFILE2.DAT. You will be taken into the data file editor and the default view will be the formatted view. This is because the .STR file is present in the same directory as the file, with the same name.

16. Add some new records to this file using all three record types.
17. Right-click the mouse; select Synchronize. What does it do?
18. Save this file using the File, Save menu and close the file.

11.6 Converting to an Index file with Alternate Keys

1. Select **Tools, Data Tools, Convert.**
2. Define the file DFILE1.DAT to be the input file. The converter recognizes it as an Indexed Sequential file.
3. Define the new file DFILE3.DAT to be the output filename.

Format	Micro Focus.
Organization	Indexed Sequential
fixed length	(Turn off Variable length records)
max length	39

4. Define keys (the default prime key is already set).
5. Insert key after: offset is: 5 with a length of 10 (customer surname), with duplicate keys allowed.
6. Click **OK.**
7. Click **Define Keys** once more to define a second alternate key.
8. Click on Alternate key 2, then click **Insert key after;** key offset is 23 with a length of 2 (country); duplicate keys allowed.
9. Click **OK.**
10. Set the check mark **Add file to project** and click **Convert.**
11. Close the Data File Convert window.
12. Double-click DFILE3.DAT to open.
13. View this file ordered by prime key, then by alternate key #1, and then alternate key #3
14. Select **File, DataTools, Load Record Layouts;** select DFILE1.STR. This layout applies to DFILE3.DAT as it did to DFILE1.DAT.
15. Right-click; add a new record to the file. You will be prompted for the prime key. Why were you not prompted for the alternate keys?
16. Close the data file editor window.

11.7 Using the Data Tools

As part of the payroll project, an employee master file has been downloaded from a mainframe to \NETXCLASS\DATAFILE\EMPL.DAT. The file is in the EBCDIC character set, and it contains packed or binary data (i.e., non-text data). It is sequential, with fixed-length, 100-byte records. You are to convert this file to ANSI for use with Net Express. In order to convert and edit this data file you will need to create a .STR file which describes it – this will prevent the File Convert utility from creating invalid data from the COMP or COMP-3 fields. The data file's record layout is defined in \NETXCLASS\OTHERS\EMPLCOPY.CPY. This copybook is used in the FD of \NETXCLASS\IDEPROGS\EMPL.CBL. Place the converted data file (also to be named EMPL.DAT) back in the \DATAFILE directory. Animate EMPL.CBL; it should execute correctly.

Optional Steps

1. Once the file is converted, edit it.
2. Scroll to the bottom, and then back to the top. Scroll to the far right, then back to the left.
3. Insert one new record before the first record in the file, and enter some data. Insert one new record after the last record in the file, and enter some data.
4. Repeat one record and change some of its fields.
5. How many records contain FIN anywhere within them?
6. Change JULIAN B. DATE's middle initial to S. Were you warned?
7. Delete one of the records you inserted earlier. Are you prompted?
8. Now we find out that JULIAN DATE really doesn't have a middle initial. First, make sure that you've in Overtype Mode (OVR appears in the status bar), and delete the middle initial. What are the risks of deleting (and inserting) characters when you're working in unformatted edit mode?

9. The 4 bytes that you viewed in Hex mode earlier are the salary. What is the salary of the first employee in the file? Give that employee a \$10 raise (not 10 cents ... can you easily tell exactly how much you're giving?)
10. Close the Data File Editor.

Module 12

OpenESQL Assistant

Overview

Net Express provides COBOL programmers with full support for accessing ODBC-enabled data sources via OpenESQL, a SQL preprocessor.

Objectives

At the end of this module you will be able to:

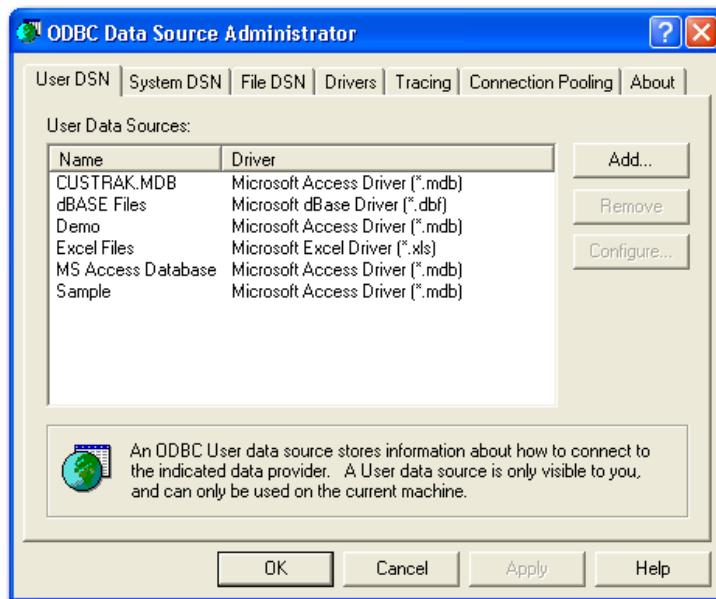
- Use SQL syntax and data types.
- Connect to Databases.
- Compile Programs with Embedded SQL.
- Navigate through OpenESQL Assistant.
- Generate code to declare and use cursors, select a Table and Query, select Columns, specify Search Criteria, create and execute Table Joins.
- Run a query.
- Add Embedded SQL to a COBOL Program (Define a Query, Insert Query into Current Program, Define a CONNECT Statement, SQLCA Declaration, Define Host Variable Declarations).

ODBC Interface

Open Database Connectivity (ODBC) is an Application Programming Interface (API) enabling applications to access data in databases (referred to as data sources) that use Structured Query Language (SQL). ODBC manages this by inserting a middle layer, called a database driver. Examples include Microsoft SQL Server, Microsoft Access, IBM DB2, Oracle, and Sybase.

ODBC Drivers

Data source drivers can be added manually through the ODBC Administrator Driver List, through the OpenESQL Assistant, or through a batch command. (Refer to Appendix A at the end of this Module for setup guidelines).



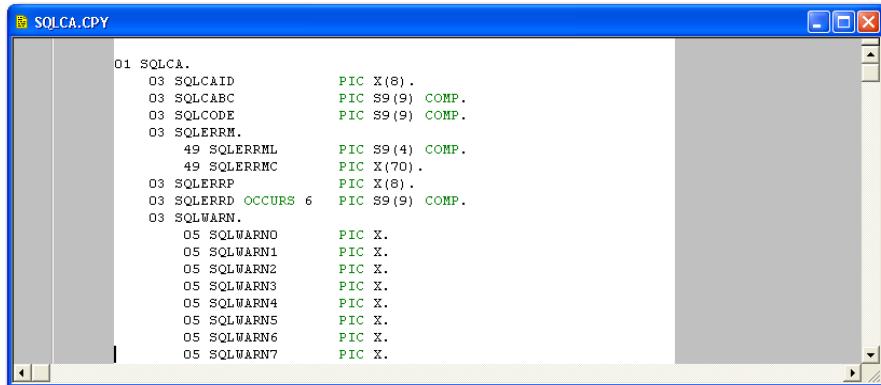
SQL in Net Express COBOL Programs

SQL statements can be embedded as part of the source code for an application. The source code is compiled and linked with the appropriate COBOL facilities. The SQL statements are then executed in the context of

the application. COBOL programs containing SQL syntax must be compiled with the SQL Compiler directive.

SQLCA

After embedded SQL statements are executed, error and status information is returned in the SQLCA. The SQLCA data structure is included in the file SQLCA.CPY in the SOURCE directory under MFSQL in the Net Express base installation directory.



The screenshot shows a window titled "SQLCA.CPY" containing COBOL source code. The code defines a record structure for the SQLCA (SQL Communication Area). It includes fields for SQLCAID, SQLCABC, SQLCODE, SQLERRM, SQLERRML, SQLERRMC, SQLERRP, SQLERRD (which occurs 6 times), and SQLWARN (which occurs 7 times). Each field is defined with its type and length.

```
01 SQLCA.
  03 SQLCAID          PIC X(8).
  03 SQLCABC          PIC S9(9) COMP.
  03 SQLCODE           PIC S9(9) COMP.
  03 SQLERRM.
    49 SQLERRML        PIC S9(4) COMP.
    49 SQLERRMC        PIC X(70).
  03 SQLERRP           PIC X(8).
  03 SQLERRD OCCURS 6  PIC S9(9) COMP.
  03 SQLWARN.
    05 SQLWARN0         PIC X.
    05 SQLWARN1         PIC X.
    05 SQLWARN2         PIC X.
    05 SQLWARN3         PIC X.
    05 SQLWARN4         PIC X.
    05 SQLWARN5         PIC X.
    05 SQLWARN6         PIC X.
    05 SQLWARN7         PIC X.
```

OpenESQL generates the INCLUDE SQLCA statement for inclusion in your program.

Host Variables

These variables are data items defined within a COBOL program. They are used to pass values to and receive values from an ODBC data source.

OpenESQL Assistant generates the INCLUDE statement for the DCLGen copybook.

Connecting to Data Sources (Databases)

Before a program can access data in a database, it must make a connection to the database. The CONNECT statement attaches to a specific database with the user name and password. Refer to Appendix A at the end of this module for coding requirements.

Methods to Connect to a Database

You can use either of two methods to connect to a database:

Explicit connection (recommended method)—Use the CONNECT statement if the program is designed to access different data sources whose names are not known at compilation time or if the program is going to access multiple databases.

Implicit connection—Use this method if the program is only going to connect to one database that is known at compilation time. If you specify the INIT option of the SQL Compiler directive, the compiler inserts a call at the start of the program. This call automatically connects the program to the data source specified in the DB option of the SQL Compiler directive. It uses the login information specified in the PASS option.

Note: Data Source Names must NOT contain spaces. When an application has finished working with a database, it should disconnect from the database. Do this by using the DISCONNECT statement. If implicit connection is being used, OpenESQL automatically disconnects from the data source when the program terminates.

SQL Compiler Directive

The SQL Compiler directive enables COBOL programs to use Embedded SQL for ODBC. There are two ways of specifying options for the SQL Compiler directive:

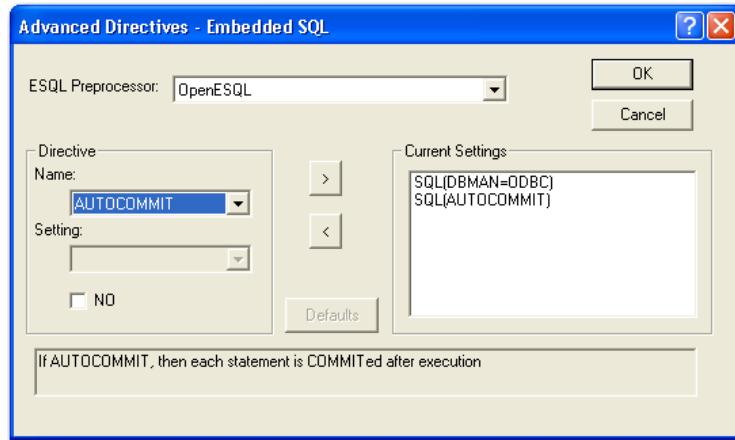
1. Code the directive at the top of the program.

```
$SET SQL(DBMAN=ODBC) SQL(AUTOCOMMIT)
IDENTIFICATION DIVISION.
PROGRAM-ID. CONVCUST.
```

2. Include with build settings. The steps follow:

- a) Right-mouse the .CBL in the Project View pane; select **Build Settings...**. Click on the **Compile** tab, then press **SQL Directives**.

- b) Pull down the **ESQL Preprocessor** list box; select **OpenESQL**.



- c) In the Directive group, pull down the **Name** list box; select required directives (DBMAN first, then AUTOCOMMIT). Press the **>** arrow to add each directive to the Current Settings list.

OpenESQL Assistant

Net Express provides COBOL programmers with full support for accessing ODBC-enabled data sources via OpenESQL, a SQL preprocessor. This enables developers to embed SQL statements directly into an application. There is no need for a separate pre-compiler or for the programmer to be familiar with database APIs.

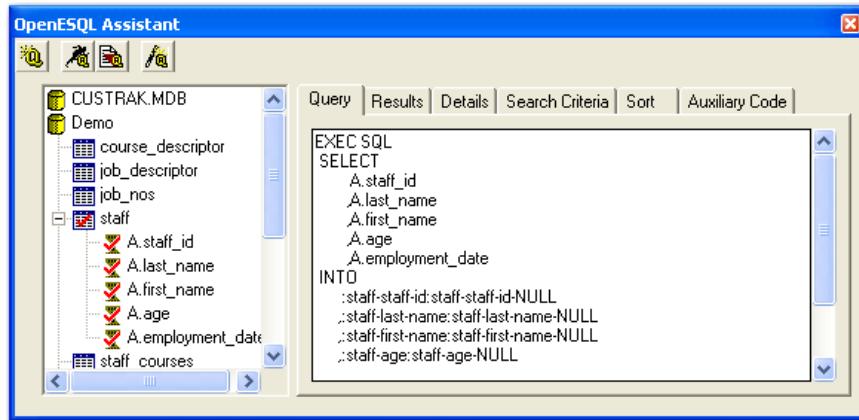
The OpenESQL Assistant provides COBOL programmers a point-and-click interface to build SQL statements.

The OpenESQL Assistant provides support for the following:

- Connect to a data source / Disconnect from a data source
- Select a table / De-select a table
- Select a column / De-select a column

- Select all the columns in a table
- Display column details (Type, Precision, Host Variable, etc.)
- Create a new query
- Run a variety of SELECT queries plus Update, Insert, and Delete
- Create and test a Cursor
- Specify search criteria (WHERE clause for the SELECT statement)
- Create and test a table join
- Add auxiliary code to a COBOL program (e.g., SQLCA, host variable definition).

These statements can be inserted into the COBOL source code at the appropriate point. The SQL statement can be tested in the OpenESQL Assistant and can be modified as needed without having to build and run the application.



OpenESQL Assistant Functionality

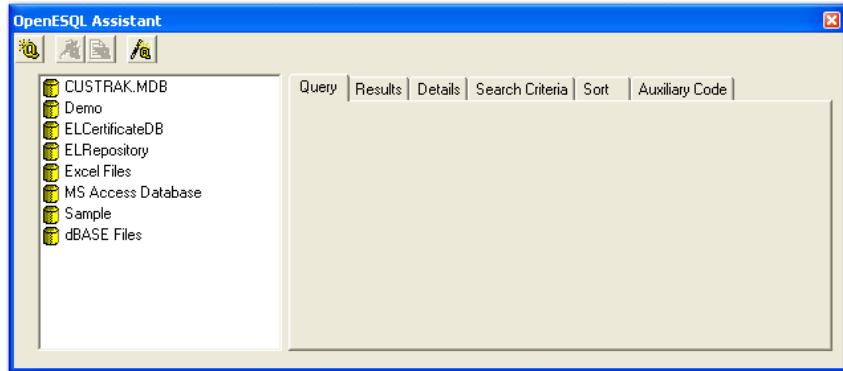
The OpenESQL Assistant action icons and tabs are described in the table that follows.

Action Icons	Description
	Create a new query
	Run the query on the query tab
	Insert query into current program
	Refresh list of available data sources
Tabs	Description
Query	Displays the current query
Results	Results of running the query
Details	Query column details: data type, precision, etc.
Search Criteria	A column search for specified a value limiting the query
Sort	Sort column data in the SELECT
Auxiliary Code	Database access code to include in the program

Starting the OpenESQL Assistant

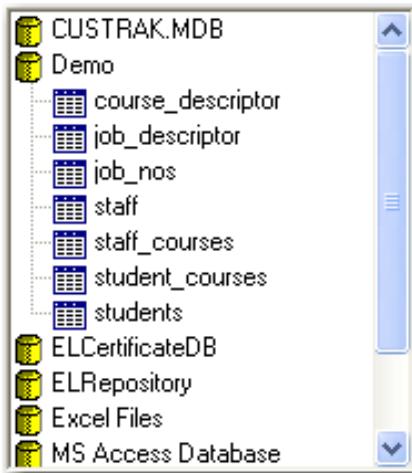
Use one of the following options to start the OpenESQL Assistant.

Select Tools, OpenESQL Assistant or View, Dockable Windows, OpenESQL Assistant. A list of data sources to connect to appears in the left panel.



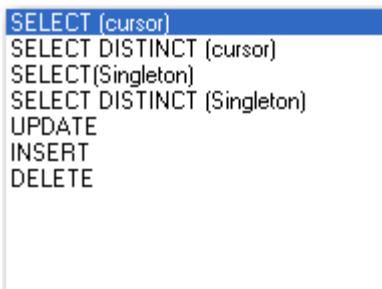
Connect to Data Source

To connect to a data source double-click on its name or icon. After connecting to a data source, the names of all the tables are displayed.



Selecting a Table and Building a Query

To select a table, double-click on its name. A prompt to select the type of query required is displayed.



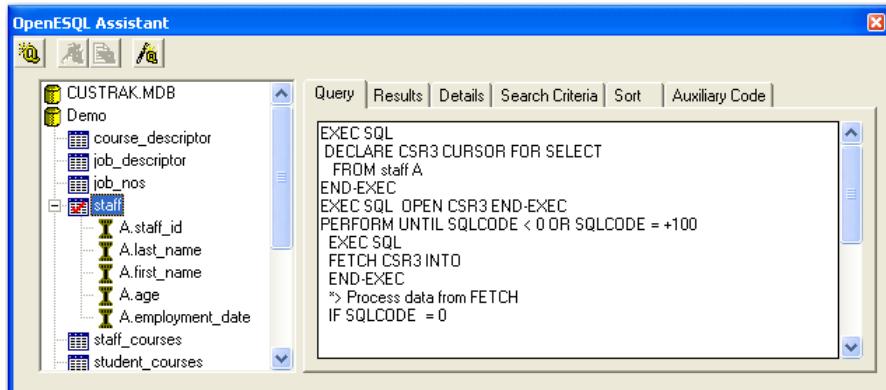
The query types are described in the table below:

SELECT (cursor)	Creates code to retrieve multiple rows of data (DECLARE, OPEN, FETCH)
SELECT DISTINCT (cursor)	Creates code to retrieve unique rows (i.e., remove duplicate rows)

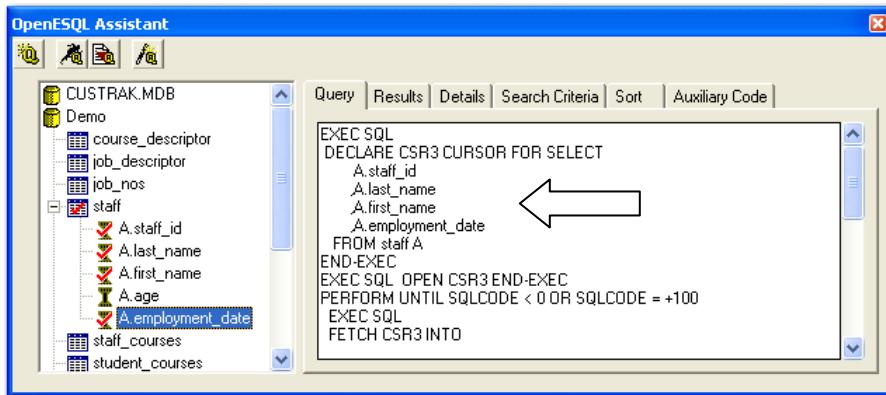
SELECT (Singleton)	Creates code to retrieve one row of data
SELECT DISTINCT (singleton)	Creates 'DISTINCT' code to retrieve 1 row of data.
UPDATE	Creates code to update one or more rows of data
INSERT	Creates code to insert one or more rows of data
DELETE	Creates code to delete one or more rows of data

The COBOL code for the selected query is generated and displayed under the Query tab. A list of all the columns in the table displays.

The OpenESQL Assistant generates an alias for the table. The first table selected is assigned A, any subsequent tables would be assigned B, C and so on. Each column name is prefixed by its alias (A.Company, A.Contact), so column names can be distinguished from those in another table.



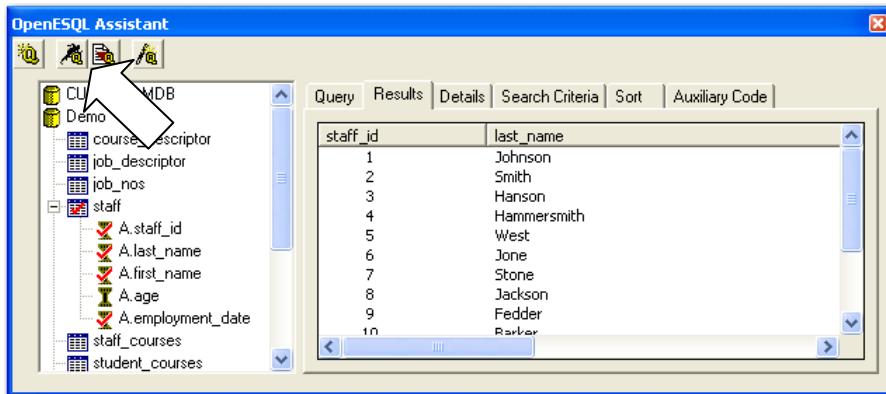
Select the columns required for the query by double-clicking on the column name. Alternatively, select all columns, right-click the table name (e.g., **Customer**) and select all columns.



The query will be updated to include the selected columns.

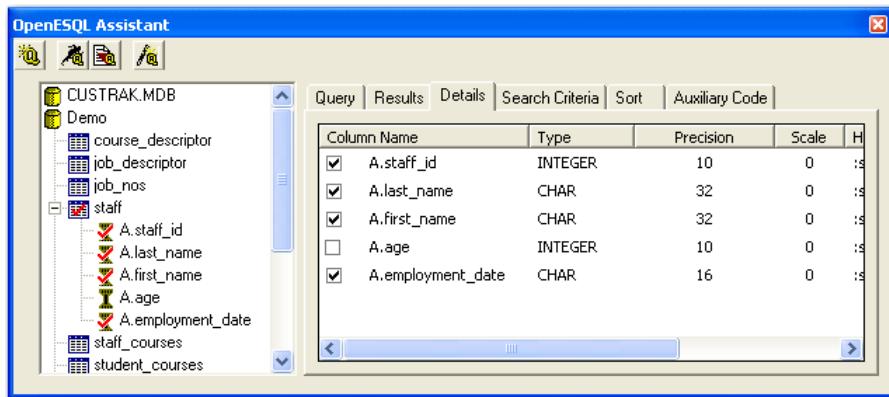
Running a query

To run a selected query, select the **Run Query** button. The results of the query will appear in the Results tab.



Query Details

Details of the table fields are displayed on the Details tab. Checkbox indicators are provided to indicate active fields in the current query. Details include: Column Name, Type, Precision, Scale, Host Variable and Indicator Variable.



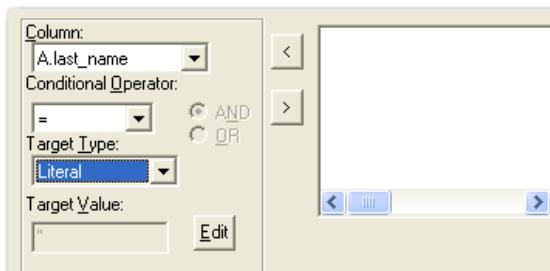
Specifying Search Criteria

The number of rows that are returned by a SELECT statement, can be limited by specifying search criteria (a WHERE clause); use the **Search Criteria** tab.

Column Name and Conditional Operator

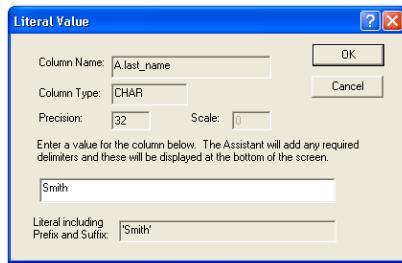
To specify search criteria:

1. Select a column name from the **Column** list box.
2. Select a conditional operator: =, <>, >, >=, <, <=, LIKE, NOT LIKE, IS NULL, IS NOT NULL.



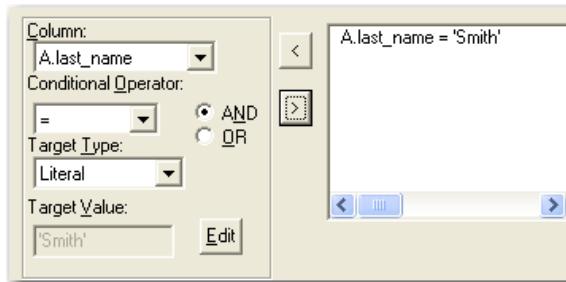
Specify Target Type and Target Value

1. Select a **Target Type**. The target type must be one of the following: Host variable, Literal, Special register, Column name.
2. Select, or enter, a **Target Value**. If a column name or host variable is selected as the target type, a list of all the valid column names or host variables are displayed in the target value list box. If a literal or special register is selected, press the **Edit** button to the right of the target. Enter a value into the target value entry field.



Update Query Window

After the search criteria have been entered, click on the right-pointing arrow button. This moves the search criteria across into the right-hand pane and updates the COBOL code in the query window:

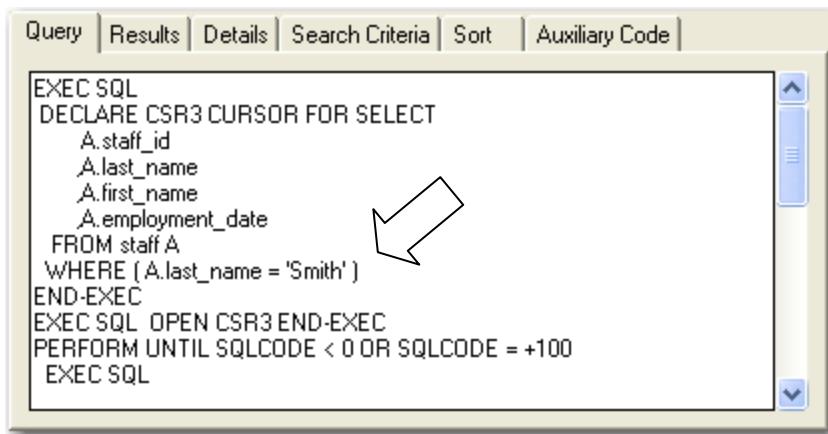


Delete Query Criteria

To delete a query criteria, click it and press the left-pointing arrow.

Check Query Results

Check the resulting Query using the the Query tab. The where clause will be added to the query code.



```

Query | Results | Details | Search Criteria | Sort | Auxiliary Code |

EXEC SQL
DECLARE CSR3 CURSOR FOR SELECT
    A.staff_id
    A.last_name
    A.first_name
    A.employment_date
FROM staff A
WHERE (A.last_name = 'Smith')
END-EXEC
EXEC SQL OPEN CSR3 END-EXEC
PERFORM UNTIL SQLCODE < 0 OR SQLCODE = +100
EXEC SQL
  
```

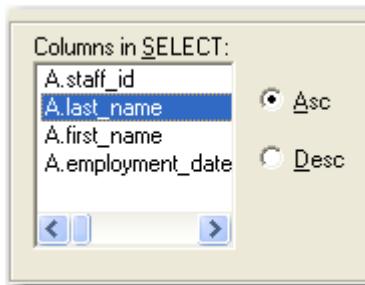
Sort

The rows that are returned by a SELECT statement, can be sorted by specifying sort criteria (an ORDER BY clause); use the **Sort** tab.

Column Name and Sort Order

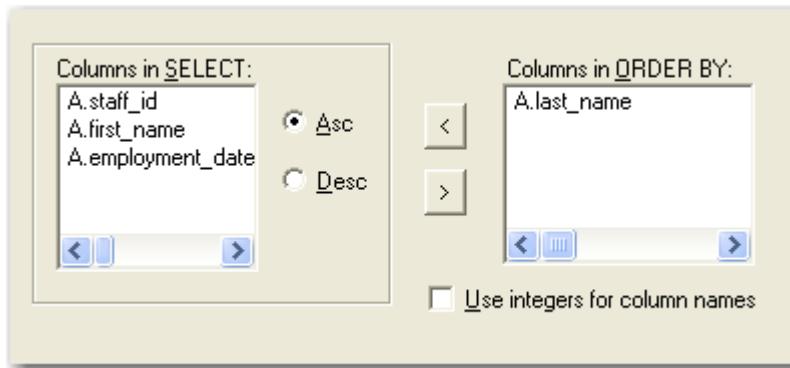
To specify sort criteria:

1. Select a column name from the **Column** list box.
2. Select order – **Ascending** or **Descending**.



Update Query Window

After the sort criteria have been entered, click on the right-pointing arrow button. This moves the sort criteria across into the right-hand pane and updates the COBOL code in the query window:



Delete Query Criteria

To delete a query criteria, click it and press the left-pointing arrow.

Check Query Results

Check the resulting Query using the the Query tab. The order by clause will be added to the query code.

The screenshot shows the 'Query' tab of the OpenESQL Assistant interface. The tab bar includes 'Query', 'Results', 'Details', 'Search Criteria', 'Sort', and 'Auxiliary Code'. The main area contains the following COBOL SQL code:

```
EXEC SQL
DECLARE CSR3 CURSOR FOR SELECT
    A.staff_id
    A.last_name
    A.first_name
    A.employment_date
FROM staff A
ORDER BY A.last_name
END-EXEC
EXEC SQL OPEN CSR3 END-EXEC
PERFORM UNTIL SQLCODE < 0 OR SQLCODE = +100
EXEC SQL
```

An arrow points to the 'ORDER BY A.last_name' line in the code.

Auxiliary Code

The Auxiliary code tab is used to generate SQL Statements that are not part of the query, but need to be defined within the program itself. The code that is generated from this tab can be directly inserted into a COBOL program. Sample of generated code follow in the table below.

SQL Statement	Code sample
CONNECT Statement	EXEC SQL CONNECT TO 'Demo' USER 'admin' END-EXEC
SQLCA Declaration	EXEC SQL INCLUDE SQLCA END-EXEC
Host Variable Declarations	EXEC SQL INCLUDE staff END-EXEC
DISCONNECT Statement	EXEC SQL DISCONNECT CURRENT END-EXEC
COMMIT Statement	EXEC SQL COMMIT END-EXEC
ROLLBACK Statement	EXEC SQL ROLLBACK END-EXEC
Generic SQL Program	\$SET SQL WORKING-STORAGE SECTION. Etc.... PROCEDURE DIVISION. Etc....
Input Interface Linkage	EXEC SQL INCLUDE InInterfaceParms.cpy END-EXEC
Output Interface Linkage	EXEC SQL INCLUDE OutInterfaceParms.cpy END-EXEC

After clicking the Auxiliary Code tab, click the desired statement to display the code in the selection window. Click the Insert Code icon to put the code into a program.

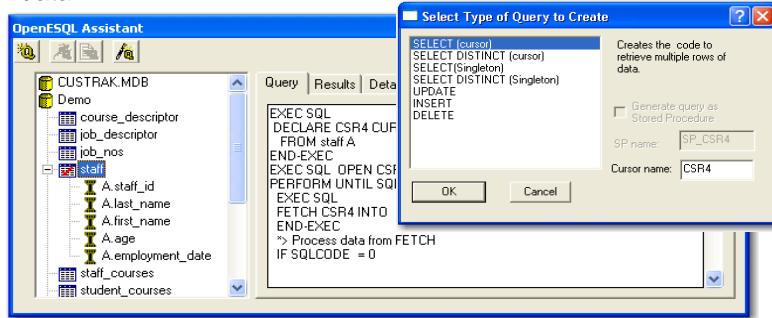
Table Joins

Information can be retrieved from more than one table when tables have at least one column in common. For example, where customer information is held in one table and product and order details are held in another, there

may be a requirement to collate information about what customers have ordered.

Select a Table and a Query

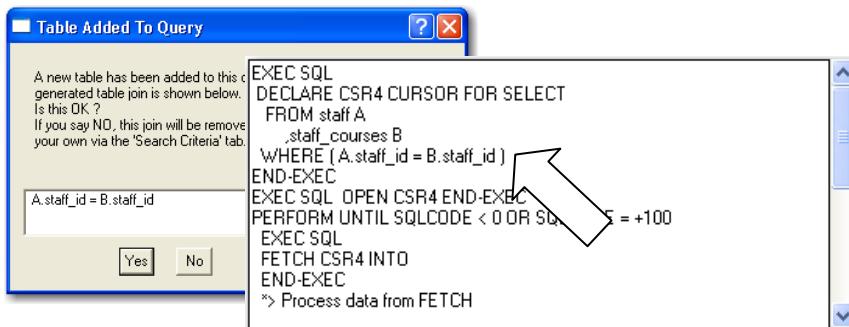
- Select a data source
- Select a table
- Select a query, in this case **SELECT(cursor)** to retrieve multiple rows of data.



Add a Second Table

Select a second table; for example, **Orders**.

The OpenESQL Assistant will create a table join using matching columns. The OpenESQL Assistant creates a table join using the first matching column it finds. (To change it, click on the **No** button, then specify a join via the Search Criteria tab). Click on the **Yes** button, the WHERE clause is added to the COBOL code:



The required columns for the query can now be added from the available tables.

```
EXEC SQL
DECLARE CSR4 CURSOR FOR SELECT
    A.last_name
    ,A.first_name
    ,B.course_no
    ,B.start_date
  FROM staff A
    ,staff_courses B
 WHERE (A.staff_id = B.staff_id )
END-EXEC
EXEC SQL OPEN CSR4 END-EXEC
PERFORM UNTIL SQLCODE < 0 OR SQLCODE = +100
```

The query can now be run giving the resulting join data.

The screenshot shows the OpenESQL Assistant interface. On the left, there is a tree view of database tables: 'staff' and 'staff_courses'. Under 'staff', there are columns: staff_id, last_name, first_name, age, employment_date. Under 'staff_courses', there are columns: staff_id, course_no, length_of_course, start_date, location. On the right, there is a grid titled 'Query Results' with the following data:

last_name	first_name	course_no	start_date
West	George	...	1997-10-10
West	George	...	1998-02-20
West	George	...	1997-06-01
Stone	Peter	...	1997-11-07
Stone	Peter	...	1997-11-25
Stone	Peter	...	1998-01-18
Jackson	John	...	1997-10-10
Jackson	John	...	1997-12-01
Jackson	John	...	1998-03-08
Nice	Brian	?	1998-01-15

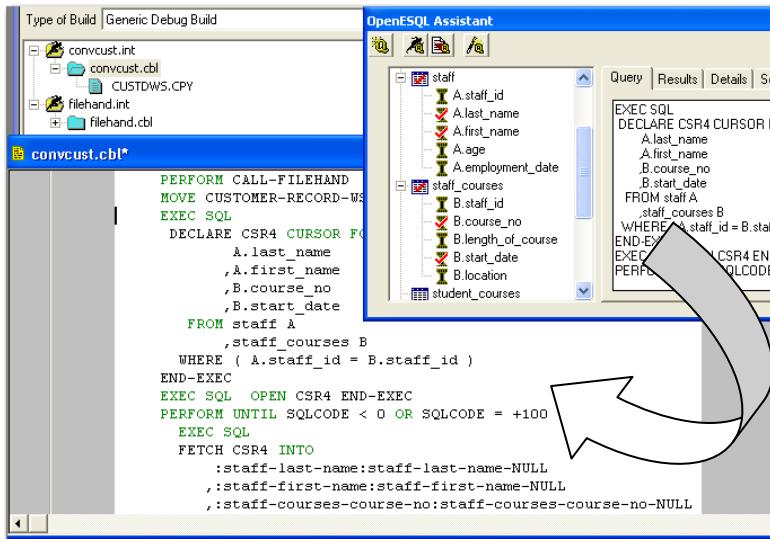
Adding Embedded SQL to a COBOL Program

The SQL query and all the Auxiliary Code can be added to a COBOL using the OpenESQL Assistant by pressing the **Insert Query into Current Program** button, after defining each SQL statement.

To insert the code into a program:

3. Open the COBOL source program for edit.
4. Define a Query.

5. Press **Insert Query into Current Program**. The code is inserted at the current cursor location line (the insert will be positioned in the line correctly).



Appendix A - Registering and Connecting a Database

Registering a Database

Command file to register a database.

```
rem * REGIT.CMD
echo .This batchfile runs the utility ODBCREG.exe, this utility
echo .registers the Custrak_Database for use with Net Express IDE
echo .OpenESQL Assistant.
echo .The Access DataBase for this class is located in the sub-
directory
echo .NetXclass\projects\skeleton\Custrak.mdb
echo .Syntax: ODBCREG Access Location Access DB Name Data
Source Name
echo . For example if Custrak.mdb is in c:\Netxclass then:
echo . ODBCREG c:\Netxclass\projects\skeleton Custrak.mdb
Custrak_DataBase
if %1.x == .x goto :usage
ODBCREG %1:\Netxclass\projects\skeleton Custrak.mdb
Custrak_DataBase
echo Finished!!
goto end
:usage
echo .**USAGE**
echo .You MUST type in a drive letter, it is not necessary to type the
colon
```

Register a Database Manually

1. Use Explorer to locate the .mdb file (the Access database) you want to register. For class, the custrak.mdb database is shipped in x:\netxclass\projects\lskeleton.
2. Click Start, Settings, Control Panel. Double-click 32bit ODBC.
3. Press **Add....** From the Driver list select **Microsoft Access**, then press **Finish**.
4. For Data Source Name, enter just the name of the .mdb file (e.g., CUSTRAK.MDB), then press **Select**. Move through the drive and directories list to locate your .mdb file. Select it and press **OK**. Your new database should be in the list. Press **OK** to close Control Panel.
5. Start Net Express. When you start OpenESQL the database will be in the list. **Note:** Net Express Sample1 and Net Express Sample2 are shipped with the product rather than with the courseware. They are normally added to the ODBC list at install time, but you can add them later if you need to, using the same steps outlined above. The physical files are named: x:...\\Net Express 5.0\\Examples\\Net Express IDE\\Smpldata\\Access\\Demo.mdb and Sample.mdb.

Connecting to a Database

Code samples and syntax are provided for reference. The code sample connects to a database named CUSTRAK with a USER ID of admin.

The Statement syntax is shown and described below:

- CONNECT TO [data source][AS name] USER [user[.password] [WITH [NO] PROMPT]]
- CONNECT user [{IDENTIFIED BY, '/') password] [AT name] [USING data source] [WITH [NO] PROMPT]
- CONNECT WITH PROMPT
- CONNECT RESET [name]

data source	The name of the ODBC data source. If you omit data source, the default ODBC data source is assumed. The data source can be specified as a literal or as a host variable.
name	A name for the connection. Connection names can have as many as 30 characters, and can include alphanumeric characters and any symbols legal in filenames. The first character must be a letter. Do not use Embedded SQL keywords, CURRENT, DEFAULT, or ALL for the connection name (they are invalid). If name is omitted, DEFAULT is assumed. Name can be specified as a literal or a host variable.
user	A valid user-id at the specified data source. (admin is the default)
password	A valid password for the specified user-id. (no password is the default for admin)
RESET	Resets (disconnects) the specified connection. You can specify name as CURRENT, DEFAULT or ALL.

When using only one connection it is not necessary to supply a name for the connection. When using more than one connection, a name must be specified for each connection. All database transactions after a successful CONNECT, other than CONNECT RESET, work through this most recently declared, current connection. To use a different connection, you must use the SET CONNECTION statement.

If WITH PROMPT is used, the ODBC run-time module will prompt for entry or confirmation of the connection details at run-time (i.e., it will ask you for connection details such as data source, name, and password). **Note:** Be sure to define the data source to the ODBC Administrator Driver List prior to opening OpenESQL.

Appendix B – Coding SQL Statements

Embedded SQL Statements

Within COBOL programs each embedded SQL statement must be preceded by the keywords EXEC SQL and followed by the keyword END-EXEC. For example:

```
EXEC SQL  
    SELECT  
        A.COMPANY  
    INTO  
        :Customer-COMPANY:Customer-COMPANY-NULL  
    FROM Customer A  
END-EXEC
```

Note: The case of embedded SQL keywords is ignored.

SQLCA

After embedded SQL statements are executed, error and status information is returned in the SQLCA. The SQLCA contains a variable, SQLCODE, as well as a number of warning flags that are used to indicate whether an error has occurred in the most recently executed SQL statement.

Testing the value of SQLCODE is the most common way of determining the success or failure of an embedded SQL statement. The possible values are:

Value	Meaning
0	The statement ran without error.
1	The statement ran, but a warning was generated. The values of the sqlwarn flags should be checked to determine the type of error.

100	Data matching the query was not found or the end of the results set has been reached. No rows were processed.
< 0 (negative)	The statement did not run due to an application, database, system, or network error.

To include it in a COBOL program, use the following statement in the data division:

```
EXEC SQL INCLUDE SQLCA END-EXEC
```

If this statement is not included, the COBOL Compiler allocates an area, but it is not addressable from within the program. If the data item MFSQLMESSAGEGETTEXT is declared, it is updated with a description of the exception condition whenever SQLCODE is non-zero. MFSQLMESSAGEGETTEXT must be declared as a character data item, PIC X(n), where n can be any legal value.

Sample code to check SQLCODE:

```
IF SQLCODE NOT = 0
    DISPLAY "Error: SQLCODE " SQLCODE
    DISPLAY SQLERRMC
STOP RUN.
```

Host Variables

Host variables are data items defined within a COBOL program. They are used to pass values to, and receive values from an ODBC data source. Host variables can be defined in the File Section, Working-Storage Section, Local-Storage Section or Linkage Section of your COBOL program and have any level number between 1 and 48. Level 49 is reserved for VARCHAR data items. When a host variable name is used within an embedded SQL statement, the data item name must begin with a colon (:) to enable the

Compiler to distinguish between host variables and tables or columns with the same name.

Host variables are used in one of two ways:

- Input host variables
- Output host variables.

For example, in the following statement, :Customer-SALESMAN is an input host variable that contains the id of the salesman to search for while :Customer-COMPANY is an output host variable that returns the result of the search:

EXEC SQL

```
SELECT A.COMPANY INTO :Customer-COMPANY FROM Customer A  
WHERE ( A.SALESMAN = :Customer-SALESMAN )  
END-EXEC
```

Declaring Host Variables

Before a host variable can be used in an embedded SQL statement, it must be declared. Declarations are preceded by the keywords EXEC SQL and followed by the keyword END-EXEC. For example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC  
03 Customer-CustID          PIC X(5).  
03 Customer-Company        PIC X(40).  
EXEC SQL END DECLARE SECTION END-EXEC
```

When using the OpenESQL Assistant to declare host variables, the following statement will be added to the COBOL program:

```
EXEC SQL DECLARE table-name TABLE ...END-EXEC
```

The DECLARE TABLE statement is treated as comment. The copyfile produced contains the Host Variable Declarations. For example:

```

EXEC SQL DECLARE Customer TABLE
( CustID          TEXT(5)
, Company         TEXT(40)
) END-EXEC.
*****
* COBOL DECLARATION FOR TABLE Customer
*****
****

01 DCLCustomer.
03 Customer-CustID    PIC X(5).
03 Customer-Company   PIC X(40).
*****
****
```

Null Values

SQL supports variables that may contain null values. A null value means that:

- The value is either unknown or is undefined, or no entry has been made
- A distinction can be made between a deliberate entry of zero (for numerical columns) or a blank (for character columns) and an unknown or inapplicable entry.

For example, a null value in a price column does not mean that the item is being given away free, it means that the price is not known or has not been set.

Together, a host variable and its companion indicator variable specify a single SQL value. A colon must precede both variables. When a host variable is null, its indicator variable has the value -1; when it is not null the indicator variable has a value other than -1.

```

EXEC SQL
UPDATE closeoutsale
SET temp_price = :saleprice:saleprice-null,
listprice = :oldprice
END-EXEC
```

Data Types

The following table shows the mappings used by OpenESQL when converting between SQL and COBOL data types:

SQL Type	COBOL Picture	Notes
SQL_CHAR(n)	PIC X(n)	
SQL_NCHAR(n)	PIC X(n) or PIC N(n)	
SQL_VARCHAR(n)	PIC X(n)	
SQL_NVARCHAR(n)	PIC X(n) or PIC N(n)	
SQL_LONGVARCHAR	PIC X(max)	max = 64K
SQL_NTEXT	PIC X(n) or PIC N(n)	
SQL_DECIMAL(p,s)	PIC 9(p-s)V9(S) COMP-3	p = precision (total number of digits). s = scale (number of digits after the decimal point).
SQL_NUMERIC(p,s)	PIC 9(p-s)V9(S) COMP-3	
SQL_SMALLINT	PIC S9(4) COMP-5	
SQL_INTEGER	PIC S9(9) COMP-5	
SQL_REAL	COMP-2	
SQL_FLOAT	COMP-2	
SQL_DOUBLE	COMP-2	
SQL_BIT	PIC S9(4) COMP-5	
SQL_TINYINT	PIC S9(2) COMP-5	
SQL_BIGINT	PIC S9(18) COMP-3	

SQL_BINARY(n)	PIC X(n)	
SQL_VARBINARY(n)	PIC X(n)	
SQL_LONVARBINAR	PIC X(max)	
SQL_DATE	PIC X(10)	yyyy-mm-dd
SQL_TIME	PIC X(8)	hh:mm:ss
SQL_TIMESTAMP	PIC X(26)	yyyy-mm-dd hh:mm:ss.fffffff

Note: p = precision (total number of digits). s = scale (number of digits after the decimal point).

Integer Data Types

TINYINT - tiny integer. A 1-byte integer SQL data type that can be declared in COBOL as PIC s9(4)COMP-5.

SMALLINT - small integer. SMALLINT is a 2-byte integer SQL data type that can be declared in COBOL with usage BINARY, COMP, COMP-X, COMP-5, or COMP-4. OpenESQL currently supports signed small integers, but does not support unsigned small integers. For example:

```

03 shortint1      pic s9(4) comp.
03 shortint2      pic s9(4) binary.
03 shortint3      pic x(2) comp-5.
03 shortint4      pic s9(4) comp-4.
03 shortint5      pic 9(4) usage display.
03 shortint6      pic s9(4) usage display.

```

INT – integer. INT is a 4-byte integer SQL data type that can be declared in COBOL with usage BINARY, COMP, COMP-X, COMP-5 or COMP-4.

OpenESQL currently supports signed integers, but not unsigned integers.

For example:

```
03 longint1      pic s9(9) comp.  
03 longint2      pic s9(9) comp-5.  
03 longint3      pic x(4) comp-5.  
03 longint4      pic x(4) comp-x.  
03 longint5      pic 9(9) usage display.  
03 longint6      pic s9(9) usage display.
```

BIGINT - big integer. An 8-byte integer SQL data type that can be declared in COBOL as PIC S9(18)COMP-3. OpenESQL supports a maximum size of S9(18) for COBOL data items used as host variables to hold values mapped from the SQL data type BIGINT. You should be aware, however, that a BIGINT data type can hold a value that is larger than the maximum value that can be held in a PIC S9(18) data item and ensure that your code checks for data truncation.

Character Data Types

CHAR - Fixed-length character strings. SQL data types with a driver defined maximum length. They are declared in COBOL as PIC X(*n*) where *n* is an integer between 1 and the maximum length.

For example:

```
03 char-field1    pic x(5).  
03 char-field2    pic x(254).
```

VARCHAR - Variable-length character strings. SQL data types that can be declared in COBOL in one of two ways:

As fixed-length character strings (PIC X(*n*)).

As group items containing only two elementary items, both of which must have a level number of 49. The first item is a 2-byte field declared with usage COMP or COMP-5 that represents the effective length of the character string. The second item is a PIC X(*n*) data type, where *n* is an integer, and holds the actual data. For example:

```
03 varchar1.  
      49 varchar1-len      pic 9(4) comp-5.  
      49 varchar1-data      pic x(200).  
  
03 Longvarchar1.  
      49 Longvarchar1-len   pic 9(4) comp.  
      49 Longvarchar1-data   pic x(30000).
```

If the data being copied to a SQL CHAR, VARCHAR or LONG VARCHAR data type is longer than the defined length, the data is truncated and the SQLWARN1 flag in the SQLCA data structure is set.

Cursors

Declaring a Cursor

A cursor is used when rows of data are to be fetched one at a time. The cursor indicates the current position in a results set, so enables updates and deletions at that position. You will declare and use a cursor when a SELECT statement fetches multiple rows of data. You will declare the cursor first. Do this by using the DECLARE CURSOR statement, which specifies a name for the cursor and either a SELECT statement or the name of a prepared SELECT statement. Cursor names must conform to the rules for identifiers on the database that you are connecting to, for example, some databases do not allow hyphens in cursor names:

```
EXEC SQL  
DECLARE CSR1 CURSOR FOR
```

```
SELECT A.COMPANY  
      FROM Customer A  
END-EXEC
```

Opening a Cursor

Once a cursor has been declared, it must be OPENed before it can be used. For example:

```
EXEC SQL OPEN CSR1 END-EXEC
```

Using a Cursor to Retrieve Data

Once a cursor has been opened it can be used to retrieve data from the database using the FETCH statement. The FETCH statement retrieves the next row from the results set produced by the OPEN statement and writes the data returned to the specified host variables. For example:

```
PERFORM UNTIL SQLCODE < 0 OR SQLCODE = +100  
      EXEC SQL  
      FETCH CSR1 INTO  
          :Customer-COMPANY:Customer-COMPANY-NULL  
      END-EXEC  
END-PERFORM
```

When the cursor reaches the end of the results set, SQLCODE is set to 100 and SQLSTATE is set to 02000.

Closing a Cursor

When the application has finished using the cursor, it should be closed using the CLOSE statement. For example:

```
EXEC SQL CLOSE Curr1 END-EXEC
```

Positioned UPDATE and DELETE Statements

Positioned UPDATE and DELETE statements are used with cursors and include WHERE CURRENT OF clauses instead of search condition clauses. The WHERE CURRENT OF clause specifies the corresponding cursor.

Update (Positioned)

Syntax:

```
UPDATE table_name SET column=expression[ ,... ] WHERE
CURRENT OF cursor_name
```

table_name	The table to be updated.
Set	Introduces the assignment of values to column names.
column=expression	A value for a particular column name. This value can be an expression or a null value.
cursor_name	A previously declared, opened, and fetched cursor.
current of	Identifies the cursor to be used in the update operation.

For example to update last_name in the row that was last fetched from the database using cursor Cur1:

```
EXEC SQL
    UPDATE emp SET last_name = :last-name
    WHERE CURRENT OF Cur1
END-EXEC
```

Delete (Positioned)

Syntax:

```
DELETE FROM table_name WHERE CURRENT OF cursor_name
 The same table used in the SELECT FROM option.
```

cursor_name A previously declared, opened, and fetched cursor.

To delete the row that was last fetched from the database using cursor Cur1:

```
EXEC SQL  
    DELETE emp WHERE CURRENT OF Cur1  
END-EXEC
```

Note: Positioned UPDATE and DELETE are part of the Extended ODBC Syntax and are not supported by all drivers.

Searched UPDATE and DELETE

Update (Searched)

UPDATE is a standard SQL statement that is passed directly to the ODBC driver. If you do not specify a WHERE clause, all the rows in the named table are updated.

Syntax:

```
UPDATE [FOR :host_integer] {table_name, view_name} SET  
[column={expression}[,...]] WHERE search_conditions ]
```

:host_integer	The maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5
table_name	The table to be updated.
view_name	The view to be updated.
column=expression	A value for a particular column name. This value can be an expression or a null value.
search_conditions	Any valid expression that can follow the standard SQL WHERE clause.

For example, to change the job (JOB) of an employee (EMPNO) in the EMPLOYEE table to Programmer:

```

EXEC SQL

    UPDATE EMPLOYEE

        SET JOB = 'Programmer'

        WHERE EMPNO = '1234'

END-EXEC

```

Delete (Searched)

DELETE is a standard SQL statement that removes table rows that meet the search criteria. If a WHERE clause is not specified, all of the rows in the named table are removed.

Syntax:

```
DELETE [FOR :host_integer] [FROM] {table_name,
view_name} [WHERE search_conditions]
```

:host_integer	The maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5.
FROM	An optional keyword. It is required for ANSI SQL 92 conformance.
table_name	The target table for the delete operation.
view_name	The target view for the delete operation.
WHERE	A standard SQL WHERE clause identifying the row to be deleted.
search_conditions	Any valid expression that can follow the standard SQL WHERE clause.

For example to delete an employee (EMPNO) from the EMPLOYEE table:

```

EXEC SQL

    DELETE FROM EMPLOYEE WHERE EMPNO = '1234'

END-EXEC

```

Module Summary

Net Express provides OpenESQL, a SQL Preprocessor that enables COBOL programmers to:

- Connect to Databases.
- Compile Programs with Embedded SQL.
- Generate code to declare and use cursors, select a Table and Query, select Columns, specify Search Criteria, create and execute Table Joins.
- Run a query.
- Add Embedded SQL to a COBOL Program (Define a Query, Insert Query into Current Program, Define a CONNECT Statement, SQLCA Declaration, Define Host Variable Declarations).

12.1 Connect to Data Source, Run a Query

Prior to doing the workshops, register the Access database with the OBDC manager. You can refer to Appendix A in this module for step-by-step instructions. The databases that need to be registered are in your Net Express install directory:

\Net Express 5.0\Examples\Net Express IDE\SMPLDATA\ACCESS\DEMO.MDB

\Net Express 5.0\Examples\Net Express IDE\SMPLDATA\ACCESS\SAMPLE.MDB

1. Load the convcust.APP from ...\\projects\\openesql\\convcust.
2. Select Tools, OpenESQL Assistant.
3. Double-click the **Demo** data source.
4. Double-click the **Staff** table.
5. Create a Singleton SELECT.
6. Right-click staff; Select All Columns. Result:

```
EXEC SQL
SELECT
    A.staff_id
    ,A.last_name
    ,A.first_name
    ,A.age
    ,A.employment_date
INTO
    :staff-staff-id:staff-staff-id-NULL
    ;staff-last-name:staff-last-name-NULL
    ;staff-first-name:staff-first-name-NULL
    ;staff-age:staff-age-NULL
    ;staff-employment-date:staff-employment-date-NULL
FROM staff A
END-EXEC
```

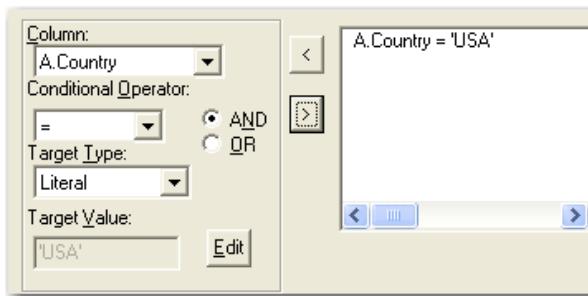
7. Run the query. Result:

The screenshot shows the OpenESQL Assistant interface with the 'Results' tab selected. The results grid displays the following data:

staff_id	last_name
1	Johnson
2	Smith
3	Hanson
4	Hammersmith
5	West

12.2 Define Search Criteria

1. Press Create a New Query.
2. Select the Net Express Sample data source.
3. Create a Singleton Query on the Customer table.
4. Select **A.CustID**, **A.Company**, and **A.Country**. Run the query.
5. Refine the search criteria to find all customers in the USA. (A.Country = literal USA).



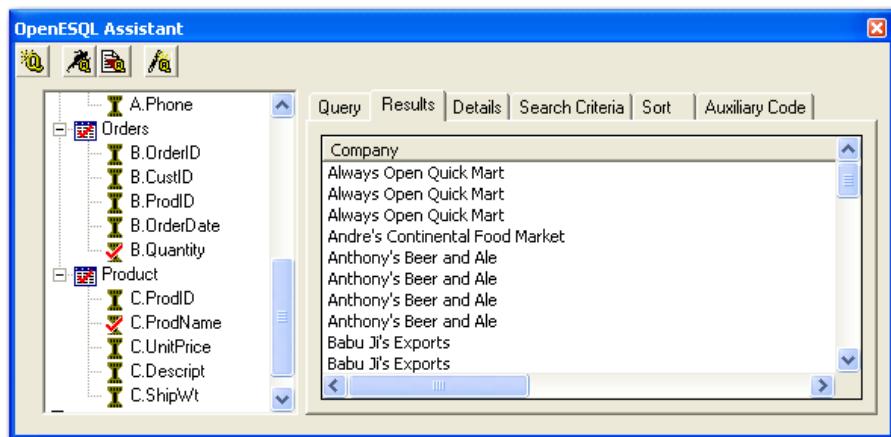
Result:

The screenshot shows the 'Results' tab of the OpenESQL Assistant interface. A grid displays the following data:

CustID	Company	Country
ALWA0	Always Open Quick Mart	USA
ANDRC	Andre's Continental Food Market	USA
ANTHB	Anthony's Beer and Ale	USA
BERGS	Bergstad's Scandinavian Grocery	USA
BLUEL	Blue Lake Deli & Grocery	USA
FITZD	Fitzgerald's Deli and Video	USA
FOODI	Foodmongers, Inc.	USA
FRUGP	Frugal Purse Strings	USA
FUJIA	Fujiwara Asian Specialties	USA
GARCA	Garcia's All-Day Food Mart	USA

12.3 Define Table Joins

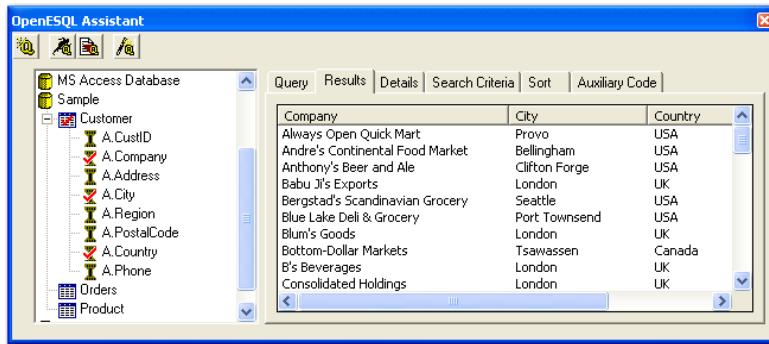
1. Press **Create a New Query....**
2. Select the **Net Express Sample** data source.
3. Create a **SELECT (Cursor)** on the Customer table.
4. Join Customer and Orders, accepting the default table join.
5. Join Orders and Product, accepting the default table join.
6. Select the following columns (in this order) to report a list of companies together with the name and quantity products ordered:
 - A. Company
 - C. ProdName
 - B. Quantity
7. Run the query. Result:



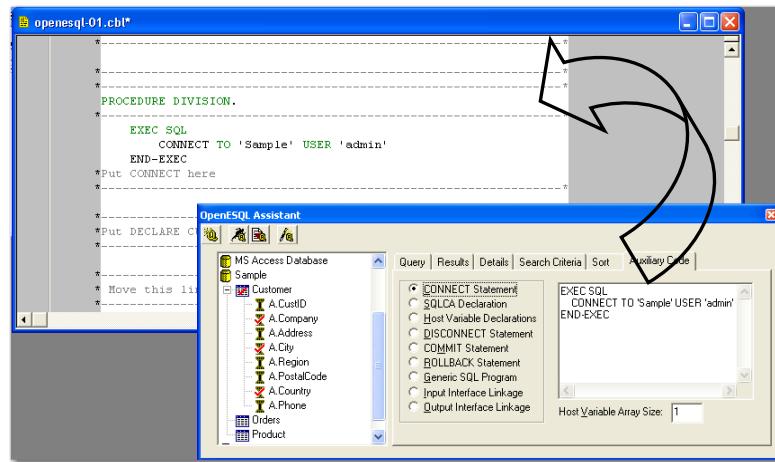
12.4 Add Embedded SQL to a COBOL Program

Change the COBOL program Openesql001.cbl to include a SQL query giving a list of customers including city and country information.

1. Use the Project Wizard to create an Empty Project under ...\\Projects named OESQL-01.
2. Add Openesql001.cbl from \\NETXCLASS\\PROJECTS\\SKELETONS. Edit Openesql001.
3. Start the OpenESQL Assistant.
4. Select **Create a New Query** for the Net Express Sample data source. Select the **Customer** table.
5. Create a query, SELECT(cursor), to retrieve multiple rows of data.
6. Select the following columns:
 - A.Company
 - A.City
 - A.Country
7. Run the query. The query and auxiliary code can now be added to the COBOL program.



8. Use the Auxiliary Code tab within the OpenESQL Assistant and follow the instructions documented within the source program.



9. Change the Build Settings (Advanced) to include the OpenESQL pre-processor and the AUTOCOMMIT directive.
10. Rebuild the project and run the program.

Application Output - Application Output		
Quality Plus Foods	New York	USA
Queensbridge Transport	London	UK
Rattlesnake Canyon Grocery	Albuquerque	USA
Reggie's Wine and Cheese	Pocatello	USA
Richmond Sugar	London	UK
Rite-Buy Supermarket	Berkeley	USA
Sawe-a-lot Markets	Boise	USA
Sawyer Hill General Store	Ithaca	USA
Seven Seas Imports	London	UK
Silver Screen Food Gems	Helena	USA
Special Delights	London	UK
Sugar and Spice	London	UK
Tommy's Imported Specialties	Denver	USA
Top of the Food Chain	Santa Fe	USA
Trade More	London	UK
ValuMax Food Stores	Austin	USA
Village Food Boutique	Los Angeles	USA
Vine and Barrel, Inc.	Seattle	USA
Walnut Grove Grocery	Buffalo	USA
Wanda's Wine and Cheese Shop	Bellingham	USA
Wellington Trading	Sevenoaks	UK
White Clover Markets	Seattle	USA
Witanowski's East-West Market	Lewiston	USA
Witanowski's East-West Market	Lewiston	USA

12.5 Define Search Criteria; Add Embedded SQL

The COBOL program needs to be changed to include an SQL query giving a list of customers in Canada including city information and telephone number.

1. Use the Project Wizard to create an Empty Project named OESQL002.
Add OESQL002.cbl from \NETXCLASS\PROJECTS\SKELETONS
2. Load the COBOL program OESQL002.CBL for edit.
3. Start the OpenESQL Assistant.
4. Select the Net Express Sample data source. If already open, press **Create a New Query...**
5. Select the **Customer** table.
6. Select a **query: SELECT(cursor)**, to retrieve multiple rows of data.
7. Select the following *columns*:
A.Company
A.Phone
A.Country
8. Run the query.
9. Add search criteria to find all customers in Canada and Run the query again. Result:



The query and auxiliary code can now be added to the COBOL program.

10. Use the Auxiliary Code tab within the OpenESQL Assistant and follow the instructions documented within the source program.
11. Change the Build Settings for the program to include the OpenESQL pre-processor and the AUTOCOMMIT directive.
12. Rebuild the project.
13. Run the program.

12.6 Define Table Join; Add Embedded SQL

Change the COBOL program to include an SQL query giving a list of customers in Canada including product and telephone number.

1. Use the Project Wizard to create an Empty Project named OESQL003.
Add Oesql003.cbl from \NETXCLASS\PROJECTS\SKELETONS
2. Load the COBOL program OESQL003.CBL for edit.
3. Start the OpenESQL Assistant.
4. Select the **Net Express Sample** data source.
5. Select the **Customer** table.
6. Create a query: SELECT(cursor), to retrieve multiple rows of data.
7. Join Customer and Orders, accepting the default table join.
8. Join Orders and Product, accepting the default table join.
9. Select the following columns in this order:
A.Company
C.ProdName
B.Quantity
10. Run the query.
11. Add search criteria to find customers that have purchased Chang.
12. Run the query. Result:

The screenshot shows the OpenESQL Assistant interface with a results grid. The grid has three columns: Company, ProdName, and Quantity. The data is as follows:

Company	ProdName	Quantity
Anthony's Beer and Ale	Chang	20
Fraser Distributors	Chang	24
Highbridge Gourmet Shoppe	Chang	35
La Playa Mini Mart	Chang	15
Richmond Sugar	Chang	20
Save-a-lot Markets	Chang	30
Sawyer Hill General Store	Chang	12

13. The query and auxiliary code can now be added to the COBOL program.
Use the Auxiliary Code tab within the OpenESQL Assistant and follow
the instructions documented within the source program.
14. Change the Build Settings for the program to include the OpenESQL
pre-processor and the AUTOCOMMIT directive.
15. Rebuild the project.
16. Run the program.

12.7 Verify Table is Empty

Project: None

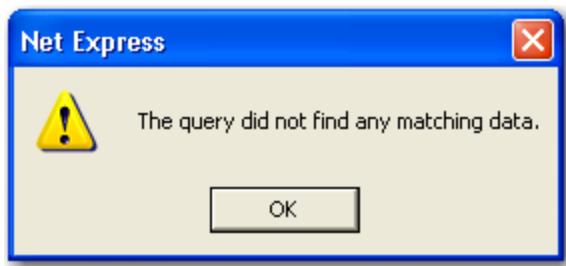
Program(s): None

Data Source: CUSTRAK

Table(s): Customer

This Exercise uses the CUSTRAK database. Using an operating system prompt, run the REGIT.BAT batch file prior to doing this workshop. The batch file is located in x:\NETXCLASS. You may need to edit the batch file (use Notepad) to point to the correct directory prior to running it: x:\NETXCLASS\PROJECTS\SKELETON. The CUSTRAK database contains a table, Customer; however this table is empty. In order to confirm the table is empty, run a query using the OpenESQL Assistant to retrieve any rows of data:

1. Start the OpenESQL Assistant from the Net Express View menu.
2. Select the **CUSTRAK Database** data source.
3. Select the **Customer** table.
4. Select a **query: SELECT(cursor)**, to retrieve multiple rows of data.
5. Select All Columns.
6. Run the query. The database should be empty.



12.8 Insert Data from Indexed Sequential File

Project: x:\NETXCLASS\PROJECTS\OPENESQL\CONVCUST

Program(s): CONVCUST, FILEHAND

Data Source: CUSTRAK

Table(s): Customer

The COBOL program CONVCUST contains code to read records from an indexed sequential file. The program needs to be changed to include an SQL query to INSERT rows of data into the CUSTRAK database. Load the project CONVCUST. The project properties need to be updated to include the location of copyfiles required by FILEHAND.CBL. These copyfiles are in x:\NETXCLASS\PROJECTS\CHARCUST.

1. Start the OpenESQL Assistant from the Net Express View menu.
2. Select **New Query**.
3. Select the **CUSTRAK** Database data source.
4. Select the **Customer** table.
5. Select a query: INSERT, to insert one or more rows of data.
6. Select All Columns. The query and auxiliary code can now be added to the COBOL program.
7. Load the COBOL program CONVCUST.CBL for edit.
8. Use the Auxiliary Code tab within the OpenESQL Assistant and follow the instructions documented within the source program.
9. Change the Build Settings for the program to include the OpenESQL pre-processor and the AUTOCOMMIT directive.
10. Rebuild the project.
11. Modify MFEXTMAP.DAT (right-click; use Shell Open) to point to the location of the data file: x:\NETXCLASS\PROJECTS\CHARCUST\DATA
12. Run the program. The records from the Indexed Sequential file should now all be in the database.

12.9 Verify Data Added to Data Source

Project: None

Program(s): None

Data Source: CUSTRAK

Table(s): Customer

1. Start the OpenESQL Assistant from the Net Express View menu.
2. Select New Query.
3. Select the **CUSTRAK Database** data source.
4. Select the **Customer** table.
5. Select a **query: SELECT(cursor)**, to retrieve one or more rows of data.
6. Select All Columns, Run the query.

12.10 Sum, Count

1. From the OpenESQL Assistant, double-click on **Net Express Sample**.
2. Double-click on **Customer**.
3. Left-click on **Select (Cursor)**, press **OK**.
4. Double-click on **Region** to add it as a column to the Select statement.
5. Right-click on **Region** and select **COUNT**.
6. Run the query.

The screenshot shows the OpenESQL Assistant interface with the 'Results' tab selected. The table displays the count of regions for each state:

Region	COUNT_Region
AK	0
AZ	1
BC	2
CA	4
CO	3
ID	2
	4

7. Press **Create a New Query**.
8. Double-click on **Orders**.
9. Left-click on **Select (Cursor)**, press **OK**.
10. Double click both **CustID** and **OrderID**.
11. Right-click on **Quantity** and select **SUM**.
12. Run the query.

The screenshot shows the OpenESQL Assistant interface with the 'Results' tab selected. The table displays the sum of quantities for each customer ID:

CustID	OrderID	SUM_Quantity
ALWAO	84	20.00
ALWAO	101	17.00
ANDRC	92	10.00
ANTHB	44	24.00
ANTHB	60	30.00
ANTHB	105	20.00
BABUJ	111	80.00

12.11 Select Distinct

13. Press **Create a New Query**. You are creating a list of unique regions.
14. Double-click on **Net Express Sample**; double-click on the **Customer** table.
15. Left click on **SELECT DISTINCT (cursor)**. Press **OK**. This generates a skeleton DECLARE CURSOR and SELECT statement.
16. Double-click on **A.Region** to add it to the DECLARE CURSOR statement and to add Customer-Region to the Fetch.
17. Run the query.

Region
AK
AZ
BC
CA
CO
ID
Kent
Lancashire
MT
NM
NY
OR
TX
UT
WA

Module 13

Using SQL Option for DB2

Overview

This module is a guide for developing and testing DB2 applications on the PC.

Objectives

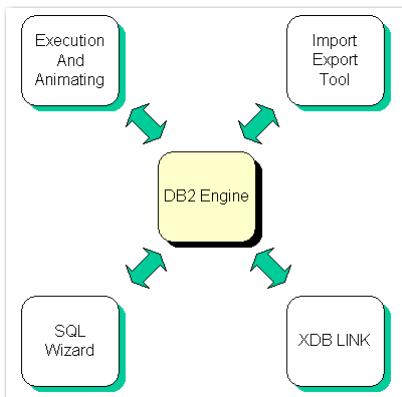
At the end of this module you will be able to:

- Set up a DB2 application on the PC.
- Develop and test DB2 applications on the PC (run DDL to create tables, populate table rows, modify table contents, compile and animate programs that contain embedded SQL code).
- Pre-test embedded SQL code interactively.
- Test the pre-tested SQL code in the program.

What is SQL Option for DB2?

SQL Option for DB2 employs the following functions.

- A DB2 emulator running on the PC against table data on a PC or a Network Server (XDB LINK allows access to data on the host).
- An Engine (Server) that provides access to the data and protects its integrity.
- A SQL preprocessor.
- An Animating tool using the original source code (i.e.: the EXEC SQL statements), not the translated source.
- SQL Wizard, which allows:
 - Creation of tables
 - Editing of table data
 - Creation, modification, and testing of SQL code.
- Import and Export Tool for:
 - Loading tables from DSNTIAUL files or text files
 - Exporting table data.
- Migrate options to copy tables between various database locations.
- XDB LINK—Connects the PC development environment to data in the DB2 subsystems on the host.



Terminology

In most places the host (DB2) and PC (SQL Option for DB2) terminology are the same. The main difference is the PC use of the term LOCATION. A LOCATION on the PC is the equivalent of a Subsystem on the Host. Each LOCATION can contain one or more Databases. One Engine can manage one or more locations (but only one can be active at a time). The databases are stored as folders on the PC. The related tables are stored in the database folders and have an extension of TAB.

Each LOCATION contains the following databases:

- DSNDB04 Default Database
- DSNDB06 System Catalog

The SYSTEM LOCATION also contains the following databases:

- XDBACF Security Tables
- DSNDDF Local/Remote Locations

Note: In order to use the SQL Option for DB2, you will need to set the required LOCATION prior to any testing activities.

Steps to Use SQL Option for DB2

1. Prepare the DB2 Subsystem:
 - Download the source files.
 - Unload table data on host using the DSNTIAUL utility.
 - Download the DSNTIAUL files.
 - Create Project making Net Express SQL tools available.
 - Create the LOCATION.
 - Set default the LOCATION.
 - Define Tables using DDL ‘Create Table’ statements.
 - Load tables by importing DSNTIAUL files.

2. Complete the Project definition:
 - Add files to the project.
 - Build the project.
3. Develop the DB2 Application:
 - Edit programs.
 - Compile programs.
 - Execute programs.
 - Pre-test embedded SQL code using the SQL Wizard.
 - Test the pre-tested SQL code in the program.
 - Use the SQL Wizard to design and test queries.
 - Add SQL queries from the SQL Wizard to programs.

Prepare DB2 Subsystem

Download the Source Files

For this course, a set of source files has already been downloaded from the host. These files are in the folder x:\netxclass\projects where x: is the class drive. The files are:

COBOL source:

GOLFJOIN.CBL
GOLFMODL.CBL
GOLFUPDT.CBL

COBOL copybooks:

CUSTOMER.CPY	DCLGEN for the Customer table
ORDERS.CPY	DCLGEN for the Orders table
PRODUCTS.CPY	DCLGEN for the Products table

JCL files

GOLFJOIN.JCL	JCL to execute GOLFJOIN
GOLFMODL.JCL	JCL to execute GOLFMODL
GO7LFUPDT.JCL	JCL to execute GOLFUPDT

SQL files containing Create Table statements for the class tables

CUST-CRE.SQL	CREATE DDL for Customer table
EMP-CRE.SQL	CREATE DDL for Employee table
ORD-CRE.SQL	CREATE DDL for Order table
PROD-CRE.SQL	CREATE DDL for Product table

Note: Please see your DB2-DBA group for more information about utilities that are available to extract the 'Create Table' statements.

Prepare DB2 Subsystem: Download DSNTIAUL Files

Unload Table Data on Host using the DSNTIAUL Utility

DSNTIAUL is a host DB2 utility that is used to select rows from tables.

Please see your DB2-DBA group for more information about its availability and usage at your shop.

Download the DSNTIAUL Files

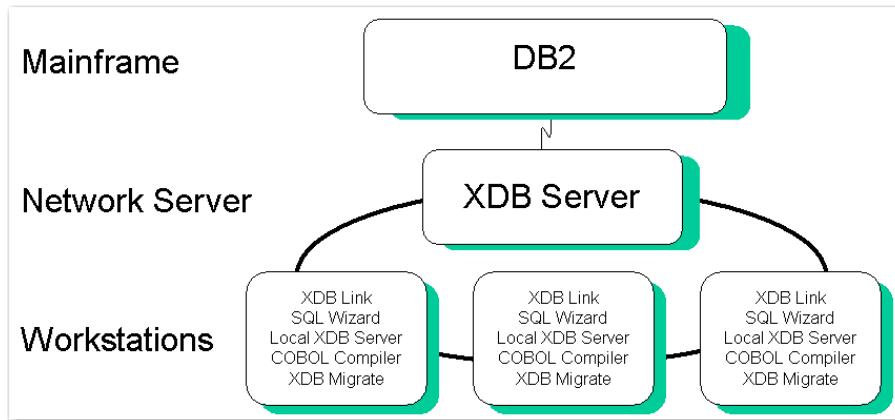
For this course, selected rows from host DB2 tables have already been downloaded and the files stored in the folder x:\netxclass\projects where x: is the class drive: The files are:

CUSTOMER.DSN	DSNTIAUL download for Customer table
ORDERS.DSN	DSNTIAUL download for Orders table
PRODUCTS.DSN	DSNTIAUL download for Products table
EMPLOYEE.DSN	DSNTIAUL download for Employee table

XDB Link

XDB Link allows PC SQL programs to access mainframe DB2 tables, connected either through a network server or by direct connection. Your system administrator will set up the TCP/IP protocol. Access to the setup is through **Options, SQL for DB2, XDB Link**.

The network server connection:



Accessing the Host

The Security option must be enabled for the XDB Server that is being used with XDB Link. This is set from the Tools, SQL for DB2, Server Configuration menu by selecting SQL Engine Options and toggling XDB-Server Security on. User names and passwords must be created for all authorized users from Tools, SQL for DB2, SQL Wizard |Admin |Users by a “super user” (e.g.: INSTALL). See Help, SQL for DB2 Help, SQL Wizard for more information. The client security options must be set from Tools, SQL for DB2, Options, Security tab, select Client Security enabled.

Table Names on the Host

When you run directly against the mainframe using an XDB Server, your user ID (AuthID) is the default qualifier for table name; e.g.: if you log on as JBLOGGS and request the PRODUCTS table, DB2 assumes you mean JBLOGGS.PRODUCTS.

If you want a different table, such as MYGOLF.PRODUCTS, then you need to either:

- Specify the table using an explicit, two-part name such as MYGOLF.PRODUCTS
- Specify the table with a one-part name and, on the Mainframe, create a synonym
- Specify the table with a one-part name and use the SET CURRENT SQLID statement in your program to specify a new default qualifier.

Module Summary

This module has reviewed the following requirements for using the SQL Option for DB2:

- Set up a DB2 application on the PC.
- Develop and test DB2 applications on the PC (run DDL to create tables, populate table rows, modify table contents, compile and animate programs that contain embedded SQL code).
- Pre-test embedded SQL code interactively.
- Test the pre-tested SQL code in the program.

13.1 Create SQL Project

To create a LOCATION you first need to start NetExpress with the SQL option selected.

1. Select **File, New, Project**. Press **OK**. Create a new GOLF project in **x:\NETXCLASS\PROJECTS\GOLF**. You can continue with the next step of creating the LOCATION. The SQL Wizard is used to create a LOCATION. Before the SQL Wizard can be started, the XDB-Server must be running. An SQL-enabled project must be open, too.
2. Select **Tools, SQL For DB2**; select **Start Server** to start the Server, if not already started. After a few seconds the server will start.
3. Minimize the server window.
4. Select **Tools, SQL For DB2, SQL Wizard**. This will connect to the Server and start the SQL Wizard.
5. The Catalog Browser window should be open inside the SQL Wizard window. If it isn't, select **View, Locations**.
6. At the left edge of the Catalog Browser window are three tabs: LOCATION, TABLE and QUERY. Click on each of these to see how the window changes. Then click on the LOCATION tab. There are already three LOCATIONS defined: MAINTAIN, SYSTEM and TUTORIAL. The first two are system locations and the third is a user sample location.
7. Click the **Create** button in the Catalog Browser window. Enter the values:
8. Name: **GOLF**
9. Creator: {MYNAME}. Enter your name for Creator.
10. Path: **x:\xdbclass**
11. Sort Sequence: **EBCDIC Sensitive**
12. Remarks: suitable remarks. *Do NOT press OK.*
13. Click **SQL** to view the SQL that is about to be executed.

14. Close the Show SQL window and press **OK**. Creating a location takes some time because the SQL Option for DB2 is creating a number of files and folders within **x:\xdbclass**.
15. When creating has completed, click the **Refresh** button in the Catalog Browser window. The GOLF location is now in the list.
16. Close the SQL Wizard.

13.2 Set Default LOCATION

In order to make sure that your GOLF Location is the one that the Server will connect to when using the SQL Wizard and other utilities follow these steps.

1. Select **Options, SQL For DB2, Client**.
2. Press the **Connect tab**. Enter:

Locations: GOLF

AuthID: {MYNAME}

Also set the default application path. This is where SQL Option searches for DDL files (e.g.: create table statements).

3. To set the default application path, select the Paths tab. Change Default Application Path to x:\xdbclass, press **OK**.
4. Select **Tools, SQL For DB2, SQL Wizard** to start the SQL Wizard again.
5. In order to confirm that the current active location is GOLF, either look in the lower center of the SQL Wizard window for 'Loc' of GOLF or select **Location, Set Location**, then click **Cancel** when you have confirmed that the GOLF location is the current active location.

13.3 Define Tables

Define the tables you will be using in class to the SQL Option. SQL files containing 'Create Table' DDL (Data Definition Language) statements have been downloaded from the host for this purpose.

1. To view or edit the DDL, from the SQL Wizard select **File, Open**. Make sure that **SQL Scripts** is selected in the List Objects of Type selection box.
2. Select the **cust-cre.sql** file and press **OK**. View the contents of the file: the SQL 'Create Table' to create the customer table. Close this window.
3. Select **File, Run Batch** to execute it.
4. In the same way use 'Run Batch' to execute the other three SQL files and create the remaining three tables: employee, orders, products.
5. Click the **Refresh** button in the Catalog Browser window.
6. Press the plus sign ('+') to expand the Golf location, your AuthID, and the Tables folder to see the four new tables listed.
7. Expand the **PRODUCTS** table and the Columns folder to look at the definition of this table. Use the minus sign ('-') to contract the Columns folder and the PRODUCTS table.

13.4 Load the Tables

The SQL Wizard utility provides a means of importing DSNTIAUL files. Four files have been downloaded from the host. These files can now be used to load the table data into the SQL Option tables you have defined.

1. In the Wizard, select **File, New, Import**. Enter the values:

Format: DSNTIAUL

File name: x:\XDBCCLASS\CUSTOMER.DSN

Table: GOLF.myname.CUSTOMER

2. Save this definition in an import file that then can be executed using 'Run Batch'. Select **File, Save As**, with the name Customer.imp to \xdbclass. Close the Import window.
3. Repeat the process.

Table	Data file	Save As
GOLF.myname.ORDERS	ORDERS.DSN	Orders.imp
GOLF.myname.PRODUCTS	PRODUCTS.DSN	Products.imp
GOLF.myname.EMPLOYEE	EMPLOYEE.DSN	Employee.imp

4. Close the Import window.
5. Select **File, Run Batch**; select **Import** in List Objects of Type Customer.imp. CUSTOMER.DSN populates the customer table.
6. Close the Import window.
7. Repeat the 'Run Batch' steps to import the Orders, Employee and Products tables.
8. Double-click **PRODUCTS** in the Catalog Browser to look at the contents of this table.
9. Close the Result Table.

13.5 Add Files to the Project

The project was created earlier so the NetExpress SQL tools would be available. Add COBOL programs to the project. To add the required files to your Golf project follow these steps.

1. From the NetExpress main window, select **Project, Add files**. Use the ‘Look in.’ drop down list to select x:\XDBCLASS; make sure ‘Files of type’ is Source Files.
2. Select all the .cbl. You do NOT have to select any .cpy files. NetExpress will add these automatically. Click **Add, Done**.
3. Click the Project folder in the left pane of the Project View window. Notice all the files that are in the project.
4. Take a brief look at one of the files in the project: double-click the filename **Golfupdt.cbl**. Please do NOT make any changes. This program contains Data Division entries for four copy files and a declare cursor. It also contains Procedure Division entries for open cursor, fetch data using that cursor, and close cursor.
5. Close the editor.

13.6 Build the Project

During the compile process it also precompiles the programs, translating the EXEC SQL statements. The reason that the precompile is done is that NetExpress recognized that there were EXEC SQL statements in these programs and automatically turned on the Build Settings EXEC SQL option which activates the precompiler.

1. Select **Project, Rebuild All.**

Develop the DB2 Application

13.7 Edit Programs

You have been requested to increase the price for the PGA Tour Shirt product by 10%. This will require adding an UPDATE statement to the GOLFUPDT program. Before editing the program, you need to do some research on the PRODUCTS table. Double-click PRODUCTS in the Catalog Browser to look at the contents of this table. Notice that the PGA Tour Shirt product has an itemno of 7 and a price of \$28.95. Close the Result Table.

The steps for adding the UPDATE statement are:

1. Double-click on the file name golfupd.cbl in the Project View window.
2. Scroll down in GOLFUPDT until you find the comment about Open Cursor.
3. Before the Open Cursor comment insert several blank lines by using the e key.
4. Enter the following UPDATE/SET/WHERE statement along with the EXEC SQL and END-EXEC code:

```
* OPEN Cursor
```

5. Save the changes to GOLFUPDT.
6. Close this edit window.

13.8 Execute Programs

Before making any further changes, execute the program GOLFUPDT so you can see what it does.

1. Make sure that the Server is running. Select **Animate, Start Animating**.
2. Right-click on PRODUCTS-ITEMNO in the FETCH statement and select **Locate “PRODUCTS-ITEMNO”**
3. Double-click on the data items: PRODUCTS-ITEMNO and PRODUCTS-PRICE. Then add each to the list.
4. Step through the code until you see the PRICE when ITEMNO is equal to. Notice that it is now \$31.84; the 10% price increase has taken effect.
5. Continue to Step through the code until STOP RUN has been executed.
6. Close the Animator.

13.9 Pre-test Embedded SQL using the SQL Wizard

At this point, you can edit the GOLFUPDT program in any way you wish. However in the case of modified or new SQL statements, the SQL Option for DB2 provides some assistance. You can make the program changes and then copy/paste them into the SQL Wizard. The SQL Wizard can then be used to 'pre-test' statements before testing them in the program. The advantage of this is that you can focus on the making sure the SQL code is correct.

Some examples:

- In the first example, you'll maintain existing SQL code using the wizard.
- In the second example, you will be modifying the GOLFUPDT program so it will use host variables. In this case you will go through the entire process of modifying, pre-testing using the SQL Wizard, and testing the program.

Example One: Test a Three Table Join

The purpose of this program is to find what products and how many were ordered by customers in the current customer table.

1. Edit golfjoin.cbl. Scroll down in GOLFJOIN to the Declare Cursor statement.
2. In the editor window, highlight ONLY the SELECT/WHERE code in the DECLARE CURSOR statement; select **Edit, Copy** to copy the code.

```
SELECT LSTNM, FSTNM, ITEMNM, ORDQTY  
      FROM CUSTOMER, ORDERS, PRODUCTS  
     WHERE CUSTOMER.CUSTNO = ORDERS.CUSTNO  
           AND ORDERS.ITEMNO = PRODUCTS.ITEMNO
```

3. Close the editor window.
4. In the SQL Wizard, select File, New, and select SQL. Select Edit, Paste to paste your copied statement here.

5. Select Query, Run to execute it. The results should show that Lee Trevino ordered one Ping Pang Putter and that Nancy Lopez ordered one set of Ping Iron/Woods.
6. Close the Result Table window.

Verify the Results for Example One

One way is to look at individual table values and ‘play computer’ (hint: close the Result Table window at the end of each step).

7. Double-click CUSTOMER in the Catalog Browser. There are three customers: Lee Trevino, Jan Stephensen, and Nancy Lopez with custnos of 1, 2, and 3 respectively.
8. Double-click ORDERS in the Catalog Browser and notice that of the custnos that existed in the CUSTOMER table:
custno 1 (Trevino) has an itemno of 4 for an ordqty of 1; and
custno 3 (Lopez) has an itemno of 3 also for an ordqty of 1.
9. Double-click PRODUCTS in the Catalog Browser and notice that itemno 4, which belongs to Lee Trevino, is a Ping Pang Putter and itemno 3, which belongs to Nancy Lopez, is the Ping Iron/Woods, so the earlier results were correct.

Another way of verifying is to break the existing SQL code into smaller sections and use them to minimize the amount of time you have to play computer:

1. Add a semi-colon (“;”) at the end of the existing SELECT/WHERE SQL code so the SQL Wizard will know where one statement ends and the next starts.
2. Paste another copy of the SELECT/WHERE and modify it so it looks like this (Note: be sure to change ITEMNM to ITEMNO).

```
SELECT LSTNM, FSTNM, ITEMNM, ORDQTY  
      FROM CUSTOMER, ORDERS, PRODUCTS  
     WHERE CUSTOMER.CUSTNO = ORDERS.CUSTNO
```

3. Select **Query, Current Query Options**; select **Current Command**, and press OK. This changes the default value of First Command.
4. Click anywhere to the left of the semi-colon (“;”) in your modified SELECT/ WHERE statement and click the Run button.
5. Notice Trevino has an itemno of 4 for an ordqty of 1; and Lopez has an itemno of 3 also for an ordqty of 1. Close the Result Table window.
Note: if you want to make ‘Current Command’ the default value, from the NetExpress main window, select **Options, SQL**, click the Query Run tab, select **Current Command**, and press **OK**.
6. Type the following SELECT/WHERE statement:

```
SELECT ITEMNO, ITEMNM  
      FROM PRODUCTS  
 WHERE ITEMNO = 3 OR ITEMNO = 4;
```

7. Click anywhere to the left of the semi-colon (“;”) in the new SELECT/ WHERE statement and click the Run button. Notice that itemno 4, which belongs to Lee Trevino, is a Ping Pang Putter and itemno 3, which belongs to Nancy Lopez, is the Ping Iron/Woods, so this verifies that the earlier results were correct. Close the Result Table window. Close the SQL window that contains the three SELECT/WHERE statements; respond **NO** to the ‘save changes’ prompt ... leave the SQL Wizard.
Note: Even though the second method took longer in this case, the more complex the SQL code is and the larger the tables are, the more this method will reduce debug time.

Example Two: Increase Price for Selected Products using a Host Variable

In practice, programs generally would not have an SQL statement that updates just one record as in the GOLFUPDT program. What is more likely is a statement that updates specific records based on the value in some data item (i.e.: host variable) obtained by reading from a file, receiving values from an input screen, doing a computation, etc.

In this example, using SQL Wizard to test embedded SQL, you will want to see how the SQL Wizard handles a host variable with an UPDATE/SET/WHERE statement.

Make the Changes to the Program

1. Edit the **golfupdt.cbl** program. Scroll down in GOLFUPDT until you find the UPDATE/SET/WHERE statement.
2. Change the WHERE clause in the UPDATE/SET/WHERE statement so it uses a host variable (i.e.: WHERE ITEMNO = :PRODUCTS-ITEMNO).
3. Save the changes.

Copy this Code to the SQL Wizard

1. From the editor, highlight ONLY the UPDATE/SET/WHERE code in the statement (i.e.: do NOT highlight the EXEC SQL and the END-EXEC) and Select **Edit, Copy** to copy the code.
2. From the SQL Wizard, select **File, New, SQL**. Paste your copied statement here.
3. Add a semi-colon (“;”) to the end of the UPDATE/SET/WHERE statement ... in case you decide to execute some other SQL code from the same SQL window.

‘Pre-test’ it

1. Click to the left of the semi-colon (“;”) in the UPDATE/SET/WHERE statement; Run.
2. Enter a value of 7 and press **OK**. Respond **OK** to the ‘Updating records’ prompt.

Verify your Results

1. Double-click **PRODUCTS** in the Catalog Browser: the price for itemno 7 is \$35.02. The price increase changes have an accumulative effect ($\$28.95 * 1.1 = 31.84$; $\$31.84 * 1.1 = 35.02$).

2. Close the Result Table. Another way to verify the results is to execute SQL code that displays just the relevant information: the itemno of 7 and its price.
3. Enter this code after the UPDATE/SET/WHERE statement:

```
SELECT ITEMNO, PRICE  
      FROM PRODUCTS  
 WHERE ITEMNO = 7;
```

4. Select **Query, Current Query Options**. Select **Current Command**, and press **OK**. This changes the default value of First Command.
5. Click anywhere to the left of the semi-colon (“;”) in the SELECT/WHERE statement and click the Run button. Notice the price for itemno 7 is \$35.02.
6. Close the Result Table.

13.10 Test the Pre-Tested SQL Code in the Program

You do not have to spend the time to add file I/O or screen access code to provide the value for the host variable itemno. For your purpose, a MOVE statement will work fine.

1. Edit; scroll down in GOLFUPDT to the modified UPDATE/SET/WHERE statement.
2. Add a MOVE statement before the EXEC SQL to move a value of 7 to products-itemno. The modified UPDATE/SET/WHERE statement along with the new MOVE statement should now look like this:

```
MOVE 7 TO PRODUCTS-ITEMNO  
EXEC SQL  
    UPDATE PRODUCTS  
        SET PRICE = PRICE * 1.1  
        WHERE ITEMNO = :PRODUCTS-ITEMNO  
    END-EXEC
```

3. Compile GOLFUPDT.
4. Start the Animator. Right-click on PRODUCTS-ITEMNO in the UPDATE/SET/WHERE statement and select Locate “PRODUCTS-ITEMNO”.
5. Double-click on the data item PRODUCTS-ITEMNO. Then add it to the list. Do the same for PRODUCTS-PRICE. Note: It is a good idea to add SQLCODE to the watch list.
6. Step through the code until you see the FETCHed PRICE when ITEMNO is equal to 7. Notice that it is now \$38.52. Remember that the price increase changes have an accumulative effect ($\$28.95 * 1.1 = 31.84$; $\$31.84 * 1.1 = 35.02$; $\$35.02 * 1.1 = 38.52$)
7. Continue to Step through the code until STOP RUN has been executed. The Application Output window will show you the Job termination information.
8. Close the Animator.

Verify the Results

1. From SQL Wizard, double-click **PRODUCTS** in the Catalog Browser and notice the price for itemno 7 is \$38.52.
2. Close the Result Table.
3. Let's also use the alternate method, so click anywhere to the left of the semi-colon (“;”) in your SELECT/WHERE statement and click the button. Notice the price for itemno 7 is \$38.52.

```
SELECT ITEMNO, PRICE  
FROM PRODUCTS  
WHERE ITEMNO = 7;
```

4. Close the Result Table. Since it might be nice to have this ‘verify’ code for future use, let’s save it:
5. Delete the UPDATE/SET/WHERE statement; leave the just SELECT/WHERE statement.
6. Select **File, Save As**. Save this file with name Verify.sql to the folder \xdbclass.
7. Close the SQL window.

13.11 Refreshing the Products Table

In order to refresh the PRODUCTS table so it has the pre-update value for itemno 7 do the following:

1. In the SQL Wizard, select **File, New, SQL**.
2. Enter the following SQL code and click the Run button

```
DROP TABLE PRODUCTS
```

3. Respond **OK** to the ‘dropping table’ prompt.
4. Close the SQL window. Respond **NO** to the ‘save changes’ prompt.
5. Select **File, Run Batch**. Because the Default Application Path was set to x:\XDBCLASS, the ‘Directories’ section should already be pointing to it. Select the file prod-cre.sql to create the PRODUCTS table and press **OK**.
6. Select **File, Run Batch**. Select **Import** in the ‘List Objects of Type’ selection box. Choose the file name: **Products.imp** and press **OK** to execute the import.
7. Close the Import window.

Verify the Results

8. From SQL Wizard, double-click PRODUCTS in the Catalog Browser and notice the price for itemno 7 is \$28.95.
9. Close the Result Table.

Additional Features

13.12 Add Additional Rows to the Tables

Rows can be added to a table by using the SQL INSERT statement in embedded COBOL SQL statements or through interactive SQL. The SQL Wizard can also be used to add rows to the table.

1. Double-click PRODUCTS in the Catalog Browser to bring up the Result Table containing the table's contents.
2. Select **Record, Allow Editing**. Selecting this will also turn on Autocommit. Notice the new row at the bottom ... with '@' as its Row #
3. Edit the table to include a new product by clicking into the new row and entering the column information. Use the tab key to move to the next column. Numeric data will be right justified. Character data will be left justified. For example:
4. Close the Result Table window for the PRODUCTS table. Respond **Yes** to the 'Insert record before closing window' prompt.
5. Close the SQL Wizard.

13.13 DCLGEN

In the examples so far, the COBOL copybooks have already been created. The DCLGEN facility creates copybooks for tables. Create a COBOL copy file for the Employee table.

1. From the NetExpress main window, select Tools, SQL For DB2, Declaration Generator.
2. Use the drop down selection list to select EMPLOYEE for the table name.
3. Click into the 'Output File' field and notice the default values that appear. Change the Structure Name to DCL-EMPLOYEE (with a dash). Enter a Field Prefix of EMPLOYEE- (with a dash). Click the Text radio button. Click Generate and then Exit. Because the Default Application Path was set to x:\XDBCLASS, that is where the copy file is generated.
4. Edit golfmod1.cbl and add this include statement after the other INCLUDES:

```
EXEC SQL
    INCLUDE EMPLOYEE
END-EXEC
```

Save the change.

5. In order to update the project, select **Build, Update Dependencies**. The copybook has now been added to your project under the copybook list.
6. Right-click **EMPLOYEE** in the INCLUDE statement; select **Show Copybook** to look at the copy file you just generated.
7. Close the editor.

13.14 SQL Compatibility

Up to now you have been using SQL Option for DB2 in its DB2 compatibility mode. It does have other modes it can run in.

1. Select **Options, SQL for DB2, Client**.
2. Click the SQL tab. Notice that the SQL compatibility mode is set to "DB2".
3. Click the drop down selection box to see the other modes.
4. Click Cancel when you have finished. Respond "**OK**" to the exit prompt.

Final Exercise

13.15 Putting It All Together

1. In the SQL Wizard, remove the EMPLOYEE table and PART tables from the GOLF location by dropping the tables. (You will need to refresh the catalog browser view to see the effect of this).
2. Create the definition of the EMPLOYEE table by running the SQL file emp-cre.sql. (Again refresh the catalog browser view to see the effect of this).
3. Check that the definitions have been correctly added by double clicking the EMPLOYEE table. Hint: open the emp-cre.sql file and compare it to EMPLOYEE table's definitions.
4. Create an import file that will use the DSNTIAUL file employee.dsn to populate the EMPLOYEE table.
5. Run this import file to populate the EMPLOYEE table.
6. Check that the table has been correctly populated by double clicking the EMPLOYEE table. There should be 31 records. Close the Result table.
7. Edit this table to: Add a couple of extra rows, Modify several existing rows, Delete some rows.
8. Use DCLGEN to generate the copy file for this table. Use the default values. (Overwrite the existing copy file).
9. Use the SQL Wizard to define a query on this table to find all first and last names of employees who live in California ('CA').
10. Run this query to make sure it is correct. Save it with a suitable name.
11. Copy this query into GOLFMODL in a new CURSOR definition. Remove the existing CURSOR definition.
12. Modify your program to use this cursor in a FETCH.
13. Compile and animate your program to ensure it returns the correct values in the FETCH.

Module 13

Using SQL Option for DB2

Overview

This module is a guide for developing and testing DB2 applications on the PC.

Objectives

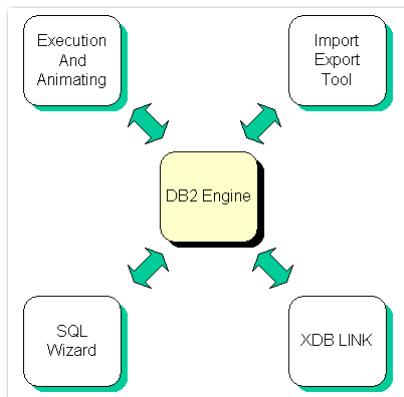
At the end of this module you will be able to:

- Set up a DB2 application on the PC.
- Develop and test DB2 applications on the PC (run DDL to create tables, populate table rows, modify table contents, compile and animate programs that contain embedded SQL code).
- Pre-test embedded SQL code interactively.
- Test the pre-tested SQL code in the program.

What is SQL Option for DB2?

SQL Option for DB2 employs the following functions.

- A DB2 emulator running on the PC against table data on a PC or a Network Server (XDB LINK allows access to data on the host).
- An Engine (Server) that provides access to the data and protects its integrity.
- A SQL preprocessor.
- An Animating tool using the original source code (i.e.: the EXEC SQL statements), not the translated source.
- SQL Wizard, which allows:
 - Creation of tables
 - Editing of table data
 - Creation, modification, and testing of SQL code.
- Import and Export Tool for:
 - Loading tables from DSNTIAUL files or text files
 - Exporting table data.
- Migrate options to copy tables between various database locations.
- XDB LINK—Connects the PC development environment to data in the DB2 subsystems on the host.



Terminology

In most places the host (DB2) and PC (SQL Option for DB2) terminology are the same. The main difference is the PC use of the term LOCATION. A LOCATION on the PC is the equivalent of a Subsystem on the Host. Each LOCATION can contain one or more Databases. One Engine can manage one or more locations (but only one can be active at a time). The databases are stored as folders on the PC. The related tables are stored in the database folders and have an extension of TAB.

Each LOCATION contains the following databases:

- DSNDB04 Default Database
- DSNDB06 System Catalog

The SYSTEM LOCATION also contains the following databases:

- XDBACF Security Tables
- DSNDDF Local/Remote Locations

Note: In order to use the SQL Option for DB2, you will need to set the required LOCATION prior to any testing activities.

Steps to Use SQL Option for DB2

1. Prepare the DB2 Subsystem:
 - Download the source files.
 - Unload table data on host using the DSNTIAUL utility.
 - Download the DSNTIAUL files.
 - Create Project making Net Express SQL tools available.
 - Create the LOCATION.
 - Set default the LOCATION.
 - Define Tables using DDL ‘Create Table’ statements.
 - Load tables by importing DSNTIAUL files.

2. Complete the Project definition:
 - Add files to the project.
 - Build the project.
3. Develop the DB2 Application:
 - Edit programs.
 - Compile programs.
 - Execute programs.
 - Pre-test embedded SQL code using the SQL Wizard.
 - Test the pre-tested SQL code in the program.
 - Use the SQL Wizard to design and test queries.
 - Add SQL queries from the SQL Wizard to programs.

Prepare DB2 Subsystem

Download the Source Files

For this course, a set of source files has already been downloaded from the host. These files are in the folder x:\netxclass\projects where x: is the class drive. The files are:

COBOL source:

GOLFJOIN.CBL
GOLFMODL.CBL
GOLFUPDT.CBL

COBOL copybooks:

CUSTOMER.CPY	DCLGEN for the Customer table
ORDERS.CPY	DCLGEN for the Orders table
PRODUCTS.CPY	DCLGEN for the Products table

JCL files

GOLFJOIN.JCL	JCL to execute GOLFJOIN
GOLFMODL.JCL	JCL to execute GOLFMODL
GO7LFUPDT.JCL	JCL to execute GOLFUPDT

SQL files containing Create Table statements for the class tables

CUST-CRE.SQL	CREATE DDL for Customer table
EMP-CRE.SQL	CREATE DDL for Employee table
ORD-CRE.SQL	CREATE DDL for Order table
PROD-CRE.SQL	CREATE DDL for Product table

Note: Please see your DB2-DBA group for more information about utilities that are available to extract the 'Create Table' statements.

Prepare DB2 Subsystem: Download DSNTIAUL Files

Unload Table Data on Host using the DSNTIAUL Utility

DSNTIAUL is a host DB2 utility that is used to select rows from tables.

Please see your DB2-DBA group for more information about its availability and usage at your shop.

Download the DSNTIAUL Files

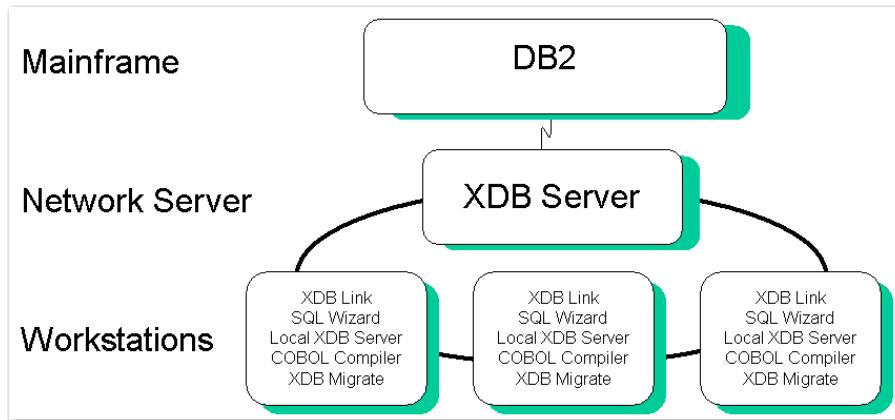
For this course, selected rows from host DB2 tables have already been downloaded and the files stored in the folder x:\netxclass\projects where x: is the class drive: The files are:

CUSTOMER.DSN	DSNTIAUL download for Customer table
ORDERS.DSN	DSNTIAUL download for Orders table
PRODUCTS.DSN	DSNTIAUL download for Products table
EMPLOYEE.DSN	DSNTIAUL download for Employee table

XDB Link

XDB Link allows PC SQL programs to access mainframe DB2 tables, connected either through a network server or by direct connection. Your system administrator will set up the TCP/IP protocol. Access to the setup is through **Options, SQL for DB2, XDB Link**.

The network server connection:



Accessing the Host

The Security option must be enabled for the XDB Server that is being used with XDB Link. This is set from the Tools, SQL for DB2, Server Configuration menu by selecting SQL Engine Options and toggling XDB-Server Security on. User names and passwords must be created for all authorized users from Tools, SQL for DB2, SQL Wizard |Admin |Users by a “super user” (e.g.: INSTALL). See Help, SQL for DB2 Help, SQL Wizard for more information. The client security options must be set from Tools, SQL for DB2, Options, Security tab, select Client Security enabled.

Table Names on the Host

When you run directly against the mainframe using an XDB Server, your user ID (AuthID) is the default qualifier for table name; e.g.: if you log on as JBLOGGS and request the PRODUCTS table, DB2 assumes you mean JBLOGGS.PRODUCTS.

If you want a different table, such as MYGOLF.PRODUCTS, then you need to either:

- Specify the table using an explicit, two-part name such as MYGOLF.PRODUCTS
- Specify the table with a one-part name and, on the Mainframe, create a synonym
- Specify the table with a one-part name and use the SET CURRENT SQLID statement in your program to specify a new default qualifier.

Module Summary

This module has reviewed the following requirements for using the SQL Option for DB2:

- Set up a DB2 application on the PC.
- Develop and test DB2 applications on the PC (run DDL to create tables, populate table rows, modify table contents, compile and animate programs that contain embedded SQL code).
- Pre-test embedded SQL code interactively.
- Test the pre-tested SQL code in the program.

13.1 Create SQL Project

To create a LOCATION you first need to start NetExpress with the SQL option selected.

1. Select **File, New, Project**. Press **OK**. Create a new GOLF project in **x:\NETXCLASS\PROJECTS\GOLF**. You can continue with the next step of creating the LOCATION. The SQL Wizard is used to create a LOCATION. Before the SQL Wizard can be started, the XDB-Server must be running. An SQL-enabled project must be open, too.
2. Select **Tools, SQL For DB2**; select **Start Server** to start the Server, if not already started. After a few seconds the server will start.
3. Minimize the server window.
4. Select **Tools, SQL For DB2, SQL Wizard**. This will connect to the Server and start the SQL Wizard.
5. The Catalog Browser window should be open inside the SQL Wizard window. If it isn't, select **View, Locations**.
6. At the left edge of the Catalog Browser window are three tabs: LOCATION, TABLE and QUERY. Click on each of these to see how the window changes. Then click on the LOCATION tab. There are already three LOCATIONS defined: MAINTAIN, SYSTEM and TUTORIAL. The first two are system locations and the third is a user sample location.
7. Click the **Create** button in the Catalog Browser window. Enter the values:
8. Name: **GOLF**
9. Creator: {MYNAME}. Enter your name for Creator.
10. Path: **x:\xdbclass**
11. Sort Sequence: **EBCDIC Sensitive**
12. Remarks: suitable remarks. *Do NOT press OK.*
13. Click **SQL** to view the SQL that is about to be executed.

14. Close the Show SQL window and press **OK**. Creating a location takes some time because the SQL Option for DB2 is creating a number of files and folders within **x:\xdbclass**.
15. When creating has completed, click the **Refresh** button in the Catalog Browser window. The GOLF location is now in the list.
16. Close the SQL Wizard.

13.2 Set Default LOCATION

In order to make sure that your GOLF Location is the one that the Server will connect to when using the SQL Wizard and other utilities follow these steps.

1. Select **Options, SQL For DB2, Client**.
2. Press the **Connect tab**. Enter:

Locations: GOLF

AuthID: {MYNAME}

Also set the default application path. This is where SQL Option searches for DDL files (e.g.: create table statements).

3. To set the default application path, select the Paths tab. Change Default Application Path to x:\xdbclass, press **OK**.
4. Select **Tools, SQL For DB2, SQL Wizard** to start the SQL Wizard again.
5. In order to confirm that the current active location is GOLF, either look in the lower center of the SQL Wizard window for 'Loc' of GOLF or select **Location, Set Location**, then click **Cancel** when you have confirmed that the GOLF location is the current active location.

13.3 Define Tables

Define the tables you will be using in class to the SQL Option. SQL files containing 'Create Table' DDL (Data Definition Language) statements have been downloaded from the host for this purpose.

1. To view or edit the DDL, from the SQL Wizard select **File, Open**. Make sure that **SQL Scripts** is selected in the List Objects of Type selection box.
2. Select the **cust-cre.sql** file and press **OK**. View the contents of the file: the SQL 'Create Table' to create the customer table. Close this window.
3. Select **File, Run Batch** to execute it.
4. In the same way use 'Run Batch' to execute the other three SQL files and create the remaining three tables: employee, orders, products.
5. Click the **Refresh** button in the Catalog Browser window.
6. Press the plus sign ('+') to expand the Golf location, your AuthID, and the Tables folder to see the four new tables listed.
7. Expand the **PRODUCTS** table and the Columns folder to look at the definition of this table. Use the minus sign ('-') to contract the Columns folder and the PRODUCTS table.

13.4 Load the Tables

The SQL Wizard utility provides a means of importing DSNTIAUL files. Four files have been downloaded from the host. These files can now be used to load the table data into the SQL Option tables you have defined.

1. In the Wizard, select **File, New, Import**. Enter the values:

Format: DSNTIAUL

File name: x:\XDBCCLASS\CUSTOMER.DSN

Table: GOLF.myname.CUSTOMER

2. Save this definition in an import file that then can be executed using 'Run Batch'. Select **File, Save As**, with the name Customer.imp to \xdbclass. Close the Import window.
3. Repeat the process.

Table	Data file	Save As
GOLF.myname.ORDERS	ORDERS.DSN	Orders.imp
GOLF.myname.PRODUCTS	PRODUCTS.DSN	Products.imp
GOLF.myname.EMPLOYEE	EMPLOYEE.DSN	Employee.imp

4. Close the Import window.
5. Select **File, Run Batch**; select **Import** in List Objects of Type Customer.imp. CUSTOMER.DSN populates the customer table.
6. Close the Import window.
7. Repeat the 'Run Batch' steps to import the Orders, Employee and Products tables.
8. Double-click **PRODUCTS** in the Catalog Browser to look at the contents of this table.
9. Close the Result Table.

13.5 Add Files to the Project

The project was created earlier so the NetExpress SQL tools would be available. Add COBOL programs to the project. To add the required files to your Golf project follow these steps.

1. From the NetExpress main window, select **Project, Add files**. Use the ‘Look in.’ drop down list to select x:\XDBCLASS; make sure ‘Files of type’ is Source Files.
2. Select all the .cbl. You do NOT have to select any .cpy files. NetExpress will add these automatically. Click **Add, Done**.
3. Click the Project folder in the left pane of the Project View window. Notice all the files that are in the project.
4. Take a brief look at one of the files in the project: double-click the filename **Golfupdt.cbl**. Please do NOT make any changes. This program contains Data Division entries for four copy files and a declare cursor. It also contains Procedure Division entries for open cursor, fetch data using that cursor, and close cursor.
5. Close the editor.

13.6 Build the Project

During the compile process it also precompiles the programs, translating the EXEC SQL statements. The reason that the precompile is done is that NetExpress recognized that there were EXEC SQL statements in these programs and automatically turned on the Build Settings EXEC SQL option which activates the precompiler.

1. Select **Project, Rebuild All.**

Develop the DB2 Application

13.7 Edit Programs

You have been requested to increase the price for the PGA Tour Shirt product by 10%. This will require adding an UPDATE statement to the GOLFUPDT program. Before editing the program, you need to do some research on the PRODUCTS table. Double-click PRODUCTS in the Catalog Browser to look at the contents of this table. Notice that the PGA Tour Shirt product has an itemno of 7 and a price of \$28.95. Close the Result Table.

The steps for adding the UPDATE statement are:

1. Double-click on the file name golfupd.cbl in the Project View window.
2. Scroll down in GOLFUPDT until you find the comment about Open Cursor.
3. Before the Open Cursor comment insert several blank lines by using the e key.
4. Enter the following UPDATE/SET/WHERE statement along with the EXEC SQL and END-EXEC code:

```
* OPEN Cursor
```

5. Save the changes to GOLFUPDT.
6. Close this edit window.

13.8 Execute Programs

Before making any further changes, execute the program GOLFUPDT so you can see what it does.

1. Make sure that the Server is running. Select **Animate, Start Animating**.
2. Right-click on PRODUCTS-ITEMNO in the FETCH statement and select **Locate “PRODUCTS-ITEMNO”**
3. Double-click on the data items: PRODUCTS-ITEMNO and PRODUCTS-PRICE. Then add each to the list.
4. Step through the code until you see the PRICE when ITEMNO is equal to. Notice that it is now \$31.84; the 10% price increase has taken effect.
5. Continue to Step through the code until STOP RUN has been executed.
6. Close the Animator.

13.9 Pre-test Embedded SQL using the SQL Wizard

At this point, you can edit the GOLFUPDT program in any way you wish. However in the case of modified or new SQL statements, the SQL Option for DB2 provides some assistance. You can make the program changes and then copy/paste them into the SQL Wizard. The SQL Wizard can then be used to 'pre-test' statements before testing them in the program. The advantage of this is that you can focus on the making sure the SQL code is correct.

Some examples:

- In the first example, you'll maintain existing SQL code using the wizard.
- In the second example, you will be modifying the GOLFUPDT program so it will use host variables. In this case you will go through the entire process of modifying, pre-testing using the SQL Wizard, and testing the program.

Example One: Test a Three Table Join

The purpose of this program is to find what products and how many were ordered by customers in the current customer table.

1. Edit golfjoin.cbl. Scroll down in GOLFJOIN to the Declare Cursor statement.
2. In the editor window, highlight ONLY the SELECT/WHERE code in the DECLARE CURSOR statement; select **Edit, Copy** to copy the code.

```
SELECT LSTNM, FSTNM, ITEMNM, ORDQTY  
      FROM CUSTOMER, ORDERS, PRODUCTS  
     WHERE CUSTOMER.CUSTNO = ORDERS.CUSTNO  
           AND ORDERS.ITEMNO = PRODUCTS.ITEMNO
```

3. Close the editor window.
4. In the SQL Wizard, select File, New, and select SQL. Select Edit, Paste to paste your copied statement here.

5. Select Query, Run to execute it. The results should show that Lee Trevino ordered one Ping Pang Putter and that Nancy Lopez ordered one set of Ping Iron/Woods.
6. Close the Result Table window.

Verify the Results for Example One

One way is to look at individual table values and ‘play computer’ (hint: close the Result Table window at the end of each step).

7. Double-click CUSTOMER in the Catalog Browser. There are three customers: Lee Trevino, Jan Stephensen, and Nancy Lopez with custnos of 1, 2, and 3 respectively.
8. Double-click ORDERS in the Catalog Browser and notice that of the custnos that existed in the CUSTOMER table:
custno 1 (Trevino) has an itemno of 4 for an ordqty of 1; and
custno 3 (Lopez) has an itemno of 3 also for an ordqty of 1.
9. Double-click PRODUCTS in the Catalog Browser and notice that itemno 4, which belongs to Lee Trevino, is a Ping Pang Putter and itemno 3, which belongs to Nancy Lopez, is the Ping Iron/Woods, so the earlier results were correct.

Another way of verifying is to break the existing SQL code into smaller sections and use them to minimize the amount of time you have to play computer:

1. Add a semi-colon (“;”) at the end of the existing SELECT/WHERE SQL code so the SQL Wizard will know where one statement ends and the next starts.
2. Paste another copy of the SELECT/WHERE and modify it so it looks like this (Note: be sure to change ITEMNM to ITEMNO).

```
SELECT LSTNM, FSTNM, ITEMNM, ORDQTY  
      FROM CUSTOMER, ORDERS, PRODUCTS  
     WHERE CUSTOMER.CUSTNO = ORDERS.CUSTNO
```

3. Select **Query, Current Query Options**; select **Current Command**, and press OK. This changes the default value of First Command.
4. Click anywhere to the left of the semi-colon (“;”) in your modified SELECT/ WHERE statement and click the Run button.
5. Notice Trevino has an itemno of 4 for an ordqty of 1; and Lopez has an itemno of 3 also for an ordqty of 1. Close the Result Table window.
Note: if you want to make ‘Current Command’ the default value, from the NetExpress main window, select **Options, SQL**, click the Query Run tab, select **Current Command**, and press **OK**.
6. Type the following SELECT/WHERE statement:

```
SELECT ITEMNO, ITEMNM  
      FROM PRODUCTS  
 WHERE ITEMNO = 3 OR ITEMNO = 4;
```

7. Click anywhere to the left of the semi-colon (“;”) in the new SELECT/ WHERE statement and click the Run button. Notice that itemno 4, which belongs to Lee Trevino, is a Ping Pang Putter and itemno 3, which belongs to Nancy Lopez, is the Ping Iron/Woods, so this verifies that the earlier results were correct. Close the Result Table window. Close the SQL window that contains the three SELECT/WHERE statements; respond **NO** to the ‘save changes’ prompt ... leave the SQL Wizard.
Note: Even though the second method took longer in this case, the more complex the SQL code is and the larger the tables are, the more this method will reduce debug time.

Example Two: Increase Price for Selected Products using a Host Variable

In practice, programs generally would not have an SQL statement that updates just one record as in the GOLFUPDT program. What is more likely is a statement that updates specific records based on the value in some data item (i.e.: host variable) obtained by reading from a file, receiving values from an input screen, doing a computation, etc.

In this example, using SQL Wizard to test embedded SQL, you will want to see how the SQL Wizard handles a host variable with an UPDATE/SET/WHERE statement.

Make the Changes to the Program

1. Edit the **golfupdt.cbl** program. Scroll down in GOLFUPDT until you find the UPDATE/SET/WHERE statement.
2. Change the WHERE clause in the UPDATE/SET/WHERE statement so it uses a host variable (i.e.: WHERE ITEMNO = :PRODUCTS-ITEMNO).
3. Save the changes.

Copy this Code to the SQL Wizard

1. From the editor, highlight ONLY the UPDATE/SET/WHERE code in the statement (i.e.: do NOT highlight the EXEC SQL and the END-EXEC) and Select **Edit, Copy** to copy the code.
2. From the SQL Wizard, select **File, New, SQL**. Paste your copied statement here.
3. Add a semi-colon (“;”) to the end of the UPDATE/SET/WHERE statement ... in case you decide to execute some other SQL code from the same SQL window.

‘Pre-test’ it

1. Click to the left of the semi-colon (“;”) in the UPDATE/SET/WHERE statement; Run.
2. Enter a value of 7 and press **OK**. Respond **OK** to the ‘Updating records’ prompt.

Verify your Results

1. Double-click **PRODUCTS** in the Catalog Browser: the price for itemno 7 is \$35.02. The price increase changes have an accumulative effect ($\$28.95 * 1.1 = 31.84$; $\$31.84 * 1.1 = 35.02$).

2. Close the Result Table. Another way to verify the results is to execute SQL code that displays just the relevant information: the itemno of 7 and its price.
3. Enter this code after the UPDATE/SET/WHERE statement:

```
SELECT ITEMNO, PRICE  
      FROM PRODUCTS  
 WHERE ITEMNO = 7;
```

4. Select **Query, Current Query Options**. Select **Current Command**, and press **OK**. This changes the default value of First Command.
5. Click anywhere to the left of the semi-colon (“;”) in the SELECT/WHERE statement and click the Run button. Notice the price for itemno 7 is \$35.02.
6. Close the Result Table.

13.10 Test the Pre-Tested SQL Code in the Program

You do not have to spend the time to add file I/O or screen access code to provide the value for the host variable itemno. For your purpose, a MOVE statement will work fine.

1. Edit; scroll down in GOLFUPDT to the modified UPDATE/SET/WHERE statement.
2. Add a MOVE statement before the EXEC SQL to move a value of 7 to products-itemno. The modified UPDATE/SET/WHERE statement along with the new MOVE statement should now look like this:

```
MOVE 7 TO PRODUCTS-ITEMNO  
EXEC SQL  
    UPDATE PRODUCTS  
        SET PRICE = PRICE * 1.1  
        WHERE ITEMNO = :PRODUCTS-ITEMNO  
    END-EXEC
```

3. Compile GOLFUPDT.
4. Start the Animator. Right-click on PRODUCTS-ITEMNO in the UPDATE/SET/WHERE statement and select Locate “PRODUCTS-ITEMNO”.
5. Double-click on the data item PRODUCTS-ITEMNO. Then add it to the list. Do the same for PRODUCTS-PRICE. Note: It is a good idea to add SQLCODE to the watch list.
6. Step through the code until you see the FETCHed PRICE when ITEMNO is equal to 7. Notice that it is now \$38.52. Remember that the price increase changes have an accumulative effect ($\$28.95 * 1.1 = 31.84$; $\$31.84 * 1.1 = 35.02$; $\$35.02 * 1.1 = 38.52$)
7. Continue to Step through the code until STOP RUN has been executed. The Application Output window will show you the Job termination information.
8. Close the Animator.

Verify the Results

1. From SQL Wizard, double-click **PRODUCTS** in the Catalog Browser and notice the price for itemno 7 is \$38.52.
2. Close the Result Table.
3. Let's also use the alternate method, so click anywhere to the left of the semi-colon (“;”) in your SELECT/WHERE statement and click the button. Notice the price for itemno 7 is \$38.52.

```
SELECT ITEMNO, PRICE  
FROM PRODUCTS  
WHERE ITEMNO = 7;
```

4. Close the Result Table. Since it might be nice to have this ‘verify’ code for future use, let’s save it:
5. Delete the UPDATE/SET/WHERE statement; leave the just SELECT/WHERE statement.
6. Select **File, Save As**. Save this file with name Verify.sql to the folder \xdbclass.
7. Close the SQL window.

13.11 Refreshing the Products Table

In order to refresh the PRODUCTS table so it has the pre-update value for itemno 7 do the following:

1. In the SQL Wizard, select **File, New, SQL**.
2. Enter the following SQL code and click the Run button

```
DROP TABLE PRODUCTS
```

3. Respond **OK** to the ‘dropping table’ prompt.
4. Close the SQL window. Respond **NO** to the ‘save changes’ prompt.
5. Select **File, Run Batch**. Because the Default Application Path was set to x:\XDBCLASS, the ‘Directories’ section should already be pointing to it. Select the file prod-cre.sql to create the PRODUCTS table and press **OK**.
6. Select **File, Run Batch**. Select **Import** in the ‘List Objects of Type’ selection box. Choose the file name: **Products.imp** and press **OK** to execute the import.
7. Close the Import window.

Verify the Results

8. From SQL Wizard, double-click PRODUCTS in the Catalog Browser and notice the price for itemno 7 is \$28.95.
9. Close the Result Table.

Additional Features

13.12 Add Additional Rows to the Tables

Rows can be added to a table by using the SQL INSERT statement in embedded COBOL SQL statements or through interactive SQL. The SQL Wizard can also be used to add rows to the table.

1. Double-click PRODUCTS in the Catalog Browser to bring up the Result Table containing the table's contents.
2. Select **Record, Allow Editing**. Selecting this will also turn on Autocommit. Notice the new row at the bottom ... with '@' as its Row #
3. Edit the table to include a new product by clicking into the new row and entering the column information. Use the tab key to move to the next column. Numeric data will be right justified. Character data will be left justified. For example:
4. Close the Result Table window for the PRODUCTS table. Respond **Yes** to the 'Insert record before closing window' prompt.
5. Close the SQL Wizard.

13.13 DCLGEN

In the examples so far, the COBOL copybooks have already been created. The DCLGEN facility creates copybooks for tables. Create a COBOL copy file for the Employee table.

1. From the NetExpress main window, select Tools, SQL For DB2, Declaration Generator.
2. Use the drop down selection list to select EMPLOYEE for the table name.
3. Click into the 'Output File' field and notice the default values that appear. Change the Structure Name to DCL-EMPLOYEE (with a dash). Enter a Field Prefix of EMPLOYEE- (with a dash). Click the Text radio button. Click Generate and then Exit. Because the Default Application Path was set to x:\XDBCLASS, that is where the copy file is generated.
4. Edit golfmod1.cbl and add this include statement after the other INCLUDES:

```
EXEC SQL
    INCLUDE EMPLOYEE
END-EXEC
```

Save the change.

5. In order to update the project, select **Build, Update Dependencies**. The copybook has now been added to your project under the copybook list.
6. Right-click **EMPLOYEE** in the INCLUDE statement; select **Show Copybook** to look at the copy file you just generated.
7. Close the editor.

13.14 SQL Compatibility

Up to now you have been using SQL Option for DB2 in its DB2 compatibility mode. It does have other modes it can run in.

1. Select **Options, SQL for DB2, Client**.
2. Click the SQL tab. Notice that the SQL compatibility mode is set to "DB2".
3. Click the drop down selection box to see the other modes.
4. Click Cancel when you have finished. Respond "**OK**" to the exit prompt.

Final Exercise

13.15 Putting It All Together

1. In the SQL Wizard, remove the EMPLOYEE table and PART tables from the GOLF location by dropping the tables. (You will need to refresh the catalog browser view to see the effect of this).
2. Create the definition of the EMPLOYEE table by running the SQL file emp-cre.sql. (Again refresh the catalog browser view to see the effect of this).
3. Check that the definitions have been correctly added by double clicking the EMPLOYEE table. Hint: open the emp-cre.sql file and compare it to EMPLOYEE table's definitions.
4. Create an import file that will use the DSNTIAUL file employee.dsn to populate the EMPLOYEE table.
5. Run this import file to populate the EMPLOYEE table.
6. Check that the table has been correctly populated by double clicking the EMPLOYEE table. There should be 31 records. Close the Result table.
7. Edit this table to: Add a couple of extra rows, Modify several existing rows, Delete some rows.
8. Use DCLGEN to generate the copy file for this table. Use the default values. (Overwrite the existing copy file).
9. Use the SQL Wizard to define a query on this table to find all first and last names of employees who live in California ('CA').
10. Run this query to make sure it is correct. Save it with a suitable name.
11. Copy this query into GOLFMODL in a new CURSOR definition. Remove the existing CURSOR definition.
12. Modify your program to use this cursor in a FETCH.
13. Compile and animate your program to ensure it returns the correct values in the FETCH.

Module 14

UNIX Option

Overview

The UNIX Option enables both off-loading application development from the UNIX environment using the tools and facilities provided by Net Express. UNIX Option enables applications that have been created using Net Express to be deployed on a UNIX system.

Objectives

At the end of this module you will be able to:

- Identify portability issues: syntax checking, syntax flagged, syntax not flagged, debugging (.idy files).
- Identify issues regarding installation of SCP and SAMBA.
- Provide Publisher setup information including configuration details and server capabilities.
- Identify Publisher's handling of filename mapping, pre-build and post-build commands, copyfile handling and system copyfiles.
- Connect to the UNIX server, log on to your server, and start Server Express.
- Set up publishing to UNIX.
- Perform remote debugging.
- Copy UNIX applications to the PC using import.

Net Express UNIX Tools

A complete set of tools is provided for developing UNIX applications, including:

- Character Dialog System for developing user interfaces.
- Publisher for transferring files to a UNIX machine, creating a make file based on the Net Express project settings and compiling the application using the appropriate Micro Focus UNIX compiler.
- An Import wizard that enables you to import an Object COBOL for UNIX application into Net Express. Object COBOL for UNIX is the Micro Focus COBOL language and compiler for the UNIX environment.
- PowerTerm from Ericom, a powerful terminal emulator on the PC allowing access to UNIX applications without leaving the Net Express development environment.
- SAMBA, a utility that enables the PC to treat UNIX directories as PC disk drives.
- SCP (Server Control Program), a program providing a standardized interface between the Net Express UNIX Option and the UNIX operating system and COBOL product. SCP is installed when you install Server Express and is required for the Publisher to work.
 - To avoid confusion when publishing to Linux platforms, (which have a conflicting 'scp' system command) the Micro Focus daemon has been renamed to cobscp'. Link to cobscp in /cobol_install_dir/bin.

The UNIX system must have Micro Focus COBOL Version 3.2 for UNIX or later. You should have Server Express on your UNIX server.

Portability Issues

The Net Express Compiler has been enhanced to identify syntax that is not portable to a UNIX system. However, since it cannot flag all incompatibilities it may be necessary to manually check and change a program before porting to a UNIX system.

For more information regarding portability, see the UNIX Option User's Guide (Net Express online Bookshelf).

Installing SCP and SAMBA

Before the Publisher can be used **SCP** must be installed on the UNIX server. While not required in order to user Publisher, SAMBA can also be installed and configured. Installation details can be found in the ***UNIX Option User Guide Appendix A.***

Access to your UNIX system is required in order to install SCP and SAMBA superuser. UNIX system administrators possess the relevant skills to configure your UNIX system. They will be familiar with:

- Using **ftp**
- Setting permissions on files and directories
- The use of rhosts files and host name entries to enable transparent log-in without requiring a password
- Providing a login for Publisher on the target UNIX system
- The effects of using DNS on UNIX systems
- Recognizing basic network problems.

Configuration Information

The following table provides definitions and guidelines for SCP and SAMBA configuration.

Server Name	The name of the UNIX system on which the project is to be published.
User Id	The login identifier to use when contacting the server specified above. By default, this is the user-id you used to login to Windows

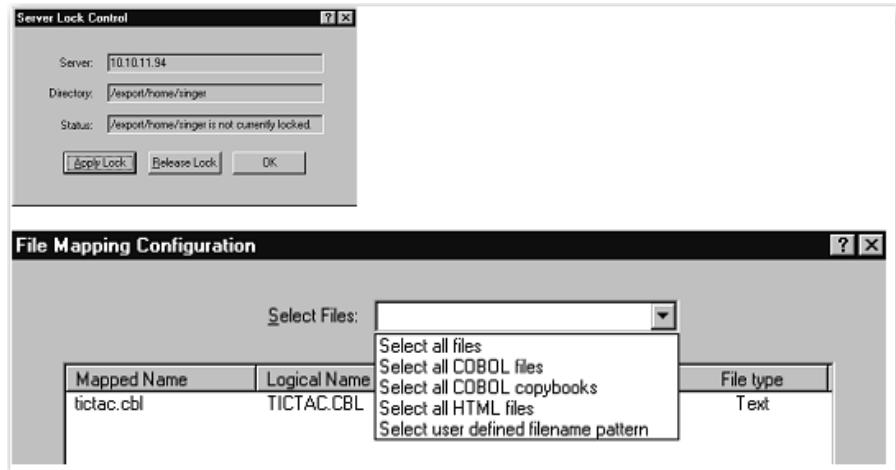
UNIX COBOL Directory	The directory on the UNIX system that contains Micro Focus COBOL for UNIX. After entering details in this field, the Server Capabilities button is enabled.
Build Directory	The directory on the UNIX system to which the project will be published. After entering details in this field, the Server Lock Control button is enabled.
Server Copyfile Directory	The directory on the UNIX system to which copyfiles will be published.
Enable CGI Support	CGI programs are copied to the directory specified in Build Directory. If a different directory is required they must be copied manually.
Enable Target Directory Build	Specifies the Net Express target type; for example, Debug. This directory will be created under the directory specified in Build Directory.
Build On Server	Copy the files to the UNIX system then rebuild them.
Enable Filename Mapping	Specifies how filenames should be mapped on the UNIX system.
Define Filename	Mapping button is enabled.
Pre-Build Command	A command that is executed on the UNIX system before the project is built.
Post-Build Command	A command that is executed on the UNIX system after the project has been built.

Server Capabilities

Verify Server displays information about COBOL for UNIX on the UNIX system.



The Server Lock Control button specifies whether the build directory on the UNIX system is locked or unlocked. A dialog is displayed showing the current lock status, and enables the build directory to be locked or unlocked.



Filename Mapping

Press the Project tab. **Filename Mapping** is used to specify how filenames should be mapped on the UNIX system. There may be a need to do this to match filenames entered on statements such as CALL statements.

Publisher Setup finds the filenames in the current application and suggests a mapping strategy. The names can be set as all upper case, or all lower case using the following dialog:

- Mapped Name shows filenames, as they are known to the file system.
- Logical Name shows the filenames as displayed by Net Express.

A name for the mapped filename can be entered, and this name used when it is copied to the UNIX system. All files created from the mapped name have the same case mapping. For example, CUST.cbl would result in CUST.int. The mapping dialog cannot be used to change the filename.

Pre-build and Post-build Commands

The Pre-Build Command and Post-Build Command allow any commands that can be executed on a UNIX system to be specified. The commands are embedded in the makefile created by the Publish operation. Because the commands are embedded in the makefile:

- The build can be recreated on the UNIX system by running the make command.
- The COBOL environment settings set on the UNIX system are available to the commands
- Any errors reported use the success / failure reporting mechanisms on the UNIX system.

Copyfile Handling

Press **Logical Directories**. The Server Copyfile Directory field is used to specify a location on the UNIX server in which to place copyfiles when publishing. However, copyfiles in the project source directory are treated as any other source file and will be copied to the Build Directory.

All copyfiles found using the COBCPY environment variable will be copied to the Server Copyfile Directory. The Makefile generated is modified to pick up copyfiles from this new location.

System Copyfiles

Some COBOL components call system copyfiles when building an application. With Net Express, all of these system copyfiles are stored in one place (NetExpress\Base\Source). On COBOL for UNIX the system copyfiles are stored in a number of directories, and may have the extension .cpy or .CPY.

When publishing, system copyfiles are not copied to the UNIX server. (COBOL for UNIX has to use its specific system copyfiles). When publishing, a message that the system copyfiles have not been copied is displayed in the Publish pane of the IDE.

Publisher cannot dynamically search for system copyfiles on the UNIX system, so it assumes that they are in the directory \$COBDIR/cpylib. The Makefile created by Publisher on the UNIX system expects system copyfiles to be in this directory. However, if the build fails due to a problem locating a required system copyfile, the file can be copied to the directory \$COBDIR/cpylib and the make command used to rebuild the application.

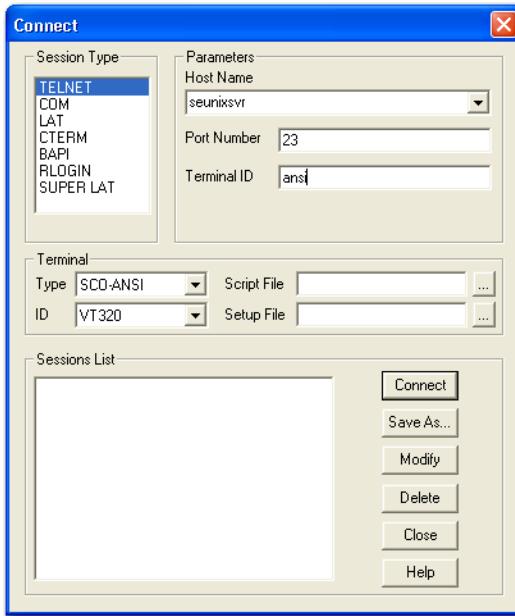
Connecting to the UNIX Server

Check with your server administrator for your UNIX home directory. This is the location you will be publishing to.

Logging on to your Server and Starting Server Express

1. Start Net Express. Select **UNIX, Terminal** to bring up PowerTerm. PowerTerm is a terminal emulator for the UNIX environment.
2. When Powerterm starts, select **Communication, Connect**.
3. Enter the following:
 - Session type: TELNET
 - Host name: UNIX server name (see your administrator, for example myunixsvr)
 - Port number: probably 23

- Terminal ID: ansi
- Click Connect.

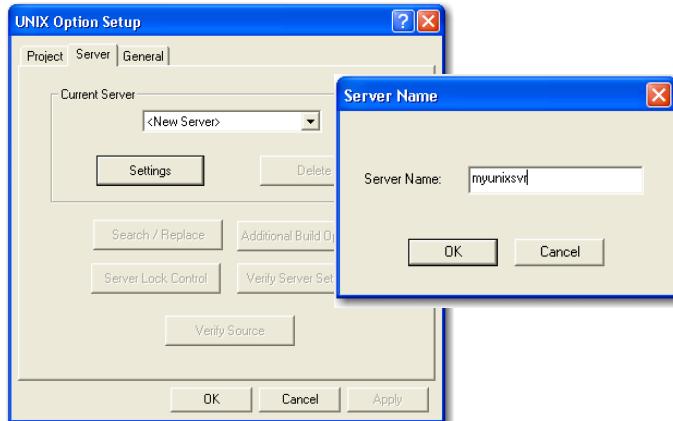


Note: If the logical namer (myunixsvr) is not seen by your machine, you may replace it with the IP address (111.222.333.444 – see your administrator).
To test whether you can see the server, try to ping it from a dos prompt.
Example: ping myunixsvr or ping 111.222.333.444.

4. You should now be connected to the Unix Box. Type in your userid and then enter a password. You will login and start in your directory
5. You should be able to type in *sde* or *tbox* to start Server Express.
6. Escape out of Server Express.

Setting up Publishing to UNIX

1. Select **Unix, Setup**.
2. Click on the Server Tab.
3. Click on Settings. Enter the server name (e.g., myunixsvr) then click OK.



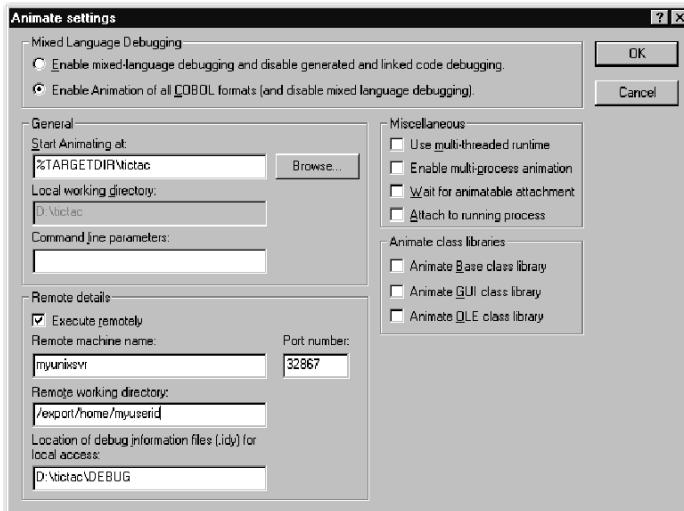
4. Enter the following:
 - Your Userid
 - The directory on the UNIX server where Server Express is located
 - The directory you are publishing to on the UNIX server
5. Click **OK**.
6. Press **Verify Server Settings** to be sure you are connected.
7. Compile your program on the workstation.
8. Select **Unix, Publish All**. This should make Net Express publish your program to your UNIX directory. **Hint:** Use explorer to examine the new files that were copied over . Or, use PowerTerm and type `ls` to view files in your directory. (The UNIX `ls` command is similar to the DOS DIR command).

Remote Debugging

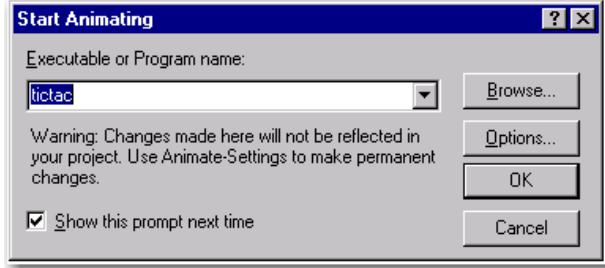
In order to do remote debugging you must have the PowerTerm session up and logged in.

1. Type **animserv** at the UNIX command line. ((Note: there are four Server Express variants of the animserv program, depending on whether you are running in 32-bit or 64-bit mode, and in single-threading or multi-threading mode. Consult your administrator for the exact command)).
2. This will give you a dynamically allocated port. Remember or write down the port number (e.g., 32800, but it changes every time).
3. In Net Express create a project and load a program into the project.
4. Compile the program.
5. Publish the program to the UNIX server.
6. Select **Animate, Settings**.

Do not forget to enter the Port Number that you obtained from your PowerTerm Session.



7. After setting up the screen start animating by pressing **Step** or selecting **Animate, Start animating**.
8. Click Yes to connect. You will receive a pop up window:



9. Enter the name as it appears on the UNIX machine (case sensitive). You do not need to include the .int extension or in the animate *settings* box, you edit the 'start animating at' and remove %TARGETDIR\TICTAC. Replace it with the name of the program you are running (e.g., tictac).
10. Click **OK**. You should begin to debug remotely. Remember all screen output will be on the PowerTerm session, not the application output window of Net Express. (Tictac is an interactive game: tic-tac-toe).
11. When you are finished debugging, select **Animate, Stop Animating**. Accept the defaults to stop animating and shut down the remote process.
12. You will have to close the project or exit from Net Express in order to stop the *animserv* running on UNIX.

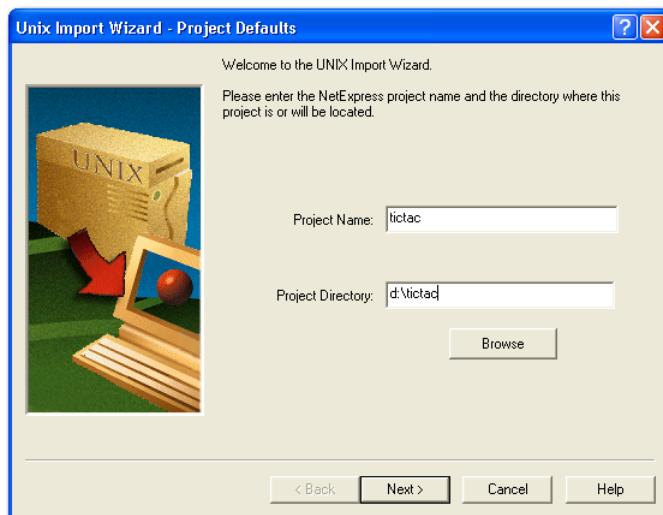
Copying UNIX Applications to the PC

Samba enables a UNIX directory structure to be accessible to a PC by mapping that directory to an available drive letter, using Windows Explorer. The files can be copied using drag and drop from the UNIX machine onto the hard drive of the PC. A Net Express project can then be created. Alternatively, the files can be left in their current location on the UNIX

machine, as the UNIX directory is now mapped as an additional area of the PC.

Using Import to Create your Project

1. Invoke Samba by mapping a drive to your UNIX server. To map a drive right-click Network Neighborhood on your Windows desktop. Select **Map Network Drive**. The first available drive will be displayed in the first input field.
2. Enter the address of your UNIX server (e.g., \myunixsvr or \1111.222.33.44). You will likely be prompted for userid and password.
Note: If \myunixsvr does not work, it may be because the dialed-up DNS server can't resolve the myunixsvr name. Get around this by adding 11.22.33.44 myunixsvr into your hosts file. It can be found in \winnnt\system32\drivers\etc\hosts.
3. Select **UNIX, Import**. Enter the project name (e.g., tictac) and its drive and directory on your workstation. Press **Next**.



4. In the Source Directories windows, press **Browse** and select the drive you mapped in Samba in step 1 above, then press **Next**.

5. In the source-files screen, select the drive you just entered in previous step.
6. After you select a drive, a file selection window will come up. Select tictac, press **Next**.
7. Make sure Convert Unix Text file to PC format is checked. Press **Next**.
8. The Miscellaneous screen is used to pass directive files. Press **Next**.
9. After the review screen, press **Finish**. The file should be copied over and the project created.
10. Compile the program. You will get punctuation warnings. When import builds the project it puts in directives that help catch potential .int portability problems. In the case of tictac the warnings can be ignored. Or, you may choose to remove the WARNINGS(3); directive from the project properties/directives.

Module Summary

This module has enabled you to do the following:

- Identify portability issues: syntax checking, syntax flagged, syntax not flagged, debugging (.idy files).
- Identify issues regarding installation of SCP and SAMBA.
- Provide Publisher setup information including configuration details and server capabilities.
- Identify Publisher's handling of filename mapping, pre-build and post-build commands, copyfile handling and system copyfiles.
- Connect to the UNIX server, log on to your server, and start Server Express.
- Set up publishing to UNIX.
- Perform remote debugging.
- Copy UNIX applications to the PC using import.

12.1 Publish Existing Application to UNIX

1. Within the Net Express IDE, Load the project UNIX-PUB01.
2. Select **Setup** from the **UNIX** menu and complete the necessary options.
3. Select **Publish All** from the **UNIX** menu to publish the application.
4. Select **Terminal** from the **UNIX** menu, to start the Ericom terminal emulator.
5. **Login** to a **UNIX** session.
6. Rename **MUSTOCK.idx** to **MUSTOCK.DAT.idx**.
7. Animate or Run the **LOCKING** program Anim **LOCKING** or cobrun **LOCKING**.

12.2 Change Existing Program and Publish to UNIX

1. Within the Net Express IDE, Load the project UNIX-PUB01.
2. Load the file LOCKING.CBL for edit.
3. Find All occurrences of Display.
4. Change the line display locking01-00 to display locking01-00 with reverse-video.
5. Rebuild the project and run LOCKING.
6. Select **Publish** from the UNIX menu to publish the changed program.
The Makefile will be updated, and only LOCKING.CBL will be published.
7. Switch to the UNIX Terminal Emulator.
8. Animate or Run the LOCKING program:
Anim LOCKING or cobrun LOCKING .

Module 15

Introduction to COM and DCOM

Overview

COM (Component Object Model) is a binary standard that allows applications to communicate with each other in a standard way. These applications use ActiveX objects. You can send messages to ActiveX objects, such as Word and Excel. ActiveX objects can also send messages to your COBOL applications.

DCOM (Distributed Component Object Model) are COM objects used over a network, using network protocols such as TCP/IP to communicate. DCOM therefore allows the client and server to be on two different machines.

Objectives

At the end of this module you will be able to:

- Employ the concepts of COM and DCOM
- Create a COBOL component
- Invoke the COBOL component from Visual Basic and COBOL
- Convert legacy COBOL code into components
- Use component automation from a COBOL program with Microsoft software such as Word and Excel

COM

To enable applications to communicate with each other using COM requires Object COBOL for the message-sending portion of the application. This includes the use of Object COBOL syntax as generated and supported by Net Express wizards. In the Visual Basic-COBOL example that follows, the Stockdemo.cbl program is an Object COBOL class program.

COM is intended for a Microsoft Windows 32 bit environment although there have been implementations in other operating systems. For the purpose of this module only Windows will be considered. In particular we will look at the "Automation" interface to COM. It is based on a single Interface, called Dispatch, which allows the dynamic binding of a client to a server. Automation servers can be DLLs, EXEs or even COBOL .INT/.GNT. COM is often referred to as OLE Automation because when Microsoft originally brought out this technology, the name it used was OLE2.

COM Example

For this example we will look at a COBOL Component and see how that Component is used from Visual Basic and from Dialog System. For the next example you will construct your own Component and interface to it from Visual Basic.

Loading a COBOL Component Application

1. Load the stockmgr project from \netxclass\projects\com-dcom\stockmgr. This project was based on a legacy application that has subsequently been changed to use Component Technology.
2. To run the application as it used to be, edit Stockmgr.cbl. Note that the line of code: [78 OLE value 1] is commented out.
3. Edit Stockfile.cbl. Change the SELECT statement for STOCK.DAT to point to the correct drive/directory.
4. Rebuild the Application and run it. Product codes are 000000001 to 000000004. Quantity and re-order are 4 digits each. Maximum price is 999.99. For any error (e.g., attempting to purchase more than is in

stock) you will receive a vague error message. It's not the prettiest application you have ever seen!

5. Return to editing Stockmgr.cbl and remove the comment on the line of code: 78 OLE value 1.
6. Rebuild the Application but do NOT run it yet. At this stage don't worry about how the Application is constructed; we will look at this later.

Registering the COBOL Component

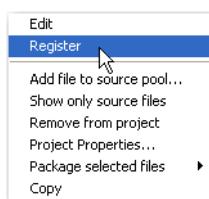
We now have built a COBOL component, but to use it we must first register it with the Windows registry. Follow these steps.

1. Right-click on the file stockdemoserver.reg (in the Source Pool View) and select Edit. This line must describe the location of the file correctly. Is the drive correct?

`@="c:\\netxclass\\projects\\com-dcom\\stockmgr\\debug\\..."` If not, change it and save the file.



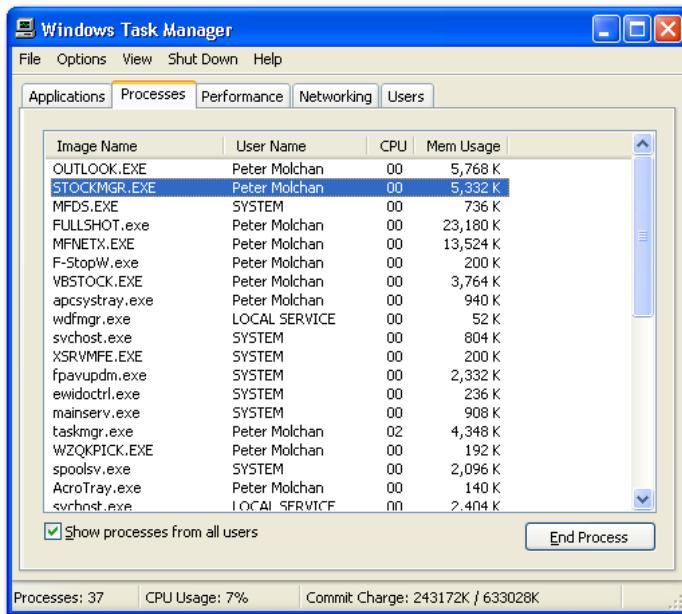
2. Right-click on the file stockdemoserver.reg and select **Register**.
3. If prompted to add this information to the registry, answer **Yes**. Your COBOL Component stockmgr is now registered. Stockdemoserver.reg was generated by Tools, OLE Registry File Generator.



Using the Component from Visual Basic

A Visual Basic Application has been supplied for you. Follow these steps to run the application.

1. Start up Windows Explorer.
2. Start the Application Vbstock.EXE from ...\\projects\\com-dcom\\stockmgr\\vb.
3. Start the Windows task list (processes) to see that VBSTOCK.EXE is executing (if you are running Windows 95 or 98, use the executable pview95.exe, supplied in \\com-dcom).
4. Click the **New** button on the VBSTOCK application and observe the task list processes (refresh the view if necessary).
5. See how the STOCKMGR.EXE Component has been started by the Visual Basic application.



6. Continue to run the Visual Basic application by clicking on one of the bar codes and then clicking on **TOTAL**.



7. Click **PAID** to complete the transaction.
8. Close the Visual Basic application and see how the STOCKMGR.EXE Component is shut down in the task list.

Using the Component from Dialog System and COBOL

A Dialog System Application has been supplied for you. Follow these steps to run this application.

1. Return to Net Express. Load the project Dsstock.app from:
c:\netxclass\projects\com-dcom\stockmgr\ds.
2. Run this application. You will see that it is not the same application as the Visual Basic version, but will use the same COBOL Component.



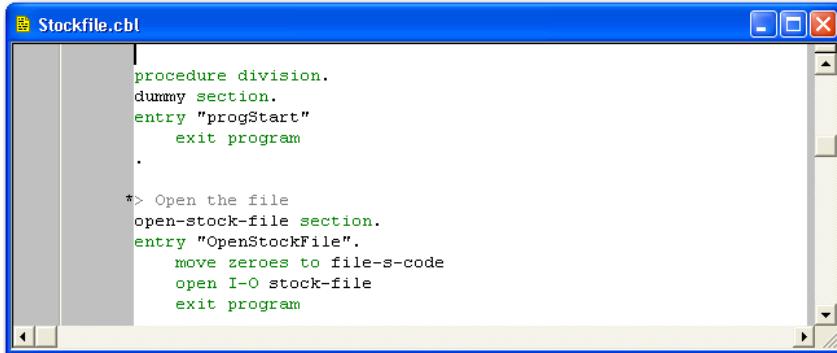
3. See how the task list now includes RUNW.EXE

4. Now click on the radio button Orlando and see how the COBOL Component is again started in the task list.
5. Start the Visual Basic application again and chose some items, Total them and Pay them.
6. Back in the Dialog Application refresh the view to see how those items have now been removed from Stock.
7. Start a second version of the Visual Basic application to see how you can run multiple versions of the Visual Basic application, but this will just use one version of STOCKMGR.EXE.
8. Close all the Visual Basic applications and the Dialog application. On the final close, STOCKMGR.EXE will be terminated.
9. Close the Process Viewer and Net Express. We will see how all this is achieved later.

More about the Stock Application

The stock application originally had just 2 COBOL programs. Oldstock.cbl and Stockfile.cbl. You can see this version by loading the project Oldstock.app from \com-dcom\oldstock into Net Express.

If you look at the code inside Oldstock.cbl you will find conventional COBOL code. If you look inside Stockfile.cbl again you will find conventional COBOL code, but not the code that was originally inside this program. The code that was not originally there are the program entry points.



```
Stockfile.cbl

procedure division.
  dummy section.
  entry "progStart"
    exit program
  .

  *> Open the file
  open-stock-file section.
  entry "OpenStockFile".
    move zeroes to file-s-code
    open I-O stock-file
    exit program
```

This program used to be called with a parameter, which then diverted the call to the appropriate section. The original code may have been something like this.

```
evaluate call-code
when "O"
    perform open-stock-file
when "C"
    perform close-stock-file
when "A"
    perform stock-change
etc
```

Entry points were added into appropriate parts of the legacy code.

Automating Word and Excel from COBOL

Automation can be used to access Microsoft Word and Excel via Com. A number of examples have been supplied with your Net Express installation. The following assumes that you have Microsoft Word and Excel installed on your machine.

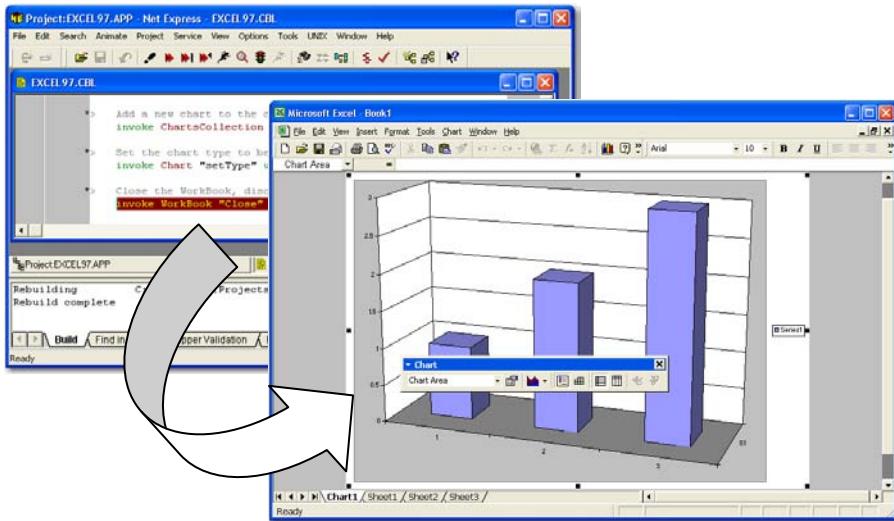
Automating Word from COBOL

Word can be controlled from a Net Express COBOL program.

1. Start Net Express and load the project Word97.app from ...\\com-dcom\\word97
2. This project contains a number of independent programs; word1.cbl thru word5.cbl. There is also an example (used, by word5) which shows how COBOL can be used to act upon events generated by Word.

Automating Excel from COBOL

Excel can be controlled from a Net Express COBOL program.



Start Net Express. Load the Excel97.app project from \com-dcom\excel97. This project contains a single program, excel97.cbl. Animate.

Finding Available Word and Excel Methods

You need to install the help for Visual Basic when Word and Excel are installed. The help within Visual Basic allows you to look at the Word and Excel methods.

DCOM

DCOM (Distributed Component Object Model) are COM objects used over a network, using network protocols such as TCP/IP to communicate. DCOM therefore allows the client and server to be on two different machines.

DCOM has many operational rules. Some of the most critical follow.

Location – You need to tell the client machine on where the server is located. You can do this in several ways:

- Run DCOMCNFG.EXE, find the description of the server, select "Properties" and "Location", ensure that the option "Run this program on the following computer" is selected then enter the name of the server.

- Add the location to your client registry file before entering it in to the registry, using the “RemoteServerName” entry under the AppID key.
- Identify the location dynamically. In Object COBOL use the “newWithServer” method.

Security Access – Make sure that each valid client user has both launch and access permissions to use the server. Use DCOMCNFG.EXE to change these.

Server identity – The server needs to know under which user identity it will run on its machine. Use DCOMCNFG.EXE to change the identity. Here are three options.

- The Launching User. The server will run under the user that the client machine is logged in with. Each client user will get a separate instance of the server launched for it. **Note:** Server machines have a VERY SMALL maximum number of simultaneous servers it can launch this way – this is because each user’s server is run under a separate Windows “WinStation” and these consume resources.
- The Interactive User. The server will run under the identity of the user logged in to the server machine. The server will only be launched once unless it is marked as a “single-use” server. It will only run if a user is logged on, and has access to the screen for GUI.
- This User. The server will run under a particular identity. The server will only be launched once unless it is marked as a “single-use” server.

NT Domain security - Clients and servers should normally be running in the same NT domain, but the same user logging on to two different NT Workstations with the same password should also allow a client to call a server, although the user must be added as a local user on an NT Server. If the client passes an automation object to the server, the server may not have the correct security privileges to call methods on that object. In this case the client should use the OLE API “CoInitializeSecurity” with appropriate parameters before passing the object.

In-process servers - In-process servers are used over DCOM using a proxy program on the server machine to load the DLL. The default is called

“Dllhost.exe”. The server machine needs the special named value “DIIISurrogate” under the AppId registry key specifying the proxy program (blank indicates using the default). Unlike on a local machine, if the server is registered as running under a single user identity, the proxy will only be started once and the instances of the server object will share data space in the same executable.

NTFS has security on directories – the identity that the server application is being run under must be given the right privileges to access and execute programs and data that have been protected in this way.

Microsoft Transaction Server

What is MTS?

MTS; Microsoft Transaction Server; (also known as MTX).

MTS is an extra layer on COM that provides for transaction integrity using COM in-process automation Components. It also provides further security abilities and a method of packaging Components for DCOM.

Net Express supports MTS using the objectContext class and methods within Object COBOL OLE automation Components.

What problems should I look out for using MTS?

Objects running under MTS must be DLLs. If you don't have a type library registered, the MTS Explorer can't show the interface for your Component. You also need to have installed the MTS component, rather than imported a pre-registered DLL.

MTS loads up all of the Components in a package at once when a client is started; therefore, all of the COBOL Components in each package must be linked with the same threading model. If you want to change the registry settings for an MTS Component, it is best to back the Component out of MTS first, make the changes and then re-import the Component into the package. This is because MTS alters the registry settings for each Component when entered.

MTS objects are stateless. That is, when the transaction under which they are running is finished (i.e., setAbort/setComplete has been called) the object is deleted. You cannot rely on the next method call following the completion of a transaction to connect to the same object.

Module Summary

In this module you have learned the do the following:

- Employ the concepts of COM and DCOM
- Create a COBOL component
- Invoke the COBOL component from Visual Basic and COBOL
- Convert legacy COBOL code into components
- Use component automation from a COBOL program with Microsoft software such as Word and Excel

15.1 The Original Stock Application

1. Animate the OLDSTOCK application to see how the code is constructed. What about the revised Application? The revised application, which is now implemented as a Component, comprises 4 programs.

Stockmgr.cbl	The replacement for Oldstock.cbl
Stockfile.cbl	The same as the Stockfile.cbl in the original application
Stockdemo.cbl	A new program, an Object COBOL Class, is used simply to redirect the invoked methods to Stockfile.cbl
StockTrigger.cbl	A new program used to set up the OLE message loop

Start the Revised COBOL Component

1. Load the StockMgr project com-dcom\stockmgr.
2. Start animating the application. The first statement call, OpenStockFile, calls the entry point in stockfile.cbl and opens the file.
3. Step through the code in Stockfile.cbl until you get back into the code in stockmgr.cbl and reach the line call stocktrigger.
4. Step on this code and you will see that this first line of code is call stockdemo. When you step on this line of code nothing appears to happen. But what has happened is that the stockdemo class has now been registered as an OLE class.
5. Step on the code, invoke olesup becomeserver. This code now causes the COBOL Component to go into a wait state.

Start the Visual Basic application that starts to use this COBOL Component

1. Start up Windows Explorer
2. Start the Application Vbstock.EXE from the directory: c:\com-dcom\stockmgr\vb.

4. Click on New and then click on one of the bar codes.
5. Return to Net Express. You will see that control is passed to the QueryItem method inside the Stockdemo class program. A method is the Object equivalent of a subroutine (e.g., a CALLED program). If you arrow key up a few lines (to method-id) you will recognize this as a small program within Stockdemo.cbl, with its own data storage section and a procedure division statement. The call “QueryItem” statement actually calls the QueryItem entry point in Stockfile.cbl.
6. Step though that code into Stockfile.cbl and then step until the COBOL Component returns to a wait state once more.
7. Return to the Visual Basic application. Click **Total** and then Click **Paid** and see you have now invoked the StockChange method inside the Stockdemo class.
8. Continue stepping through the code and finally close the Visual Basic application and step to the end of the COBOL Component.

How Can COBOL Use a Visual Basic Server?

This may be not something you would do very often, but it is quite straightforward if required. Visual Basic servers can be used by COBOL clients using the Object COBOL syntax.

```
$set ooctrl(+p)
class-control.

Vbserver is class "$OLE$Visual.Basic.Server".
...
01 theServer object reference.
...
invoke Vbserver "new" returning theServer
invoke theServer "finalize" returning theServer
```

“Visual.Basic.Server” is the programmatic id (ProgID) of the server. ProgIDs are stored in the Windows registry.

Module 16

Using Services with COBOL

Overview

This module provides an introduction to creating, deploying and consuming Services using Net Express.

Objectives

At the end of this module you will be able to:

- Use the Mapping Wizard to create a Web service with a default interface mapping and use the Interface Mapper to define your own operations and mappings for a Web service.
- Explain the use of an enterprise server to deploy and manage services.
- Understand the use of the Micro Focus Enterprise Server product.
- List the deployment for Web services, EJBs, and COM interfaces.
- Use Enterprise Server Administration to start an enterprise server.
- Deploy a Web service using an enterprise server.
- Use the Net Express mapping to create a client application that calls a service created in Net Express.
- Use a WSDL file to generate a client application in Net Express.
- Run client applications to access services deployed in an enterprise server.

Services Review

As defined previously, services are applications that process a client's requests. Services provide access to COBOL code from new technology client applications.

Services created with Net Express have three types of interfaces, Web services, COM, or EJB. The process to create each type of interface is similar. This section focuses on creating a Web service. The *Getting Started* documentation details the creation of each type of interface.

Web Services Review

Web services are software components that perform a certain task. They are used by client applications across the internet or intranet. Typical applications for Web services include credit card and check authorizations, weather information programs, package tracking, etc.

Web services make integration with COBOL code much easier by using the Internet and XML to create software components that communicate with others regardless of language, platform, or culture.

Web services use HTTP, XML, SOAP, WSDL, and UDDI to communicate and transfer data.

Using Net Express for Services

Net Express can be used to create services that expose existing COBOL applications and to define custom mapping.

Interface Mapping Toolkit

The Interface Mapping Toolkit, (IMTK), creates services that expose existing COBOL programs to non-COBOL client applications. These services can be Web services, EJBs or COM. Using this facility, COBOL programmers can build and use services without the need of another programming language to interface to the web.

The Toolkit works on programs designed to be subprograms (they have a Linkage Section). It maps the entry points and data items in the COBOL program onto an external interface. The original top-level program of the application is not used.

The Toolkit consists of the following tools:

- **Mapping Wizard.** The Mapping Wizard generates external service interface definitions based on existing procedural COBOL program interfaces. It extracts the COBOL linkage section and entry points from a COBOL program.
- **Interface Mapper.** The Interface Mapper defines which of the fields sent between the Web service and a client is to be mapped onto which of the COBOL program's Linkage Section parameters.
- **Deploy Tool.** This tool deploys the mapping information to an enterprise server for Web services and EJB interfaces. For EJBs, only the COBOL package is deployed to the enterprise server. The EJB is deployed onto a Java application server.

Creating a Web Service

To create and provide a service fields need to be mapped to create the service interface. The service is then deployed to an enterprise server. In this module we look at creating a Web service interface by mapping the fields using the Interface Mapping Toolkit. Creating an EJB or COM service is a similar process. The process for creating an EJB service is detailed in *Introduction to Using Java with COBOL*.

Mapping the Fields

Use the Interface Mapper to define the mapping between fields in the client program and fields in the COBOL subprograms Linkage Section. This interface mapping is stored in a mapping repository. By default, the mapping repository has the same filename as the project with an extension of .mpr.

In COBOL, you describe a data item's data type precisely, using its picture-string. XML (used for sending data between a client and a Web service),

Java and COM each define their own sets of data types. The Interface Mapper maps the COBOL types in your program onto the types available in XML, Java, or COM.

Using the Mapping Wizard

The Net Express Mapping Wizard generates a default mapping that reflects the current interface of the COBOL program.

Sections that follow in this module use a project called the Calculator project. The calculator program, calc.cbl, uses a parameter to determine whether to add, subtract, multiply or divide two arguments supplied by the user.

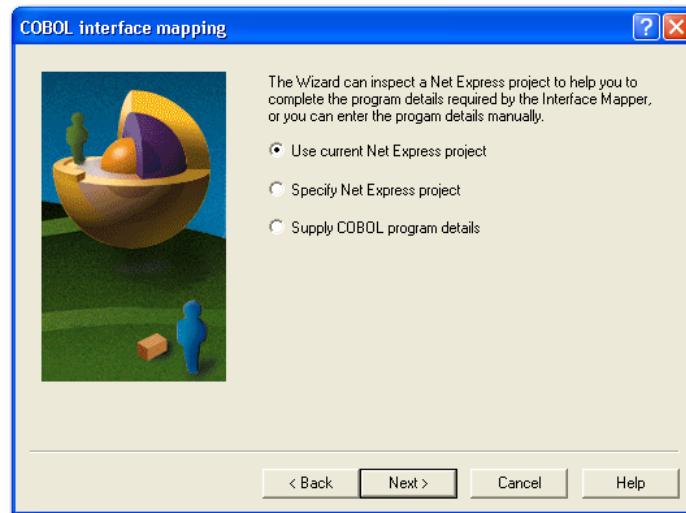
Follow these steps to create a Web service from calc.cbl using the default mapping supplied by the Mapping Wizard.

1. Open the **Calculator project** from the folder **\NetXClass\Projects\CALCULATOR**. **Rebuild** the project.
2. To create the Web service, select **File, New, Service Interface** and click **OK**.

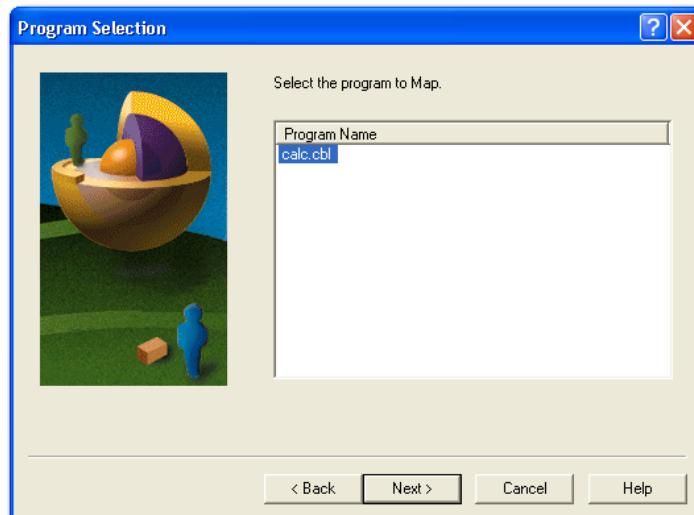


3. The Mapping Wizard steps you through the mapping automatically and displays a window for you to choose what type of service you wish to map. Note the four types of services. Choose **Map COBOL as a Web Service** and click **Next>**.

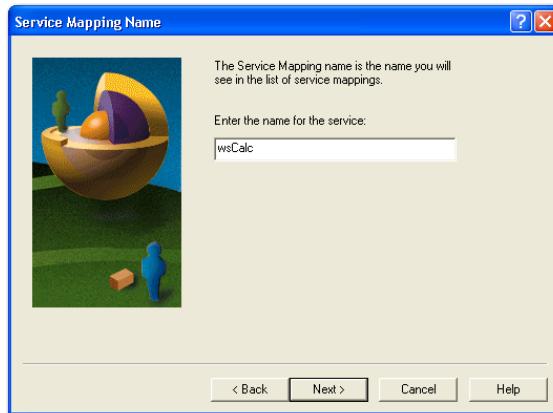
4. Select **Use current Net Express project** and click **Next>**.



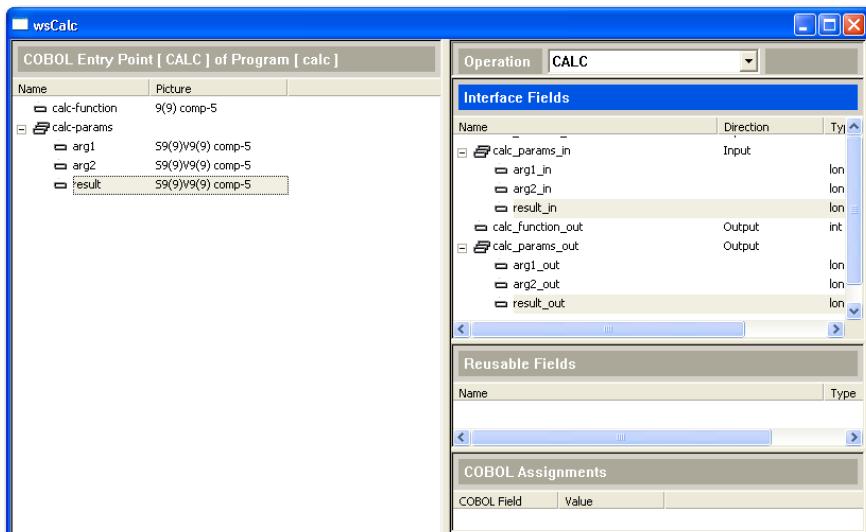
5. The Program Selection window displays. Select **calc.cbl** as the program to map and click **Next>**.



6. Specify the name for the service mapping. Enter **wsCalc** and click **Next>**.



7. Select **Default Mapping**, click **Next>** and then **Finished**. The Interface Mapper displays and shows the default mapping created for the Web service wsCalc.



The Web Service wsCalc

The Web service created, wsCalc, has one operation, CALC. This operation performs one of the four arithmetic functions depending on the parameter passed.

The Interface Fields

An input and output interface field is created for each Linkage Section item. The input fields represent incoming messages to the service. The output fields represent the response sent back to the client application by the service. The interface fields have XML data types and the hyphens are replaced with underscores. Note the corresponding fields below.

```

working-storage section.
linkage section.
01 calc-function      pic x(4) comp-5.
78 78-CALC-FN-ADD   value 0.
78 78-CALC-FN-SUB   value 1.
78 78-CALC-FN-MUL   value 2.
78 78-CALC-FN-DIV   value 3.

01 calc-params.
  05 Arg1            pic s9(9)v9(9) comp-5.
  05 Arg2            pic s9(9)v9(9) comp-5.
  05 Result          pic s9(9)v9(9) comp-5.

```

Linkage Section Item	Input Field	Output Field
calc-function	calc_function_in	calc_function_out
Arg1	arg1_in	arg1_out
Arg2	arg2_in	arg2_out
Result	result_in	result_out

New Files Created

XML is the basis for data transmission and communication between the Web service and client applications. Notice there are two XML documents created for this service.

This new file...	Contains...
calc.xml	XML text describing the calc program.
wsCalc.xml	XML text describing the wsCalc service.
Calculator.mpr	All service interfaces defined for the project. This list populates the Service Interfaces window.

Calculator.mpr is created in the project directory. The XML files are created in the \NetXClass\Projects\CALCULATOR\Calculator\REPOS folder.

New Menu Items

Four menus, Operation, Field Mapping and Assignment, are added to Net Express as a result of creating the service. These menu options are available when the Interface Mapper is open. Use the Operation menu to add, remove, or change an operation. The Field menu provides functions to add, remove, or change fields and their mapping.



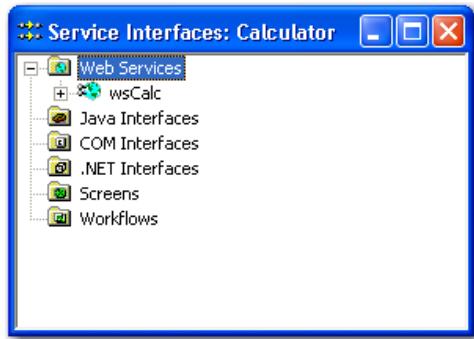
Defining Your Own Interface Mapping

More specifically, you can create a Web service with operations to perform each function of the Calculator program. Then the client application can call only the operation it needs.

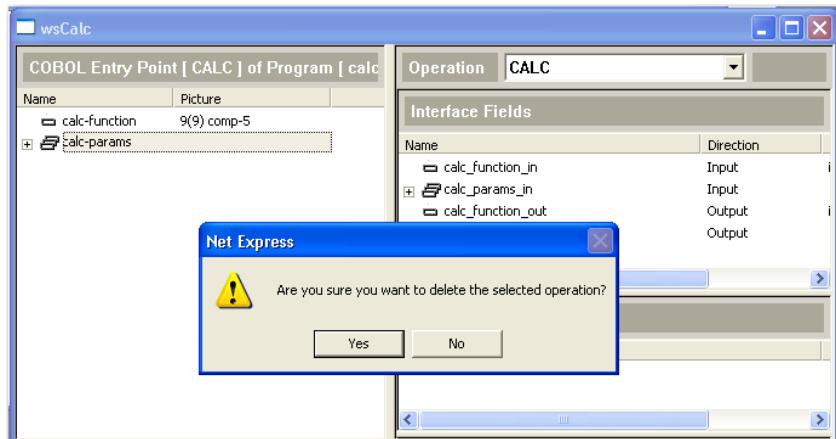
Follow these steps to create operations for multiplication and division.

1. Close all windows except the Calculator project window. Select **File, Open Service Interfaces**. Select **Calculator.mpr** and click **Open**. The

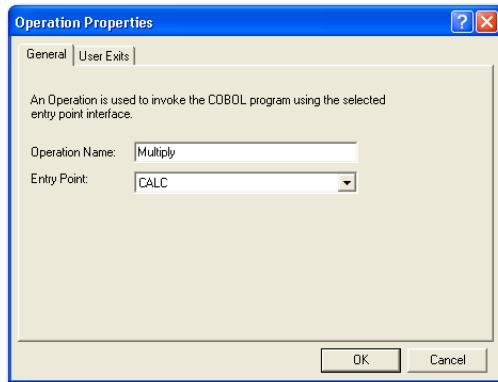
Service Interfaces window displays. Note the wsCalc service is the only service defined for the Calculator project.



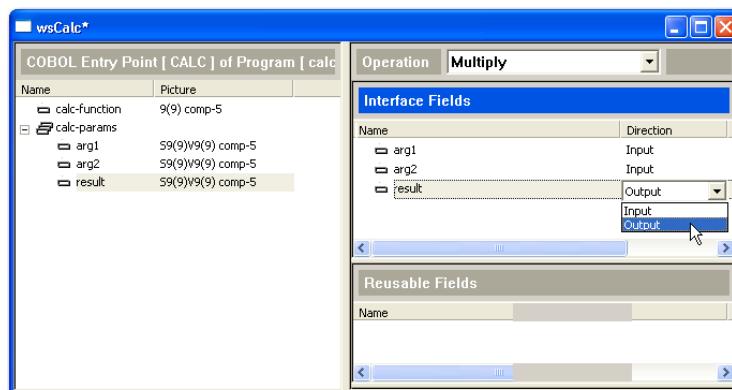
2. Double-click **wsCalc** to display the Interface Mapper window for this service (or select **Service, Edit**). Currently, there is one operation, **CALC**.
3. Remove the CALC operation created by the default mapping since we are replacing it with two operations – one dedicated to multiplication and one dedicated to division (we could also create operations for Add and Subtract). Make sure the Operation says **CALC** and select **Operation, Remove** and respond **Yes** to remove CALC.



4. To add an operation to perform multiplication, select **Operation, New**. The Operation Properties window displays to define the operation.
5. Enter **Multiply** as the Operation Name and **CALC** as the entry point. Click **OK**.



6. The Interface Mapper window displays. Expand **the fields in the linkage section (calc-params)**.
7. Drag the fields **arg1**, **arg2**, and **result** over to the Interface Fields pane. Notice by default they are all input parameters. Double-click on **the result field's direction**. Use the drop-down to change it to an output field.



Using Preset COBOL Values

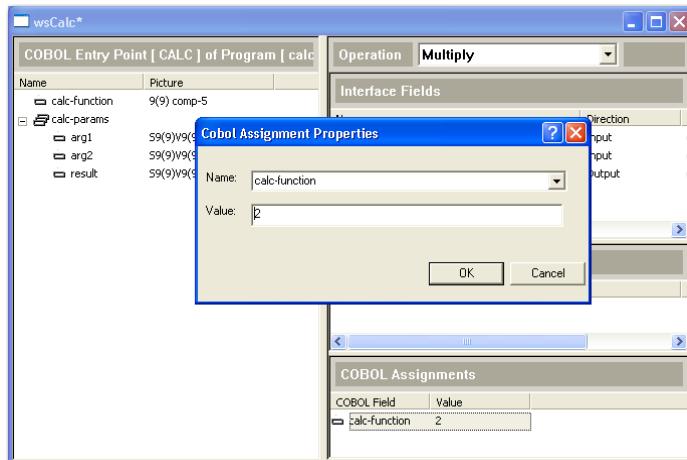
The Interface Mapper allows you to provide values for arguments that would otherwise be passed as parameters. This eliminates the need for the client application to have to pass the value as a parameter.

Since the Multiply operation always performs only multiplication, we can preset the value of calc-function to reflect that. Then there is no need for the client application to pass this value. Looking at the calc program, if the calc-function parameter has a value of 2, the operation performed by the program is multiplication.

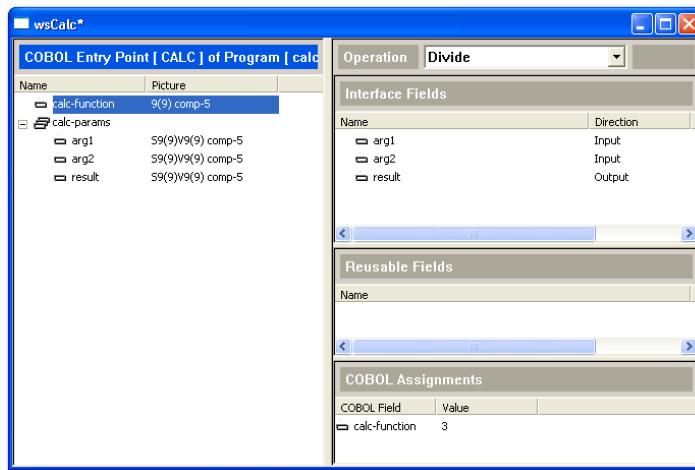
```
linkage section.
01 calc-function    pic x(4) comp-5.
78 78-CALC-FN-ADD  value 0.
78 78-CALC-FN-SUB  value 1.
78 78-CALC-FN-MUL  value 2.
78 78-CALC-FN-DIV  value 3.
```

Follow the steps below to preset the value for calc-function to 2 for the operation Multiply. That will make the calc program perform only multiplication.

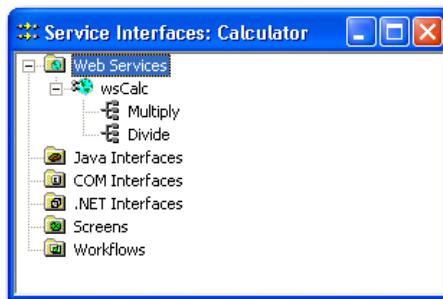
1. Drag **calc-function** over to the COBOL Assignments pane. Set the Value to **2** and click **OK**.



2. The Multiply operation is complete. The Interface Mapper window re-displays.
3. Repeat the process to create a Divide operation. This time, drag **calc-params** into the Interface Fields pane, right-click and select **Ungroup** to get the interface fields. Make sure to change result to an output field and preset the calc-function to 3.



4. Close the Interface Mapper window and save changes. Open the Service Interfaces window and note the new operations for Multiply and Divide.



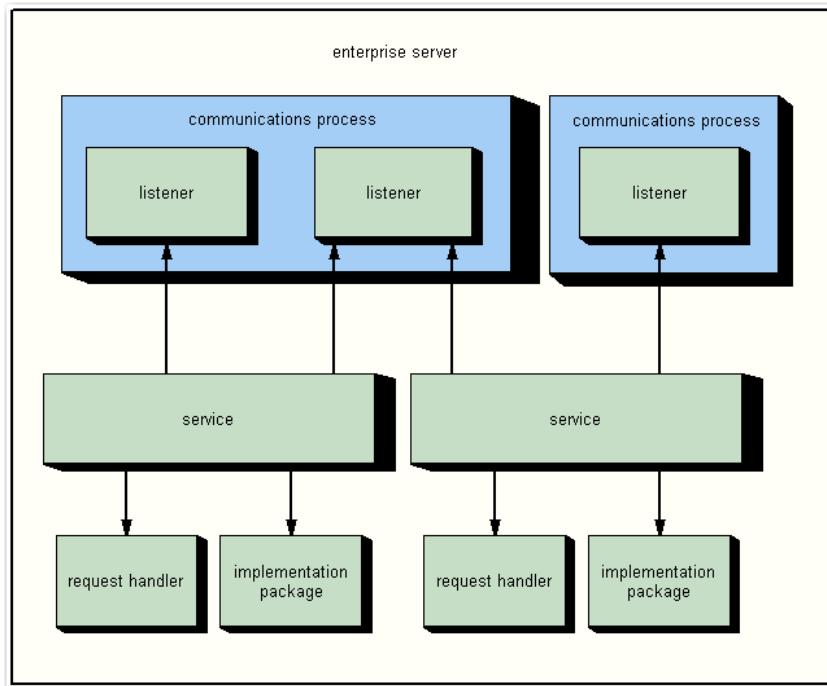
5. The Web service is ready to be deployed and be available for a client application to use. Save and close the **Calculator** project.

Introduction to Deploying Services

Micro Focus Enterprise Server

Micro Focus Enterprise Server is an execution environment for COBOL applications running as services. This provides a scalable, managed and transactional environment for the deployment of COBOL services, COBOL/J2EE applications and for hosting COBOL Web services to Windows, LINUX and UNIX.

Enterprise Server includes the Micro Focus Application Server providing a high performance, robust, proven and portable platform for the deployment of Net Express COBOL applications to Windows, LINUX and UNIX. It has the run-time support files for applications developed using Net Express.



Using Enterprise Server

Enterprise Server provides a complete infrastructure for publication of business services. Use Enterprise Server to:

- Deploy applications as services.
- Manage deployed services.
- Expose services to remote clients.
- Execute services in a stable environment.

Deploying Applications as Services

Use Net Express tools to develop service interfaces for existing COBOL programs and then deploy the services to Enterprise Server. These COBOL services can be invoked by clients as Web services, through a J2EE connector, or through COM. Enterprise Server also acts as an overall COBOL deployment environment for service-orientated applications.

Managing Deployed Services

Enterprise Server provides a web-based interface facility, Enterprise Server Administration, to assist you with the administration of:

- Server configuration
- Listeners
- Request handlers
- Services
- Implementation (Deployment) Packages

Additionally, Enterprise Server provides live monitoring of the server, logging and diagnostic capabilities.

Exposing Services to Remote Clients

Enterprise Server listens for requests from the client applications to invoke the services and handles these requests coming in through a mapped interface from the client. It passes the requests to the COBOL run-time

system to be processed and then returns the response back through the interface to the client.

Executing in a Stable Environment

Enterprise Server provides a stable execution environment. A COBOL application that is deployed as an Enterprise Server service is always run inside one of the servers' service execution processes (SEPs). The part of the service execution process that runs the application is a tightly managed COBOL environment, or COBOL container. Enterprise Server guarantees maintaining a consistent state for this container between application invocations to ensure consistent execution behavior.

Getting Started with Enterprise Server

Installing Enterprise Server

Enterprise Server accompanies Net Express and may be installed on your development machine for use in testing. In a production environment, Enterprise Server is typically set up on a dedicated server and the site administrator performs server administration. For production use, a licensed copy is required.

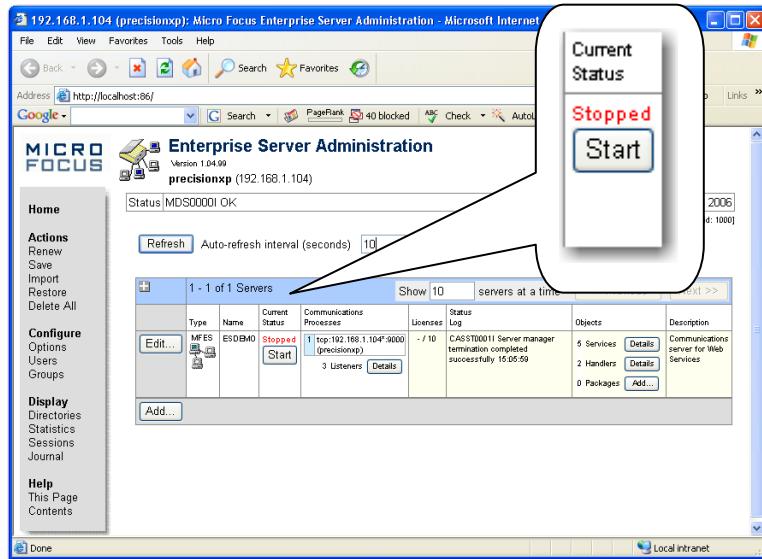
Starting Enterprise Server

Follow these steps to start Enterprise Server Administration and start an enterprise server session.

- Select **Start, Programs, Micro Focus Net Express, Configuration, Enterprise Server Administration.**
- Start **Net Express** and select **Tools, Enterprise Server Administration.**
- Start a web browser and enter the URL <http://localhost:86>.

The Enterprise Server Administration home page displays.

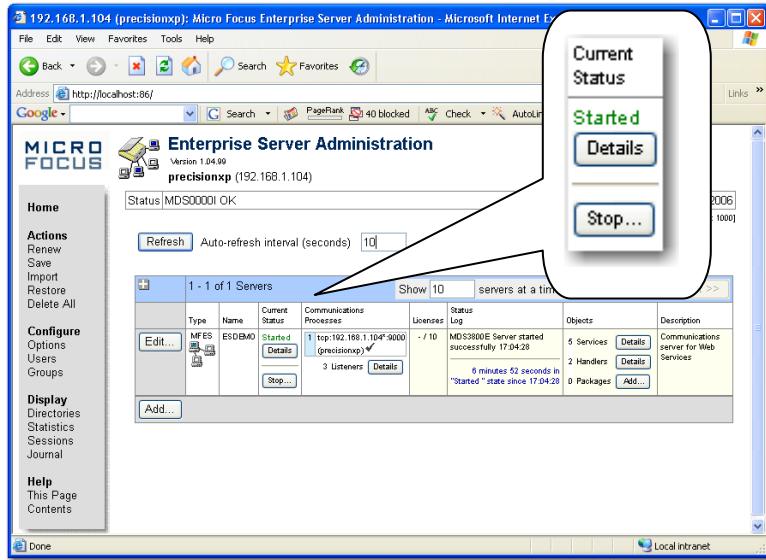
A default server configuration, ESDEMO, is created for you. Start the server from this configuration by clicking **Start** in the Current Status column.



The Enterprise Server Console Daemon window displays. When the message, **Communications Process instance 01 is ready to accept requests** displays, minimize the window.

```
3740 17:04:17 CASSCD0127I SEP 00002 created for ES ESDEMO, process-id = 3484 17:04:17
3484 ESDEMO CASSII1500I SEP initialization started 17:04:18
3740 ESDEMO CASSII1500I SEP initialization started 17:04:18
3160 ESDEMO CASSII1500I Event Manager initialization started 17:04:17
3160 ESDEMO CASCSD1038I ES Communications Server created, ES ESDEMO, process-id = 2312 17:04:18
3160 ESDEMO CASSII1503I Event Manager initialization complete 17:04:18
3840 ESDEMO CASJC0001I Journal control initialized 17:04:18
2140 ESDEMO CASKC1000I ES concurrent request limit: <unlimited> 17:04:18
2140 ESDEMO CASSII1000I Server Manager initialization completed successfully 17:04:18
3484 ESDEMO CASSII1600I SEP initialization completed successfully 17:04:21
3740 ESDEMO CASSII1600I SEP initialization completed successfully 17:04:22
2312 ESDEMO CASCSS5001I Communications interface 01 initialization started 17:04:24
2312 ESDEMO CASCSS5003I Communications interface 01 initialization complete 17:04:26
2312 ESDEMO CASCSS5100I Communications Process instance 01 is ready to accept requests 17:04:29
```

The Current Status for ESDEMO changes to Started.



When a COBOL service (such as one created using the Interface Mapping Toolkit) is deployed, the service and its components (operations and packages) are received and added it to the server execution environment.

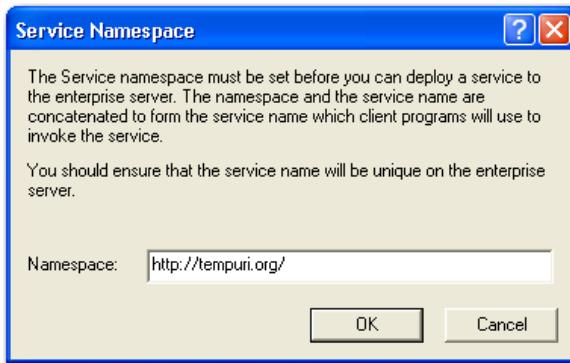
Configuring the Enterprise Server

Enterprise Server Administration is detailed in the online document *Configuration and Administration Guide*, section under Micro Focus Server. For this session, we set the necessary deployment options to deploy a Web service using the default configuration ESDEMO.

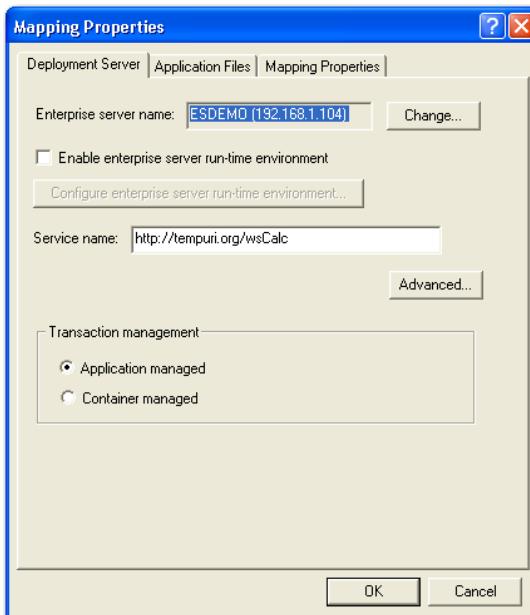
Note: examples below will use the same project (Calculator project) as the previous section of this module.

6. Open the **\NetXClass\Projects\CALCULATOR\Calculator** project.
7. Select **File, Open Service Interfaces**. Select **Calculator.mpr** and click **OK** to display the Service Interfaces window.

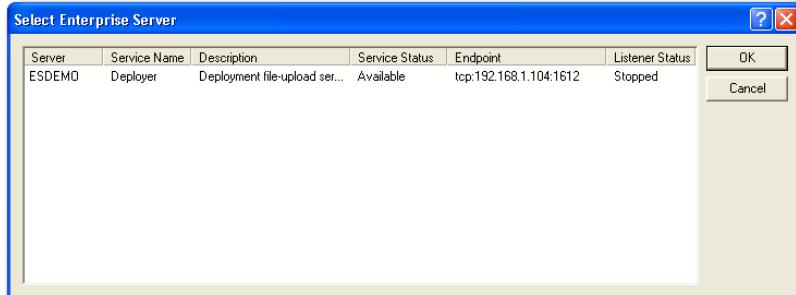
8. Right-click the service **wsCalc** and select **Settings**. The Service Namespace window displays.



9. Leave the default Namespace and click **OK**. The Mapping Properties window displays.

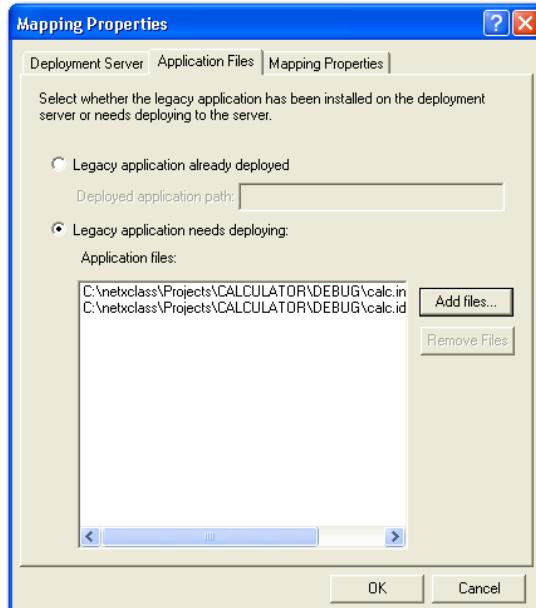


10. The enterprise server we want to use is ESDEMO. Click **Change** for the enterprise server name field. The list of available enterprise servers displays. Select **ESDEMO** and click **OK**.



11. Select the **Application files tab**. Select **Legacy application needs deploying**. Click **Add Files...** and add these files.

\NetXClass\Projects\CALCULATOR\DEBUG\calc.int
\NetXClass\Projects\CALCULATOR\DEBUG\calc.idy



12. Click **OK**. The deployment settings are now set for the wsCalc service.

Deploying Services with Enterprise Server

The Interface Mapping Toolkit can be used to create services that are Web services, EJBs or COM. Deployment information is listed below for these services.

To deploy the	Use...
Web service	a Micro Focus enterprise server.
EJB	a Micro Focus enterprise server to deploy the mapping and the application service, and a Java application server to deploy the EJB.
COM interface	Net Express to deploy the component and register it with Windows.

COBOL Web Services

Use Enterprise Server to deploy COBOL Web services created with Net Express. These services are standards based and can be consumed by any language on any platform that acts as a Web service client including Java, .NET languages and Net Express COBOL.

Enterprise Server acts as a SOAP Server and requires no third party application or web servers when deploying Direct COBOL Web Services.

COBOL/J2EE Applications

When deployed with Enterprise Server and the Micro Focus J2EE Connector, COBOL applications can be used in a J2EE compliant manner from a Java Application Server such as IBM WebSphere or BEA Weblogic.

Transactional COBOL Applications

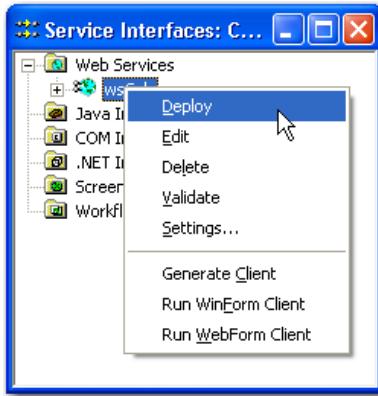
Use Enterprise Server to deploy transactional COBOL applications in a managed environment without the purchase of third-party software. Transaction support for the applications is provided when they are a part of a container-managed Web service or a transactional J2EE/connector request. The applications must be deployed as services and invoked from either Web services or J2EE.

Deploying a Web Service

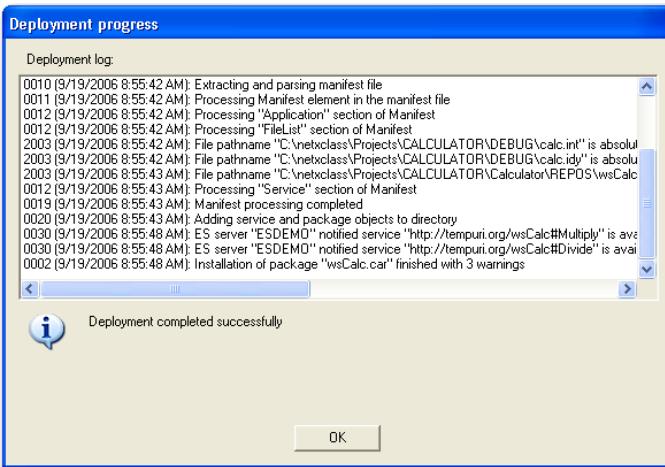
In this session, we will look specifically at deploying a Web service created earlier in this module. **Note:** be sure the Server is started.

Follow these steps to deploy the wsCalc Web service.

1. In the Service Interfaces window, make sure the service, **wsCalc** is selected. Select **Service, Deploy**. Alternately, right click on **wsCalc** and select **Deploy**.



2. The Deployment Progress window displays the Deployment Log. The service has deployed successfully.



3. Click **OK** to close the Deployment progress window and minimize the IDE.
4. Return to the **Enterprise Server Administration window**. Click **Details** for Services. A window displays with details for each service. Notice there are several system services and wsCalc with the two operations, Multiply and Divide.

#Multiply Edit...	1	1 CP 1 Web Services tcp:192.168.1.104*:9003 (precisionxp)	MFRHSOAP	http://tempuri.org/wsCalc	Available	OK
#Divide Edit...	1	1 CP 1 Web Services tcp:192.168.1.104*:9003 (precisionxp)	MFRHSOAP	http://tempuri.org/wsCalc	Available	OK

5. Click **Edit** for the Multiply operation. Notice the details about the operation you can change such as transaction management, status, and listeners.

Name:

Search Order:

Service Class:

Version: . .

Logging:

Transaction Management: Current Status: Available

Application New Status: Available

Accepting Listener(s) Available Listener(s)

CP 1 - (Web Services) tcp:192.168.1.104*:9003	CP 1 - (HTTP Echo) tcp:192.168.1.104*:9002 CP 1 - (Web) tcp:192.168.1.104*:2241*
---	---

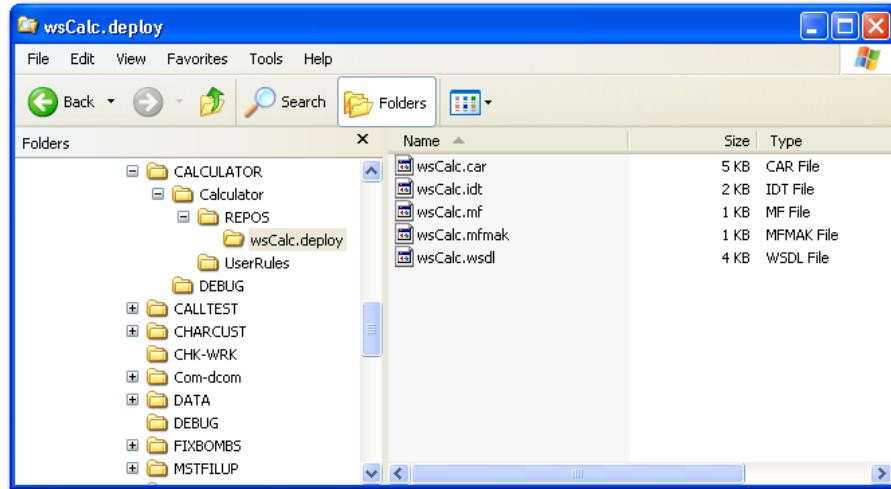
[Remove ...>](#) [<- Add](#)

New Files Created

Files created as a result of deployment vary by type of service (Web service, EJB, or COM). For the Web service deployed, wsCalc, a new folder, wsCALC.deploy has been created under the project's REPOS folder

This folder contains all files created to support deployment of the service and are listed below.

This new file...	Is the...
wsCalc.car	COBOL archive file for the service. It contains all the generated files that need to be deployed, including the mapping information, the COBOL executable and the data files. View the contents using the WINZIP.EXE utility.
wsCalc.idt	File containing the mapping information. It is packaged within the above .car file.
wsCalc.mf	XML file defining files associated with the service.
wsCalc.mfmak	Makefile used by the deployer.
wsCalc.wsdl	XML file defining the interface to the service. Give this file to people writing client applications.



Consuming Client Applications

Services that have been created and deployed are ready for client applications to access them. These services can be accessed by the following client applications.

This service type ...	Can be invoked by...
Web service	Any type of client that supports remote invocation of Web services.
EJB	Another EJB, JSP, or other J2EE component.
COM	.NET or the usual COM mechanisms.

During the discussion that follows, the focus will be on generating client applications to access a Web service.

Net Express and Client Generation

When a service is created using Net Express, the Interface Mapping Toolkit creates the mapping information for the service. If a service is deployed using Net Express, a WSDL file is created at deployment time. This file contains XML that defines the interface to the service.

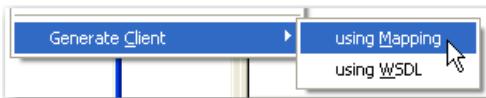
The Net Express client generation tool generates a COBOL client application and proxy program. These may be generated from a mapping (if service was created in Net Express) or from a WSDL file (service does not have to be created/deployed by Net Express).

The proxy program does the call to the service. There is no need for the programmer to change this program. The client application calls the proxy program. The programmer may optionally modify this application to add more complex features if required.

Generating a Client Using Mapping

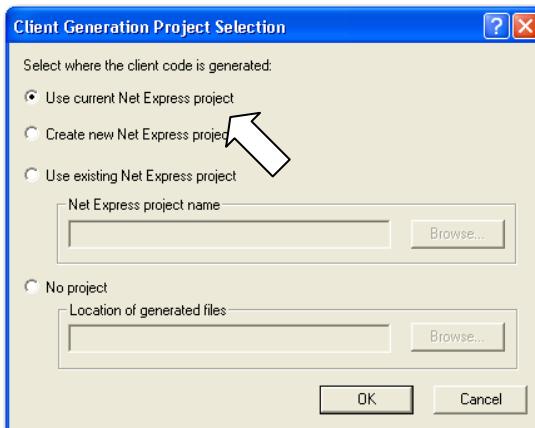
Follow these steps below to generate a client application for the service wsCalc using the mapping produced when the service was created in Net Express.

1. Open the **\NetXClass\Projects\CALCULATOR\Calculator** project.
2. Select **File, Open Service Interfaces**. Select **Calculator.mpr** and click **OK** to display the Service Interfaces window.
3. Make sure the enterprise server, **ESDEMO**, is started.
4. Select **wsCalc** in the Service Interfaces window. Select **Service, Generate Client, using Mapping**. **Note:** If you right-click on wsCalc and select Generate Client, it always uses the mapping, there is no option to use the WSDL.

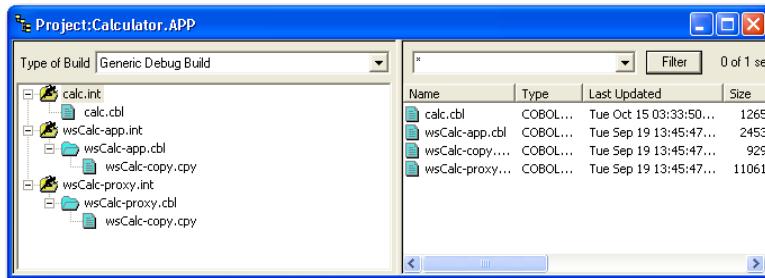


The Client Generation Project Selection window displays prompting you for the project location of where the client code should be generated.

5. Select **Use current Net Express project** and click **OK**. This places the generated code in the Calculator project.



6. A message indicating that the client generation was successful will appear in the message pane. .
7. Minimize the **Service Interfaces window** and look at **the project window**.



These files have been added to the Calculator project along with their build objects.

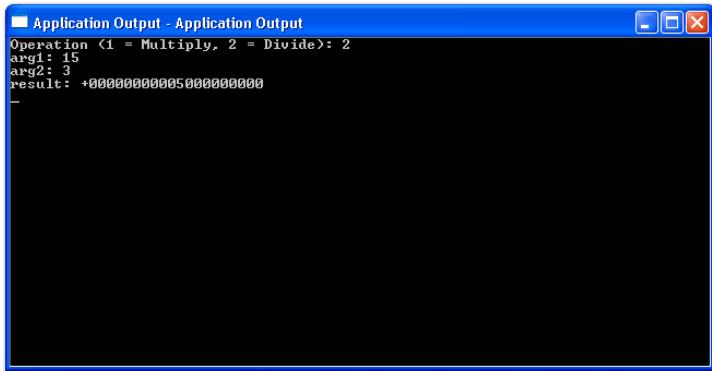
This file ...	Contains the ...
wsCalc-copy.cpy	Copybook information to support the generated application.
wsCalc-proxy.cbl	Proxy program that actually invokes the service.
wsCalc-app.cbl	Generated client application. This program calls the proxy program.

8. Edit the client application, **wsCalc-app.cbl** and notice that Net Express has assigned each operation, Multiply and Divide, a numeric code. The code for Multiply is 1 and Divide is 2. These do not relate to the preset COBOL values in any way, simply the order the operations are defined by the client program.

A screenshot of a COBOL code editor window titled "wsCalc-app.cbl". The code is as follows:

```
procedure division.  
    display "Operation ("  
        "1 = Multiply, "  
        "2 = Divide): "  
    no advancing  
    accept wsc-op-num  
  
    set wsc-proc_ptr to entry "wsCalc-proxy"  
  
    evaluate wsc-op-num  
        when 1  
            perform wsc-op-1  
        when 2  
            perform wsc-op-2  
        when other  
            display "Invalid operation"  
    end-evaluate
```

9. Click **Rebuild**  to rebuild the project.
10. Right-click **wsCalc-app.int** and select **Run**.
11. The Application Output window displays prompting you to enter the operation you would like to perform associated with the wsCalc service. Enter **2** for Divide. Supply **15** and **3** as the arguments. The result returned is 5.



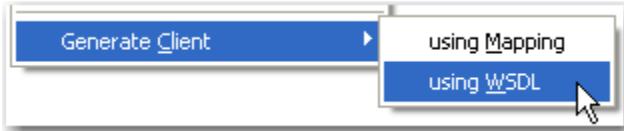
12. Try other combinations and the Multiply operation as well.

Generating a Client using WSDL

Client applications can be generated with most any WSDL file you have regardless of where the service was created or deployed.

Follow these steps to generate a client using the WSDL file for the wsCalc service.

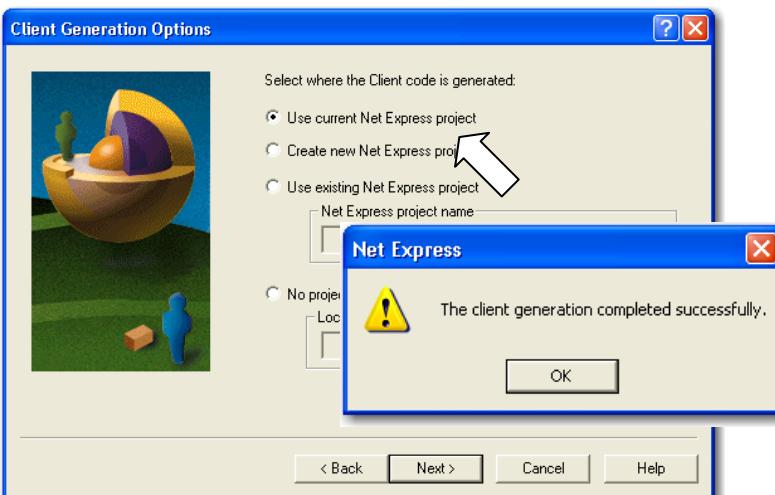
1. Remove the generated client files from the project (wsCalc-app.cbl, wsCalc-copy.cpy, wsCalc-proxy.cbl and their build objects).
2. Open the **Service Interfaces** window. Select **wsCalc** then **Service, Generate Client, using WSDL**.



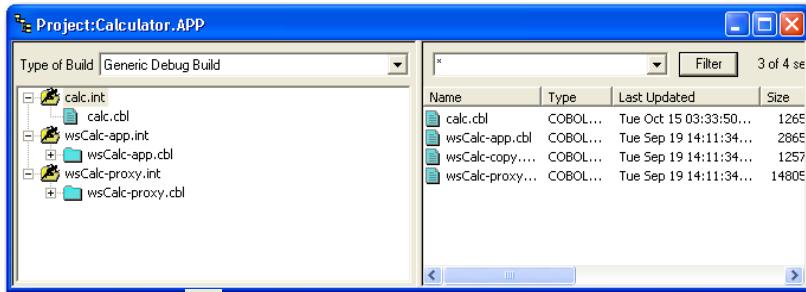
3. The Generate Client window displays. Click **Browse** and navigate to **\NetXClass\Projects\CALCULATOR\Calculator\REPOS\wsCalc.deployment\wsCalc.wsdl** and click **Open**.



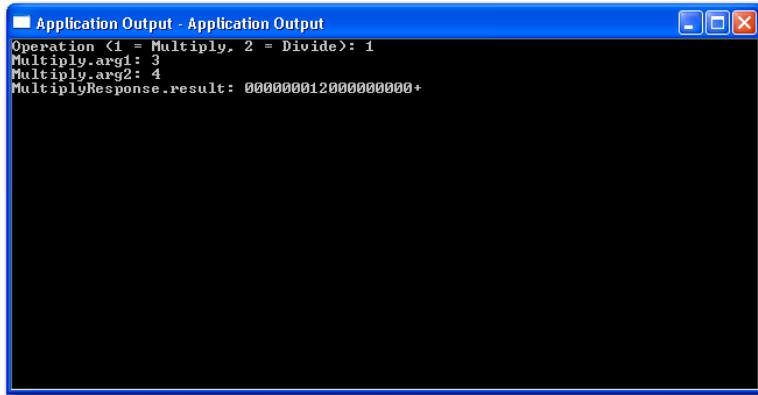
4. Click **Next**. Choose **Use Current Net Express Project**.
5. Click **Next**, then **Finished**. A window displays telling you the client has been generated successfully. Click **OK**.



6. Close the Service Interfaces window. Notice the same files in the project window as before.



7. Click **Rebuild**  to rebuild the project.
8. Right-click **wsCalc-app.int** and select **Run**. Click **OK** to begin program execution.
9. The Application Output window displays as before. Enter **1** for Multiply and enter **3** and **4** as the arguments. The result displays.



Module Summary

Services provide a way for COBOL code to be available to client applications that are written in various languages on various platforms.

The Net Express Interface Mapping Toolkit can be used to create Web services, EJB or COM interfaces.

An enterprise server is required to deploy a Web service or EJB interface. Micro Focus Enterprise Server allows you to create, configure, and manage enterprise servers.

Net Express supports client generation using a WSDL file (regardless of the origination) or a mapping file for services produced by Net Express.

Net Express client generation produces a proxy program that interfaces with the service. This proxy program is called by the COBOL client application.

Module 17

Using Java with COBOL

Overview

This module provides an introduction to Using Java with COBOL. It introduces the Net Express facilities for using COBOL and Java together and provides an example of how these facilities are used.

Objectives

At the end of this module you will be able to:

- List the ways Net Express supports using COBOL and Java together.
- Understand the deployment process for an EJB service interface.
- Run a client program to access a COBOL program using an EJB service interface.

Using COBOL and Java Together

Calling COBOL from Java

There are three ways to call COBOL from Java.

- **Accessing procedural COBOL with a service** – Use Net Express to create a service with an EJB interface to access the COBOL program. Deploy the interface mapping using an enterprise server and the EJB with a Java application server to J2EE.
- **Calling procedural COBOL from Java** - Use the support provided in the com.microfocus.cobol.RuntimeSystem class. This class is extended to provide the CobolBean class.
- **Calling OO COBOL from Java** - Use the Class wizard to create a Java wrapper class that provides a function for each method in the OO COBOL class.

Calling Java from COBOL

There are two ways to call Java from COBOL.

- **Calling Java from procedural COBOL** - Use the support provided in the com.microfocus.cobol.RuntimeSystem class to call Java through the Java Native Interface (JNI).
- **Calling Java from OO COBOL** – Use the Java object domain for OO COBOL.

The remainder of this module focuses on creating and deploying an EJB interface using an enterprise server and a Java application server.

Accessing COBOL with an EJB Service Interface

The simplest way of sending requests from Java to COBOL is to use the Interface Mapping Toolkit to generate an EJB. Deploy the EJB under a J2EE application server, which manages the connection through the resource adapter to an enterprise server.

The process steps for accessing COBOL using an EJB service interface follow below.

- Use the Interface Mapper to map an EJB interface onto the COBOL program.
- Deploy the mapping information and COBOL program to an enterprise server.
- Deploy the generated EJB to a third party application server running on J2EE (ie, WebSphere or WebLogic).
- Deploy the resource adapter to the J2EE application server.
- Call the EJB with a client program, for example a JSP client.

Mapping the EJB and COBOL Program

Use the Interface Mapping Toolkit to create the interface mapping for the EJB service and generate the EJB.

The COBOL Program

To demonstrate the generation and deployment of an EJB service interface, this section uses a sample COBOL program, book.cbl. This program maintains an indexed stock file that could be used by a bookstore. Follow the steps below to get familiar with the project and the program.

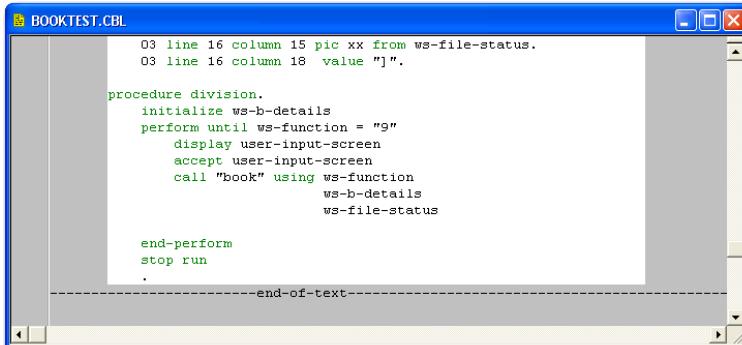
1. Open **\NetXClass\Projects\MAPDEMO\MAPDEMO** project.
2. Open **BOOK.CBL** and look at the logic of this program.

It has four functions - Read record, Add record, Delete record, and Get next record. It has three parameters - the function required, the data to be processed, and a data item in which to return the file status. The source identifies the function required by means of an EVALUATE statement testing the Ink-function parameter.

Close **BOOK.CBL**.

3. Open **BOOKTEST.CBL**.

This is a test program for book.cbl. It has a main loop that requests input from a screen interface and then calls the subprogram to process the data received.



```
BOOKTEST.CBL
03 line 16 column 15 pic xx from ws-file-status.
03 line 16 column 18 value "]".

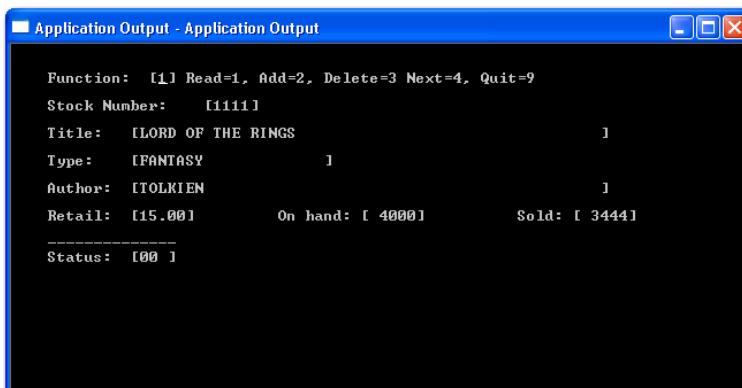
procedure division.
    initialize ws-b-details
    perform until ws-function = "9"
        display user-input-screen
        accept user-input-screen
        call "book" using ws-function
        ws-b-details
        ws-file-status

    end-perform
    stop run

.
-----end-of-text-----
```

Close **BOOKTEST.CBL**.

4. Click **Rebuild**  to rebuild the project.
5. Right-click **BOOKTEST.int** and select **Run**.
6. In the Application Output window, enter **1** for Read record. Enter **1111** for the book's stock reference number. The program looks it up in the indexed file and returns full details in Ink-b-details. These details display in the Application Output window.

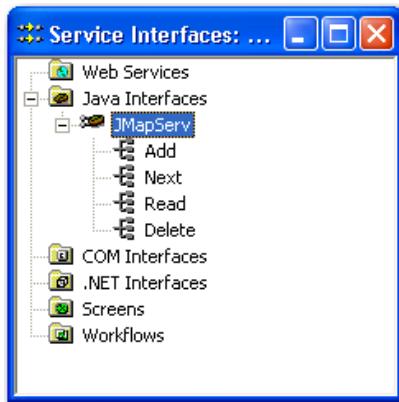


```
Function: [1] Read=1, Add=2, Delete=3 Next=4, Quit=9
Stock Number: 11111
Title: [LORD OF THE RINGS]
Type: [FANTASY]
Author: [TOLKIEN]
Retail: [15.00] On hand: [ 4000] Sold: [ 3444]
Status: [00 ]
```

Creating an EJB Service Interface

An EJB service interface (similar to the Web service created in Module 16) has already been created for book.cbl. Follow these steps to look at the interface.

1. With the map demo project open, select **File, Open Service Interfaces**. Open **\NetXClass\Projects\MAPDEMO\MAPDEMO.mpr**.
2. The Service Interfaces window shows the EJB service interface, JMapServ. Expand **the service** to see the four operations for the book program, Add, Delete, Next, and Get.

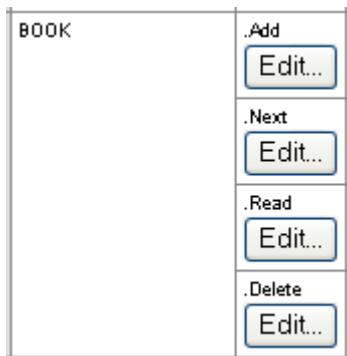


Deploying the Mapping

Deploy the interface mapping (COBOL side of the EJB) to an enterprise server. When you use Net Express to deploy the interface mapping to an enterprise server, it also generates and packages the EJB into a Java archive file, JMapServ.jar.

We have deployed the service to ESDEMO already.

1. Return to **the Enterprise Server Administration window** and click **Refresh**. You will see the BOOK service with four operations, Add and Next.



New Files Created

For the EJB deployed, JMapServ, a new folder, JMapServ.deploy has been created under the project's REPOS folder

(\NetXClass\MAPDEMO\MAPDEMO\REPOS\JMapServ.deploy). This folder contains all files created to support deployment of the service.

This new file...	Is the...
JMapServ.car	COBOL archive file for the service. It contains all the generated files that need to be deployed, including the mapping information and the COBOL and the data files. View the contents using Winzip.
JMapServ.idt	File containing the mapping information. It is packaged within the above .car file.
JMapServ.mf	XML file defining files associated with the service.
JMapServ.mfmak	Makefile used by the deployer.
JMapServ.jar	Java archive file containing the generated EJB. View the contents of this file using the jar command if you have a Java system installed.
com\mypackage\JMapServ	Directory containing several class files for the service.

Deploying to the Application Server

Next, you must deploy the Java part of the generated EJB and a resource adapter to a J2EE application server (such as IBM WebSphere or BEA WebLogic).

Resource Adapters

Before an EJB can communicate with the COBOL running on an enterprise server, you need to deploy a resource adapter on the J2EE application server.

Net Express provides resource adapters, or J2EE connectors, to enable EJBs deployed on J2EE to communicate with COBOL deployed on an enterprise server. Resource adapters are stored in a resource adapter archive (.rar) file.

At run time, the Java client software calls the EJB, which passes the request to the resource adapter, which communicates with the enterprise server to execute the COBOL service.

The resource adapter has already been deployed to WebLogic.

1. Using the WebLogic Server Console, expand Micro Focus, Deployments. Under Applications, notice the MapDemo service is deployed. Under Connectors, notice the mfcobol-notx connector is deployed.

The screenshot shows the WebLogic Server Console interface. On the left, there is a navigation tree with categories like Deployments, Applications, EJB, Web Applications, Connectors, Services, and Security. Under Applications, 'MapDemo' is listed. Under Connectors, 'mfcobol-notx' is listed. The main right pane has tabs for Configuration, Deploy, and Notes. The Deploy tab is active, showing the 'Deployment Status by Target' table and 'Deployment Activity' table.

Component	Component Type	Target	Target Type	Deployed	
JMapServ.jar	EJB	MF-server	Server	true	Undeploy Redeploy
MapDemo.war	Web Application	MF-server	Server	true	Undeploy Redeploy

Buttons below the table: [Undeploy Application](#) (Undeploy the entire application), [Deploy Application](#) (Deploy or redeploy the entire application).

Description	Status	BeginTime	EndTime
Activate application MapDemo on MFserver	Completed	Mon May 19 13:40:08 PDT 2003	Mon May 19 13:40:31 PDT 2003

For details about deploying these pieces, refer to your *Getting Started, Deploying Services*.

Deploying with an Unmanaged Connection

You can bypass the EJB and the J2EE application server by writing your own code to call the resource adapter and send the request to an enterprise server. In this case, the connection is unmanaged. Your code runs in a Java 2 Standard Edition (J2SE) environment with the libraries from the J2EE server, rather than under a J2EE application server.

To deploy an unmanaged connection, add the Micro Focus classes to your classpath. See the chapter EJBs and Resource Adapters in your *Enterprise Server Deployment Guide*.

The Client Program

Once the EJB is deployed, you can access it with a client program. A JSP client to invoke the MapDemo EJB is supplied.

The JSP takes a request from the end user and passes it to the EJB you generated. The EJB in turn passes the request to the enterprise server, ESDEMO. The original COBOL program runs on the enterprise server and processes the request, returning the results back to the JSP.

These files make up the client program and are provided in
\\NetXClass\\MapDemo\\Client\\Java\\src\\com\\package\\JMapServ.

This file...	Contains the...
MapDemoServlet.java	Servlet. It processes the incoming requests from the JSP, invokes an instance of the stateless EJB, and forwards the response back to the JSP.
MapservJspBean.java	Getter and setter methods that are used to move information back and forth between

	the servlet and JSP.
SessionMonitor.java	Helper class, which stores the EJB instances for the stateful session bean.
MapDemo.jsp	JSP. It passes data back and forth between the end-user and the servlet.

Follow these steps to access the EJB using this client program.

1. Make sure the Enterprise Server is running and JMapServ is deployed on ESDEMO.

Also make sure you have successfully deployed the relevant MapDemo.ear to the J2EE application server. This .ear file contains the JSP client (packaged in a Web archive file MapDemo.war). Make sure the application server is running.

2. Open a browser. For WebLogic, enter the URL, <http://localhost:7001/MapDemo/MapDemo.jsp>. The client window displays.

Map Demo

The Map Demo program has a tiny book database that you can browse. You can look up books by title, author, or stock number, and you can add and delete books. To start, try selecting the **Read** operation and entering 1111 in the **stockno** field, and then click **Go**.

Operation: **Read**

Lnk_b_title:	<input type="text"/>	Lnk_b_stockno:	<input type="text"/>
Lnk_b_author:	<input type="text"/>	Lnk_b_retail:	<input type="text"/>
Lnk_b_type:	<input type="text"/>	Lnk_b_sold:	<input type="text"/>
Lnk_b_onhand:	<input type="text"/>		

Results:

Lnk_b_title:	Lnk_b_stockno:
Lnk_b_author:	Lnk_b_retail:
Lnk_b_type:	Lnk_b_sold:
Lnk_b_onhand:	
Lnk_file_status:	

SessionCreationTime:
SessionLastAccessTime:
SessionInactiveInterval:

3. Test out the 4 operations beginning with the Read operation, testing stockno 1111.

Summary

The simplest way to send requests to a COBOL program from a Java application is to use Net Express and create a service with an EJB interface to access the COBOL program.

The COBOL part of the EJB deploys to an enterprise server.

The Java part of the EJB and a resource adapter deploy to a J2EE application server.

Module 18

Using XML with COBOL

Overview

This module provides an introduction to Using XML with COBOL. It introduces XML syntax, Net Express facilities for using XML with COBOL, and provides examples of these facilities.

Objectives

At the end of this module you will be able to:

- Understand the purpose of XML.
- Compare and contrast XML and HTML.
- Describe and recognize basic XML syntax.
- Discuss how structure is imposed on XML documents.
- Describe an XML instance document.
- Discuss the concept of well-formed and valid XML documents.
- Discuss the uses of the Net Express tool, CBL2XML.
- Discuss the COBOL syntax extensions to support XML.
- Discuss Net Express support of XML PARSE and XML GENERATE.

About XML

XML provides a way for any application or system that reads text to access your data. XML documents are text documents containing data surrounded by tags. The user-defined tags describe the data and the application using the data determines what to do with it. XML documents are parsed or processed by an XML Processor. The parser checks the code for errors.

XML and HTML

XML is similar to HTML. However, HTML provides presentation of data while XML only describes and provides structure. XML does nothing but describe the data. This table compares HTML and XML.

In HTML...	While in XML...
Data presentation is the end result.	Data description and structure are the end result.
Tags and structure are predefined in HTML standard.	Tags and structure are user-defined to more accurately describe the data.
Tags are not case sensitive.	Tags are case sensitive.
Some elements do not have a closing tag.	All elements must have a starting and closing tag.
Some elements can be improperly nested.	All elements must be properly nested.
White space is stripped out.	All white space is preserved.

XML Syntax

XML Tags and Elements

The author of an XML document creates tags to specifically describe the content. The tags are case-sensitive. The starting and ending tag must match exactly and there must be an ending tag. A starting and ending tag

surrounding content is referred to as an *element*. In the example below, the starting tag <date> is followed by the content or data 03/15/03 and then the closing tag </date>. This forms a date element.

```
<date>03/15/03</date>
```

Example

Below is a sample XML document for a reminder notice to be sent from a dental office. This reminder could be sent by an email application, printed on note cards, or delivered by a phone reminder application. Each application can access the same XML content.

```
<?xml version="1.0"?>
<reminder date="03/03/03">
    <from>The Shiny Tooth</from>
    <to>Paula</to>
    <phone>999-555-1212</phone>
    <subject>Reminder</subject>
    <body>Time to schedule your visit to the dentist.
        Please call our office at your earliest
        convenience.
    </body>
</reminder>
```

XML Declaration

The *XML declaration* (although not required) is the first statement in the XML document. It indicates this is an XML document and the specification level. It must be lower case. This is the XML declaration from the example above.

```
<?xml version="1.0"?>
```

There are optional items that the declaration may contain. These include statements for external structure files and character set for the document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

In this example, the encoding attribute specifies a character set that is compatible with Windows ANSI. Native English words uses only characters with an ASCII code 64 = “A” to 122 = “z”, which do not need the encoding.

Languages requiring accents or umlauts (French accents like éé, Spanish Ñ, German äöüÄÖÜß, will require the encoding.

Root Element

Each XML document must have a *root element*. The root element is the first tag in the document and indicates the purpose of the content. The example above is a reminder from a dental office and the root element `<reminder date="03/03/03">` clearly indicates the purpose of the data. The root element also has an ending tag `</reminder>`. All other elements must be nested within the root element.

XML Attributes

An element tag may have *attributes* that further define the element. Attributes have a name and a value. If attributes are used, they must be specified in the element's start tag. Values for the attributes must be enclosed in quotes (double or single). If the value contains quotes, then use the other type of quote to delimit the value.

The root element from the dental office example contains an attribute whose name is `date` and value is `03/03/03`.

```
<reminder date="03/03/03">
```

Multiple attributes may be specified.

```
<shirt style='golf' size='large' color='blue'>
</shirt>
```

Data represented by attributes could also be represented as elements. For example, the statement above may also be written as follows.

```
<shirt>
  <style>golf</style>
  <size>large</size>
  <color>blue</color>
</shirt>
```

Both representations are correct, choose the one to use based on your XML style and needs.

Comments in XML

Comments in an XML document are formatted like this.

```
<!-- My comments here. -->
```

Using Special Characters

Some characters are considered illegal in XML due to reserved uses or simply good practice not to be used in data. To represent these characters in data, XML provides *entity references*. Entity references always begin with an ampersand (&) and end with a semi-colon (;). Here are the predefined entity references in XML.

Use this entity reference...	To represent this character...
<	Less than (<)
>	Greater than (>)
&	Ampersand (&)
'	Apostrophe ('')
"	Quotation mark ("")

XML Document Structure

Nesting

All elements must be nested within the root element. These elements are *children* or *sub-elements* of the root element. Elements that have child elements nested under them are known as *parent* elements. Elements at the same level are known as *sibling* elements. Elements may be nested as deeply as required to define the data.

Well-formed Documents

XML documents must be *well-formed*. Although the XML standard indicates specifically what constitutes a well-formed document, simply stated, well-

formed documents follow the XML syntax rules and have no errors. In general, well-formed documents should have the following.

- An XML declaration
- Correct encoding attribute
- A root element containing all other elements
- Properly nested elements.

Valid XML Documents

A *valid* XML document is well-formed and also follows the rules of a *Document Type Definition* (DTD) file or *XML Schema*.

If a DTD is being used, the XML document can directly encode the DTD or contain a reference to a location or file that contains the DTD specification.

Alternatively, an XML Schema may be used to describe the structure of the data. An XML schema (usually an .xsd file) contains element definitions.

XML instance documents (.xml files) are built using the elements declared in a schema and also contain data. If schemas are being used, the XML document contains reference to the schema that the document conforms to.

The XML schema is a more modern, powerful and prevalent technology for dictating XML document structure than the older DTD technology. With this in mind, Net Express provides comprehensive support for XML schemas that enable conversion to, input from, and output to XML documents from a COBOL application. You can use Net Express tools to validate an XML schema or type-check an instance document against its declared schema to determine if it is well formed.

Using Namespaces

Since tag names in XML are user-defined, there is potential for conflict between documents with common tag names that have completely different meanings. For example, `<phone></phone>` might refer to a phone number in one document and a piece of physical hardware used for making a phone call in another.

In a complex document, use a *namespace prefix* to distinguish between otherwise identical tag names. A namespace prefix is declared in the element's start tag and can be used in that tag or any sub-tags.

This example uses a namespace, myaddr to indicate a phone number.

```
<myaddr:addr xmlns:myaddr="http://addr/namespace">
    <myaddr:phone>555 555 1234</myaddr:phone>
</myaddr:addr>
```

Alternatively, if a default namespace is used, prefixes are not required and the namespace applies to all children of the element containing the declaration. This example declares and uses a default namespace (equivalent to the example above).

```
<addr xmlns="http://addr/namespace">
    <phone>555 555 1234</phone>
</addr>
```

The namespace attribute must be a *Uniform Resource Identifier* (URI). A URI identifies an internet resource, the most common of which is a *Uniform Resource Locator* (URL) as seen in this example.

```
<customerdb xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="customer.xsd">
```

Net Express and XML

Net Express provides facilities to enable XML data exchange with existing and new COBOL applications. The facilities used to accomplish XML enablement are listed below.

- cbl2xml utility or CBL2XML Wizard
- XML syntax extensions
- XML preprocessor

XML-enabling a COBOL application

The following defines the process of XML-enabling a COBOL application.

1. Map COBOL data records to an XML schema.

2. Modify the COBOL code using XML syntax extensions.
3. Compile the code in Net Express, which invokes the XML preprocessor (requires \$set p(prexml) to recognize the ORGANIZATION IS XML statement).
4. Test and deploy the application.

CBL2XML Wizard and Utility

The CBL2XML Wizard reads a COBOL record and generates several files. The same functionality is available in the `cbl2xml` command-line utility. Use the utility or the wizard to do the following.

- Generate a schema based on existing COBOL records.
- Generate COBOL records based on an existing schema.
- Map existing COBOL records to an existing schema.
- Validate XML schemas and instance documents.

Check *Distributed Computing*, “Generating COBOL Structures and XML Schemas” for command-line utility syntax.

Generating an XML Schema and Mapping Records

Net Express can be used to generate an XML schema from an existing copyfile or COBOL program. Once the schema has been generated, an XML document can be created that conforms to the schema and contains data. The XML document can then be used as input to a COBOL application.

The process for using XML with Net Express is described using two sample programs - `customerdb.cbl` and `migrate.cbl`.

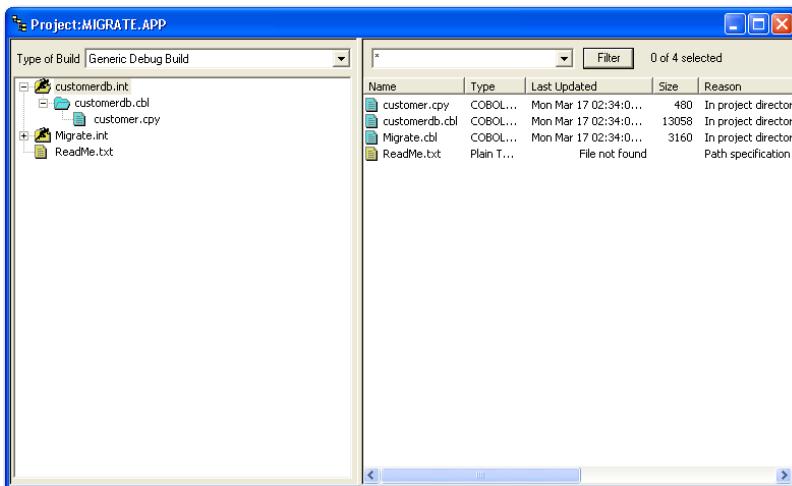
Customerdb.cbl - This is a simple legacy program that reads from, writes to, and deletes from a customer database indexed file, `customers.dat`. This example uses the copyfile, `customer.cpy`, that contains the file definition used to access the indexed database.

Migrate.cbl - `migrate.cbl`, takes the `customer.dat` file and migrates it to an XML instance document, `customers.xml`. This program:

- Demonstrates the process of XML-enabling a program by modifying an existing record.
- Provides a mechanism for generating an XML instance document from an indexed file.

To migrate the indexed file to XML, follow these steps to first convert the records contained in the copyfile to an XML equivalent.

1. Start **Net Express** and open the **MIGRATE project** from the **\NetXClass\Projects\XMLDEMONS\Migrate** folder.



2. Double-click **customer.cpy** and notice the contents. This copyfile contains the file definition used to access the indexed database.

```

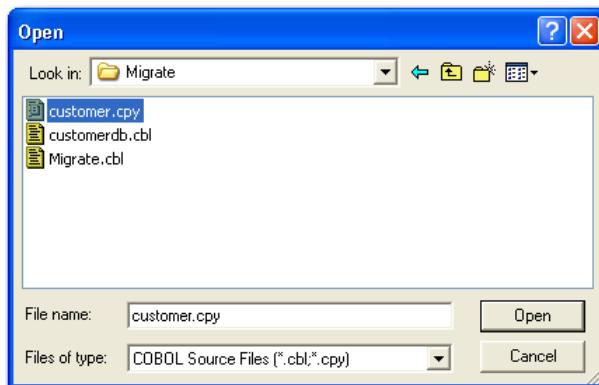
customer.cpy
01 customerdb.
  05 customer-record.
    10 customer-id      pic x(5).
    10 customer-name    pic x(50).
    10 customer-address.
      15 address-line-1 pic x(50).
      15 address-line-2 pic x(50).
      15 town-city       pic x(50).
      15 state-county    pic x(50).
      15 zip-postcode   pic x(10).
    10 phone-number     pic x(15).
  end-of-text

```

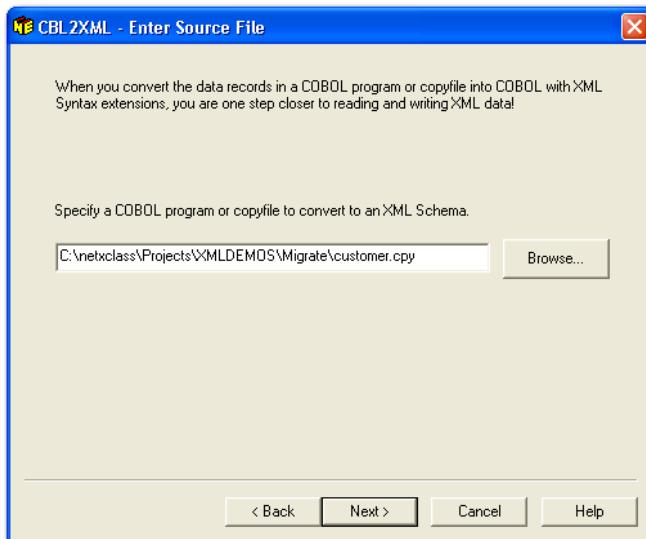
3. Select **Tools, CBL2XML Wizard**. The CBL2XML Wizard window displays.
4. Click **Convert COBOL to an XML Schema** and then click **Next>**.



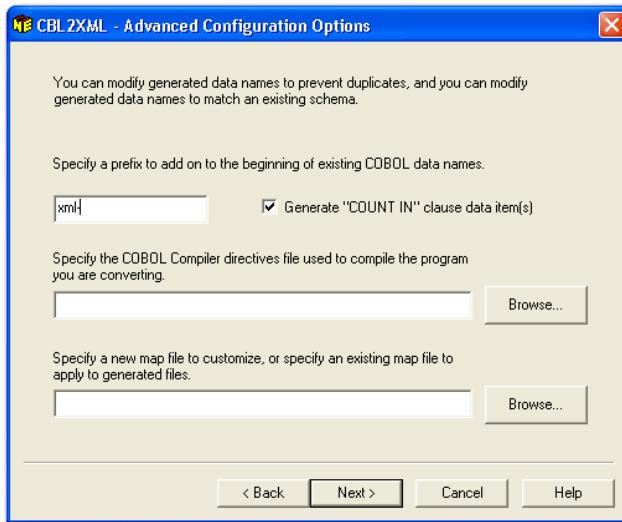
5. Browse and navigate to the **\NetXClass\Projects\XMLDEMONS\Migrate** folder. Click **customer.cpy** and **Open**.



6. This populates the Specify a COBOL program or copyfile to convert to an XML Schema field. Click **Next>**.

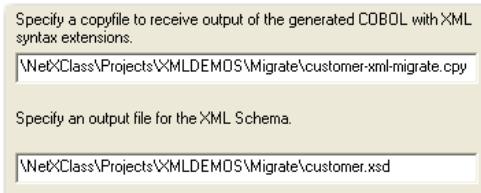


7. The Advanced Configuration Options window displays. Enter **xml-** as the prefix to add on to the beginning of existing COBOL data names field and click **Next>**.



The prefix is necessary to avoid data-name conflicts when this XML-enabled generated copyfile is integrated into migrate.cbl, which also includes the customer.cpy file.

8. The Enter Target File(s) window displays.

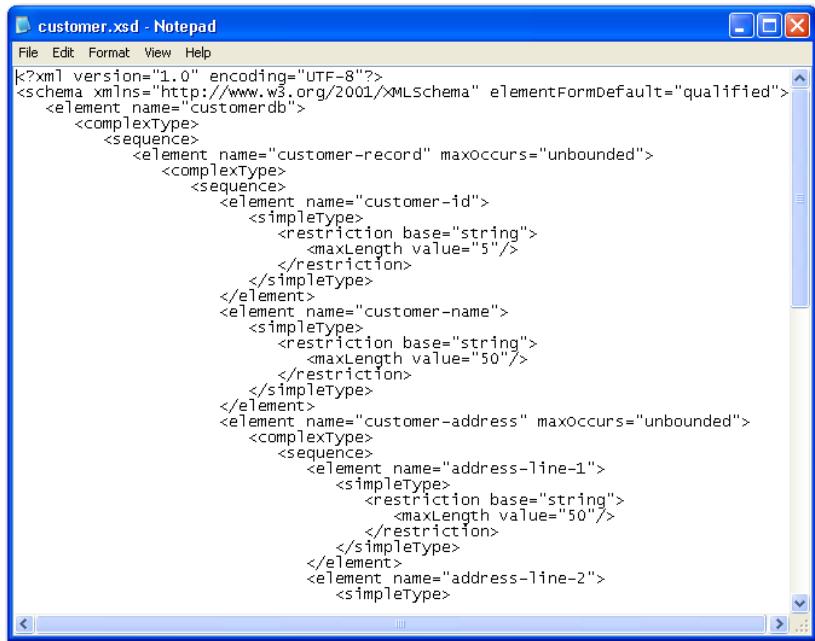


Enter **\NetXClass\Projects\XMLDEMONS\Migrate\customer-xml-migrate.cpy** as the copyfile to receive output of the generated COBOL **\NetXClass\Projects\XMLDEMONS\Migrate\customer.xsd** as the output file for the XML Schema. Click **Next>**.

9. The Finished window displays to indicate your final options. Click **Finish**. CBL2XML generates a new copyfile (customer-xml-migrate.cpy) and left the original intact.
10. Customer-xml-migrate.cpy is a dependent of the migrate program. add it to the project window, update the project dependencies by clicking **migrate.cbl** and select **Project, Update All Dependencies**.

```
01 xml-customerdb identified by "customerdb" count in
  xml-customerdb-count.
02 xml-customer-record identified by "customer-record" count in
  xml-customer-record-count.
03 xml-customer-id pic X(5) identified by "customer-id" count
  in xml-customer-id-count.
03 xml-customer-name pic X(50) identified by "customer-name"
  count in xml-customer-name-count.
03 xml-customer-address identified by "customer-address" count
  in xml-customer-address-count.
04 xml-address-line-1 pic X(50) identified by
  "address-line-1" count in xml-address-line-1-count.
04 xml-address-line-2 pic X(50) identified by
  "address-line-2" count in xml-address-line-2-count.
04 xml-town-city pic X(50) identified by "town-city" count in
  xml-town-city-count.
04 xml-state-county pic X(50) identified by "state-county"
  count in xml-state-county-count.
04 xml-zip-postcode pic X(10) identified by "zip-postcode"
  count in xml-zip-postcode-count.
03 xml-phone-number pic X(15) identified by "phone-number"
  count in xml-phone-number-count.
-----end-of-text-----
```

11. All of the data names originated in customer.cpy. Each data name in customer-xml-migrate.cpy has the xml- prefix specified when converting the file. Also note the XML syntax extensions generated for each data name. The structure of the record is identical to the original.
12. An XML schema was also generated in the project directory. This schema dictates the structure of any XML instance document that interfaces with the XML-enabled COBOL record. Display the file, \NetXClass\XMLDEMONS\Migrate\customer.xsd, using Notepad.



The screenshot shows a Windows Notepad window with the title bar 'customer.xsd - Notepad'. The menu bar includes File, Edit, Format, View, and Help. The main content area contains the following XML schema code:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <element name="customerdb">
    <complexType>
      <sequence>
        <element name="customer-record" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="customer-id">
                <simpleType>
                  <restriction base="string">
                    <maxLength value="5"/>
                  </restriction>
                </simpleType>
              </element>
              <element name="customer-name">
                <simpleType>
                  <restriction base="string">
                    <maxLength value="50"/>
                  </restriction>
                </simpleType>
              </element>
            </sequence>
            <elements>
              <element name="customer-address" maxOccurs="unbounded">
                <complexType>
                  <sequence>
                    <element name="address-line-1">
                      <simpleType>
                        <restriction base="string">
                          <maxLength value="50"/>
                        </restriction>
                      </simpleType>
                    </element>
                    <element name="address-line-2">
                      <simpleType>
```

XML Syntax Extensions

The Net Express XML syntax extensions enable COBOL programs to perform input and output on an XML instance document rather than using a traditional file method such as an indexed file.

The syntax extensions support creation of relationships between COBOL data structures and element definitions in XML schemas. Using these syntax extensions, you may code references to XML structures directly into

COBOL definitions. These XML structures are used to input and output information in an XML format.

COBOL Verbs

The following COBOL verbs and Syntax Extensions enable use with XML. Code samples may be found in “XML Syntax Extensions”.

Use this COBOL XML verb...	To...
START	Reset the position of a key.
OPEN	Open the I/O stream containing XML data.
READ	Read an XML document and create an internal tree-based representation of that document.
WRITE	Write an XML document to an I/O stream or add an XML element to the in-memory representation of the XML document.
REWRITE	Modify an XML element in the in-memory representation of the XML document.
DELETE	Delete an XML element and all sub-elements from the in-memory representation of the XML document.
CLOSE	Close the XML stream.
Use the ...	To...
SELECT clause	Assign a file name to an XML stream.
FILE SECTION and XML Description Paragraph (XD)	Specify the record definitions to receive data in the XML stream.

IDENTIFIED BY and IS ATTRIBUTE Clauses	Map the record to an XML element (tag) or an XML attribute.
PROCESSING-INSTRUCTION Clause	Embed XML processing instructions directly within the data associated with a tag.
COUNT IN Clause	Enables the program to determine the number of occurrences just read, or to specify the number of element occurrences to be output.
NAMESPACE Clause	Set or identify the namespaces that are contained in an XML document.

Generating an XML Instance Document

The Migrate program uses some XML syntax extensions to allow COBOL to read and write XML instance documents. Migrate.cbl uses both copies of the record - the original record that describes the indexed file customers.dat, and the newly generated record that describes the XML instance document, customers.xml.

Follow these steps to generate an XML instance document.

1. Make sure Net Express and the migrate project are open. Open **migrate.cbl** and notice the modified SELECT statement and the XD statement.

```

select xml-customer-file assign to "customers.xml"
      organization is xml
      document-type is "customer.xsd"
      file status is file-status-spec.

*****
*file definition for the existing customer database
*****
fd customer-file.
copy "customer.cpy".

*****
*file definition for the new XML customer database. Note the
*usage of "x1" to denote an XML format.
*****
xd xml-customer-file.
copy "customer-xml-migrate.cpy".

```

2. Close **migrate.cbl**.

3. Run migrate.cbl to generate the XML instance document. In the Project window, click **migrate.cbl**. Click **Rebuild** . Click **Run** . Ensure that **DEBUG\migrate** is specified and click **OK**.
4. To view the XML instance generated, **customers.xml**, select **File, Open**. Scroll down and select **XML file (*.xml)**. Double-click **customers** from the list.

The screenshot shows a Windows application window with a title bar 'customers.xml'. The main area contains the following XML code:

```
<customer-record>
  <customer-id>00003</customer-id>
  <customer-name>Thomas Jefferson</customer-name>
  <customer-address>
    <address-line-1>Monticello</address-line-1>
    <address-line-2/>
    <town-city>Charlottesville</town-city>
    <state-county>VA</state-county>
    <zip-postcode>22902</zip-postcode>
  </customer-address>
  <phone-number>434-984-9800</phone-number>
</customer-record>
<customer-record>
  <customer-id>00004</customer-id>
  <customer-name>James Madison</customer-name>
  <customer-address>
    <address-line-1>Montpelier</address-line-1>
    <address-line-2/>
    <town-city>Orange</town-city>
    <state-county>VA</state-county>
    <zip-postcode>22960</zip-postcode>
  </customer-address>
  <phone-number>540-672-2728</phone-number>
</customer-record>
```

Using XML PARSE and XML GENERATE

Net Express fully supports the IBM implementations of XML - XML PARSE and XML GENERATE.

The XML PARSE statement provides a mechanism for processing XML. The statement causes XML document data to be split into component fragments by the system's XML parser, and to be made available to a user-defined routine. Each fragment is labeled with a tag or event to indicate the nature of the fragment.

The general format of the XML PARSE statement is shown below:

```
XML PARSE identifier-1
PROCESSING PROCEDURE [ IS ] procedure-name-1
  [ { THROUGH } , procedure-name-2 ]
  [ ON EXCEPTION imperative-statement-1 ]
  [ NOT ON EXCEPTION imperative-statement-2 ]
  [ END-XML ]
```

Communication between the parser and the user routine is handled through four special registers.

Use this register...	To...
XML-CODE	Determine the status of the XML parsing.
XML-EVENT	Receive the name of each XML event.
XML-TEXT	Receive XML document fragments from an alphanumeric document.
XML-NTEXT	Receive XML document fragments from a national document.

The XML GENERATE statement converts data to XML format. The general format of the XML GENERATE statement is shown below:

```
XML GENERATE identifier-1 FROM identifier-2
  [COUNT IN identifier-3]
  [ON EXCEPTION imperative-statement-1]
  [NOT ON EXCEPTION imperative-statement-2]
  [ END-XML ]
```

Details on using XML PARSE and XML GENERATE may be found in the *Language Reference*.

Module Summary

XML documents contain data with user-defined starting and ending tags.

Every XML document must have a root element.

XML instance documents may have structure imposed on them by a DTD or an XML schema. Net Express supports using the modern mechanism of XML schemas.

Well-formed XML documents follow all XML syntax rules including proper nesting of elements.

Net Express provides a tool, CBL2XML, to assist in preparing COBOL records for use with XML.

Net Express supports COBOL syntax extensions that enable the COBOL code to work with XML data.

Net Express supports the IBM implementation of XML, XML PARSE and XML GENERATE.

Module 19

Using Consolidated Tracing Facility

Overview

The Consolidated Tracing Facility (CTF) is a tracing infrastructure that enables quick and easy access to diagnostic information detailing the operation of Micro Focus software components. CTF is an alternate tool for production level debugging of applications. The CTF interface is consistent across all platforms and is fully configurable.

Objectives

At the end of this module you will be able to:

1. Manipulate the parameters of a configuration file
2. Enable tracing and define the tracing information you want to capture
3. Set the severity level of tracing messages
4. Control emitters used to output the tracing information
5. Examine relevant diagnostic information produced by CTF to help troubleshoot problems.

About CTF

CTF is a highly configurable application tracing engine; activated by setting the MFTRACE_CONFIG environment variable to point to a valid CTF configuration file. It generates trace files on a ‘per-process’ basis containing trace records output from individual applications, the run-time system and environment specific CTF-enabled components. CTF provides tracing of applications in development or in production under Application Server and Enterprise Server environments. Via the CTF callable library routines, users can CTF-enable their own applications to output trace events and data at critical points within their program code.

Benefits

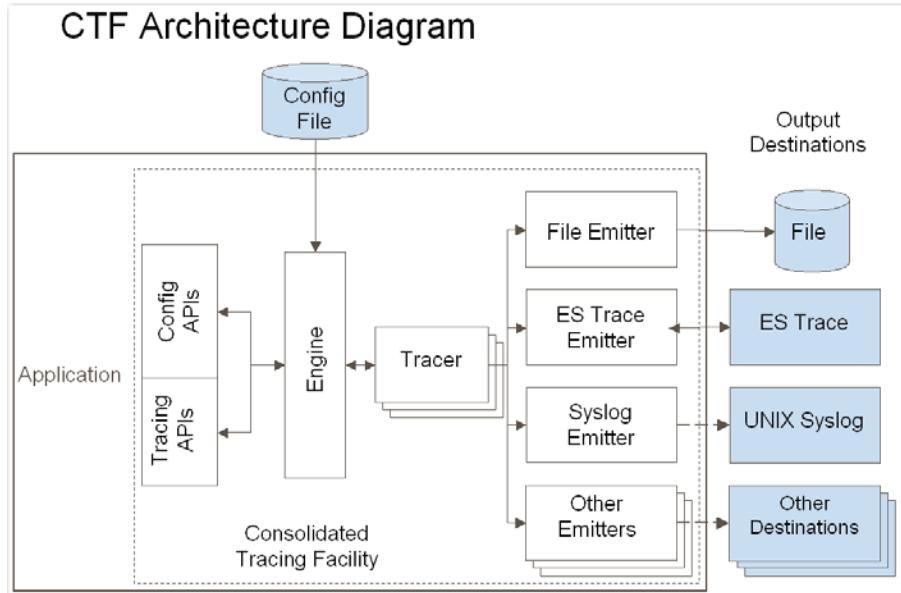
Micro Focus provides several other tracing facilities such as Object Oriented Messaging Tracing, ODBC & SQL Tracing and File Handler Tracing. CTF offers the following advantages to help debug applications and isolate issues:

- CTF Application tracing in a production environment is started by simply setting an environment variable
- CTF is always enabled and deployed in an application with no special requirements such as compiler directives or development IDE
- Reliable, fast, and highly configurable tracing capabilities
- Interleaved diagnostic information output from different components, processes and threads
- Diagnostic information generated by components is captured in a single place
- Multiple output formats/destinations are available
- Customers can enable their own application components allowing them to configure and generate trace output along with MF component trace information.

CTF Architecture

The CTF logging engine is controlled by parameters passed via a configuration file. The configuration file can be customized to control triggers and outputs. CTF components are listed and defined below.

- Tracers –Tracers are associated with a component. Properties can be set to control tracing levels, tracing areas, etc.
- Emitters – Output trace information can be directed to different destinations (e.g. text file, system event log, third-party system management tools, etc). Properties can be set to control emitter characteristics.
- Viewers –Tools used to view and analyse trace output.



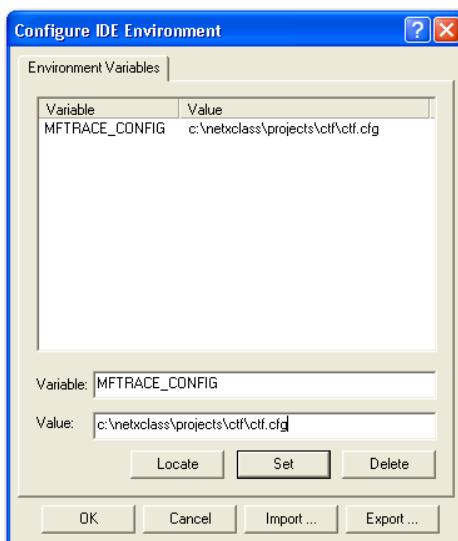
CTF Static Configuration Method

Setting the MFTRACE_CONFIG environment variable enables CTF tracing. This setting identifies the location of a configuration file (ctf.cfg) that contains specific tracing instructions. Since tracing options will vary by application, the configuration file is usually placed in the current project or execution directory.

Setting the MFTRACE_CONFIG Environment Variable

The MFTRACE_CONFIG environment variable can be set in several ways.

1. **Command prompt:** Enter the following command to set the MFTRACE_CONFIG environment variable to point at the ctf.cfg file in the current folder.
`MFTRACE_CONFIG=%cd%\ctf.cfg`
2. **Net Express IDE:** Specify the variable and parameters using **Project, Properties, Environment, IDE.**
3. Click **IDE** in the Environment section.
4. For the Variable, supply **MFTRACE_CONFIG** and supply the current project **path\ctf.cfg** for the Value. Click **Set**.



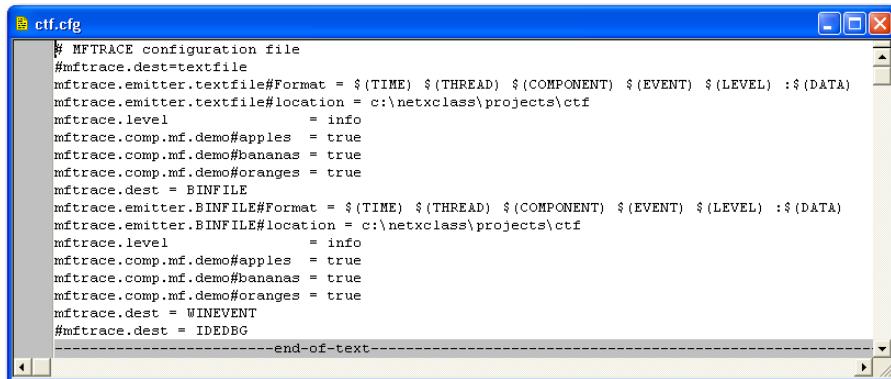
Other environment variables that can optionally be set are:

- **MFTRACE_LOGS** environment variable sets the default location where CTF log files should be generated.
- **MFTRACE_ANNOTATIONS** sets the default location of CTFViewer annotation XML files.

CTF Configuration File

The CTF configuration is typically named ctf.cfg and is enabled via the MFTRACE_CONFIG environment variable. The configuration file contains instructions for:

- Setting trace levels - (Default and per component)
- Setting component tracer properties
- Defining which emitter(s) are to be used
- Setting emitter properties
- Restricting tracing to a named application.

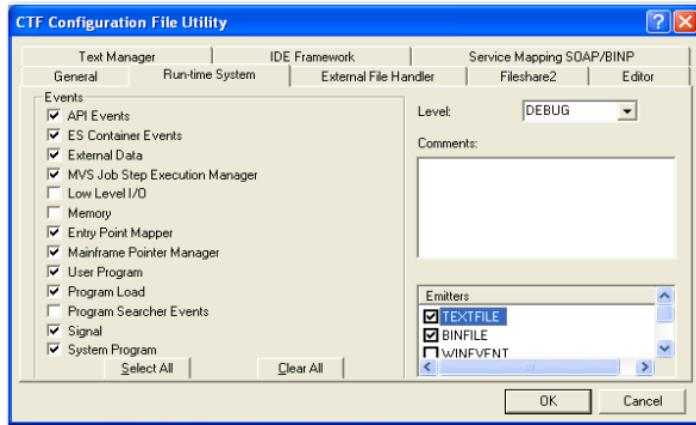


```
# MFTRACE configuration file
#mtrace.dest=textfile
mftrace.emitter.textfile#Format = $(TIME) $(THREAD) $(COMPONENT) $(EVENT) $(LEVEL) :$(DATA)
mftrace.emitter.textfile#location = c:\netxclass\projects\ctf
mftrace.level = info
mftrace.comp.mf.demo#apples = true
mftrace.comp.mf.demo#bananas = true
mftrace.comp.mf.demo#oranges = true
mftrace.dest = BINFILE
mftrace.emitter.BINFILE#Format = $(TIME) $(THREAD) $(COMPONENT) $(EVENT) $(LEVEL) :$(DATA)
mftrace.emitter.BINFILE#location = c:\netxclass\projects\ctf
mftrace.level = info
mftrace.comp.mf.demo#apples = true
mftrace.comp.mf.demo#bananas = true
mftrace.comp.mf.demo#oranges = true
mftrace.dest = WINEVENT
#mftrace.dest = IDEDBG
-----end-of-text-----
```

CTF Configuration File Generator

If available, CTFCFG.EXE is a tool that generates CTF configuration files from various user selected options. The tool options are read dynamically from existing XML files. Programmers can create template CTF configuration files supporting different tracing scenarios and existing CTF configuration

files can be loaded into CTFCFG.EXE for modification. The tool can run standalone or from Tools menu option of IDE



CTF Dynamic Configuration Method

In addition to producing tracing information for Micro Focus software components, Consolidated Tracing Facility includes a number of library routines that produce tracing information from applications. The routines that follow enable use of the Consolidated Tracing Facility infrastructure in applications.

Calls to the Consolidated Tracing Facility library routines yield return codes (success, failure, errors, etc.). Refer to the Consolidated Tracing Facility section of Net Express Help for a full description of the return codes. These return codes are provided in the MFCTF.cpy copybook for COBOL programs, and MFCTF.h include file for C programs.

CTF Library Routine	Description
CBL_CTF_COMP_PROPERTY_GET	Returns a named property value for a component.
CBL_CTF_COMP_PROPERTY_SET	Specifies a value for a component's property.

CBL_CTF_DEST	Specifies an output destination by associating (or disassociating) an emitter with (or from) a component or the default emitter list.
CBL_CTF_EMITTER_PROPERTY_GET	Returns a named property value for an emitter.
CBL_CTF_EMITTER_PROPERTY_SET	Sets a named property value for an emitter.
CBL_CTF_LEVEL	Specifies the trace level to be used for a specific component, or the default trace level if a component name is not specified.
CBL_CTF_TRACE	Outputs an event to one or more output destinations.
CBL_CTF_TRACER_GET	Returns a tracer handle that can be passed to all of the other tracing routines.
CBL_CTF_TRACER_LEVEL_GET	Returns a tracer's current threshold trace level.
CBL_CTF_TRACER_NOTIFY	Installs or uninstalls a tracer configuration callback function.

CTF Emitters

The Consolidated Tracing Facility can output its trace information using any of the emitters listed below.

- **TEXTFILE:** Writes tracing information to a standard text file using a default or a user-specified record format and filename.
- **BINFILE:** Writes tracing information to a binary file that can be viewed with the CTFViewer utility.
- **WINEVENT:** Writes tracing information to the Windows event log using a default or a user-specified record format.
- **IDEDBG:** Writes tracing information directly to a debugger such as Net Express or Visual Studio as you are stepping or running through the application.

CTF TEXTFILE Emitter

TEXTFILE writes tracing information to a standard text file using a default or a user-specified record format and filename. The file can be viewed with a text editor such as Notepad or added to the Project source pool pane and viewed with the Micro Focus Editor.

The TEXTFILE emitter is specified with the following statement in the Consolidated Tracing Facility configuration file:

```
mtrace.dest = TEXTFILE
```

```
# MFTRACE Text Emitter-
# CommandLine      = "C:\Program Files\Micro Focus\Net Express 5.0\Base\BIN\runw.exe DEBUG\ctfdemo"
# ProcessId        = 3588
# Date             = 2006/09/26
# Time             = 10:44:05.022
# Operating System = Windows XP Service Pack 2 (Build 2600)
# Computer Name    = PRECISIONXP
# Format           = ${TIME} ${THREAD} ${COMPONENT} ${EVENT} ${LEVEL} :${DATA}
10:44:05.022 3792 MF.RTS 128 1 :
10:44:05.353 3792 MF.RTS 129 1 :
10:44:09.639 3792 MF.DEMO 0 1 :
10:44:09.639 3792 MF.DEMO 2 1 :
10:44:09.639 3792 MF.DEMO 3 1 : "some lovely apples"
10:44:09.689 3792 MF.DEMO 4 1 :
10:44:09.689 3792 MF.DEMO 5 2 : "some ripe bananas" 12
10:44:09.689 3792 MF.DEMO 6 1 :
10:44:09.689 3792 MF.DEMO 7 3 : "some juicy oranges" 9 0x0012FC28
10:44:09.689 3792 MF.DEMO 1 1 :
10:44:09.689 3792 MF.RTS 254 1 : 0
10:44:09.689 3792 MF.RTS 130 1 :
10:44:09.779 3792 MF.RTS 131 1 :
```

The following table lists the properties you can set to control the behavior of the TEXTFILE emitter. Refer to the Consolidated Tracing Facility section of Net Express Help for a full description of the TEXTFILE emitter properties.

Property	Description
DELIMCHAR	The character used in the output file to delimit trace data parts for a trace event.
FILE	The name of the trace file to be written. The following pseudo-variables for the construction of the trace file name are supported: \$(APPNAME) - the basename of the executable invoked to start the current process \$(EMITTER) - always "TEXTFILE" for this emitter.

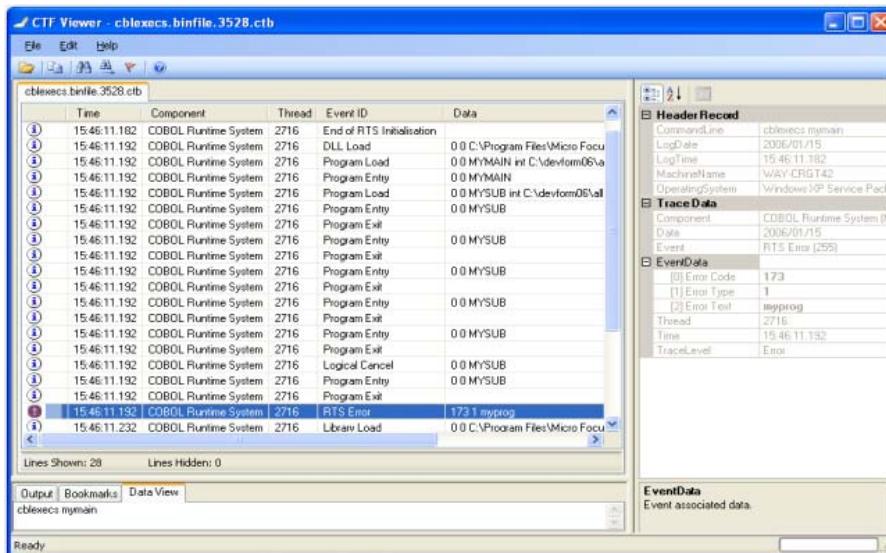
	<p>\$GEN) - the generation of the file, starting from 1.</p> <p>\$PID) - the operating system identifier for the current process</p>
FLUSHEVERY	The number of trace records that will be output before the file is flushed.
FORMAT	<p>The format to be used for each trace data record written to the file. The following pseudo-variables are supported in the format specification:</p> <p>\$(COMPONENT) - The name of the component outputting the trace event.</p> <p>\$(DATA) - The trace data specified by the component to be output for the trace event.</p> <p>\$(DATE) - The current date, output as yyyy/mm/dd.</p> <p>\$(EVENT) - The event identifier as specified by the component outputting the trace event.</p> <p>\$(LEVEL) - The tracing level: : 0 debug, 1 info, 2 warning, 3 error, 4 fatal</p> <p>\$(THREAD) - The current operating system thread identifier.</p> <p>\$(TIME) - The current time, output as hh:mm:ss.nnn.</p>
HEXBLOCKSIZE	The number of hexadecimal bytes to be output in each block when outputting binary trace data.
LOCATION	The folder where the trace files are to be written.
MAXFILESIZE	The maximum amount of data (in Kb) that will be written to the trace file before the file is closed and the next trace file in the generation sequence is opened..
MAXGENERATION	The maximum number of trace files that will be written to while tracing is enabled..
QUOTESTRING	Whether or not string trace data is to be output enclosed in double-quote characters.

CTF BINFILE Emitter

You can view the trace files output by the BINFILE emitter using the CTFViewer.exe utility that is supplied with Net Express V5.0. The format of the trace files is platform independent, so you can use CTFViewer on Net Express to view binary trace files that were created on Unix, for example.

The BINFILE emitter, is specified using the following statement in the Consolidated Tracing Facility configuration file:

```
mftrace.dest = BINFILE
```



The following table lists the properties you can set to control the behavior of the BINFILE emitter:

Property	Description
FILE	The name of the trace file to be written. The following pseudo-variables for the construction of the trace file name are supported: \$(APPNAME) - The basename of the executable invoked to start the current process.

	<p>\$EMITTER - Always "BINFILE" for this emitter.</p> <p>\$GEN - The generation of the file, starting from 1.</p> <p>\$PID - The operating system identifier for the current process</p>
FLUSHEVERY	The number of trace records that will be output before the file is flushed.
LOCATION	The folder where the trace files are to be written.
MAXFILESIZE	The maximum amount of data (in Kb) that will be written to the trace file before the file is closed and the next trace file in the generation sequence is opened..
MAXGENERATION	The maximum number of trace files that will be written to while tracing is enabled.

CTF WINEVENT Emitter

The WINEVENT emitter is provided to allow trace records to be output to the Windows event log. By default, to prevent the event log from being filled with unnecessary information, this emitter only outputs trace records with a trace level of error or above.

To enable the WINEVENT emitter, specify the following statement in the Consolidated Tracing Facility configuration file:

```
mftrace.dest = WINEVENT
```

The behavior of the WINEVENT emitter and the format of the output it creates are controlled by the following properties:

Property	Description
DELIMCHAR	The character used in the output file to delimit trace data parts for a trace event.
FORMAT	The format to be used for each trace data record written to the file. The following pseudo-variables may be used in the format specification:

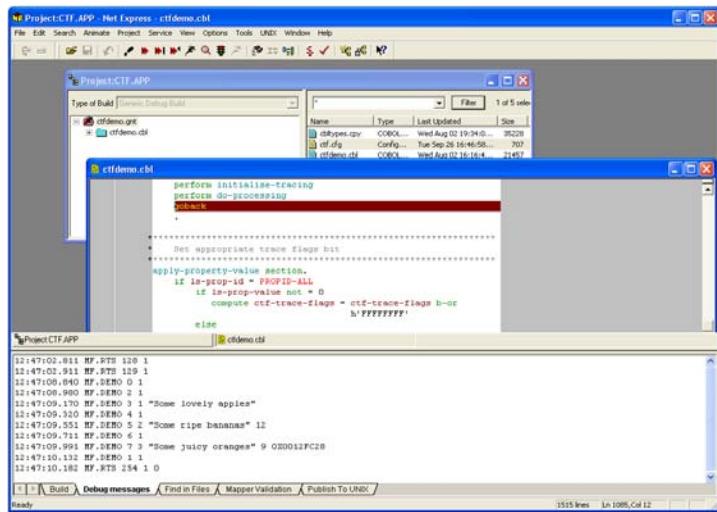
	<p>\$(COMPONENT) - The name of the component outputting the trace event.</p> <p>\$(DATA) - The trace data specified by the component to be output for the trace event</p> <p>\$(DATE) - The current date, output as yyyy/mm/dd.</p> <p>\$(EVENT) - The event identifier as specified by the component outputting the trace event.</p> <p>\$(LEVEL) - The tracing level: 0 debug, 1 info, 2 warning, 3 error, 4 fatal</p> <p>\$(THREAD) - The current operating system thread identifier.</p> <p>\$(TIME) - The current time, output as hh:mm:ss.nnn.</p>
HEXBLOCKSIZE	The number of hexadecimal bytes to be output in each block when outputting binary trace data.
LEVEL	The trace level below which this emitter will not output trace records.
QUOTESTRING	Whether or not string trace data is to be output enclosed in double-quote characters.

CTF IDEDBG Emitter

The IDEDBG emitter is provided to allow trace records to be output by a debugger, such as Net Express or Visual Studio, at the point at which they occur in the application being debugged. To enable the IDEDBG emitter, specify the following statement in the Consolidated Tracing Facility configuration file:

```
mftrace.dest = IDEDBG
```

To use this emitter with the Net Express IDE, you must set the MFTRACE_CONFIG environment variable for the IDE (as shown earlier). Trace events appear in the Debug messages tab when the application is debugged.



To use this emitter with a third-party debugger such as Visual Studio, you must set the MFTRACE_CONFIG environment variable in the environment before the debugger process is started. Trace events appear in the debugger's window where other system debug events would normally be displayed. For example, under Visual Studio 6, debug messages are displayed in the Debug tab.

The following table lists the properties you can set to control the behavior of the IDEDBG emitter:

Property	Description
DELIMCHAR	The character used in the output file to delimit trace data parts for a trace event
FORMAT	The format to be used for each trace data record written to the file. You can use the following pseudo-variables in the format specification: \$(COMPONENT) - The name of the component outputting the trace event. \$(DATA) - The trace data specified by the component to be output for the trace event.

	<p>\$(DATE) - The current date, output as yyyy/mm/dd.</p> <p>\$(EVENT) - The event identifier as specified by the component outputting the trace event.</p> <p>\$(LEVEL) - The tracing level: 0 debug, 1 info, 2 warning, 3 error, 4 fatal</p> <p>\$(THREAD) - The current operating system thread identifier.</p> <p>\$(TIME) - The current time, output as hh:mm:ss.nnn.</p>
LEVEL	The trace level which this emitter will not output trace records: 0 debug, 1 info, 2 warning, 3 error, 4 fatal, >4 no trace records are output.
HEXBLOCKSIZE	The number of hexadecimal bytes to be output in each block when outputting binary trace data
QUOTESTRING	Whether or not string trace data is to be output enclosed in double-quote characters.

CTF Enabled Product Components

The Consolidated Tracing Facility can produce tracing output for any of the components listed below. For details, refer to the components listed in the Consolidated Tracing Facility section of your Net Express Help Reference. A partial list follows:

Component	Description
MF.RTS: Runtime System Tracing	API calls, CBL_ functions, ES container events, memory management, loading/unloading of user/system programs and more
MF.IDMR: SOAP/BINP Request Handlers and Mapping	Client/Server request handler events, IDT generation/processing, run-time mapping between IDT and COBOL fields
MF.MFFH.XFH/FS: External File Handler and Fileshare2	Entry/exits from file handling I-O routines, byte stream functions, turn tracing on/off for individual files Fileshare2 including comms calls like send/receive in fhrsub

MF.ESODBCXA/ESMSSQL: XA Switch Modules	Database transaction processing using XA under Enterprise Server
MF.MVS.CASSPOOL: JCL	Job Execution Tracing
MF.MVS.MVSCATIO: JCL	System Catalog Tracing
MF.EDE.MFEDITOR	MF Editor Function Tracing
MF.EDE.TEXTMGR:	MF Text Manager Tracing
MF.IDE.IDELOAD	Core IDE Framework Event Tracing
MF.IDE.MVSPROJ	Project Handling Event Tracing
MF.IDE.VSXREST	Editing and Debugging Event Tracing

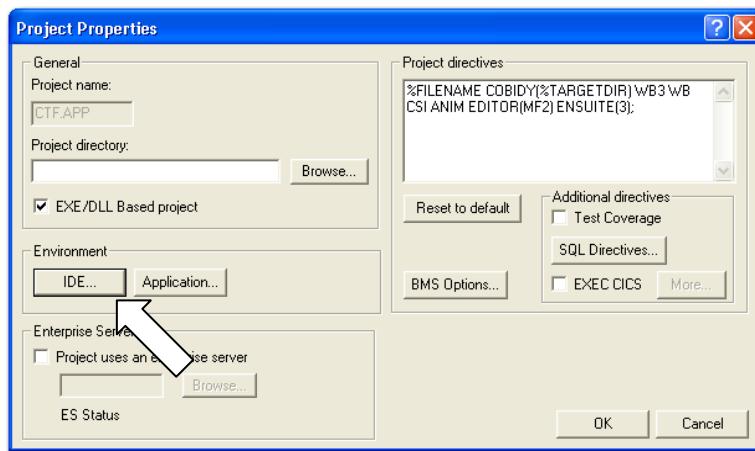
Module Summary

This overview of CTF tracing has shown you how to do the following:

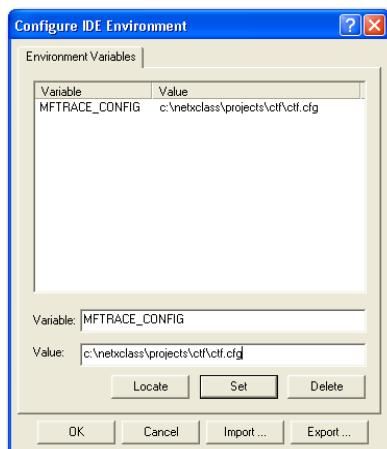
1. Manipulate the parameters of a configuration file
2. Enable tracing and define the tracing information you want to capture
3. Set the severity level of tracing messages
4. Control emitters used to output the tracing information
5. Examine relevant diagnostic information produced by CTF to help troubleshoot problems.

19.1 Using CTF - Workshop

1. Open the **CTF project** in the netxclass\projects\CTF folder. The configuration file can be found in the source pool view of the CTF project. Open **ctf.cfg** and review the code.
2. From the **Project menu**, select **Properties**.
3. Click **IDE** in the Environment section.



4. For the Variable, supply **MFTRACE_CONFIG** and supply the current project **path\ctf.cfg** for the Value. Click **Set**.



5. The CTF Project ctf.cfg file is set up to create a text file named **runw.textfile.9999.log** in your project directory. (Other output options are commented out with a # sign). **Rebuild** the project and **run** the program. **Inspect** the output file with Notepad.

```

runw.textfile.3588.log - Notepad
File Edit Format View Help
# -MFTRACE Text Emitter-
# CommandLine = ""C:\Program Files\Micro Focus\Net Express 5.0\Base\BIN\runw.exe DEBUG\ctfdemo"
# ProcessId = 3588
# Date = 2006/09/26
# Time = 10:44:05.022
# Operating System = Windows XP Service Pack 2 (Build 2600)
# Computer Name = PRECISIONXP
# Format = $(TIME) $(THREAD) $(COMPONENT) $(EVENT) $(LEVEL) :$(DATA)
10:44:09.639 3792 MF..RTS 129 1 :
10:44:09.639 3792 MF..RTS 129 1 :
10:44:09.639 3792 MF..DEMO 0 1 :
10:44:09.639 3792 MF..DEMO 2 1 :
10:44:09.639 3792 MF..DEMO 3 1 : "Some lovely apples"
10:44:09.689 3792 MF..DEMO 4 1 :
10:44:09.689 3792 MF..DEMO 5 2 : "Some ripe bananas" 12
10:44:09.689 3792 MF..DEMO 6 1 :
10:44:09.689 3792 MF..DEMO 7 3 : "some juicy oranges" 9 0x0012FC28
10:44:09.689 3792 MF..DEMO 8 1 :
10:44:09.689 3792 MF..RTS 254 1 : 0
10:44:09.689 3792 MF..RTS 130 1 :
10:44:09.779 3792 MF..RTS 131 1 :

```

6. **Edit** the ctf.cfg file.
7. **Put a # sign** in front of the live code and **Remove the # sign** from mftrace.dest = IDEDBG. **Save** and **Close**.
8. **Rebuild** the Project and **Animate** the application noting the log display in the **Build** window.

