



Microsoft® Dexterity
Programmer's Guide Volume 1
Release 12

Copyright

Copyright © 2012 Microsoft Corporation. All rights reserved.

Limitation of liability

This document is provided “as-is.” Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

Intellectual property

This document does not provide you with any legal rights to any intellectual property in any Microsoft product.

You may copy and use this document for your internal, reference purposes.

Trademarks

Microsoft, Dexterity, Excel, Microsoft Dynamics, Visual Basic, Visual SourceSafe and Windows are trademarks of the Microsoft group of companies. FairCom and c-tree Plus are trademarks of FairCom Corporation and are registered in the United States and other countries.

All other trademarks are property of their respective owners.

Warranty disclaimer

Microsoft Corporation disclaims any warranty regarding the sample code contained in this documentation, including the warranties of merchantability and fitness for a particular purpose.

License agreement

Use of this product is covered by a license agreement provided with the software product. If you have any questions, please call the Microsoft Dynamics GP Customer Assistance Department at 800-456-0025 (in the U.S. or Canada) or +1-701-281-6500.

Publication date

December 2012

Contents

- Introduction 2**
 - What’s in this manual 2
 - What’s new in this release of Dexterity 3
 - Finding information 4
 - Prerequisites 6
 - Symbols and conventions 7
 - Product support 8
 - What to do next 8
- Part 1: Overview 10**
 - Chapter 1: The Dexterity System 11**
 - What is a Dexterity application? 11
 - The application dictionary 12
 - Dexterity 12
 - The runtime engine 13
 - The DEX.DIC dictionary 13
 - Dexterity Utilities 14
 - Chapter 2: Tools and Features 15**
 - Tools 15
 - Features 17
 - Integration capability 20
 - Chapter 3: Application Development 21**
 - Application types 21
 - Basic development process 24
 - Interface design 24
 - Database development 27
 - Report generation 29
 - Scripting 30

Part 2: Basics.....32

Chapter 4: Getting Started with Dexterity 33

 Components..... 33

 Launching Dexterity..... 34

Chapter 5: The Dexterity Interface..... 37

 The main menu..... 37

 The toolbar..... 38

 The menu bar..... 38

 Windows..... 42

 Standard buttons..... 45

Part 3: Building an Application.....48

Chapter 6: Data Types 49

 Data type elements..... 50

 Static values 51

 Control types..... 54

 Using data types..... 71

 Procedure: Creating data types..... 73

Chapter 7: Formats..... 77

 Format elements..... 78

 Format string 79

 The Multiple Format Selector..... 80

 Procedure: Creating formats..... 84

Chapter 8: Global Fields 89

 Field elements..... 89

 Using global fields 91

 Array fields 93

 Procedure: Creating global fields..... 94

Chapter 9: Tables..... 97

 Table concepts..... 97

 Table elements 100

 Table fields 104

 Keys..... 105

Key options	106
Segment options.....	107
SQL Auto Procedure Disable Options	109
Table options.....	109
Temporary tables	110
Table locations	111
Table groups	111
Procedure: Creating tables.....	112
Chapter 10: Forms	117
Form overview	118
Form elements.....	119
Procedure: Creating forms.....	122
Chapter 11: Windows	125
Window elements	126
Creating a window layout.....	128
The Toolbox	129
The layout area.....	131
The layout grid.....	132
Adding fields to a window layout	132
The Properties window.....	136
Window properties.....	137
Field properties	138
Drawn object properties	142
Previewing a window	142
Tab sequence.....	143
Adding pictures	144
Linking fields to prompts	145
Positioning the window	145
Resizing windows.....	146
Setting required fields	149
System color support - automatic.....	149
System color support - customizable.....	152
Custom color support	153
Procedure: Creating windows	154

Chapter 12: Using Test Mode 157

Starting and stopping test mode..... 157

Test mode tools 158

Script lookup in test mode..... 159

Limitations 160

Part 4: Navigation 162

Chapter 13: Main Menu 163

The main window 163

The menu bar 164

Chapter 14: Form-based Menus 165

Menu types..... 165

Menu elements 166

Cascading menus 171

Default menu set 172

Using menus 173

Application-level menus and sanScript..... 174

Displaying form-level menus..... 176

Procedure: Creating form-based menus 177

Chapter 15: Commands 181

Command elements 181

Using commands..... 184

Controlling command access..... 185

Built-in commands..... 185

Procedure: Creating a command 188

Chapter 16: Command-based Menus 191

Implementing command-based menus 191

Sample menu script 191

Automatic menu clean-up 194

Chapter 17: Context Menus 195

Using context menus 195

Displaying a context menu 199

Context menus and macros 199

Chapter 18: Toolbars	201
Implementing toolbars.....	201
Sample toolbar script.....	203
Re-creating toolbar arrangement.....	203
Automatic toolbar clean-up	204
 Part 5: Additional Resources	 206
 Chapter 19: Messages	 207
Messages overview	207
Messages examples.....	208
Using product names	209
Procedure: Creating messages	210
 Chapter 20: Pictures and Native Pictures	 211
Pictures	211
Procedure: Adding a picture to the picture library	212
Procedure: Using a picture from the picture library.....	212
Native pictures	213
Procedure: Creating native pictures.....	214
Procedure: Synchronizing native pictures	215
 Chapter 21: Constants	 217
Global constants	217
Procedure: Creating global constants	218
Form-level constants	218
Procedure: Creating form-level constants	219
Predefined constants	219
Constant examples.....	220
 Chapter 22: Global Variables	 223
Global variables overview	223
Global variables example	223
Procedure: Creating global variables	224
 Chapter 23: Strings	 225
Strings overview	225
Procedure: Modifying a string	226

Chapter 24: Libraries 227

Library types 227

Procedure: Adding a library reference 228

Creating a resource library 229

Chapter 25: Icons 233

Icon elements 233

Procedure: Creating an icon resource 234

Chapter 26: Table Groups 237

Using table groups 237

Procedure: Defining a table group 237

Chapter 27: Virtual Tables 239

Virtual table concepts 239

Virtual table elements 240

Member table relationships 242

Virtual tables and SQL views 243

Using virtual tables 244

Procedure: Creating a virtual table 246

Part 6: Working with Dictionaries 252

Chapter 28: Resource Explorer 253

Resource Explorer window 253

Resource Explorer toolbar 256

Explorer menu 257

Chapter 29: Worksets 259

Creating worksets 259

Adding resources to a workset 260

Viewing worksets 261

Removing resources from a workset 261

Deleting worksets 262

Chapter 30: Search and Replace 263

Resource Find window 263

Procedure: Using Find 264

Procedure: Using Replace 265

Chapter 31: Importing and Exporting	267
Textual representation of resources.....	267
Importing	268
Exporting	269
 Part 7: Reports	 274
Chapter 32: Report Writer Overview	275
Reports and the Report Writer.....	275
Report types	275
The Dexterity Report Writer and the Runtime Report Writer.....	276
Planning a report	277
 Chapter 33: Table relationships	 279
Table relationship overview	279
Types of table relationships	279
Defining a table relationship	280
 Chapter 34: Report Definition	 285
Report elements	285
Report options	288
Printing a report definition	291
 Chapter 35: Report Layout	 293
Creating a report layout.....	293
Layout sections.....	294
The Toolbox	296
Adding fields to a report layout	299
The Properties window.....	300
Report properties	301
Field properties	302
Drawn object properties	303
Report field characteristics	304
Applying drawing options.....	306
Viewing your report	308

Chapter 36: Sorting 309
 Using a main table key 309
 Creating a sorting definition 310

Chapter 37: Restrictions 313
 Defining report restrictions 313
 Restriction functions 315

Chapter 38: Calculated Fields 323
 Creating a calculated field 324
 Operators 325
 Fields tab 327
 Constants tab 329
 Functions tab 330
 System-defined functions 330
 User-defined functions 336
 Evaluation order 338

Chapter 39: Additional Headers and Footers 341
 Overview of headers and footers 341
 The order of the headers and footers 342
 Creating additional headers or footers 343
 Group overview 345
 Sorting for groups 346
 Group headers 347
 Group footers 347
 Counting items in a group 348
 Counting groups 348
 Totaling and subtotaling 349

Chapter 40: Legends 351
 Creating legends 351
 Legends example 352

Chapter 41: Modifying Fields 353
 Field visibility 354
 Field formatting 355
 Changing display types 357

Display type summary	360
Chapter 42: Using Reports in Applications	361
Running reports	361
Exporting data to a file	362
Using temporary tables	364
Mail connectivity	364
Part 8: SQL Database Manager	368
Chapter 43: Data Sources	369
Installing Microsoft SQL Server	369
Overview of data sources	369
Configuring ODBC data sources	370
Chapter 44: Logins and Connections	375
Logins	375
Connections	377
Chapter 45: Cursors	381
Cursor overview	381
Cursor block size	382
Stale data	383
Cursor refreshing	383
Chapter 46: Locking	385
Table definition requirements	385
Session management	386
The DEX_LOCK table	387
Creating the DEX_LOCK and DEX_SESSION tables	388
Clearing stranded active locks	389
Chapter 47: SQL Tables	391
Creating SQL tables	391
Auto-generated stored procedures	391
Granting access to tables	393
Paths for SQL tables	395
SQL error handling	395
Encrypted fields for SQL Server	399

Accessing existing SQL data..... 402

Chapter 48: Stored Procedures407

 Using stored procedures 408

 Creating a prototype procedure..... 409

 Stored procedure parameters 410

 Pathname support for stored procedures..... 413

 Stored procedure time limits 414

Chapter 49: Pass-through SQL 415

 Pass-through SQL connections 415

 Executing SQL statements 416

 Working with results sets..... 416

 Specifying a database 417

Part 9: Source Code Control 420

Chapter 50: Introduction to Source Code Control421

 Benefits of Source Code Control 421

 Source code control terms 422

 Configurations..... 422

 Providers 423

Chapter 51: Setting up Source Code Control..... 425

 Connectivity 425

 Installing components 425

 Configuring the repository 428

 Configuring the Source Code Control Server 429

 Configuring Dexterity 431

Chapter 52: Source Files.....435

 Source file contents 436

 Source file structure 436

 Source file names..... 437

Chapter 53: Using Source Code Control..... 439

 Checking in a new dictionary..... 440

 Examining the repository..... 441

 Checking out source files 442

Making changes to resources	445
Creating new resources.....	445
Checking in source files	446
Locking and unlocking source files.....	448
Fetching source files	451
Updating a dictionary	453
Creating and updating the index file.....	455
Building a dictionary.....	456
Viewing version information	459
Source code control errors	460
Chapter 54: Maintaining the Repository.....	461
Deleting resources from a dictionary.....	461
Renaming resources	462
Labels.....	464
Updating source file states	464
Chapter 55: Source Code Control for Integrating Applications... 465	
Configuring Dexterity	465
Developing an application	465
Checking in a development dictionary	467
Moving to new versions of Microsoft Dynamics GP.....	467
Chapter 56: Generic Provider.....	469
Specifying the project location.....	469
Creating a project.....	470
Checking in a new dictionary	470
Checking out source files.....	471
Checking in source files	471
Part 10: Using the Runtime Engine	474
Chapter 57: Setting Product Information	475
Product information	475
Procedure: Setting product information	477

Chapter 58: Using Launch Files.....479
 Creating a launch file..... 479
 Contents of a launch file..... 480
 Using the launch file to start your application 481

Part 11: Packaging Applications.....484

Chapter 59: Making Installation Files.....485
 Installation scripts..... 485
 Building chunk dictionaries 486
 Creating an empty launch file..... 493
 Testing the installation process 493
 Additional components..... 495
 Runtime command line parameters 496

Chapter 60: Updating an Application.....499
 Changing table definitions..... 499
 Converting c-tree and Pervasive.SQL tables..... 500
 Converting SQL tables 502
 How to package an update..... 504
 Creating an update dictionary chunk 504
 Updating forms and reports dictionaries 508
 Updating forms dictionaries..... 509
 Updating reports dictionaries 510

Part 12: Utilities.....514

Chapter 61: Synchronize Utility515

Chapter 62: Duplicate utility517
 Procedure: Duplicating resources..... 517
 Additional resources duplicated..... 518

Chapter 63: Integration Reports.....519
 Procedure: Creating integration reports 519
 Creating a submission file..... 521

Index.....523

Introduction

Welcome to Microsoft® Dexterity, a powerful application development system you can use to develop software applications. Originally designed to create Microsoft Dynamics™ GP, Dexterity is ideal for creating transaction-based business applications, such as accounting and business management products. If you use Dexterity to create applications that integrate with other products like Microsoft Dynamics GP, Dexterity allows you to add to the power of these core applications with solutions fitted to the industries in which you have expertise.

Before you put Dexterity to work for you, take a few moments to review the information presented here. Understanding the organization can provide you with the proper approach to the Dexterity documentation.

What's in this manual

The Dexterity Programmer's Guide is designed to give you an in-depth understanding of how to use the Dexterity system. It is divided into two volumes. Volume 1 contains information about creating resources and working with dictionaries. Volume 2 contains information about adding scripts to an application. Even if you've used other development programs before, we recommend using the Programmer's Guide as a foundation for understanding application development using Dexterity. Volume 1 of the Programmer's Guide contains the following parts:

- [Part 1, Overview](#), provides a comprehensive look at the Dexterity system, the components that make up a Dexterity application and the basic development process used to create applications with Dexterity.
- [Part 2, Basics](#), shows you how to start using the Dexterity system and explains the components of the Dexterity interface.
- [Part 3, Building an Application](#), describes the basic resources used to create Dexterity applications. It also includes detailed procedures designed to help you complete fundamental tasks in Dexterity.
- [Part 5, Additional Resources](#), describes additional resources that you can use to enhance the features of your applications.
- [Part 6, Working with Dictionaries](#), explains how to use the Resource Explorer and describes methods to work with resources in dictionaries.

- [Part 7, Reports](#), introduces you to the Report Writer built into Dexterity, which allows you to create reports for your application.
- [Part 8, SQL Database Manager](#), explains how to use Microsoft SQL Server™ with Dexterity applications.
- [Part 9, Source Code Control](#), explains how to use a source code control system to manage development of a Dexterity application.
- [Part 10, Using the Runtime Engine](#), describes how to use a Dexterity application with the runtime engine.
- [Part 11, Packaging Applications](#), explains how to package your application for delivery to customers.
- [Part 12, Utilities](#), describes utilities that are included with Dexterity and used at various stages of application development.

What's new in this release of Dexterity

If this is the first time you've used Dexterity, you may want to skim this section and move to the section titled [Finding information](#) on page 4.

Refer to the following list for more information about major new features in this release of Dexterity and where you can look to find out more about each:

1. *Web client*

Dexterity has additional new runtime components that allow Dexterity-based applications like Microsoft Dynamics GP to be accessed as a Silverlight application in a web browser.

2. *Per-user DEX.INI file*

The Dexterity runtime now supports per-user DEX.INI files. This allows the application to divide the defaults file settings into those that apply to the entire application and those that apply to individual users.

3. *Support for Exchange Web Services*

Dexterity-based applications can send mail through Exchange Web Services (EWS).

4. *Scripting enhancements*

Enhancements include new statements and functions, as well as additions and enhancements to several function libraries.

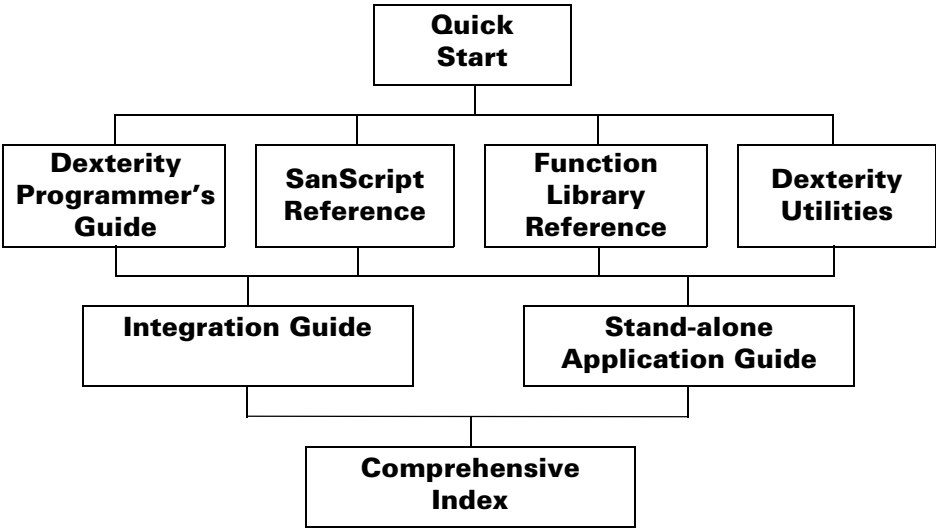
Refer to the "What's new?" topic in the Dexterity online help for a list of the other features added, and links to more information about each enhancement.

Finding information

We’ve provided several tools to help you understand the Dexterity development environment. These include sample applications, online help files, manuals designed to address specific needs, and a comprehensive index to help you find information anywhere in the Dexterity documentation. Your individual learning style and skill level will dictate how extensively you will use these tools.

Manual organization

The Dexterity documentation is divided into several manuals. An overview of the manuals is shown in the following illustration:



Start with the Quick Start manual. Once you begin developing applications, use the Dexterity Programmer’s Guide, SanScript Reference, Function Library Reference and Dexterity Utilities manuals. If you’re creating a stand-alone application, refer to the Stand-alone Application Guide; if you’re creating an application that integrates with Microsoft Dynamics GP, refer to the Integration Guide. A brief description of each manual follows.

Standard information

The following documentation components are used for standard development tasks with Dexterity.

- The **Dexterity Quick Start** manual contains overview information and lessons to introduce you to creating an application with Dexterity.
- The **Dexterity Programmer's Guide** is divided into two volumes. Volume 1 contains information about the Dexterity development system, a detailed explanation of the resources used to create applications, information about working with dictionaries, and instructions about packaging applications.

Volume 2 contains information about adding scripts to a Dexterity application, describes how to work with controls and tables, and describes advanced scripting tools.

- The **SanScript Reference** contains information about the Dexterity script language, sanScript.
- The **Function Library Reference** describes the various libraries of functions available to use in your applications.
- The **Dexterity Utilities manual** contains an overview of how to use Dexterity Utilities and instructions that can help you perform dictionary maintenance tasks.
- The **Comprehensive Index** contains the combined index of all the Dexterity manuals. Use this when you are searching for a specific topic, but don't know in which manual you should look.

Online help

Dexterity has a comprehensive online help system. It contains information about application development, the resources used to create applications, a complete reference to the sanScript language, and descriptions of all windows in Dexterity.

Integration information

Use the **Integration Guide** if you will be creating applications that integrate with Microsoft Dynamics GP. This manual describes how to make your application look and function like these products, how to use tools that are part of the platform for your application, and how to package your application.

Stand-alone information

Use the **Dexterity Stand-alone Application Guide** if you're creating applications that function on their own. This manual is a comprehensive set of instructions for creating a stand-alone application. Specific topics include developing pathname support, online help, security, and using tools such as the Report Writer or Modifier.



Prerequisites

Although the information in the Dexterity manuals will help you use the Dexterity system, an understanding of basic programming concepts and database design is also important.

- A cursory understanding of an application development environment such as Microsoft's Visual Basic[®] is helpful. Knowledge of third-generation programming languages, such as C or Pascal, is also helpful.
- If you're developing applications that operate over a network, a thorough understanding of your network environment is useful.
- If you're developing applications that will function in a client/server environment, a knowledge of client/server database architecture is recommended.

Symbols and conventions

To help you use the Dexterity documentation more effectively, we’ve used the following symbols and conventions within the text to make specific types of information stand out.

Symbol	Description
<code>↳ of table Items;</code>	A continuation character indicates that a script continued from one line to the next should be typed as a single line in the Script Editor.
	The light bulb symbol indicates helpful tips, shortcuts and suggestions.
	Warnings indicate situations you should be aware of when completing tasks with Dexterity.
<i>Margin notes summarize important information.</i>	Margin notes call attention to critical information and direct you to other areas of the documentation where a topic is explained.
New	The new symbol indicates additions and changes available with this release of Dexterity.
SQL	The SQL symbol indicates information that applies only when a SQL database type is used.

Convention	Description
Part 2, Basics	Bold type indicates a part name.
Chapter 10, “Forms”	Quotation marks indicate a chapter name.
<i>Applying formats</i>	Italicized type indicate a section name.
<code>'l_Item' = 1;</code>	This font is used to indicate script examples.
RUNTIME.EXE	Words in uppercase indicate a file name.
Software Development Kit (SDK)	Acronyms are spelled out the first time they’re used.
TAB or ALT+M	Small capital letters indicate a key or a key sequence.

Product support

Dexterity technical support can be accessed using the following methods.

- **Telephone support** – Technical Support can be reached at (888) 477-7877 between 8:00 a.m. and 5:00 p.m. Central Time, Monday through Friday. International users can contact Technical Support at (701) 281-0555.
- **Internet** – Technical Support is also available online through CustomerSource or PartnerSource, and is accessible from msdn.microsoft.com/dynamics/gp.

What to do next

To get the full benefit of the software, complete the following steps:

- 1. Install the Dexterity software.**

If you haven't already done so, install the Dexterity software.

- 2. Go through the lessons in the Quick Start manual.**

This will introduce you to basic concepts in Dexterity.

- 3. Read Part 1, Overview.**

This information provides a comprehensive look at the Dexterity development system and explains processes that can help you understand the basics involved in developing applications.

- 4. Read Part 2, Basics.**

This information provides information about using Dexterity for the first time and takes you through the Dexterity interface.

- 5. Continue reading the Programmer's Guide.**

The remaining chapters in the Programmer's Guide give you basic information about how to create an application with Dexterity.

- 6. Browse through the other manuals.**

Begin looking through the SanScript Reference and the other manuals included with Dexterity. This will help you become familiar with all of the pieces of the development environment.

Part 1: Overview

This portion of the manual describes the components that make up the Dexterity system. The information is divided into the following chapters:

- [Chapter 1, “The Dexterity System,”](#) describes what a Dexterity application is and introduces the basic components of the system.
- [Chapter 2, “Tools and Features,”](#) describes the tools and features that make Dexterity applications unique.
- [Chapter 3, “Application Development,”](#) describes the two types of applications you can create with Dexterity and provides an overview of developing an application using Dexterity.

Be sure to review this information before you begin using Dexterity. Understanding the basic functionality available with Dexterity can give you better insight into how you can develop your own applications. Once you’ve completed the overview, continue to [Part 2, Basics](#), to start becoming familiar with Dexterity.

Chapter 1: The Dexterity System

The unique architecture of a Dexterity application is the key to the many features Dexterity applications provide. Information about the Dexterity system is divided into the following sections:

- [What is a Dexterity application?](#)
- [The application dictionary](#)
- [Dexterity](#)
- [The runtime engine](#)
- [The DEX.DIC dictionary](#)
- [Dexterity Utilities](#)

What is a Dexterity application?

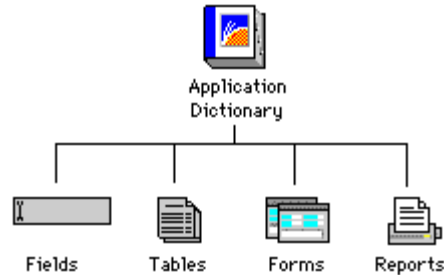
A Dexterity application is a combination of a few basic components that work together to present a functioning application.

- An *application dictionary* (a file with the extension .DIC) is the file that stores all of the resources and code for your application. When you use Dexterity to create new windows, tables and reports, you're adding objects, or *resources*, to the application dictionary.
- The *runtime engine* (RUNTIME.EXE) is an executable application that interprets the contents of the dictionary to present a functioning application to users.
- The dictionary *DEX.DIC*, which contains resources used by the runtime engine as it interprets an application dictionary.

Although additional files are required for a Dexterity application to operate properly, these are the basic components. The following sections explain each of these components and the role each plays in a Dexterity application.

The application dictionary

The characteristics of an application created in Dexterity are stored in a file called an *application dictionary*. The dictionary stores the *resources* that make up the application. Some of the types of resources an application dictionary can have include fields, tables, forms and reports.



You will use Dexterity to create application dictionaries. Refer to [Chapter 3, “Application Development,”](#) for more information.

For instance, the Real Estate Sales Manager sample application uses a single application dictionary named RESM.DIC. This dictionary stores the resources that make up the product.

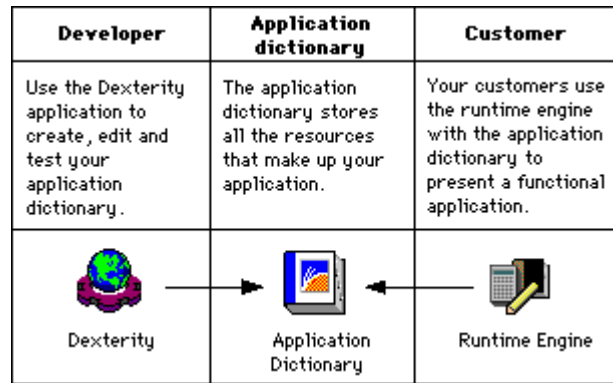
Storing resources in the dictionary, in contrast to compiling functionality into an executable application, allows the dictionary to be platform independent. In addition, the dictionary structure allows resources to be reused, which reduces coding required for an application and helps ensure consistency. For example, you can define a field resource once, and all tables, reports, and forms that use the field will refer to that specific resource. Resource relationships also allow quick global changes – a single global static text change will change the text value in all windows and reports where it is used.

Dexterity

You use the Dexterity application to build the application dictionary. It contains the tools you will use to create resources in the application dictionary. The Dexterity application also has the ability to run your application in test mode so you can determine whether the application is operating properly.

The runtime engine

Just as an everyday dictionary is used to define specific words so we can interpret meaning and usage when communicating to others, the application dictionary is interpreted by a platform-specific *runtime engine* to present an application to end-users. Since the runtime engine is platform-specific, the dictionary resources are interpreted appropriately for each platform – you won't be required to recompile or convert your application dictionary to provide cross-platform functionality. The following illustration shows the relationship among the Dexterity application, the application dictionary and the runtime engine:



When you complete the development process, you can rename the runtime engine (for instance, if your application is used to track customer leads, you can name the runtime engine LEADS.EXE), then provide the application dictionary and the runtime engine to your customers.



If you're developing applications for use with Microsoft Dynamics™ GP, RUNTIME.EXE is the same runtime engine used to run that product.

The DEX.DIC dictionary

The DEX.DIC dictionary is required by both Dexterity and the runtime engine. It contains resources used both by the Dexterity application to create application dictionaries, and by the runtime engine when it presents the application to users.

Dexterity Utilities

The Dexterity Utilities application contains several utilities and reports that you will use while you are developing your application. These utilities and reports are described in detail in the Dexterity Utilities manual. They are also described in other places in the Dexterity documentation where you will use them during the development process.

Chapter 2: Tools and Features

The primary goal of the Dexterity system is to allow you to design graphical business applications that have powerful tools and features, but allow you to concentrate on the application, not on the low-level implementation details. Information about tools and features is divided into the following sections:

- [Tools](#)
- [Features](#)
- [Integration capability](#)

Tools

The tools built into the Dexterity system can be utilized by any Dexterity application. They allow you to extend the capability of your application to provide features such as end-user customizations.

Modifier with VBA

The Modifier is an interface customization tool that's available to your application's users. This tool allows users to customize forms, windows and other resources in your application dictionary without changing the logic or functionality of the application. In this respect, the Modifier functions much like Dexterity, but without the ability to write or change scripts.

The Modifier also has embedded support for Visual Basic[®] for Applications (VBA). You can license VBA and include it with your application to provide increased customization capabilities.

Refer to [Chapter 15, "Using the Modifier,"](#) in the Stand-alone Application Guide for more information about using the Modifier and VBA in your application.

Report Writer

Report Writer is a report generation and customization tool available to your application's users. This tool is similar to the Dexterity Report Writer, allowing users access to the same tools you use to create your application's reports. Users can design reports based on original reports in your application, create copies of your original reports, change a report's layout, add new functionality and add fields from multiple tables without affecting your original report.

Your users can refer to the Modifier User's Guide for more information about using the Modifier's features.

Your users can refer to the Report Writer User's Guide for more information about using Report Writer's features.

Your users can refer to the Import online help for more information about using the Import Utility.

For more information about the Resource Descriptions tool, refer to the Resource Descriptions online help.

Refer to [Chapter 16, “Using the Report Writer,”](#) in the Stand-alone Application Guide for more information about using the Report Writer in your application.

Import Utility

The Import Utility is a tool that allows users to import data into your application. To do this, users create an import definition that maps data in a comma- or tab-delimited text file to the appropriate table in your application. The Import Utility uses an intuitive point-and-click method that allows data from a text file to be easily viewed and associated with fields from your application’s tables. Refer to [Chapter 14, “Using the Import Utility,”](#) in the Stand-alone Application Guide for more information about using the Import Utility in your application.

Resource Descriptions

The Resource Descriptions tool allows your users to view information about your application dictionary. This tool is designed to be used with a variety of products that enhance your application, including Dexterity, the Import Utility, the Report Writer and the Modifier. This information can also be a valuable tool for users to gain a better understanding about how data is stored in your application. The Resource Descriptions tool draws information about windows, tables and fields directly from your application dictionary, so users always view the most current technical information available for your application. Refer to [Chapter 17, “Using the Resource Descriptions,”](#) in the Stand-alone Application Guide for more information about using the Resource Descriptions in your application.

COM

Dexterity applications support Microsoft’s *Component Object Model (COM)*. COM allows external applications like Microsoft’s Visual Basic to integrate with Dexterity-based applications through Automation. Products in the Continuum series of products use this COM interface to integrate with Microsoft Dynamics GP.

Mail

Dexterity-based applications can send e-mail. For example, reports generated in the application can be mailed. Mail can be sent through Exchange Web Services (EWS) or through a MAPI-compatible mail client. MAPI is the Mail Application Program Interface.

Exchange Web Services uses a web service interface to communicate directly with a Microsoft Exchange or Office 365 server to send mail. No e-mail client is required. This is the preferred way to send mail.

When a 32-bit compatible MAPI mail client is installed, mail can be sent through the local mail client.. This is still supported for backward-compatibility.

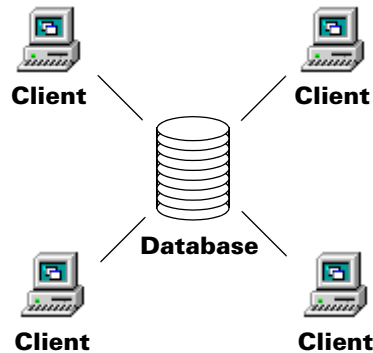
Features

Several features are built into the Dexterity system to allow you to easily make powerful, multiuser applications.

Network operation

Applications you create in Dexterity can be configured to operate across a variety of local area networks (LANs). In this environment, your application's data is located on the network, allowing multiple workstations to read and write data from the database. Each workstation in a LAN uses a platform-specific runtime engine and a copy of the application dictionary.

The following illustration shows the relationships between various workstations and an application's data.



Although a network server environment is an optimal configuration, data can also be shared between workstations using a peer-to-peer network environment, such as Windows networking.

Multiuser operation

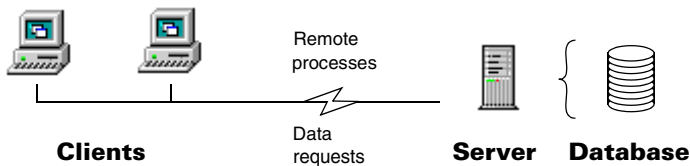
Dexterity supports multiple users accessing the same data at the same time. To accommodate this, Dexterity applications apply Optimistic Concurrency Control (OCC), a form of record locking that allows multiple users to work in the same tables and access the same records with minimal restrictions, while helping to ensure data integrity.

Database independence

Internally, Dexterity has a Data Management Subsystem that takes generic data requests and translates them into the specific calls needed to access data from the chosen database. You write your application independent of a specific database type. This allows your application to use any of the database types supported by Dexterity.

Client/server architecture

Dexterity applications can operate in a client/server environment, which allows you to separate data management tasks from processing and display tasks. By distributing these tasks between a client computer and a server computer, you optimize the use of your computing resources. The following illustration shows a client/server configuration.

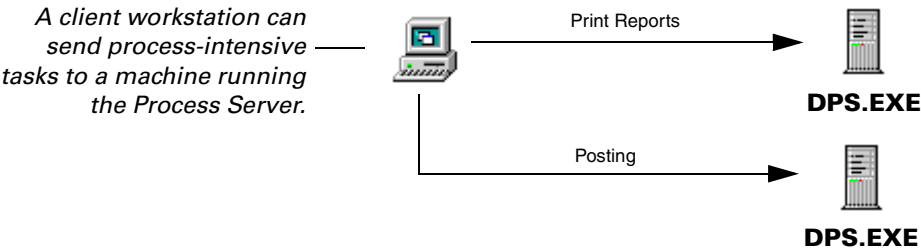


Refer to [Part 8, SQL Database Manager](#), for more information about SQL.

Data is located at the server, while the application (the runtime engine and dictionary) is located at the client. Client computers can then request data from or write data to the server computer. Dexterity supports Microsoft SQL Server 7 and SQL Server 2000.

Process Server

Processes are distributed between client and server workstations using the Process Server. With the Process Server, you can send certain processes – such as posting data or printing reports – to another computer that’s running the Process Server (either the server or another client):



Distributing your application's processes allows you to take advantage of faster processing capabilities that you may have on more powerful computers in your network. It also can reduce the work load and improve the performance of busier workstations by off-loading processes to computers that are idle.

Macro system

The Dexterity macro system allows your users to play and record macros, allowing your users to automate tasks throughout your application. The macro system is also used while testing your application. Refer to [Chapter 34, "Testing Your Application,"](#) in Volume 2 of the Dexterity Programmer's Guide for information about using the Macro system.

Security

You can implement security in any Dexterity application. Security allows you to control exactly which forms, reports and tables users of your application can access. Security is described in [Chapter 5, "Security,"](#) in the Stand-alone Application Guide.

Pathname support

Pathname support allows your application to store information in designated locations. Dexterity applications have built-in support for pathnames. Pathname support is described in [Chapter 2, "Pathnames,"](#) in the Stand-alone Application Guide.

Online help support

Dexterity applications can support Windows Help and HTML Help. For more information about online help, refer to [Chapter 7, "Windows Help,"](#) and [Chapter 8, "HTML Help,"](#) in the Stand-alone Application Guide.

Integration capability

*The process of using
Dexterity to create add-
on products is
described in the
Integration Guide.*

Integration with other products such as Microsoft Dynamics GP is available to Dexterity developers. These two products are created and maintained using Dexterity. The *multidictionary* capability of the runtime engine allows multiple third-party dictionaries to function with these two accounting products.

You use Dexterity to create your application directly in the Dynamics.dic dictionary. This allows you to use existing resources and create applications that integrate seamlessly with Microsoft Dynamics GP. You then use a special process to “extract” your application from the dictionary you developed it in. The runtime engine allows you to operate your application dictionary and the Dynamics.dic dictionary simultaneously.

If you use Dexterity to create applications that integrate with Microsoft Dynamics GP, you can easily add to the power of these core applications with solutions perfectly fitted to a particular industry.

Chapter 3: Application Development

There are two types of applications you can develop with Dexterity. The following sections describe these application types and introduce the process of creating an application with Dexterity. Information about application development is divided into the following sections:

- [Application types](#)
- [Basic development process](#)
- [Interface design](#)
- [Database development](#)
- [Report generation](#)
- [Scripting](#)

Application types

Depending upon the type of application you're developing, the development process you use will vary. Dexterity allows you to create two different types of applications:

- Stand-alone applications are created by Dexterity developers who want to develop applications that are independent of any other application.
- Integrating applications are created by Dexterity developers who want to integrate their business applications with Microsoft Dynamics GP.

Stand-alone applications

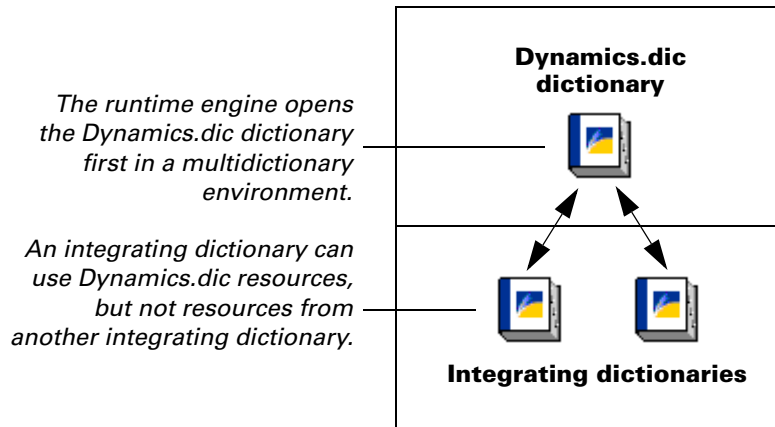
Although Dexterity was initially designed to create Microsoft Dynamics GP, you can use Dexterity to develop stand-alone applications. Many standard application features that are available to developers who create applications that integrate with Microsoft Dynamics GP, such as security, pathname support and online help, must be developed separately for a stand-alone application. These development processes and other information about creating stand-alone applications are described in the Stand-alone Application Guide.

For detailed information about creating stand-alone applications, refer to the Stand-alone Application Guide.

For detailed information about creating an integrating application, refer to the Integration Guide.

Integrating applications

The runtime engine included with Dexterity allows one or more dictionaries to function at the same time. This type of development environment is known as a multidictionary environment. If your application integrates with Microsoft Dynamics GP, this feature allows your application to share resources with the Microsoft Dynamics GP system and function simultaneously. If you're developing a stand-alone application, other dictionaries may operate simultaneously with your application, but not share resources.

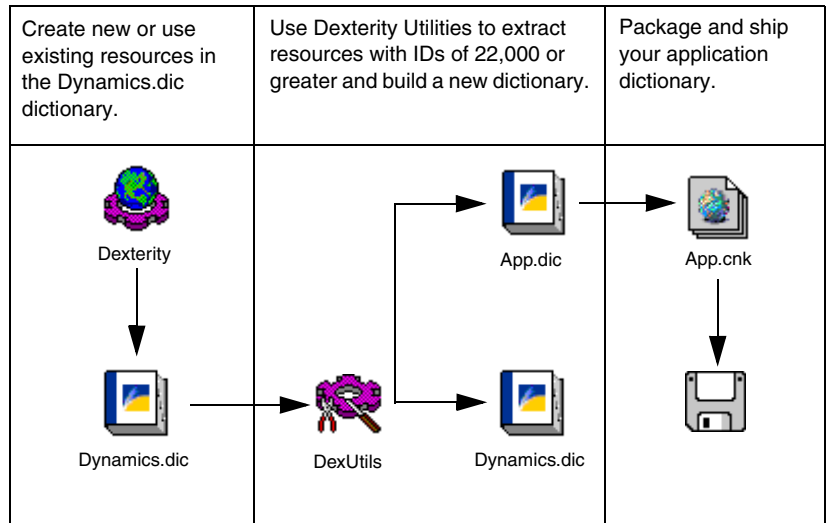


You can use many existing Dynamics.dic resources to add to the functionality of your own application. For instance, if you create a point-of-sale application, you can add fields from the Inventory Master Table directly to your application's sales entry window, allowing users to view, manipulate and save item information to the Microsoft Dynamics GP table as well as your own. Developing your application in this manner lets you take advantage of existing information in the Microsoft Dynamics GP database and create a seamless integration.

Integration also means that your application can benefit from system features available in Microsoft Dynamics GP. For instance, system security allows users to control access to forms and reports in either the integrating dictionary or the Dynamics.dic dictionary. Customization tools allow users to modify forms and reports in the integrating dictionary.

In order to integrate with Microsoft Dynamics GP, your application's resources must be developed directly in the Dynamics.dic dictionary. This allows you to use Dynamics.dic resources to help you create a seamless integration. To differentiate between your application and Microsoft Dynamics GP, the version of Dexterity you've received creates resources with resource IDs starting at 22,000. In contrast, all Microsoft Dynamics resources have resource IDs less than 20,000.

After you have finished developing your application, you will use Dexterity Utilities to "extract" your application from the Dynamics.dic dictionary. This extraction process doesn't remove your resources from the dictionary, but instead makes copies. It is the extracted dictionary that you will package and send to your customers. This process is shown in the following illustration.



Basic development process

This section outlines the development process for creating an application with Dexterity. You will use this same basic process when you develop your own application dictionary.

- In *interface design*, you define the look and feel of your application and determine how users navigate to other areas of your application using form and window resources.
- In *database development*, you determine how your application stores data using data type, field and table resources.
- In *report generation*, you design the layout and content of your application's reports using the Dexterity Report Writer.
- In *scripting*, you determine your application's logic and control using sanScript, the Dexterity programming language.

The following sections explain of each of these elements in more detail.

Interface design

Dexterity uses a combination of two different dictionary resources to establish your application interface.

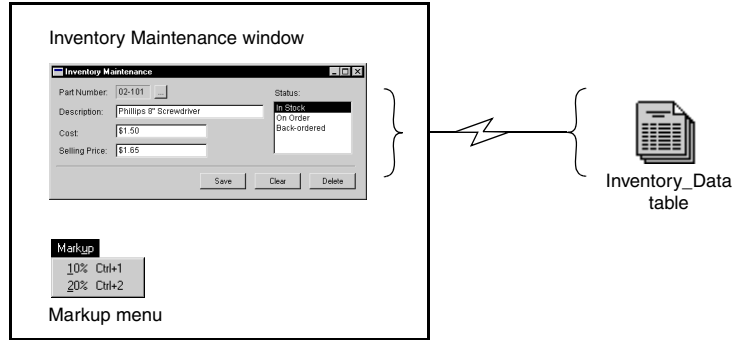
- *Forms* are used to organize windows, menus, tables and scripts into logical groups based upon the tasks being performed.
- *Windows* are used to display fields and controls, such as push buttons, list boxes, scrolling windows and check boxes.

Forms

It's important to remember that forms don't appear directly in your application the same way a window or a report does. Instead, they're used to organize other resources that function together to complete a specific task, such as allowing a user to enter, maintain and print item records. Although forms aren't a visual element of your application's interface, they provide the organization and structure that is the basis of how your application presents itself to the user.

The following illustration shows the components that make up the sample application's Inventory Maintenance form.

Inventory_Maintenance form

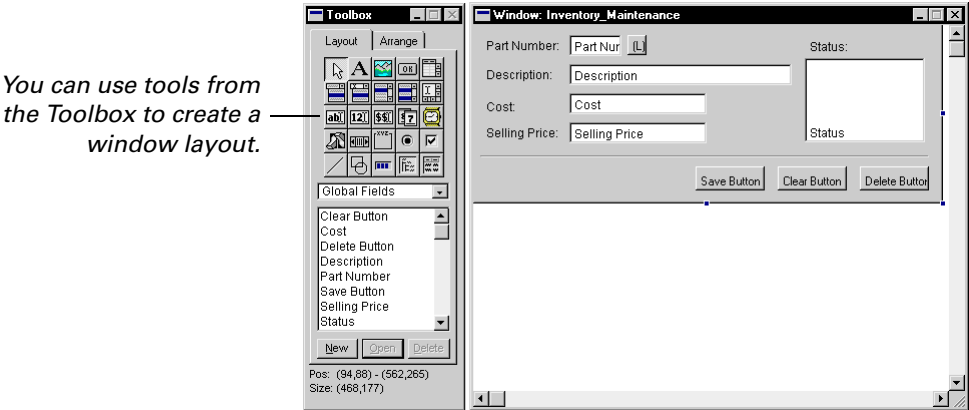


A form groups together one or more windows, tables, menus and scripts, all of which work together to perform a specific task in your application. In the previous example, the sample Inventory application uses the Inventory_Maintenance form to track new inventory items and update information about existing ones. This form has a single window that's used to enter and retrieve data, and references a single table that's used to store information entered in the form. A menu allows a specific option to be set for an item's price.

Windows

Windows allow users to interact with your application. They are the primary mechanism an application's user will use to enter and manipulate data.

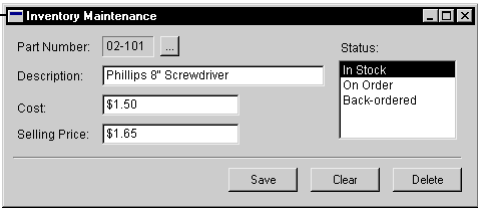
Dexterity uses a WYSIWYG (what you see is what you get) window layout tool that allows you to quickly and easily create windows. This drawing environment allows you to quickly combine data-entry fields and window controls with static items such as lines, rectangles, pictures and text.



The layout window allows you to add a variety of window controls, such as push buttons, list boxes, multi-select list boxes, radio buttons, drop-down lists, scrolling windows and visual switches. You can design windows quickly using an assortment of drawing tools similar to those offered in many graphics applications. Graphics, such as your company logo, can also be added to a window using standard cut-and-paste techniques.

When you design windows using Dexterity, you don't have to add native window controls, such as Windows resize controls. These controls are displayed automatically when Dexterity or the runtime engine is used to run your application.

Window controls are drawn automatically for the platform on which the application is being run.



If you are using the Themes service provided by the operating system, the controls in the window will be drawn to match the theme.

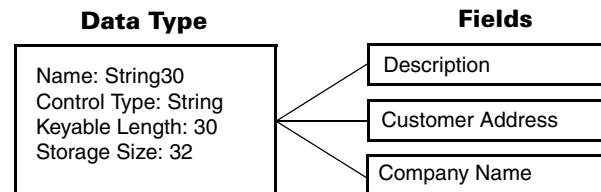
Database development

Dexterity uses a combination of dictionary resources to specify how data is stored in an application.

- *Data types* define how individual pieces of information are stored.
- *Fields* are the individual pieces of information in the application. Each field in the application uses a data type to specify its characteristics. Fields are used to build tables.
- *Tables* are collections of related fields.

Data types

Dexterity allows you to define a wide range of standard data types, including string, integer, currency, text and boolean data types. A data type resource establishes a set of data storage and display instructions for fields that use that data type. Several fields can use the same data type. For instance, all fields used to store 30 alphanumeric characters can use the String30 data type:

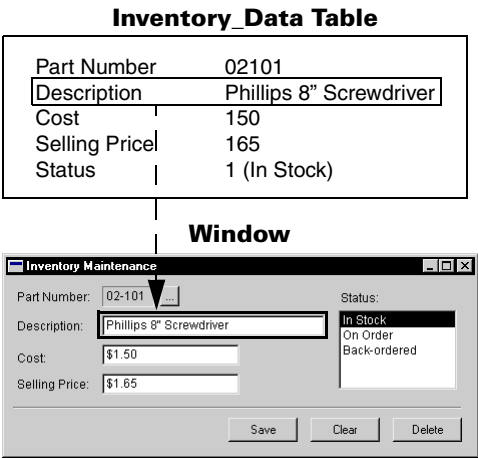


Fields

Fields represent individual pieces of information in an application. Each field in an application uses a data type to specify its characteristics. Many fields can use the same data type. Fields can be used on windows and stored in tables. In the previous illustration, the Description, Customer Address and Company Name fields use the String30 data type.

Tables

When you define a table resource for your application, you group related field resources. This grouping is called a record. For instance, a record could include fields for a unique part number, a description, an item cost, selling price and a status. Together, these fields are used to create the Inventory Data table. The following illustration shows the fields in the Inventory_Data and how they relate to the fields on the Inventory Maintenance window.

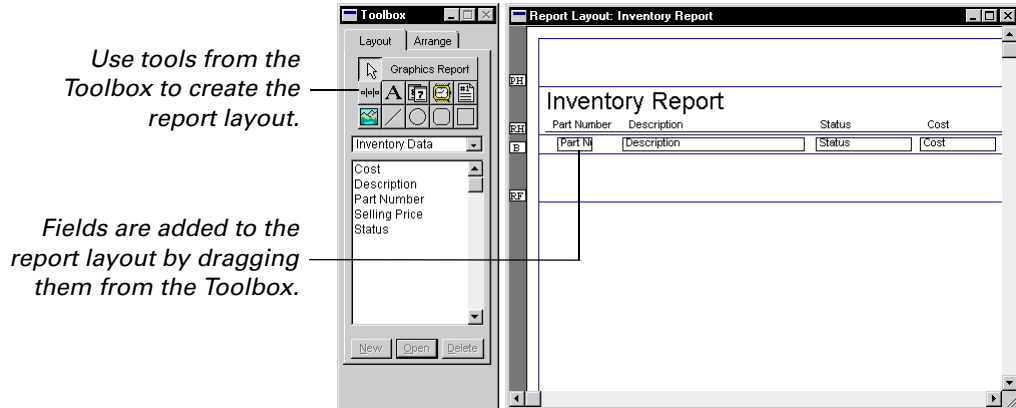


Currently, Dexterity supports three database managers that manage the tables in your application: Pervasive.SQL, c-tree Plus, and Microsoft SQL Server.

- The Pervasive.SQL database manager is used by applications that access data on a local workstation or on a network. It is also used by applications that access data on servers running Pervasive.SQL for Windows NT.
- The c-tree Plus database manager is used by applications that use c-tree Plus. It supports single-user operation and multiuser operation.
- The SQL database manager supports access to Microsoft SQL Server 7.0 and Microsoft SQL Server 2000.

Report generation

You will use the Dexterity Report Writer to create reports. The report layout allows you to design reports that present information from your application's tables. Reports can be printed to a screen, printer or data file. Reports are composed of several sections, each representing an area on the report, such as the report header, report footer, and the main body. The following illustration shows the layout of the sample Inventory Report.



Printing reports in your application is relatively easy. Simple sanScript commands allow you to print reports from objects such as menus or push buttons. For instance, the Inventory Report is printed using the **run report** sanScript statement, as shown in the following illustration.

```
run report 'Inventory Report';
```

↓

Inventory Report				
Part Number	Description	Status	Cost	Selling Price
01-100	16 oz. Hammer	In Stock	\$6.50	\$7.80
01-101	12 oz. Hammer	In Stock	\$6.00	\$7.20
02-100	Standard 8" Screwdriver	In Stock	\$1.50	\$1.80
02-101	Phillips 8" Screwdriver	In Stock	\$1.50	\$1.65
03-101	10 Grit Sandpaper	In Stock	\$5.25	\$6.30
02-102	Torx Screwdriver	On Order	\$4.75	\$5.70
05-100	1/2 HP Orbital Sander	On Order	\$45.00	\$49.50
03-100	20 Grit Sandpaper	Back-ordered	\$4.75	\$5.22

Scripting

You will use sanScript, Dexterity's scripting language, to add functionality and logic to your application. SanScript was developed specifically for Dexterity to provide functionality tailored to the requirements of graphical business applications. Unlike many traditional third-generation languages (such as Pascal or C) which require you to write long pieces of program code, sanScript is written in small segments, or scripts, that are attached to resources, such as fields, windows, forms and menus, in your application.

This script-based approach and sanScript's English-like instructions are designed to make Dexterity applications easier to write than applications written with other languages. If you've had experience writing programs in other languages such as Pascal or BASIC, you will find sanScript very easy to learn.

Dexterity has a built-in editor you can use to write scripts. It also has a source level debugger to aid in debugging scripts, as well as profiling tools to optimize scripts.

Part 2: Basics

Use this part of the documentation to understand how you can start using Dexterity. Following is a list of the items that are discussed, with a brief explanation of each:

- [Chapter 4, “Getting Started with Dexterity,”](#) shows you how to start a Dexterity session to create a new application dictionary or edit an existing one.
- [Chapter 5, “The Dexterity Interface,”](#) describes the elements that make up the Dexterity interface.

After you review this information, begin reading [Part 3, Building an Application](#). It describe the resources used to create Dexterity applications.

Chapter 4: Getting Started with Dexterity

You can begin using the Dexterity software immediately after you install it. This portion of the documentation lists the basic components of Dexterity and shows you how to launch it. If you haven't already done so, install Dexterity. Information is divided into the following sections:

- [Components](#)
- [Launching Dexterity](#)

Components

Refer to the COMPNT.PDF file included with Dexterity for a complete list of the components included with Dexterity.

This section describes the basic components required to use Dexterity. Depending on the installation options you selected when you installed Dexterity, some additional software, such as the Process Server, may have been installed as well. After you complete the Dexterity installation, the directory where you installed Dexterity will contain the following files:

File	Description
DEX.EXE	The Dexterity application. You can double-click this file to launch Dexterity.
DEX.DIC	A dictionary used by the Dexterity and Dexterity Utilities applications.
RUNTIME.EXE	The runtime engine used to run your application dictionary in a runtime environment.
DEXUTILS.EXE	The Dexterity Utilities application. Although this software isn't necessary for Dexterity to operate properly, it is necessary to perform dictionary maintenance.
DEXUTILS.DIC	A dictionary used by the Dexterity Utilities application.
DEX.CHM	The help file for Dexterity.
DEXUTILS.CHM	The help file for Dexterity Utilities.
IMPORT.CHM	The help file for the Import Utility.
RESDESC.CHM	The help file for the Resource Descriptions tool.

Several Dynamic Link Libraries (files with the extension .DLL) are installed with Dexterity. Refer to the COMPNT.PDF file included with Dexterity for a detailed list of DLLs.

Launching Dexterity

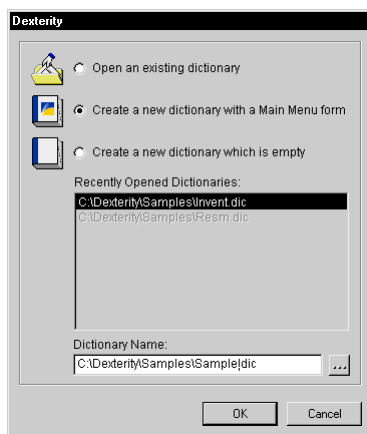
This section describes how to launch Dexterity and provides instructions for creating a new dictionary or editing an existing one. To ensure that Dexterity starts properly, be sure the Dexterity application is always in the same location as the Dexterity dictionary (DEX.DIC). Use the Dexterity program group in the Start menu to launch Dexterity.

Creating a new dictionary

Complete the following steps to create a new dictionary.

1. Launch Dexterity.

A dialog will appear, prompting you to open an existing dictionary or create a new one:



2. Specify the type of dictionary.

You can create a new dictionary that has a Main Menu form or one that has no resources. In most cases, you will want a dictionary that has a Main Menu form.

3. Specify the dictionary name and location.

In the Dictionary Name field, specify the name and complete path to the dictionary you want to create. You can type the value directly, or click the Dictionary Name lookup button to display a dialog that allows you to specify the name and location of the new dictionary.

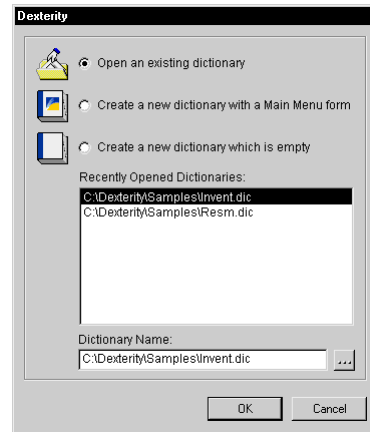
Click OK. The dictionary will be created in the location you specified.

Editing an existing dictionary

Complete the following steps to edit an existing dictionary.

1. Launch Dexterity.

A dialog will appear, prompting you to open an existing dictionary or create a new one:



Mark the option indicating you want to open an existing dictionary.

2. Specify the dictionary name and location.

In the Dictionary Name field, specify the name and complete path to the dictionary you want to open. You can type the value directly, or click the Dictionary Name lookup button to display a dialog that allows you to select a dictionary. You could also pick a dictionary from the Recently Opened Dictionaries list.

Click OK. The dictionary you selected will be opened.

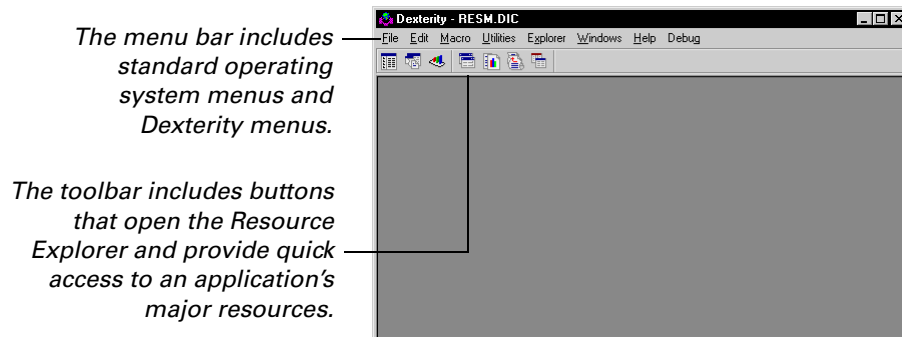
Chapter 5: The Dexterity Interface

This portion of the documentation describes the Dexterity-specific interface conventions you should be familiar with. Information about the Dexterity interface is divided into the following sections:

- [The main menu](#)
- [The toolbar](#)
- [The menu bar](#)
- [Windows](#)
- [Standard buttons](#)








The main menu

The main Dexterity workspace is called the *main menu*. When you launch Dexterity and either open an existing dictionary or create a new one, the main menu will appear. This window allows you access to all the components you need when creating an application dictionary. The following illustration shows how the main menu will appear when you start Dexterity:



The toolbar

The buttons on the Dexterity toolbar allow you to open the Resource Explorer, or provide quick access to primary resources in the dictionary. The following table lists the buttons in the toolbar and a short description of each:

Button	Description
	Opens the Resource Explorer window.
	Displays worksets in the Resource Explorer.
	Displays base resources in the Resource Explorer.
	Displays forms in the Resource Explorer.
	Displays reports in the Resource Explorer.
	Displays scripts (global procedures and global functions) in the Resource Explorer.
	Displays tables in the Resource Explorer.

Each of these resources are described in detail later in this manual.

The menu bar

This section describes the items available in the Dexterity menu bar.

File: New Dictionary

This menu item opens a dialog box that allows you to create a new dictionary. You can create a dictionary that contains a Main Menu form, or an empty dictionary that contains no resources. Empty dictionaries are useful when you're using source code control to manage application development.

File: Open Dictionary

This menu item opens a dialog box that allows you to open an existing application dictionary. You can also choose from the list of the dictionaries that were recently opened.

File: Login

This menu item opens a dialog box that allows you to log into a SQL data source.

File: Logout

This menu item logs out of the current SQL data source.

File: Print Setup

This menu item opens the printer setup dialog box. This dialog box allows you to configure the currently selected printer.

File: Process Monitor

This menu item opens the Process Monitor window. This window displays activity for tasks that you choose to process in the “background” within your application.

File: Resource Explorer

This menu item opens the Resource Explorer window. The Resource Explorer is the primary window for working with resources in a dictionary.

File: Table/Field/Window Descriptions

These menu items open forms in the Resource Descriptions tool. This tool displays information about all the tables, fields and windows used in the current dictionary.

File: Generate Resource Reports

This menu item prints a resource report for the application. The resource report is a text file that lists all resources in the current dictionary, their internal resource IDs and any resources associated with the listed resource. When you choose Generate Resource Reports, a dialog box will appear and allow you to name the report and select its location.

File: Generate Help Files

This menu item opens the Generate Help Files window. You will use this window to generate .RTF, .MAP and .HPJ files used when implementing Windows Help for your application.

File: Exit

This menu item allows you to exit the current session of Dexterity.

Edit: Undo

This menu item will undo the last keyboard entry in an editable field. It will also undo field movement and sizing in a layout window, but not the addition or removal of fields, text or graphics from a layout window.

Edit: Cut/Copy/Paste

These menu items allow you to copy text or graphics to the Clipboard, and then paste it in a different location. You cannot cut, copy or paste fields.

Edit: Clear

This menu item allows you to remove text from an editable field or remove selected items from the layout area.

Edit: Select All

This menu item allows you to select the entire entry in a field or all items in the layout area for a window, report or scrolling window.

Edit: Find

This menu item opens the Resource Find window, which allows you to search through the scripts, strings and messages in a dictionary.

Edit: Options

This menu item opens the Options window, which allows you to specify Dexterity preferences and characteristics of the current application dictionary.

Macro: (all)

The items available from the Macro menu allow you to record and play macros in tools mode, test mode or at runtime. For more information, search the online help for *macro menu*, or refer to [Chapter 34](#) in Volume 2 of the Dexterity Programmer's Guide.

Utilities: All

This menu lists various utilities that are available in Dexterity. These utilities are used at various stages of application development. Refer to [Part 12, Utilities](#), for more information.

Explorer: (all)

The items available from the Explorer menu allow you to work with items in the Resource Explorer window. Refer to [Chapter 28, "Resource Explorer,"](#) for more information about using the Resource Explorer.

Windows:

This menu displays a list of the windows currently open in Dexterity. Selecting a window from this menu will make the window active.

Help: Lookup

This menu item opens any lookup window for the current field.

Help: Contents

This menu item displays the contents topic for Dexterity online help.

Help: Search for Help On

This menu item displays the search window for the help system, allowing you to search the Dexterity online help.

Help: Window Help

This menu item displays help for the current window in Dexterity.

Help: How to Use Help

This menu item displays information describing how to use the help system.

Help: Online Manuals

This menu item lists the online manuals available for Dexterity. The manuals are in Adobe Portable Document (PDF) format.

Help: About Dexterity

This menu item displays the About Dexterity window.

Debug: (all)

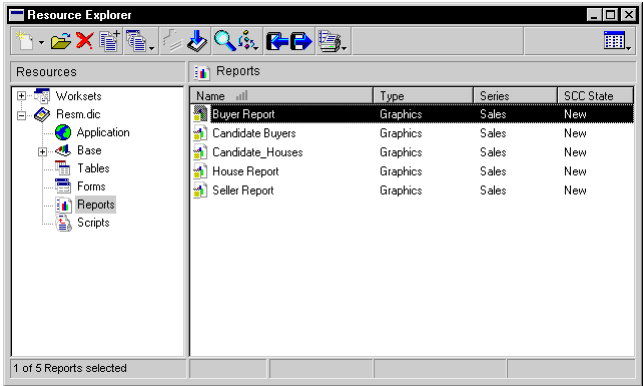
The first item the Debug Menu, Test Mode, allows you to switch Dexterity between tools mode and test mode. (Refer to [Chapter 12, “Using Test Mode,”](#) for more information.) The remaining items provide access to the Script Debugger and other analysis tools in Dexterity. These tools are described in [Part 27, Script Debugger](#), of Volume 2 of the Dexterity Programmer’s Guide.

Windows

Dexterity uses several types of windows. Review the following descriptions of windows and window controls for information about how each is used in Dexterity.

Resource Explorer

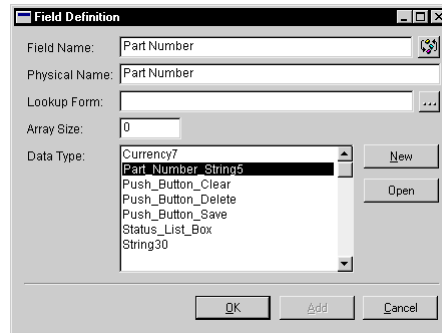
The Resource Explorer is the primary window used in Dexterity. It lists all of resources that make up a dictionary. The window is divided into two panes. The left pane contains a tree view that displays the various categories of resources. The right pane lists all of the resources in the selected category. The Resource Explorer is shown in the following illustration.



Refer to [Chapter 28, “Resource Explorer,”](#) for more information about using the Resource Explorer.

Resource definition windows

Resource definition windows allow you to create a new resource or edit an existing resource. Most Dexterity resources have a definition window. For example, if you select the Part Number field in the Fields resource list window and click Open, the Field Definition window will open and display the field's definition. You use the definition window to specify the characteristics of the field. The following illustration shows the field definition for the Part Number field.



Layout windows

Layout windows are the workspaces you'll use to design your application's windows, scrolling windows and reports. Layout windows allow you to place objects, such as fields, in the layout area. Although layout windows for scrolling windows and reports vary slightly in function, all layout windows allow you to perform similar layout tasks.

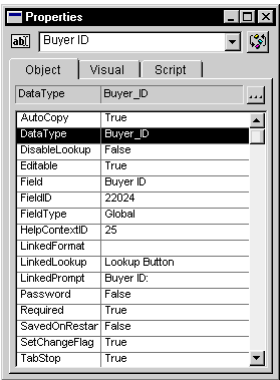
Toolbox

The Toolbox is available whenever a layout window is open. It contains tools you will use to add objects to the layout and manipulate objects in the layout. The toolbox is shown in the following illustration.










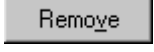

Properties window

The Properties window is used to specify characteristics of windows, fields, and drawn items when you're creating a window layout.



Standard buttons

The following buttons are used throughout Dexterity:

Button	Description
	Saves changes and closes the current window.
	Closes the current window without saving the changes to the window.
	Allows you to save information for the current window without closing the window.
	Creates a new resource.
	Opens the selected resource and displays its definition.
	Deletes the selected resource.
	Allows you to insert items in a list, such as fields into a table definition.
	Allows you to remove items from a list, such as the static text values in a list box.
	Opens another window, allowing you to select a value to return to the current field. Typically, clicking the lookup button displays a list of resources. In other cases, clicking the lookup button opens some other window, such as the Script Editor, that allows you to define a resource or specify additional characteristics.

Part 3: Building an Application

This portion of the documentation provides detailed information about the resources used to create Dexterity applications. The resources are described in the order they are typically created in during application development. Each chapter contains detailed information about the resource and a step-by-step procedure describing how to create or use the resource. Refer to the procedural information when creating your own applications. Following is a list of the topics that will be discussed, with a brief explanation of each:

- [Chapter 6, “Data Types,”](#) describes data types and how to create them for a Dexterity application.
- [Chapter 7, “Formats,”](#) introduces formats and how to use them.
- [Chapter 8, “Global Fields,”](#) explains how to create global fields for a Dexterity application and how they’re used.
- [Chapter 9, “Tables,”](#) explains how to create tables for an application.
- [Chapter 10, “Forms,”](#) explains what forms are and how to create and use them in an application.
- [Chapter 11, “Windows,”](#) describes how to create windows for an application.
- [Chapter 12, “Using Test Mode,”](#) introduces Test Mode, a special mode in Dexterity that allows you to operate your application.

Chapter 6: Data Types

The initial step in creating Dexterity applications is to create the data types your application will use. The type of information displayed or stored in a field is determined by its data type. You'll use data types to specify how data entry fields, such as string and currency fields, will store and display the information entered in them. In addition, you'll use data types to specify the characteristics of several interactive objects in your application, such as push buttons, check boxes, radio buttons and list boxes.

All fields require a data type to determine how the field will be used in your application. Without a data type, a field doesn't have the ability to store, control or display information. In this respect, most of a field's functionality is based upon the data type it uses, including the field's data storage capabilities for use in a table and information display and control characteristics when added to a window.

Information about data types is divided into the following sections:

- [*Data type elements*](#)
- [*Static values*](#)
- [*Control types*](#)
- [*Using data types*](#)
- [*Procedure: Creating data types*](#)

Data type elements

A data type can have a name, control type, keyable length, storage size, static values, a format and a composite definition. The Data Type Definition window, shown in the following illustration, is used to set the characteristics of a data type.

The data type name is entered here.

The control type is the main characteristic of the data type.

Name

Each data type in your application must have a name. Typically, the name indicates the characteristics of the data type, such as the control type used or the size of the data type. For instance, the name String30 indicates that the data type has a string control type that can display 30 characters.

Control type

The control type determines the function of the data type and how it will store and display data. The control type is the main characteristic of a data type. Dexterity has a variety of control types; a comprehensive list of them can be found in the next section.

Keyable length and storage size

Some control types allow you to specify the *keyable length* of a data type, which is the number of characters a user can enter in a field with that particular data type. Control types such as currency, integer and string have a keyable length.

The *storage size* for a data type is the number of bytes needed to store a field using the data type in a table. The storage size is typically calculated by Dexterity and can't be changed, except for string data types. The storage size for strings can be increased beyond the minimum amount calculated by Dexterity to allow room for expansion, in case you need to increase the keyable length of the data type.



Changing the storage size for a data type after you've entered data in an application may cause errors for existing records in a table. For instance, if you change the storage size for a data type used to store customer names, you won't be able to read existing customer records until data in the table is converted. Data conversion is described in [Chapter 60, "Updating an Application."](#)

Static values

Static values are any text or pictures associated with a data type that are displayed by a field using that data type. They are described in detail in [Static values](#) on page 51.

Format

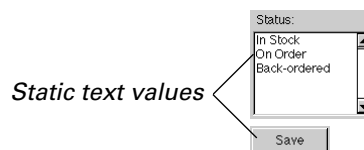
A *format* contains extra characters, spacing and attributes that can be applied to a data type when data is entered or displayed. Formats are described in the next chapter.

Composite

A *composite* is a group of fields and their associated data types that are combined to form a single data type. Composites are described in [Chapter 9](#).

Static values

Certain control types allow you to specify *static values* for the data type. Static values are any text or pictures associated with a data type that are displayed by a field using that data type. For example, the text on a push button and the items in a list box are static values.



To set static values for a data type, first use the Static Values drop-down list to set the type of static value.



Use the Static Values drop-down list to specify the type of static value to use with the data type.

You can choose one of the following:

- Text
- Picture
- Native Picture
- Text - Picture
- Text - Native Picture

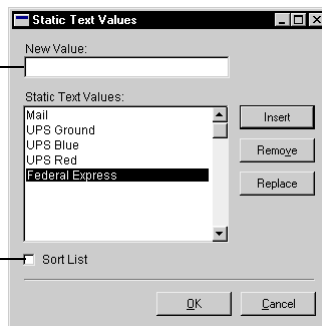
Only those selections available for the control type you selected will appear. Once you've selected the static value type, click the Static Values lookup button. A window will appear allowing you to enter or select static values.

Static text values

If you want to add text to push buttons, radio buttons, check boxes or list boxes, you can add static text values to the data type. The following illustration displays static values for a list box used to select shipping methods.

To add a new static text value, type the item name in the New field, and then click Insert.

Items in the list box can be sorted alphabetically at runtime by marking the Sort List option.



The following table lists common uses for static text values.

Control type	Static text used to:
Button Drop List	Indicate the caption that appears on the button, as well as the selections in the list.
Check box	Indicate the check box prompt name.
Drop-down list	Indicate the selections in the drop-down list.
List box	Indicate the selections in the list box.
Push button	Indicate the caption displayed on the button.
Radio button	Indicate the caption displayed next to the button.
Visual switch	Display two or more text values that will be displayed in sequence as the user clicks the visual switch.

Static picture values

If you want to add graphics to push buttons, button drop lists, visual switches, list views or tree views, you will use static picture values.

Refer to [Chapter 20, “Pictures and Native Pictures,”](#) for more information about creating picture and native picture resources.

You can use two kinds of resources as static picture values:

- *Native picture* resources are specific, or *native*, to a particular operating environment.
- *Picture* resources are pictures you’ve added to the Dexterity picture library.

The following table lists common uses for static picture values.

Control type	Static picture usage
Push button	Pictures and native pictures can be used for images on the button.
Button drop list	Pictures can be used to indicate the images for the button portion of a button drop list.
List view	Pictures can be used for items appearing in the list.
Tree view	Pictures can be used with nodes appearing in the tree.
Visual switch	Can display two or more pictures or native pictures to set up a sequence of images that will change as the user clicks the visual switch.

Control types

A data type is assigned one of several predefined control types, which controls the main characteristic of the data type. This section lists the control type, the storage size, keyable length, an example or illustration, types of static values used, and a description of the control type's function. The following control types are described:

- [Boolean](#)
- [Button drop list](#)
- [Check Box](#)
- [Combo Box](#)
- [Composite](#)
- [Currency](#)
- [Currency \(variable\)](#)
- [Date](#)
- [Drop-down list](#)
- [Horizontal list box](#)
- [Integer](#)
- [List box](#)
- [List view](#)
- [Long integer](#)
- [Multi-select list box](#)
- [Non-native list box](#)
- [Picture](#)
- [Progress indicator](#)
- [Push button](#)
- [Radio button](#)
- [Radio group](#)
- [Reference](#)
- [String](#)
- [Text](#)
- [Time](#)
- [Tiny integer](#)
- [Tree view](#)
- [Visual switch](#)

Boolean

Example	None
Storage size	2 bytes
Keyable length	Not applicable
Static values	None
Function	Stores a boolean (true or false) value. The default value is false.

Button drop list

Example



Storage size	2 bytes
Keyable length	Not applicable
Static values	Pictures, text, or both pictures and text for the button. Text for the items in the list.
Function	Allows one item to be selected from the list. The value in the field is an integer that corresponds to the position of the last item selected in the list. The items in the list are numbered starting with 1. This control is used as a means of navigation. Its value can't be stored in a table.

The items in the list are specified by the data type definition or at runtime, using the [add item](#) statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run. The button drop list does not scroll, so the maximum number of items is limited by screen size.

Refer to [Button drop lists](#) on page 67 of Volume 2 of the Dexterity Programmer's Guide for more information about using button drop lists.

Check Box

Example



Storage size 2 bytes

Keyable length Not applicable

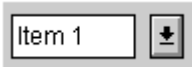
Static values Text, pictures or native pictures for the prompt

Function Stores and displays a boolean (true or false) value. The value in the field is true if marked and false if unmarked.

Refer to [Check boxes](#) on page 69 in Volume 2 of the Dexterity Programmer's Guide for more information about using check boxes.

Combo Box

Example



Storage size The keyable length + 1 + pad if not even

Keyable length Up to 255

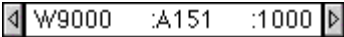
Static values Text for items in the list

Function Allows a text item to be entered by a user or chosen from the list. The value in the field is a string.


The items in the list are specified by the data type definition or at runtime, using the [add item](#) statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run.

The change script for a combo box field runs each time a new value is chosen in the list, including each time the up or down arrow key is pressed to move to a different value.

Composite

Example	
Storage size	Total of components
Keyable length	Not applicable
Static values	None
Function	Groups fields and their data types to form a single data type. The total length of the composite must be less than 128 characters.

Currency

Example	
Storage size	10 bytes
Keyable length	Up to 19
Static values	None
Function	Displays a value as a currency amount, with a currency symbol and thousands separator if specified in the data type's format.

The currency value can be in the range [-99,999,999,999,999.99999 to 99,999,999,999,999.99999]. The decimal point is implied in the number, but not actually stored. For display purposes, currency values are limited to 14 digits to the left of the decimal and 5 digits to the right.

When specifying the keyable length for a currency value, note that all five decimal places must be accounted for. For instance, a currency value that will have the form \$XXX.XX (two decimal places) will actually need to have a keyable length of 8 to account for the three additional decimal places that aren't shown.


Currency (variable)

Example	<div>\$33,340.82</div>
Storage size	8, 10 or 12 bytes
Keyable length	Up to 23
Static values	None
Function	Displays a value as a currency amount, with a currency symbol and thousands separator if specified in the data type's format.

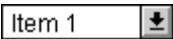
The Currency (variable) control type replaces the Currency (8-byte) control type available prior to release 5.5 of Dexterity. You can re-create an 8-byte currency value using the variable currency control type. Set the Storage size to 8 and the number of decimals to 0.

When you create a variable currency data type, you specify the keyable length and number of decimals. The keyable length is the total number of digits the field will have. Variable currency fields can have up to 23 digits. The Decimals field indicates the number of decimal places the field will have. Variable currency fields can have up to 15 decimal places. The storage size is calculated automatically, based on the keyable length. It can have the value 8, 10 or 12. It must be large enough to hold the number of digits specified in the keyable length.


Date

Example	<div>8/17/1993</div>
Storage size	4 bytes
Keyable length	Up to 10
Static values	None
Function	<p>Stores and displays a date. The date is entered in MMDDYYYY form and is displayed according to the system settings, in short form. The year values can range from 1800 to 9999. You can use the calendar drop-down in the field to select a date.</p> <p>Dexterity automatically checks each date value to ascertain whether its values are within the acceptable ranges of a date value. If the date is not valid, a message is displayed.</p> <p>If only two digits are entered in the date field, the month and year will be defaulted. If the date field previously contained a value, the month and year from the previous value will be used. If the date field was empty, the month and year from the current system date will be used.</p> <p>If only four digits are entered in the date field, the year will be defaulted. If the date field previously contained a value, the year from the previous value will be used. If the date field was empty, the year from the current system date will be used.</p> <p>If only six digits are entered in the date field, the century will be defaulted. Dates entered with year values 36 to 99 have 1900 as the default century value. Dates with year values from 0 to 35 have 2000 as the default century value.</p> <p>An uninitialized date field (one that hasn't been set to a value) will have the value 00000000. Dexterity allows this as an acceptable date value.</p>


Drop-down list

Example	
Storage size	2 bytes
Keyable length	Not applicable
Static values	Text for items in the list
Function	<p>The items in the list are specified by the data type definition or at runtime, using the add item statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run.</p> <p>The value of the field is an integer corresponding to the position of the selected item as it appears in the Static Text Values window. The items in this list are numbered sequentially so that the first item in the list is 1, the second is 2, and so on. If the static text items are sorted for display, the value of the field still is based upon the selected item's position in the Static Text Values window, not its position as displayed in the list at runtime. Up to 32,767 items can be displayed in the list.</p> <p>The change script for a drop-down list runs each time a new value is selected in the list, including each time the up or down arrow key is pressed to move to a different value.</p> <p>You can select an item in the drop-down list by typing the first several characters of the item.</p>


Horizontal list box

Example	
Storage size	2 bytes
Keyable length	Not applicable
Static values	Text
Function	Used only in windows for the Report Writer.

Integer

Example	
Storage size	2 bytes
Keyable length	Up to 5
Static values	None
Function	Displays and stores integers from -32,768 to 32,767.

List box

Example	
Storage size	2 bytes
Keyable length	Not applicable
Static values	Text for items in the list
Function	Allows only one item to be selected in the list box. The items in the list are specified by the data type definition or at runtime, using the add item statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run.

The value of the field is an integer corresponding to the position of the selected item as it appears in the Static Text Values window. Items are numbered sequentially so that the first item in the list is 1, the second is 2, and so on. If the static text items are sorted for display, the value of the field is still based upon the selected item's position in the Static Text Values window, not its position as displayed in the list box at runtime. Up to 32,767 items can be displayed in the list.

The change script for a list box runs each time a new value is selected in the list, including each time the up or down arrow key is pressed to move to a different value.

List view

Example

Address	Type	ID	City	State	Asking Price
150 North 2nd Street	Foursquare	1004	Moorhead	MN	\$50,000.00
419 Roberts Street	Split Level	1001	Fargo	ND	\$70,000.00
4750 9th Avenue South	Split Level	1006	Fargo	ND	\$75,000.00
4756 9th Avenue South	Split Level	1007	Fargo	ND	\$65,000.00
810 Santa Cruz Drive	Ranch/Rambler	1000	Fargo	ND	\$70,000.00
RR 3 - West of City	Ranch/Rambler	1003	West Fargo	ND	\$95,000.00

- Storage size**Not applicable
- Keyable length**Not applicable
- Static values**None
- Function**Displays data in icon form or as a multicolumn list.

Long integer

Example

2,123,875

- Storage size**4 bytes
- Keyable length**Up to 10
- Static values**None
- Function**Displays and stores integers from -2,147,483,648 to 2,147,483,647.

Multi-select list box

Example



Storage size

4 bytes

Keyable length

Not applicable

Static values

Text for items in the list

Function

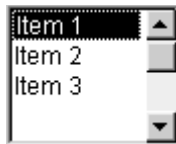
Allows multiple items to be selected in the list box. The items in the field are marked or unmarked by the [check field](#) and [uncheck field](#) statements or by the user. The SHIFT key is used to select consecutive items. The CONTROL key is used to select non-consecutive items.

The status of the items in the list must be examined using the [checkedfield\(\)](#) function.

The items in the list are specified by the data type definition or at runtime, using the [add item](#) statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run. Up to 32 items can be displayed in the list.

The change script for a multi-select list box field runs each time a new value is selected in the list, including each time the up or down arrow key is pressed to move to a different value.

Non-native list box

Example**Storage size**

2 bytes

Keyable length

Not applicable

Static values

Text for items in the list

Function

Allows only one item to be selected in the list box. The items in the list are specified by the data type definition or at runtime, using the [add item](#) statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run.

The value of the field is an integer corresponding to the position of the selected item as it appears in the Static Text Values window. Items are numbered sequentially so that the first item in the list is 1, the second is 2, and so on. If the static text items are sorted for display, the value of the field is still based upon the selected item's position in the Static Text Values window, not its position as displayed in the list at runtime. Up to 32,767 items can be displayed in the list.


The change script for a non-native list box runs each time a new value is selected in the list, including each time the up or down arrow key is pressed to move to a different value.

A non-native list box control is created completely by Dexterity.

Picture

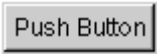
Example	Displays the picture pasted into the field.
Storage size	Variable
Keyable length	Up to 32,767
Static values	None
Function	Allows a picture to be pasted into and displayed by an application, and stored in a table. The keyable length should be set to 32,767. Pictures up to the size of the keyable length, typically 32,767 bytes, can be pasted into a picture field.

Progress indicator

Example	
Storage size	2 bytes
Keyable length	Not applicable
Static values	None
Function	<p>Allows progress to be shown visually. The indicator position is based on the value of the progress indicator field. The valid values range from 0 to 100. When set to 0 or less, no indicator is displayed. When set to 100 or greater, the entire indicator is displayed. For values in between, the amount displayed is proportional to the value.</p> <p>Several properties control the display characteristics of the progress indicator, such as style and indicator color. If the indicator is composed of blocks, the block size is set automatically based upon the height of the progress indicator.</p>

Push button

Example



Storage size 2 bytes

Keyable length Not applicable

Static values Text, pictures or native pictures

Function Provides a means of starting scripts. The change script associated with the field is run when the button is clicked. The field doesn't have a value and can't be stored in a table.

Refer to [Push buttons](#) on page 70 in Volume 2 of the Dexterity Programmer's Guide for more information about using push buttons.

Radio button

Example



Storage size 2 bytes

Keyable length Not applicable

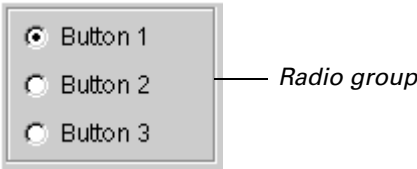
Static values Text for each radio button prompt

Function Radio button values are stored using a radio group. The value of the radio group is an integer that corresponds to the radio button currently selected. A specific radio button's value is determined by its position in the tab sequence; the first radio button's value is 0, the second is 1, and so on. The radio group must come immediately before the radio buttons in the tab sequence.

Refer to [Radio buttons](#) on page 73 in Volume 2 of the Dexterity Programmer's Guide for more information about radio buttons.

Radio group

Example



Storage size	2 bytes
Keyable length	Not applicable
Static values	None
Function	<p>Groups and stores a single value for the radio buttons inside the radio group.</p> <p>The value stored is an integer corresponding to the position of the selected radio button in the tab sequence; if the first radio button is selected, the value 0 is stored; if the second one is selected, the value 1 is stored, and so on. The radio group must precede the radio buttons in the tab sequence.</p>

Reference

Example	None
Storage size	Not applicable
Keyable length	Not applicable
Static values	None
Function	Temporarily stores a reference to a resource in the application.

String

Example	<div>This is a string.</div>
Storage size	Length + 1 + pad if not even
Keyable length	Up to 255
Static values	None
Function	Allows entry and display of strings of up to 255 characters.

Text

Example	<div>This is a text field.</div>
Storage size	Length + 2 + pad if not even
Keyable length	Up to 32,000
Static values	None
Function	<p>Displays text and allows a user to enter text into the application. The text will wrap only if the WordWrap property for the field using the data type is set to true. Scroll bars can be turned off by setting the ScrollBars property for the field using the data type to false.</p> <p>Up to 32,000 characters can be stored in the field.</p> <p>In the current version of Dexterity, text fields should <i>not</i> be used in tables that are checked for multiuser error conditions in a multiuser environment. Records can't be locked properly if text fields are included. Refer to Chapter 18, "Multiuser processing," in Volume 2 of the Dexterity Programmer's Guide for more information.</p> <p>Refer to Text fields on page 77 in Volume 2 of the Dexterity Programmer's Guide for more information about using text fields.</p>

Time

Example	<div>6:17:32 PM</div>
Storage size	4 bytes
Keyable length	Up to 6
Static values	None
Function	<p>Time is entered in 24-hour format, such as 181732, and is displayed in 12-hour format, such as 6:17:32 PM. Dexterity automatically checks each time value to ascertain whether it's within the proper range of a time value.</p> <p>An uninitialized time field (one that hasn't been set to a value) will have the value 000000. Dexterity allows this as an acceptable time value.</p>

Tiny integer

Example	<div>25</div>
Storage size	1 byte
Keyable length	Up to 3
Static values	None
Function	Displays and stores integers from 0 to 255.

Tree view

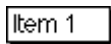
Example



Storage size	Not applicable
Keyable length	Not applicable
Static values	None
Function	Displays data as an expandable outline.

Visual switch

Example

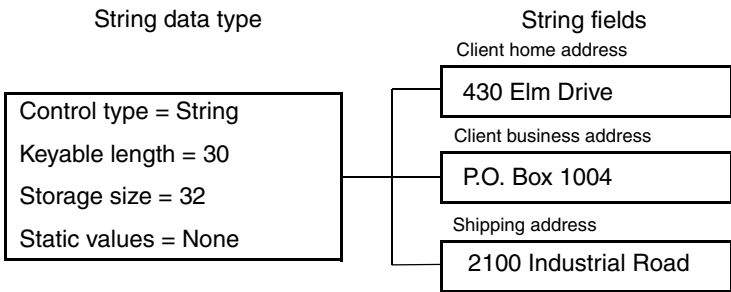


Storage size	2 bytes
Keyable length	Not applicable
Static values	Text, picture or native picture
Function	<p>Displays a series of items. The next item in the field is displayed when the field is clicked. The value of the field is an integer corresponding to the position of the currently-displayed item in the series, starting with 1 and incremented by 1.</p> <p>The items in a visual switch are defined by the static values of the data type or when the application is running, using the add item statement. Items added at runtime aren't saved in the application dictionary; they must be added each time the application is run.</p> <p>It's sometimes necessary to use a script to assign the field a value to initially display an item in the field. This typically occurs when the value of the visual switch won't be retrieved from a table.</p>

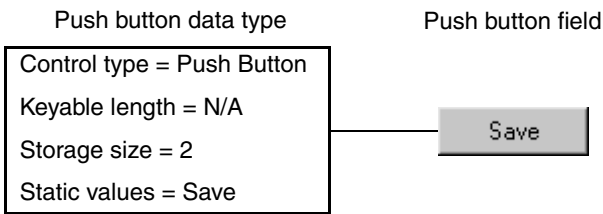
Using data types

Each field can either use its own data type or share a data type with other fields that have similar display and storage characteristics. For instance, a single data type can be used by several string fields to specify how those fields function in your application. The fields sharing the same data type will have the same keyable length, storage size and static values. If you change one of these characteristics for the data type, such as the keyable length, all fields using this data type will reflect the change.

In the following example, three address fields use the same data type. Although the fields belong to different tables and display different information, the information is stored and displayed in the same manner because each uses the same data type:

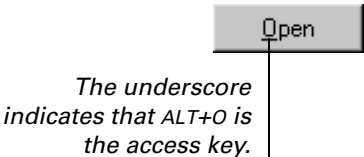


The following data type defines the characteristics of a Save button.



Push buttons

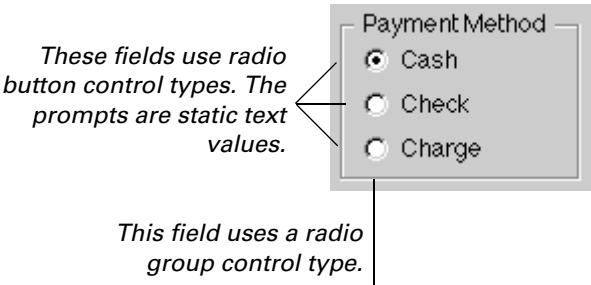
Push buttons can be clicked using access keys (pressing ALT in combination with a letter). At runtime, an underscore beneath one of the letters in the static text for the push button indicates the button has an access key. In the following illustration, typing ALT+O will click the Open button.



To define the access key, place an ampersand (&) in the static text value for the button before the letter that will act as the access key. For the Open button, &Open is the static text value used.

Radio buttons

To make radio buttons work properly in your application, you need to understand how they operate. Two control types are used: radio button and radio group. You create a radio button data type for each radio button that will appear in the group. You also create a radio group data type that will group the radio buttons. For example, the Payment Method radio group shown in the following illustration uses the data types listed in the table.



Data type	Control type
Payment Method	Radio Group
Cash	Radio Button
Check	Radio Button
Charge	Radio Button

Procedure: Creating data types

To create a data type, point to New in the Explorer menu and choose Data Type. The Data Type Definition window will appear as shown in the following illustration.

The data type name is entered here.

The control type is the main characteristic of the data type.

The screenshot shows the 'Data Type Definition' dialog box. It contains the following fields and controls:

- Data Type Name:** A text box containing 'STR30'.
- Control Type:** A dropdown menu showing 'String'.
- Keyable Length:** A text box containing '30'.
- Storage Size:** A text box containing '32'.
- Decimals:** A text box containing '0'.
- Static Values:** A text box with a dropdown arrow and a '...' button.
- Format:** A text box with a '...' button.
- Composite:** A text box with a '...' button.
- Allow Odd Length:** A checkbox that is currently unchecked.
- Options:** A section containing two checkboxes: 'Auto Add Field' (unchecked) and 'Call Component Scripts' (unchecked).
- Buttons:** 'OK', 'Add', and 'Cancel' buttons at the bottom right.

1. Name the data type.

Enter the name in the Data Type Name field. The data type name will appear in a list of data types that will be used later when you're defining fields.

The data type name should be descriptive enough to allow you to choose the appropriate data type when defining fields, without having to open the data type to see its characteristics.

2. Select the control type.

The control type specifies how the data type will store and display data, and is the main characteristic of a data type.

3. Enter a keyable length (if required).

Data types that have certain control types allow you to define the data to be entered. The keyable length sets the maximum length of the data that can be entered.

You can define the keyable lengths for the following control types:

- Combo box
- Currency
- Currency (variable)
- Date
- Integer
- Long integer
- Picture
- String
- Text
- Time
- Tiny integer

4. Enter static values (if required)

Static values are any static text or pictures associated with a data type. Static values are text, pictures or native pictures that can be displayed on fields associated with the data type. Examples of static values include text or pictures on push buttons, or the items in a list box. The following data types can use static values:

- Button drop list
- Check box
- Combo box
- Drop-down list
- Horizontal list box
- List box
- List view
- Multi-select list box
- Non-native list box
- Push button
- Radio button
- Tree view
- Visual switch

Only those selections available for the control type you selected will appear. Once you've selected the static value type, click the Static Values lookup button. A window will appear allowing you to enter static values for the type of static value you selected.

*Formats are described
in the next chapter.*

5. Choose or create a format for the data type (if necessary).

A format contains the extra characters, spacing and attributes that can be applied to a data type when data is entered or displayed. For example, a phone number can be displayed as (555) 123-4567. The actual phone number stored in the field is 5551234567. The format applied to the number is (XXX) XXX-XXXX. By using a format, only the phone number itself needs to be stored, not the extra characters for the formatting.

6. Select a composite if the data type is for a composite field.

A *composite* is a group of fields and their associated data types that are combined to form a single data type. You may also choose to mark the Call Component Scripts option for the composite data type. Composites and the Call Component Scripts options are described in [Chapter 9, "Composites."](#)

7. Mark the Auto Add Field option (optional)

If this selection is marked and the data type is saved, Dexterity will automatically create a field that has the same name as the data type and is assigned the data type. (The field name will be the same as the data type name; the field's physical name will be created using the first 25 characters of the data type name. You can use the Field Definition window to change either name.)

8. Save the data type.

Click OK to save the data type. The Data Type Definition window will close. If you are creating a new data type, you can click the Add button; the data type will be saved, but the Data Type Definition window will remain open, allowing you to create another data type.

Related items

Use the information listed to learn more about creating and using data types.

Commands

[add item](#), [check field](#), [checkedfield\(\)](#), [countitems\(\)](#), [delete item](#), [finditem\(\)](#), [itemname\(\)](#), [str\(\)](#), [uncheck field](#), [value\(\)](#)

Additional information

[Appendix A, "Naming Conventions."](#) in the Integration Guide.

Chapter 7: Formats

Formats are the extra characters, spacing and attributes that can be applied to a data type to format data when it is entered or displayed. For example, a string data type for a phone number can have a format applied to it so a field using the data type will display a phone number as (555) 123-4567 instead of 5551234567.

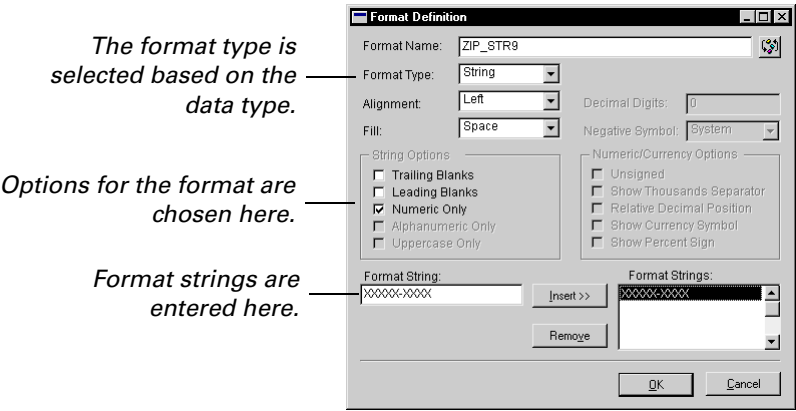
Formats are stored as separate resources, but are applied to data types to help define how information is displayed in a field at runtime. A single format resource can be used by several data types. Perhaps the easiest way to understand what formats are and how they're used is to view them as data "masks" that simply change the look of the information in a field without changing the actual information itself.

Information about formats is divided into the following sections:

- [*Format elements*](#)
- [*Format string*](#)
- [*The Multiple Format Selector*](#)
- [*Procedure: Creating formats*](#)

Format elements

A format can have a name, formatting options and a format string. Formats can be used for currency, string, numeric and composite data types. Depending upon the data type, different formatting options are available. The Format Definition window, shown in the following illustration, is used to set the characteristics of a format.



Name

Each format in your application must have a name. Typically, the name indicates the data type the format is applied to. For instance, the name Phone_Number indicates this format will be applied to the Phone_Number data type.

Formatting options

You can use several formatting options to change how specific types of data will appear in your application. You can use format options to do the following:

- Align information to the left, right or center of a field.
- Determine what characters will appear in unused parts of a field.
- Specify how you want currency fields to appear.
- Specify how numeric fields display information.

Format string

The Format Definition window allows you to specify a *format string* for the string and composite data types you create.

String formats

Format strings are used with string data types to add static elements to a field, such as parentheses or static text. Dexterity uses the capital X as a place holder to represent alphanumeric characters that will appear in the field. All other characters will be displayed as they are typed.

For instance, suppose you're using a specific data type to store information entered in phone number fields. This Phone_Number data type uses a format and a format string to determine how phone numbers will appear at runtime:

Format string	Data entered	Data displayed
(XXX) XXX-XXXX	7015550100	(701) 555-0100
ext. XXXX	6590	ext. 6590

The Xs are placeholders indicating where the digits will be displayed, while the parentheses and dash are displayed as they were typed. At runtime, the phone number can be entered with the format string displaying the special characters in the field.

Also note that a lowercase x is used in the "ext." abbreviation in the second format string. Only uppercase Xs are treated as place holders, so the lowercase x is displayed in the field, instead of being replaced when data is entered.

Phone:

(701) 555-0100



Static characters in a format string aren't actually stored with the data in the table. This allows you to change the format string without affecting how data is stored.

For information about composites, refer to [Chapter 9, “Composites.”](#)

Composite formats

A format string is used with a standard composite data type to indicate the size and order of the components, and to add static elements, such as parentheses or static text. The numeric characters 1 through 9 are used to represent the characters of each component of the composite. All other characters will be displayed as they are typed.

A format string is used with an extended composite data type to specify a “starting” format for the composite. The final format for the composite will be specified using script commands. This is described in [Chapter 9, “Composites.”](#)

The following example shows the use of a format for a standard composite. A composite must have a format and a format string. This composite contains three parts, as indicated by the format string.

Format string	Data entered	Data displayed
1111-22-3333	1000ND5050	1000-ND-5050

Note that the 1s, 2s and 3s are used as placeholders.

The Multiple Format Selector

This is an advanced formatting feature. You may want to skip this section until you have a better understanding of Dexterity.

The Multiple Format Selector allows you to change the format applied to a currency or string field while the application is running. Multiple formats are set up by selecting Link Format Selector from the Tools menu while the layout window is active. Dexterity will switch to *format linking mode*, allowing you to link the string or currency field to an integer field, called a *format field*. Dexterity will stay in this mode until you choose Link Format Selector from the Tools menu again.

While Dexterity is in format linking mode you can link fields to format fields by dragging from a currency or string field to the integer field you want to link it to. A flashing border around the integer field will indicate that the link was successful. The value of the integer field indicates which format string should be used for the currency or string field. This integer field can be an invisible field or a field such as a drop-down list, allowing the user to specify the format.

Formatting currency fields

The possible formats for currency fields and the integer value associated with them are shown in the following table.

Integer	Format	Integer	Format
0	Control panel defaults	10	\$1,234.567
1	1,234	11	\$1,234.5678
2	1,234.5	12	\$1,234.56789
3	1,234.56	13	1,234%
4	1,234.567	14	1,234.5%
5	1,234.5678	15	1,234.56%
6	1,234.56789	16	1,234.567%
7	\$1,234.	17	1,234.5678%
8	\$1,234.5	18	1,234.56789%
9	\$1,234.56		

The possible formats for variable currency fields and the integer value associated with them are shown in the following table.

Integer	Format	Integer	Format	Integer	Format
100	1	200	\$1	300	1%
101	1.1	201	\$1.1	301	1.1%
102	1.12	202	\$1.12	302	1.12%
103	1.123	203	\$1.123	303	1.123%
104	1.1234	204	\$1.1234	304	1.1234%
105	1.12345	205	\$1.12345	305	1.12345%
106	1.123456	206	\$1.123456	306	1.123456%
107	1.1234567	207	\$1.1234567	307	1.1234567%
108	1.12345678	208	\$1.12345678	308	1.12345678%
109	1.123456789	209	\$1.123456789	309	1.123456789%
110	1.1234567890	210	\$1.1234567890	310	1.1234567890%
111	1.12345678901	211	\$1.12345678901	311	1.12345678901%
112	1.123456789012	212	\$1.123456789012	312	1.123456789012%
113	1.1234567890123	213	\$1.1234567890123	313	1.1234567890123%
114	1.12345678901234	214	\$1.12345678901234	314	1.12345678901234%
115	1.123456789012345	215	\$1.123456789012345	315	1.123456789012345%

The integer value that indicates the format used can be stored in a table along with the currency field being formatted. When the record is retrieved from the table, the currency field will display as it did when the record was saved. The format field can also be used for a report, allowing the currency field to appear in the report as it did when it was saved in the table.



You can use functions from the [Currency function library](#) to create your own multiformat values. Refer to the Currency library in the Function Library Reference manual for more information.

The following example uses the Multiple Format Selector for a Quantity field to specify the number of decimal places it will display. The Quantity field uses a currency data type and is linked to a hidden integer field that specifies which format will be applied when the field is displayed in a window or printed on a report. The Quantity field is shown in the following illustration when its linked field has a value 3, indicating that no currency symbol and only two decimal places will be displayed.

Quantity:

Formatting string fields

For string fields, the integer value indicating the format to use corresponds to the position of the format string in the format definition window.

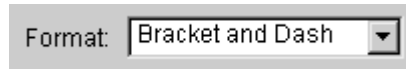
The following example uses the Multiple Format Selector for a Part Number field that has three different format strings. The user selects the format to apply by selecting the corresponding format in a drop-down list linked to the Part Number field. The three format strings for the Part Number field, entered in the Format Definition window, are shown in the following illustration.

- This format string is used when the format selector value is 0 or 1.*
- This format string is used when the format selector value is 2.*
- This format string is used when the format selector value is 3.*

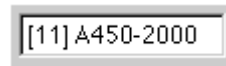
Format Strings:

- XX-XXXXXXXXXX
- XX-XXXX-XXXX
- [XX] XXXX-XXXX

The drop-down list is set to the following value (integer 3), corresponding to the third format string:

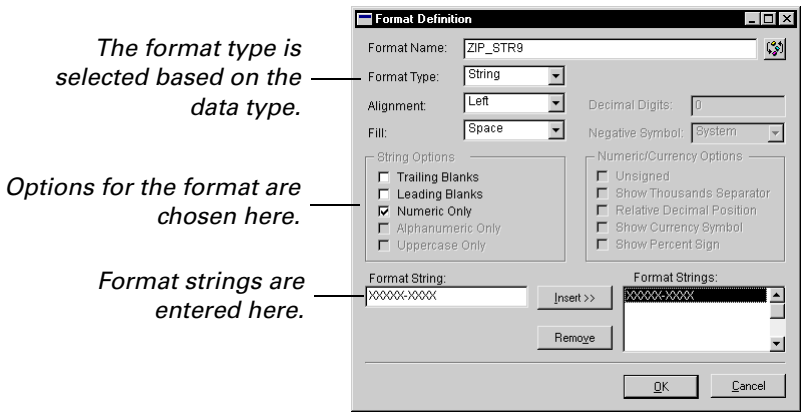


The Part Number field will display the following format:



Procedure: Creating formats

To create a format, point to New in the Explorer menu and choose Format. The Format Definition window will appear.



The Format Definition window can also be opened from the Data Type Definition window. Open the data type to be formatted and click the Format lookup button in Data Type Definition window to view the list of existing formats. Click New to open the Format Definition window.

1. Name the format.

Enter the name in the Format Name field. The format name will appear in a list of formats so the format can be selected easily and used again.

Like the data type name, the format name should be descriptive but not too long. When re-using a format, a descriptive name will allow you to pick out the appropriate format from the format list without opening the format to see its characteristics.

2. Select the format type (if necessary).

If you opened the Format Definition window by using the Resource Explorer or the Explorer menu, specify which type of format you are creating - currency, string, composite or numeric. If you opened the Format Definition window from the Data Type Definition window, the type of format has already been specified.

3. Select the necessary formatting options.

The formatting options available depend upon the type of format you are creating. The following charts show which options are available for each type:

Numeric formats	
Decimal Digits	Number of decimal places (0 to 5).
Negative Symbol	The operating system setting, a minus sign, the letters CR, or parentheses.
Alignment	Number is left-, center- or right-aligned.
Fill	Unused spaces are filled with asterisks, zeros or spaces.
Unsigned	If marked, the negative symbol won't be displayed.
Show Thousands Separator	If marked, the field will show thousands separators in the number.
Show Percent Sign	If marked, the field will show the percent sign.

Currency formats	
Decimal Digits	Number of decimal places (0 to 5).
Negative Symbol	The operating system setting, a minus sign, the letters CR, or parentheses.
Alignment	Number is left-, center- or right-aligned.
Fill	Unused spaces are filled with asterisks, zeros or spaces.
Unsigned	If marked, the negative symbol won't be displayed.
Show Thousands Separator	If marked, the field will show thousands separators in the number.
Relative Decimal Position	If marked, the number of decimal digits selected is added to the number in the operating system setting. The total can be up to 5 decimal digits.
Show Currency Symbol	If marked, the currency symbol specified in the operating system settings is displayed.

String and composite formats	
Alignment	String is left-, center- or right-aligned.
Fill	Unused characters are filled with asterisks, zeros or spaces.
Trailing Blanks	If marked, any spaces that follow the contents of the field will be saved in the file. If the selection isn't marked, any spaces following the item in the field won't be saved in a file. This selection can be left unmarked to ensure that the same entry made with or without trailing blanks will be stored with the same value.
Leading Blanks	If marked, spaces can be entered as the first characters in the field. These spaces will be saved in a file.
Numeric Only	If marked, only numbers can be entered in the field.
Alphanumeric Only	If marked, only letters and numbers can be entered in the field.
Uppercase Only	If marked, all alphabetic characters will be displayed in uppercase.

4. Enter the format string.

The format string controls the formatting for strings or composites. It contains any extra characters used in the format, along with placeholders that show where the stored data will be displayed. The following table shows the use of placeholders for strings and composites:

Type	Placeholder	Example
String	Capital Xs	(XXX)-XXX-XXXX
Composite	Numbers indicating the components to display.	1111-2222-33333

After entering the format string, click Insert to save it. Any additional format strings you add will be inserted below the format selected in the Format Strings list. If no format string is selected, the additional format string will be inserted at the top of the list.

Click OK to save the format definition.

Related items

Use the information listed to learn more about creating and using formats.

Commands

[**format\(\)**](#)

Additional information

[Currency function library](#) in the Function Library Reference manual.

[Appendix A, “Naming Conventions.”](#) in the Integration Guide.

[Chapter 41, “Modifying Fields.”](#) in Volume 1 of the Dexterity Programmer’s Guide.

Chapter 8: Global Fields

Fields you create using Dexterity represent individual pieces of information in an application, and can appear in windows and be stored in tables. Each field in an application uses a data type to specify its characteristics. Many fields can use the same data type. Two types of fields are used in an application dictionary: *global fields* and *local fields*.

- *Global fields* can be used in windows for the application, and can be stored in tables. They can also be used as global variables. Global fields are the subject of this chapter.
- *Local fields* can be used only in windows. Local fields are discussed in detail in [Chapter 11, “Windows.”](#)

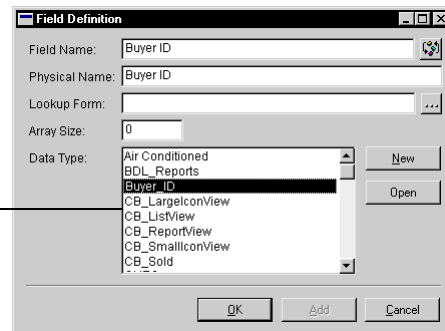
Information about global fields is divided into the following sections:

- [Field elements](#)
- [Using global fields](#)
- [Array fields](#)
- [Procedure: Creating global fields](#)

Field elements

Each field in an application has a field name, a physical name, an associated data type, an array size, and a lookup form. (Specifying a lookup form is optional.) The Field Definition window, shown in the following illustration, is used to create fields and specify their characteristics.

The data types you’ve set up using the Data Type Definition window will appear in the data type list.



Names

Each field can have two names, a *field name* and a *physical name*. The field name is required for all fields. The physical name is a shortened version (up to 24 characters) of the field name. It is used as the column name if the field is used in a SQL table.

Data type

Each field you create must be associated with one of the data types previously defined for the application, listed in the Field Definition window. The data type, which specifies most of the characteristics of the field, can be used by other fields, as well.

To view or change the settings for a data type, select a data type and click Open. The Data Type Definition window for the data type you selected will appear, allowing you to view or change the options for the data type.

Array size

An array field contains multiple occurrences of the same type of information. The individual pieces of information stored by the array field are referred to as *elements*. When you create a field, you can enter an *array size* for the field. The array size of a field indicates the number of elements that will be included in the array field. If the field won't be an array, set the array size to 0.

For example, you could create an array field to store two phone numbers by specifying an array size of 2. The following illustration shows how the array field would store data.

Name	Address	Phone [1] Phone [2]	City	State
Steve Santos	11 Ash Lane	(913) 555-3843 (913) 549-9362	Topeka	KS

Note that “Phone” is a single field, but that it has two elements, each containing a phone number.

Lookup form

You can also specify a lookup form for the field. A lookup form allows a user to easily enter information into the field. Lookups are described in detail in [Chapter 11](#) of Volume 2 of the Dexterity Programmer's Guide.

Using global fields

The fields you create using the Field Definition window are called *global fields*, meaning they can be displayed in windows or stored in tables. In contrast, *local fields* can be used only in windows. Local fields are discussed in [Chapter 11](#).

Window fields

When you add global fields to a window layout, *window fields* are created. Window fields using data types with push button, button drop list, check box, radio button or visual switch control types can act as window controls. Window fields using data types with string, integer, currency, list box or other similar control types can display information and are the method by which your application's users can enter new information or select predefined options.

For example, when users enter customer information in a window, they're entering information in *window fields*.

Seller ID: 200 ...

Seller Name: Mike and Fran Gray

Address: 810 Santa Cruz Drive

City, State, ZIP: Fargo ND 58102-5600

Phone: (701) 255-1000

Window fields

Table fields

Global fields you create and use in tables are called *table fields*. These fields are used to store the pieces of information that make up a table. After you create fields using the Field Definition window, each can be added to a table definition so you'll be able to store the contents of the field in a table.

For example the following table lists fields that would be used to store seller records in a Seller_Data table. Each field has a data type, and some use the same data type.

This field	Stores this data	Using this data type
Seller ID	String of 6 characters	Seller ID
Seller Name	String of 30 characters	STR30
Address	String of 30 characters	STR30
City	String of 20 characters	STR20
State	String of 2 characters	STR2
ZIP Code	String of 9 characters	STR9_ZIP With formatting for a ZIP code
Phone Number	String of 10 characters	STR10_Phone With formatting for a phone number

In addition, note:

- The same global field can be used as a window field and a table field. For instance, in the previous example the same global fields are used as window fields and as table fields.
- All global fields can be used as window fields, but fields using data types with push button, button drop list and radio button control types can't be used as table fields.

Global variables

With the Resource Explorer, you can create global variables from global fields. Global variables are special variables available to any script in the application dictionary at any time. Refer to [Chapter 22, "Global Variables,"](#) for more information.

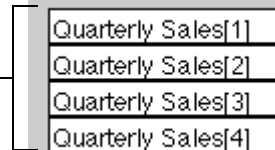
Array fields

Refer to the [Chapter 1, "Syntax,"](#) in Volume 2 of the *Dexterity Programmer's Guide* for information about referencing array fields in scripts.

If several fields in your application have the same data type and contain similar related information, it may be more convenient to use an array, rather than individual fields.

For example, if your application displays quarterly sales, it may be more convenient to use an array field with four elements, rather than four separate fields. The array is shown in the following illustration.

*These are the four
elements of the
Quarterly Sales field.*



The diagram shows a vertical stack of four rectangular boxes, each containing a text label. A horizontal line extends from the text to the left of the first box, and a bracket-like shape connects this line to the text on the left. The labels in the boxes are: Quarterly Sales[1], Quarterly Sales[2], Quarterly Sales[3], and Quarterly Sales[4].

Quarterly Sales[1]
Quarterly Sales[2]
Quarterly Sales[3]
Quarterly Sales[4]

To create an array field, set the Array Size field in the Field Definition window to the number of elements you want in the array. In the previous example, the array size was set to 4.

Procedure: Creating global fields

To create a new field, point to New in the Explorer menu and choose Global Fields. The Field Definition window will appear as shown in the following illustration.

The data types you've set up using the Data Type Definition window will appear in the data type list.

The screenshot shows a 'Field Definition' dialog box. It contains several input fields: 'Field Name' with 'Buyer ID', 'Physical Name' with 'Buyer ID', 'Lookup Form' (empty), and 'Array Size' with '0'. Below these is a 'Data Type' list box containing several items, with 'Buyer_ID' currently selected. To the right of the list are 'New' and 'Open' buttons. At the bottom of the dialog are 'OK', 'Add', and 'Cancel' buttons.

1. Name the field.

Enter the name of the field in the Field Definition window. This name will appear in a fields list when you create tables and windows.

The physical name is a shortened version (up to 24 characters) of the field name, and is used as the column name if the field is used in a SQL table.



Use caution when changing the name of a field. If the field name has already been used in scripts, they won't compile properly until the field name is changed in them, as well.

2. Choose the data type associated with the field.

Every field must have a data type associated with it. The data type indicates what kind of information the field will display or store. Any formatting applied to the data type also will be applied to the field. Select the data type from the Data Type list in the Field Definition window.



You can select a data type and click Open to open the Data Type Definition window and modify the selected data type.

3. Select an array size.

The Array Size field allows you to enter the number of items to include in an array of the selected data type. The default value is 0, indicating the field will not be an array.

If there will be several fields that will have the same data type and contain similar information, it may be more convenient to use an array, rather than individual fields. For example, if an application keeps track of monthly sales for a year, an array of size 12 could be used instead of 12 separate fields.



You don't have to set the array size to 0 if the field won't be an array field; 0 will appear as the default automatically if the size is not set to another value.

4. Select a lookup form (optional).

Use the Lookup Form lookup button to specify a lookup form for the field. A lookup form allows a user to easily enter information into the field. Lookups are described in detail in [Chapter 11](#) of Volume 2 of the Dexterity Programmer's Guide.

5. Save the global field.

Click OK to save the global field. The Field Definition window will be closed. If you are creating a new field, you can click the Add button; the field will be saved, but the Field Definition window will remain open, allowing you to create another field.

Related items

Use the information listed to learn more about creating and using global fields.

Additional information

[Appendix A, "Naming Conventions,"](#) in the Integration Guide.

Chapter 9: Tables

Understanding how data is stored in tables will help you better visualize how your application will process and maintain information. Once tables have been defined, you can link them to Dexterity resources such as windows and reports, allowing several parts of your application to access the table's information. Information about tables is divided into the following sections:

- [*Table concepts*](#)
- [*Table elements*](#)
- [*Table fields*](#)
- [*Keys*](#)
- [*Key options*](#)
- [*Segment options*](#)
- [*SQL Auto Procedure Disable Options*](#)
- [*Table options*](#)
- [*Temporary tables*](#)
- [*Table locations*](#)
- [*Table groups*](#)
- [*Procedure: Creating tables*](#)

Table concepts

For tables to store information properly, you will need to define the following items in your application:

Fields A field represents one category of information that can be stored in a table, such as a customer name or a customer address. For instance, if you were to track customer names and addresses, you could use the following fields:

Fields

Customer ID	Customer Name	Address	City	State

Records A record is made up of one instance of each field in a table. All of the records in a table contain the same fields (categories of data). For keeping track of customer names and addresses, think of a record as one row in a table containing the information. Each row (record) contains the information for one customer.

Record —

Customer ID	Customer Name	Address	City	State
C1002	Jean Thompson	P.O. Box 8392 82 101 Ave.	Kansas City	MO
C1392	Serge Lemieux	11 Ash Lane	Edmonton	AB
C4432	Dan Smith	239 Hampton Village	Fargo	ND
C4499	Cheryl Miner	9800 Woodland Drive	Springfield	IL

Tables A table is a collection of records, such as your business customer records. Tables in Dexterity group related records the same way the table in the following illustration groups customer records.

Customer ID	Customer Name	Address	City	State
C1002	Jean Thompson	P.O. Box 8392 82 101 Ave.	Kansas City	MO
C1392	Serge Lemieux	11 Ash Lane	Edmonton	AB
C4432	Dan Smith	239 Hampton Village	Fargo	ND
C4499	Cheryl Miner	9800 Woodland Drive	Springfield	IL

Table

Keys A key is a field or combination of fields within a record that is used as a basis by which to store, retrieve and sort records. Typically, the value of the key field is unique for each record so a particular record can be located quickly.

For instance, to locate a particular customer in the customer name and address table, you could search the table alphabetically using the customer name column. In this case, you’re using the customer name field as the key. To make searching easy, you would want the customer records indexed by the customer name.

Key —————

Customer ID	Customer Name	Address	City	State
-------------	---------------	---------	------	-------

The following example shows how Dexterity could store information for customer records in a table:

Fields —————

Key 1 —————

Key 2 —————

Customer ID	Customer Name	Address	City	State
C1002	Jean Thompson	P.O. Box 8392 82 101 Ave.	Kansas City	MO
C1392	Serge Lemieux	11 Ash Lane	Edmonton	AB
C4432	Dan Smith	239 Hampton Village	Fargo	ND
C4499	Cheryl Miner	9800 Woodland Drive	Springfield	IL

Record —————

Table

This table has five fields: Customer ID, Customer Name, Address, City and State. It also has four records, each containing one Customer ID, Customer Name, Address, City and State field. The table has two keys. The first key is composed of the Customer ID field. Because no two customers can have the same Customer ID, this key ensures each customer is uniquely identified. The second key is composed of the Customer Name field. This key allows a customer record to be easily located, based upon the customer’s name.

Table elements

Tables store data generated and used by your application. When you create tables in Dexterity, you’ll use or define several elements required for tables to store and access information properly, such as fields, keys and table relationships. Before you create tables, be sure you’ve defined all the fields that you’ll be using to store information in the table.



If you’re creating applications that integrate with Microsoft Dynamics GP, be aware that Microsoft Dynamics GP table structures can’t be modified. Additional information must be stored separately in tables you add.

Each table has a table name, display name, physical name, series, database type, table fields, keys, key options, segment options and table options. Use the Table Definition window to create tables.

The names for the table are entered here.

The keys you’ve set up for the table will appear in the Keys list.

To add a field to the table, select a global field and then click the Insert button.

Names

Three names are required for each table. The *table name* is the name that is used in scripts to refer to the table. To make the name easier to use in scripts, you can use underscores (_) between parts of the name instead of spaces.

The *display name* is the name that appears when the name of the table is displayed on the screen in an application. This name is usually the same as the table name, but typically has spaces between parts of the name, rather than underscores.

You create temporary tables by specifying "temp" as the table's physical name. Temporary tables are described later in this chapter.

The *physical name* is the name under which the table is stored by the operating system or database. By default, this name can be up to eight characters long and must conform to the operating system standards for the system running Dexterity. The appropriate extension, if required, is added automatically.

The following names are for a table containing customer information:

Table Name	Customer_Data
Display Name	Customer Data
Physical Name	CUSTDATA

If you're developing an application for an operating system that supports long file names, you can mark the Allow Long Physical Names option in the Options window. (Choose Options from the Edit menu to open the Options window.) Marking this option allows you to enter table physical names up to 24 characters long.

Mark this option in the Options window to allow long physical names for tables.

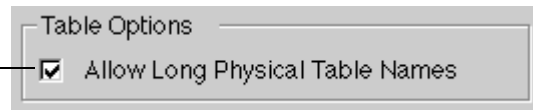


Table series

When you create a table, you must specify the series the table is part of. The table series allows you to group related tables in a Dexterity application together using series categories like Sales, Financial and System. For example, you could assign all of the tables used in a Sales Order Processing application to the Sales series. You will use these series groupings when you use the Report Writer, set up pathnames, and perform table maintenance routines.

Database type

The following database types are supported by Dexterity: c-tree Plus, Pervasive.SQL, and Microsoft SQL Server. Use the following guidelines when selecting a database type:

c-tree Choose c-tree to use the c-tree Plus database type.

If you use the c-tree Plus database type, each table will have two files created. One will have the physical name you specified plus the extension .DAT (data file), while the other will have the physical name plus the extension .IDX (index file).

Pervasive.SQL Choose Pervasive.SQL to use the Pervasive.SQL database type. This database type was formerly known as Btrieve.

If you use the Pervasive.SQL database type, each table will be created with the physical name you specified, plus the extension .BTR. You can use the **BtrieveFileNameExtension** defaults file setting to specify a different extension.

SQL Choose SQL to use the SQL database type. If you use the SQL database type, each table will be created with the physical name you specified.

Memory Choose Memory to have the table created and stored in memory. Memory-based tables act just like c-tree tables, with the exception that they are stored in memory, rather than written to a physical disk. For more information about using memory-based tables, refer to [Chapter 17](#) in Volume 2 of the Dexterity Programmer's Guide.

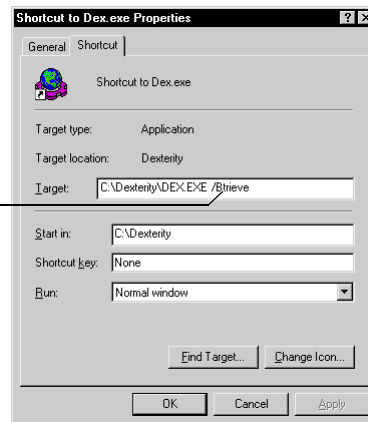
Default Choose Default to allow the database type to be determined at runtime. At runtime, Dexterity or the runtime engine will ascertain which database type to use based upon several system settings. The **DatabaseType** defaults file setting or the [Table SetDBType\(\)](#) function can be used on any platform to set the default database type.

A command line option can also be used to specify the default database type. The following table lists the command line options and the default database they specify.

Command line option	Default database type
/Ctree	c-tree Plus
/Btrieve	Pervasive.SQL (Btrieve)
/SQL	SQL

The following illustration shows the command line option to specify Pervasive.SQL (Btrieve) as the default database type.

This command line option specifies Pervasive.SQL (Btrieve) as the default database type.



The command line option overrides the **DatabaseType** defaults file setting. The [Table SetDBType\(\)](#) function overrides both the command line option and the defaults file setting.

Table fields

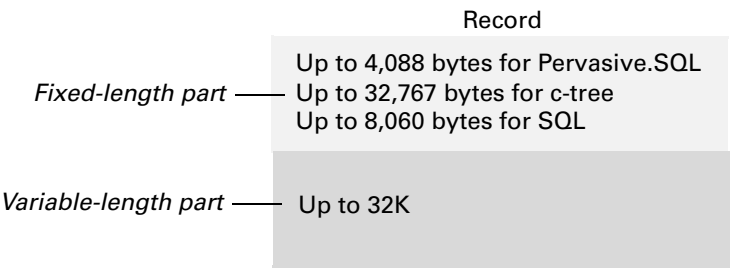
Before defining a table, you should have already used the Field Definition window to create the global fields that will be used for the table. To add global fields to the table, simply select the global field in the Global Fields list and click Insert; the field will be added to the table definition after the item currently selected in the Table Fields list. If no field is selected in the Table Fields list, the global field will be inserted at the top of the list.

The order in which you insert fields into the Table Fields list is not critical. There is only one significant consideration when inserting fields into a table definition. For optimal performance, it is desirable to group string fields next to each other in the table definition. This is particularly true for string fields that will be used in keys for the table. Through a process called segment combining, Dexterity treats adjacent string fields as a single item, allowing it to perform some table operations more efficiently.

The position of the fields in the Table Fields list is significant. Data in the table is stored according to the position of the fields in the Table Fields list. If you insert additional table fields or change the position of the fields, any existing data will need to be converted in order to be accessed properly. Data conversion is described in [Chapter 60](#) of this manual.

Record size

A record is made up of two parts: a fixed-length part and a variable-length part.



The fixed-length portion of the record stores all fields with fixed lengths. For Pervasive.SQL tables, the fixed-length portion can be up to 4,088 bytes (approximately 4K). For c-tree tables, it can be up to 32,767 bytes (32K). For SQL tables, the fixed-length portion can be up to 8,060 bytes. In all cases you should try to keep the fixed-length portion as small as possible for maximum performance.

The variable-length portion of the record stores all variable-length fields such as pictures and big text fields. These fields share 32K of variable space. For example, if a table stored a 10K picture and a big text field, the text field could store up to 22K of text for a total of 32K.

The record size of the fixed-length portion of the record is calculated automatically as fields are added or removed; the total record size equals the combined storage size of all the fixed-length fields that have been inserted into the table.

Encryption

You can encrypt fields by selecting them in the Table Fields list and marking the Encrypt option. An asterisk will appear next to each encrypted field. Encryption is used for fields such as password fields to prevent users from reading the passwords by viewing the table data with a file viewing utility.

Keys

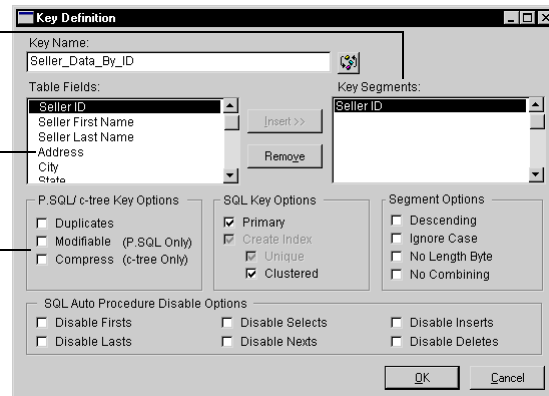
A key is a field or combination of fields in a table used to sort and locate specific records in the table. Typically, the value of the key field or fields is unique for each record in a table so a specific record can be located.

Using the Key Definition window, shown in the following illustration, you specify the fields that will make up the keys used to sort, store and retrieve records in the table you are creating. Each field in a key is referred to as a *key segment*. Dexterity allows you to create multiple keys for each table, with multiple segments in each key.

The Key Segments list displays the segments of the key in the order of sorting.

The Tables Fields list displays fields in the table.

Depending on the database type you specified, you can specify PSQL/c-tree or SQL key options.



Refer to [Appendix A, “Dexterity Limits.”](#) in Volume 2 of the Dexterity Programmer’s Guide for more information about limits to the number of keys and key segments that can be defined for a table.

For example, a customer table typically would have two keys defined for it. One key could contain the Customer Name field, because this is the field a user would most likely use to search for a record. However, a customer name may not be unique. For instance, there may be more than one “John Smith”. To avoid this problem, each customer could be assigned a unique customer ID. A second key composed of the Customer ID field could be added to the table. This key would uniquely identify customers, even if they have the same name.

The Key Segments list displays the list of fields that will be used as segments in the key. Records will be sorted primarily according to the first key segment. Records that are identical for the first key segment will be sorted again by the second key segment. For example, if the Key Segments list contains two segments: [last_name, first_name], in that order, records will be sorted primarily by the last name, and records with the same last name entry would be sorted again by the first name.

Key options

The database type you specified in the Table Definition window indicates which sets of key options you can specify for each key. If the database type for the table is c-tree or Pervasive.SQL, you can specify P.SQL/c-tree key options. If the database type is SQL, you can specify SQL key options. If the database type is Default, you can specify both sets of options.

P.SQL/c-tree key options

The following key options apply when you are using the Pervasive.SQL or c-tree database types.

Duplicates Specifies whether multiple records in the table can have the same key value. If this option is marked, new records that have the same key values as records already in the table can be saved. If the selection isn’t marked, new records with the same key values as existing records won’t be saved.

Modifiable If this option is marked, the values of key fields can be modified after a record has been saved. If this selection isn’t marked, and you attempt to change the value of a key segment and resave the record, an error will occur and the record won’t be saved. This selection applies only to the Pervasive.SQL database manager. The c-tree database manager always allows you to modify key segments of records.

Compress Marking this option allows the database manager to compress information for the key and save space in the index file. Currently, this is supported only by the c-tree database manager.

SQL key options

The following key options apply when the SQL database type is used.

Primary Marking this option indicates this is the primary key for the table. The primary key must ensure that each record in the table is uniquely identified, so by marking the Primary selection, the Create Index and Unique options are marked automatically. A table can have only one primary key.

Create Index Marking this option will create an index for the table based upon this key definition. The index contains a copy of the data in the key's key segments and pointers to the associated records in the table. An index allows data to be retrieved from the table more quickly.

If Create Index is marked, the following options are available:

- **Unique** Marking this option ensures that the records in this table will have unique values for this key. This option must be marked when the Primary option is marked. Also, the Duplicates option in the Btrieve/c-tree Key Options group can't be marked when Unique is marked.
- **Clustered** Marking this option sorts and physically stores the data in the table according to the order defined by this key. For example, the Customer table has a key based upon the Customer Name field. If this key was based on the Customer Name field, and the Clustered option were marked for the key, the data in the Customer table would be physically stored in the order indicated by the key. Only one key per table can have the Clustered option marked.

Segment options

The segment options specify how the key will handle the contents of the selected key segment.

Descending This option specifies the order in which the records will be sorted. If the selection is marked, records will be sorted in order of highest value to lowest (3, 2, 1 or c, b, a). If the selection isn't marked, records will be sorted lowest to highest (1, 2, 3 or a, b, c).

Ignore Case This selection is available only if the selected segment is a string field. The Ignore Case option specifies whether case determines how the table will be sorted. If the selection is marked, the case of the data in the key segment won't affect the sorting process. If the selection isn't marked, uppercase characters will have a lower value than lowercase characters for sorting purposes. This option does not apply for tables that use the SQL database type.

The following table shows how a set of last names would be sorted, depending upon how the Ignore Case option is marked.

Ignore Case marked	Ignore Case not marked
adams	Beaulieu
Beaulieu	JOHNSON
douglas	Jones
hagelle	Simpson
JOHNSON	adams
Jones	douglas
Simpson	hagelle

No Length Byte If a key segment has a string data type, Dexterity normally includes the length byte in the key. However, if your product integrates with an application that doesn't include length bytes in keys, mark this option. For example, *most* (but not all) Great Plains Accounting keys don't include the length byte. Be sure to verify whether the key you're working with uses a length byte. If you mark No Length Byte, No Combining will be marked automatically.

No Combining If there are two string segments next to each other in the key, Dexterity normally combines them into one. If your product integrates with an application that doesn't combine string segments (for example, Great Plains Accounting), mark this option. This option is available only for string segments.



We recommend that you use the No Length Byte and No Combining selections only if you're creating an application that integrates with Great Plains Accounting files. These selections are not intended for use in other Dexterity-based applications.

SQL Auto Procedure Disable Options

Refer to the section titled [Auto-generated stored procedures](#) in [Chapter 47, “SQL Tables,”](#) for more information about auto-generated stored procedures.

These options allow you to disable creation of the auto-generated stored procedures for specific table operations. In special circumstances, disabling the auto-generated stored procedure for a particular table operation can improve the performance of the application. Refer to [Auto-generated stored procedures](#) on page 391 for more information about auto-generated stored procedures.

Auto-generated stored procedures provide the greatest performance benefits in tables with more than 100 records. The benefit is less significant for small tables. Creating auto-generated stored procedures also adds to the time required to create a table. Based on these two factors, you may want to disable the creation of the auto-generated stored procedures if the table meets one of the following criteria:

- The table will contain a small number of records. The auto-generated stored procedures won't dramatically improve the performance, but will take up additional space in the database and increase the time required to create the table.
- The table is a temporary table. Typically, temporary tables are used once and then deleted. Creating the stored procedures increases the time required to create the temporary table, but they provide little benefit because most of them won't be used to access data before the table is deleted. This is especially true of small temporary tables and temporary tables that will be used for reports.

Table options

Several options can be specified for each table. These options control how data is stored in the table and how records can be accessed in the table.

Allow Active Locking Mark this option to allow active locking for this table. If you don't mark this option, any [change](#) or [edit table](#) statements that reference this table and include the **lock** keyword will produce errors when you attempt to compile them.



We strongly recommend that you avoid using active locking in your application. Application performance can degrade when active locking is used, especially when your application uses a SQL database.

Use 4 byte header Virtually all Dexterity applications use a four-byte header at the beginning of each table. However, Great Plains Accounting files do not. This option should *always* be marked, unless you are creating a product that integrates with Great Plains Accounting files.



Unmark the Use 4 byte header option only if you're creating an application that integrates with Great Plains Accounting files. This option is not intended for use with any other Dexterity-based applications.

Use Restriction Mark this option if you want to create a restriction that is always applied to the table. Rarely will you mark this option.

If you're creating a product that integrates with Great Plains Accounting files, and the file you're integrating with uses variant records, mark this selection. Before you mark this selection, create the table definition and save it, and then reopen it. Mark the Use Restriction option; the Default Restriction window will appear, allowing you to create a default restriction for the table.



We recommend that you don't mark the Use Restriction selection unless you're creating an application that integrates with Great Plains Accounting files. This selection is not intended for use with other Dexterity-based applications.

Use Row Timestamp Mark this option to add a field named DEX_ROW_TS to the table definition in the SQL database. This additional column stores the UTC date and time the record was created or updated. This option applies only when the SQL database type is used.

Table relationships

Table relationships link tables that have fields in common. These relationships allow the Dexterity Report Writer to select fields from all of the related tables and use them on a single report. For more information about creating table relationships, refer to [Chapter 33, "Table relationships."](#)

Temporary tables

To create a temporary table in Dexterity, create the table definitionl, but enter "temp" in the Physical Name field, as shown in this illustration.

Enter "temp" in the Physical Name field of the Table Definition window to create a temporary table.

Table Name:	Candidate_Buyers_List
Display Name:	Candidate Buyers List
Physical Name:	temp

Dexterity will create temporary tables when they are first accessed. A unique physical name will be assigned to each temporary table when it's created. If a specific temporary table is opened more than one time, each instance will have a unique physical name. Temporary tables will appear in the \WINDOWS\TEMP subdirectory for Windows, or in the *tempdb* database for SQL. Each table will be deleted automatically when it's closed.

Temporary tables can be either c-tree tables or SQL tables, depending upon the Database Type you specified in the table definition for the temporary table. If you specified c-tree or Pervasive.SQL, the temporary table will be a c-tree table. If you specified SQL, the temporary table will be a SQL table. If you specified Default, the temporary table will be a c-tree table, unless SQL has been specified as the default database type for the application.

Table locations

Refer to [Chapter 2, "Pathnames,"](#) in the *Stand-alone Application Guide* for information about specifying table locations.

After a table is defined and is opened the first time by your application, the actual physical table will be created automatically. Depending upon the database type you specified, the table will be an operating system file in the same location as your dictionary, or a table in the default database on the SQL server. If you're integrating with Microsoft Dynamics GP, the location will be determined by the series the table is associated with.

Table groups

Some tables in your application may have a logical relationship to each other. For example, invoice information is often divided into two tables. One table contains the header information for each invoice, while the other table contains the line items for the invoice. These two tables have an inherent relationship; each table requires the other.

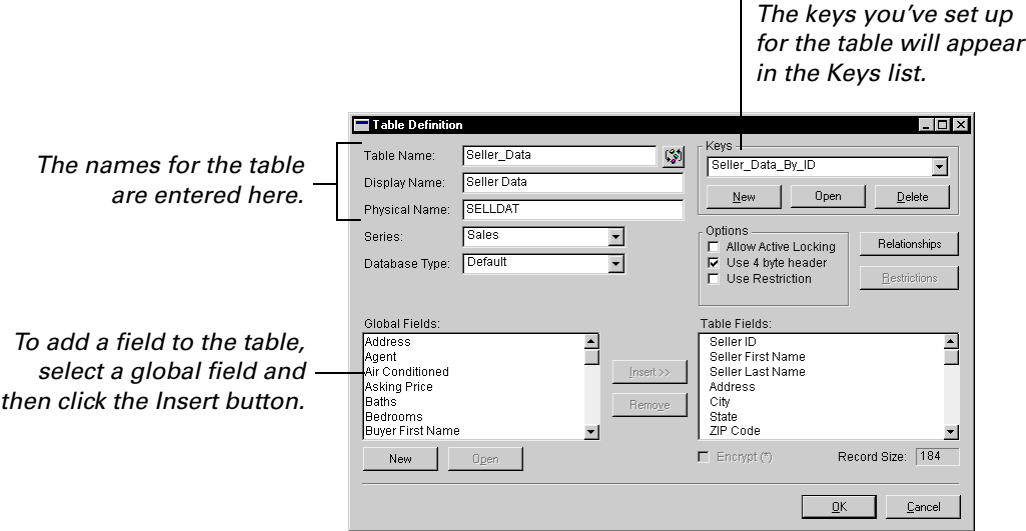
Often it is useful to work with related tables as a group. For example, when you use security in your application to set access for a table, you will want to set the same security access for all related tables. Dexterity allows you to create a *table group*. A table group allows you to group logically related tables under a single name. Refer to [Chapter 26, "Table Groups,"](#) for information about creating table groups.

Procedure: Creating tables

To define a new table, point to new in the Explorer menu and choose Table. The Table Definition window will appear.

1. Name the table.

Three names are required for the table, as shown in the following illustration.



The name entered in the Table Name field is the *table name* that is used in scripts. The *display name* is how the table name will be displayed on the screen in the application. The *physical name* is the name under which the table is stored by the operating system or database. By default, this name can be up to eight characters long.

If you're developing an application for an operating system that supports long file names, and you've marked the Allow Long Physical Names option in the Options window, you can enter table physical names up to 28 characters long.

2. Select the table series.

The table series allows you to group related tables in a Dexterity application together using series categories like Sales, Financial and System. You will use these series groupings when you use the Report Writer, set up pathnames, and perform table maintenance routines.

3. Select the database type.

The database types specifies how Dexterity will store the table. You can choose memory-based tables, which act just like c-tree tables, with the exception that they are stored in memory, rather than written to a physical disk.

Refer to the section titled [Table elements](#) on page 100 for more information about setting the default database type.

If you are unsure which database type to use, or your application will be used on different platforms, choose Default; your application will use the default database type. The default database type is specified by the **DatabaseType** defaults file setting, the [Table SetDBType\(\)](#) function, or the command line option included when Dexterity or the runtime engine is started.

4. Insert the fields that will be stored in the table.

In the Global Fields list, highlight each field you want to include in the table and click Insert. The field names will be displayed in the Table Fields list in the order they will be saved in the table you're defining. The order isn't critical, but to maximize performance, group string fields next to each other in the Table Fields list.

5. Define table keys.

A table key is the field or group of fields that are used to sort and locate specific records in the table. The table keys are always composed of fields from the table. Each table must have at least one key and can have several keys. The fields in the keys specify how you want to sort and retrieve information.

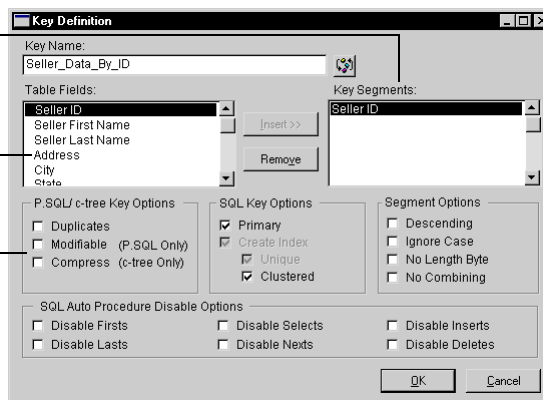
For example, if a table contains the fields Name, Address, City, State and ZIP Code, one table key could be Name. Using this key, a specific address could be retrieved by name. Another key could be the ZIP Code. This key would allow you to retrieve records from the table based upon the ZIP Code.

To create a key, click New in the Table Definition window. The Key Definition window will open, as shown in the following illustration:

The Key Segments list displays the segments of the key in the order of sorting.

The Tables Fields list displays fields in the table.

Depending on the database type you specified, you can specify P.SQL/c-tree or SQL key options.



Enter the name of the key. This name will be used in scripts, so you may want to substitute underscores for spaces.

In the Table Fields list, highlight each field you want to include in the key and click Insert. The key segment will be inserted after the item currently selected in the Key Segments list. If no item is selected in the Key Segments list, the new segment will be inserted as the first item.

If the key has more than one segment, insert the key segments in the order you want the information in the table to be sorted. Records will be sorted primarily by the first key segment. Records that are identical for the first key segment will be sorted again by the second key segment, and so on.

Select the options for the key. The key options you can set depend upon the database type you specified in the Table Definition window. If the database type for the table is c-tree or Pervasive.SQL, you can specify P.SQL/c-tree key options. If the database type is SQL, you can specify SQL key options. If the database type is Default, you can specify both sets of options. Refer to [Key options](#) on page 106 for a detailed description of the key options you can specify.

If necessary, select options for key segments. The following table lists the common segment options.

Option	Description
Descending	Sorts records in descending rather than ascending order for this segment.
Ignore Case	Sorts records without regard to case.



Use the No Length Byte and No Combining options only if you're creating an application that integrates with Great Plains Accounting files. These options are not intended for use in other Dexterity-based applications.

Click OK to save the key. The Table Definition window will re-appear.

Active locking is described in [Chapter 46, "Locking,"](#) of Volume 1 of the *Dexterity Programmer's Guide*.

6. Specify options for the table.

Typically, the only option you specify for the table is whether to allow active locking. Mark the Allow Active Locking option in the Table Definition window to allow active locking. If you don't mark this option, any [change](#) or [edit table](#) statements that reference this table and include the **lock** keyword will produce errors when you attempt to compile them.



We strongly recommend that you avoid using active locking in your application. Application performance can degrade when active locking is used, especially when your application uses a SQL database.

Changing the default settings of the Use 4 byte header and Use Restriction options is necessary only if you're integrating with Great Plains Accounting files. We strongly recommend that you don't unmark the Use 4 byte header option or mark the Use Restriction option for your Dexterity-based application.

7. Define any table relationships.

Table relationships are created between tables that have fields in common. These relationships allow the Dexterity Report Writer to select fields from all of the related tables and use them on a single report. For more information about creating table relationships, refer to [Chapter 33, "Table relationships."](#)

8. Encrypt fields (optional).

Encryption is used for fields such as password fields to prevent users from viewing passwords by looking at the table with a file editing utility. To encrypt a field, select it in the Table Fields list and mark the Encrypt option.

Click OK to save the table definition and close the Table Definition window.

Chapter 10: Forms

Dexterity allows you to define your application's interface through the use of forms and windows. Understanding how to use forms and windows will help you design an intuitive, logical interface, allowing users to easily enter, view and change information.

Information about forms is divided into the following sections:

- [*Form overview*](#)
- [*Form elements*](#)
- [*Procedure: Creating forms*](#)

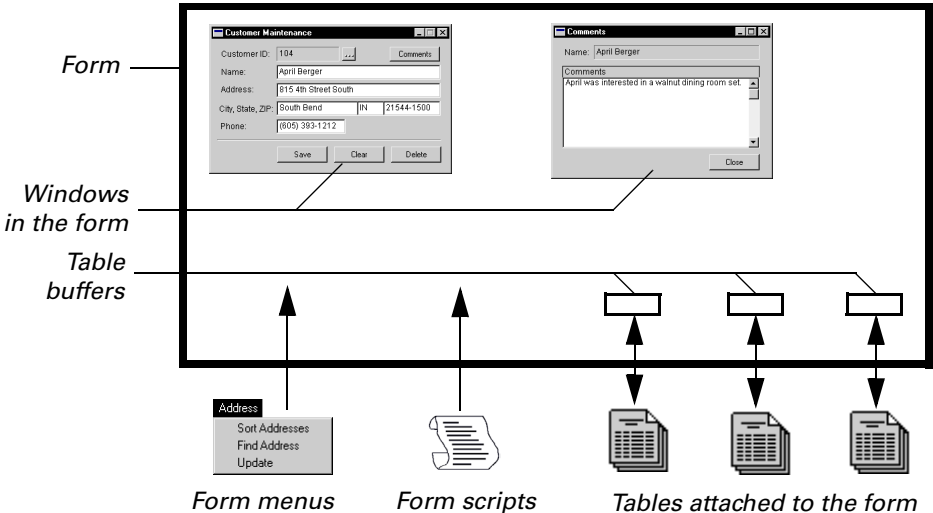
Form overview

After you create data types, fields and tables, you can begin setting up your application interface by first defining the forms and then the windows your application will use. Although forms and windows are stored as separate resources in the Dexterity development system, both work together to give structure to the tasks you’re designing your application to complete.

Windows are discussed in the next chapter. Menus are discussed in [Chapter 14, “Form-based Menus.”](#) Table buffers are explained in Volume 2 of the Dexterity Programmer’s Guide.

A form groups related objects and resources, such as windows, menus, scripts and tables, that function together for a common purpose. A single form may have several windows that can be displayed and used separately, but together all of the windows in the form function for a specific purpose. For example, a form could use one window to store information about customer addresses and another to store your comments for the customer. Together, both windows allow you to maintain customer records. The information entered in the windows is stored in tables attached to the form.

The overall structure of the form is shown in the following illustration.

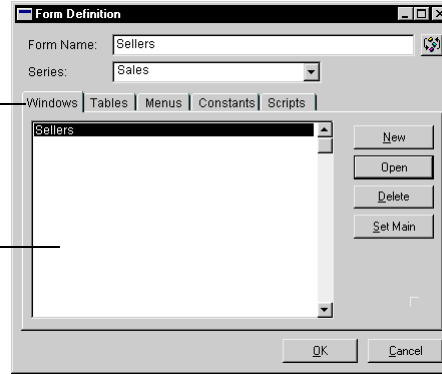


Form elements

Each form in an application can have a form name, form series, attached tables, windows, menus, constants and form scripts. The Form Definition window, shown in the following illustration, is used to create forms.

These tabs allow you to see which windows, tables, menus, constants and scripts are part of the form.

When the Windows tab is selected, the windows associated with the form are listed here.



Before you create forms, outline the goals of the form – which windows will the form use, which tables will be accessed by the windows in the form, what additional menus could help users complete tasks in the form, and what scripts will need to run for the form to operate properly in your application. For example, if your goal is to create a form to maintain customer records, plan how you want users to complete tasks for entering and maintaining customer records.

Name

Forms can be given any name, but if your application is stand-alone, there are two specially named forms that should be part of your application.

- The form named Main Menu is the form that automatically opens when your application is launched. It is used by the runtime engine as a “base” that your application dictionary can use when it starts up for the first time. A main menu form is created automatically when you create a new dictionary.
- The form named About Box will be opened if the user selects the “About” menu item from within the running application.



If your application will integrate with Microsoft Dynamics GP, you won't need to create the Main Menu or About Box forms because Microsoft Dynamics GP already has them.

Series

When you create a form, you must specify the series the form is part of. The form series allows you to group related forms in a Dexterity application together using categories like Sales, Financial and System. For example, you could assign all of the forms used in an Inventory application to the Inventory series. You will use these series groupings when you use the Modifier or use Dexterity Utilities to perform dictionary maintenance activities.

Attached tables

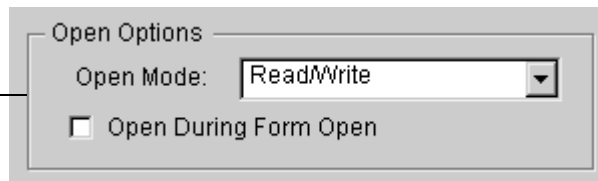
Tables must be attached to a form so the windows associated with the form will have access to the information in the tables. You must attach the tables to the form before you can reference the tables in scripts.

To attach tables to a form, click the Tables tab in the Form Definition window. The Form Definition window will list the tables attached to the form and allow you to attach additional tables.

Click Attach to open the Table Lookup window. Select a table to attach and click OK; the table will be attached to the form. To detach the table from the form, select the table in the list of tables in the Form Definition window and click Detach.

Clicking Options opens the Table Options window, which allows you to specify how the form will access a particular table. The form can access the table in read-only, read/write, or exclusive use modes. You also can choose whether the table will be opened the first time it's accessed by a script or when the form is opened. If the table is a SQL table, the cursor block size can also be specified.

These are some of the options you can set regarding how the form accesses the table.



Windows, menus, constants, commands and scripts

Each form can have windows, menus, constants and scripts associated with it. Adding windows to forms is discussed in [Chapter 11](#), form-based menus are discussed in [Chapter 14](#), and constants are described in [Chapter 21](#) of this manual. Commands are discussed in [Chapter 15](#). Form scripts, which allow you to add functionality when a form opens or closes, are discussed in [Chapter 3](#) of Volume 2 of the Dexterity Programmer's Guide.

Procedure: Creating forms

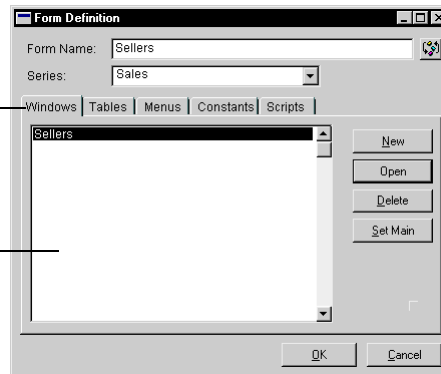
To create a form, point to New in the Explorer menu and choose Form.

1. Name the form.

Enter the name of the form in the Name field of the Form Definition window as shown in the following illustration. Since the form name will be used in scripts, you may want to substitute underscores for spaces in the name.

These tabs allow you to see which windows, tables, menus, constants and scripts are part of the form.

When the Windows tab is selected, the windows associated with the form are listed here.



2. Select a form series.

The form series provides a method of grouping related forms of a Dexterity application together under general headings, such as Sales, Financial and System.

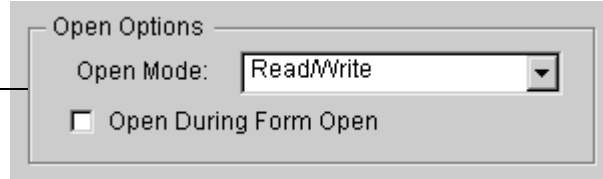
3. Attach any tables to the form and specify table options.

Tables must be attached to the form so the windows associated with the form will have access to the tables. Click the Tables tab in the Form Definition window to display the list of tables attached to the form. To attach a table to the form, click Attach. The Table Lookup window will appear, allowing you to select a table. Select a table and click OK. Repeat this process until all the necessary tables are attached.

To select options for a table, select the table in the Form Definition window and click Options. The Table Options window will appear.

The Table Options window allows you to specify how the form will access a particular table. The form can access the table in read-only, read/write, or exclusive use modes. You also can choose whether the table will be opened the first time it's accessed by a script or when the form is opened.

These are some of the options you can set regarding how the form accesses the table.



The screenshot shows a dialog box titled "Open Options". Inside, there is a label "Open Mode:" followed by a dropdown menu currently displaying "Read/Write". Below this is a checkbox labeled "Open During Form Open", which is currently unchecked.

The following table options are available:

Table access modes	
Read Only	The table can only be read.
Read/Write	The table can be read and written to.
Exclusive Use	When this form is accessing the table, no other forms or scripts can access the table.

Mark the Open During Form Open selection to open this table when the form opens, rather than when the table is first accessed by a script. You can use this selection when some of the tables are set for exclusive use, because then the form can't be opened unless all the exclusive use tables can be opened.

If the table is a SQL table, the cursor block size for the table can also be specified. Refer to [Chapter 45, "Cursors,"](#) in this manual for additional information on setting the cursor block size.

Click OK to save the displayed table options and return to the Form Definition window.

4. Create any form menus.

You can add menus or menu items to the menu bar at the top of the screen. These menus will appear when the form opens. Refer to [Chapter 14](#) for information about using menus.

5. Create windows for the form.

To create windows for the form, click the Windows tab in the Form Definition window. Click the New button; the layout window will appear, allowing you to create a window. Refer to [Chapter 11, “Windows,”](#) for more information about creating windows.

6. Specify the main window.

At runtime, when the window designated as the main window is closed, the form and any other windows associated with the form will close. By default, the first window in the list is the main window for the form. To select a different main window for the form, be sure the Windows tab is selected in the Form Definition window. Select the window you want as the main window and click Set Main; the window will be move to the top of the list, indicating it is the main window.

7. Add scripts to the form.

Click the Scripts tab in the Form Definition window to add scripts to the form. You can add scripts that run when a form opens or closes. These will be discussed in [Chapter 3, “Attaching Scripts,”](#) of Volume 2 of the Dexterity Programmer’s Guide. You can also add form-level procedures and form-level user-defined functions. These are described in [Chapter 20](#) and [Chapter 21](#) of Volume 2 of the Dexterity Programmer’s Guide, respectively.

8. Save the form definition.

Click OK to save the form definition.

Related items

Use the information listed to learn more about creating and using forms.

Additional information

[Chapter 11, “Windows,”](#) in Volume 1 of the Dexterity Programmer’s Guide.

[Chapter 14, “Form-based Menus,”](#) in Volume 1 of the Dexterity Programmer’s Guide.

[Appendix A, “Naming Conventions,”](#) in the Integration Guide.

Chapter 11: Windows

After you create a form, you can create the windows that will be part of that form. A window is the main method of communication between the user and your Dexterity application. It consists of the visual area displayed on the screen that contains editable fields, interactive controls and static items such as text or pictures.

When you display the Form Definition window, all of the windows in that form will be listed when you select the Windows tab. When you add a window to the form, it will be added to the top of the list if no window is selected, or directly below the window selected in the list. The first window in the list is considered the main window. If a user closes the main window in the application, the entire form will be closed. Windows are specific to a single form, so a single window can't be used by several forms.

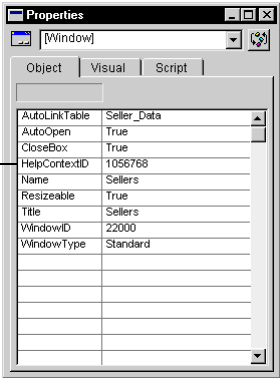
Information about creating windows is divided into the following sections:

- [*Window elements*](#)
- [*Creating a window layout*](#)
- [*The layout area*](#)
- [*The Toolbox*](#)
- [*The layout grid*](#)
- [*Adding fields to a window layout*](#)
- [*The Properties window*](#)
- [*Window properties*](#)
- [*Field properties*](#)
- [*Drawn object properties*](#)
- [*Previewing a window*](#)
- [*Tab sequence*](#)
- [*Adding pictures*](#)
- [*Linking fields to prompts*](#)
- [*Positioning the window*](#)
- [*Resizing windows*](#)
- [*Setting required fields*](#)
- [*System color support - automatic*](#)
- [*System color support - customizable*](#)
- [*Custom color support*](#)
- [*Procedure: Creating windows*](#)

Window elements

Each window in your application is composed of the window’s names, window type, scripts attached to the window, and the window layout. You will use the layout window and toolbox to create the window. You will use the Properties window to specify additional characteristics of the window. The Properties window is shown in the following illustration.

Use the Properties window to set properties for the window.



Names

Two names are required for the window. The window *name* is used in scripts to refer to the window. Window names are easier to use in scripts when they have underscores (_) between parts of the name instead of spaces.

The window *title* appears in the title bar of the window when it’s displayed on the screen. This name can be the same as the window name, but typically should contain spaces rather than underscores.

When you create a window, the window name and title are set to default values. You will want to change these settings to names appropriate for the window you are creating. The following names are for a window containing customer information:

Name	Customer_Maintenance
Title	Customer Maintenance

Window type

The type of window you create is specified by the `WindowType` property in the Properties window. Dexterity allows you to choose from eight window types: primary, modal dialog, modeless dialog, lookup, wizard, palette, toolbar, and internal. The following are the commonly used window types:

- *Primary windows* are the most common type of window and appear with all operating system controls, such as close boxes and minimize controls.
- *Modal dialogs* don't contain any operating system controls and can't be resized. These windows can be used when you require the user to complete an action in the window before closing the window.
- *Palette windows* can be used as a device for navigation throughout your application. Palette windows are similar to the tear-off menus found in other applications. Palettes are used to group windows or tasks that perform similar functions, such as windows you use to maintain inventory items.
- *Toolbar windows* are displayed across the top of the screen, expanding automatically to fit the size of the screen. Only one toolbar window can be open at a time. A toolbar window can't be closed and always appears in front of other windows at runtime.
- *Internal windows* are those that are used only for storing application state information. They are never intended to be displayed to the user. When a window has the type `internal`, the Dexterity runtime can optimize how the window is processed.

The following table lists the characteristics of the various types of windows:

	System menu	Title	Minimize/Maximize	Close box
Primary	X	X	X	X*
Modal Dialog		X	X	
Modeless Dialog	X	X	X	X*
Lookup		X		X*
Palette				X*
Wizard	X	X	X	

* Controlled by a property

Scripts

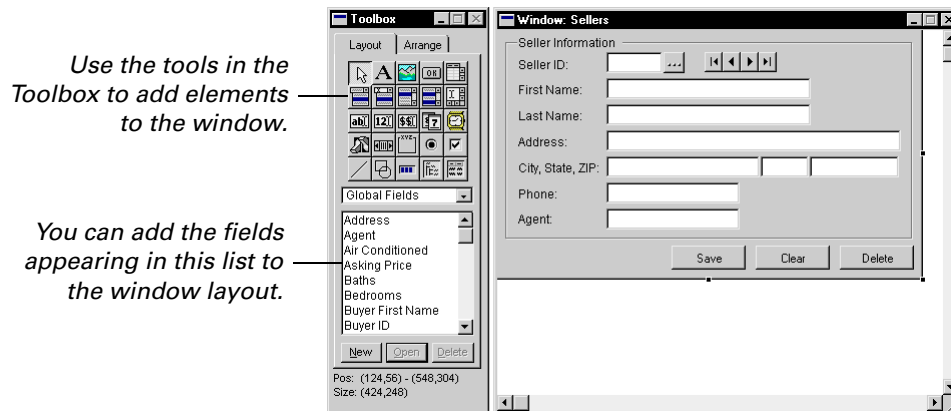
Each window can have several scripts associated with it. The window pre script and window post script run when the window opens and closes, respectively. Refer to [Chapter 3, “Attaching Scripts,”](#) in Volume 2 of the Dexterity Programmer’s Guide for more information about window scripts.

The window layout

You will use the window layout to add fields, interactive controls, and static items such as text and pictures to the window. These are the items that make the window functional.

Creating a window layout

When you create a new window or choose to open an existing window, the Toolbox and Layout window will appear, as shown in the following illustration.









The layout window and toolbox use a graphics metaphor you may already be familiar with: an assortment of tools you can use for drawing and creating text objects; a built-in grid that allows easy alignment of objects on the window; moveable objects in a layout like object-oriented drawing programs; and standard cut-copy-and-paste techniques so you can import graphics you create in other applications.






















Windows are saved only when the layout window is closed. Save your work often by closing the layout window and then reopening it to continue designing the window.

The Toolbox












To add and manipulate objects in the layout window, use tools from the Toolbox. Dexterity has the following tools available:

Tool	Name	Use
	Arrow tool	Select, resize and move objects in the window layout. Objects highlighted by the arrow tool can be dragged into position in the layout area.
	Text tool	Add text objects to the layout area. Select the text tool, then click in the appropriate location to type text in the layout.
	Picture tool	Place pictures from the picture library in the layout area.
	Scrolling window tool	Adds a scrolling window to the window layout.
	Line tool	Add lines to the layout area.
	Shape tool	Add rectangles, circles and other shapes to the layout area.

The toolbox also contains tools used for prototyping an application. These tools allow you to easily add local fields to a window layout.

Tool	Name	Tool	Name
	Push button tool		Drop-down list tool
	Combo box tool		List box tool
	Multi-select list tool		Big text tool
	String tool		Integer tool
	Currency tool		Date tool
	Time tool		Visual switch tool
	Horizontal list box tool		Radio group tool
	Radio button tool		Check box tool
	Progress indicator tool		TreeView tool
	List View tool		

Clicking the Arrange tab in the Toolbox displays an additional set of tools used to align, resize or tile fields in the window. These tools are described in the following table.

Category	Tool	Name	Use
Align		Align to Top	Aligns the selected objects with the top object in the group.
		Align to Left	Aligns the selected objects with the leftmost object in the group.
		Align to Right	Aligns the selected objects with the rightmost object in the group.
		Align to Bottom	Aligns the selected objects with the bottom object in the group.
Size		Size to Shortest	Shrinks the selected objects to the height of the shortest object in the group.
		Size to Narrowest	Shrinks the selected objects to the width of the narrowest object in the group.
		Size to Widest	Enlarges the selected objects to the width of the widest object in the group.
		Size to Tallest	Enlarges the selected objects to the height of the tallest object in the group.
		Size to Default	Resizes the field to its default size, based upon how the visual properties for the field are set.
Tile		Tile Horizontally	Tiles the selected objects horizontally. The value in the Space field specifies the space between objects.
		Tile Vertically	Tiles the selected objects vertically. The value in the Space field specifies the space between objects.



Arranging objects can't be undone. Be sure to save your window layout before arranging objects. If you aren't satisfied with the result of an arrangement, you can close the window without saving the changes.



The position and size coordinates for the currently-selected object are displayed at the bottom of the Toolbox. Use these values to accurately position and size objects.

The layout area

The layout area is where you add fields, controls, graphics and text used by the window. This area represents the size of the window as it will appear at runtime, exclusive of native operating system controls such as title and scroll bars. To change the size of the window, drag the resize handles to adjust the width and height of the window.

The layout area represents the size of the window as it will appear at runtime.

The window background color is controlled by the BackColor property.

The resize handles allow you to change the dimensions of the window.

Controls that are specific to a particular operating systems, such as scroll bars, resize boxes, minimize controls and title bars, don't appear in the layout window. Instead, operating system elements are created automatically by the runtime engine.

Remember that items appearing in the Dexterity layout window can look somewhat different than how they will appear at runtime. At runtime, all objects added to the layout area will appear in the window drawn properly.

Use the Show Field Names and Show Invisible Fields items in the Layout menu to specify how fields will be displayed in the layout window. If you mark the Show Field Names menu item, fields will show their field names, rather than their normal display characteristics. If you choose not to show field names, fields will display much like they do at runtime. The following table shows how fields appear in the layout and at runtime:

Showing names	Not showing names	At runtime

If you mark the Show Invisible Fields item in the Layout menu, those fields whose Visible property is set to false will be displayed in the layout.

Each object in the window layout can be selected with the arrow tool and then dragged to a different position in the layout area. You also can move selected graphics, text or fields in front of or behind other objects by choosing Send To Front or Send To Back from the Layout menu.

The layout grid

The layout area has a grid that's activated automatically when you open the layout window. When the grid is active, a check mark appears next to the Grid item in the Layout menu. Objects you add to the layout area will be aligned to the grid automatically, but existing objects in the layout area won't be aligned to the grid. To align existing objects to the grid, be sure the grid is active, and then select the objects with the arrow tool and choose Align To Grid from the Layout menu.

It's a good idea to use the layout grid when adding or rearranging items in the layout area, so items will be aligned automatically. You specify the size of the grid blocks using the Options window (accessed from the Edit menu). If you want to display the grid in the layout area, choose Show Grid from the Layout menu. To display the grid behind objects in the layout window, choose Grid to Back. To deactivate the grid, choose Grid from the menu to remove the check mark.



You can use the arrow keys to move selected items in the window one pixel at a time, regardless of whether the grid is turned on.

Adding fields to a window layout

Data entry fields and other window controls, such as push buttons and check boxes, can be added to a window by selecting the field from the fields list in the Toolbox and dragging it to the appropriate area of the window. Fields added to a layout area are *window fields*. Their appearance can be modified using field properties, which are discussed later in this chapter.

There are two types of fields that can be added to a window layout: *global fields* and *local fields*.

- A global field is any field that's already been created in the Field Definition window in your application dictionary.
- A local field is a field created for use with windows in the current form. Local fields can't be stored in tables, but you can set their values to update table fields.

Adding global fields

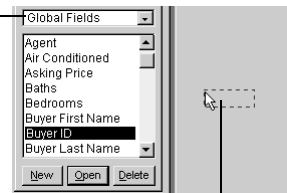
You add global fields to a layout by dragging them from the list of fields in the Toolbox. To display all of the global fields defined for the application, choose Global Fields in the drop-down list in the Toolbox. This list will then contain all global fields created using the Field Definition window.

You can also view a list of the global fields that are part of a table attached to the current form. To do this, use the drop-down list in the Toolbox and select the table whose fields you want to list. The fields list in the Toolbox will then display only the fields that are part of that table. To add a global field to the window, complete the following steps.

1. Drag the field to the window.

Select the appropriate field name and drag it to the layout area. You can choose any field from the fields list, or you can use the drop-down list above the fields list to view all global fields or only those from tables linked to the current form:

Use this drop-down list to view all global fields or just those for a particular table attached to the current form.



The field you drag to the layout area automatically will appear with the length you've set up in the field's data type.

The size of the box displayed in the window layout is the default size based on the field's data type and the current display properties. For instance, if you add a check box to a window layout, the check box will be large enough to display the static text value you applied to the check box data type, in the current font and style.



Fields that appear in tables linked to the form are the same fields as those in the list of global fields. Listing the fields associated with a specific table is simply a shortcut, allowing you to view a smaller list of fields.

2. Set properties for the field.

Setting field properties is described in the next section.

Adding local fields

Local fields are specific to a single form, and all windows from the form will have access to the local fields. They can be used only as window fields. Local fields can't be stored in a table. Typically, controls such as push buttons are created as local fields, since push buttons aren't stored in a table. To add a local field to the window, perform the following steps:

1. Create a new local field.

In the Toolbox, choose Local Fields in the drop-down list specifying the type of fields to display.

Choose Local Fields in this drop-down list.

Use the arrow tool to select the field, then drag it to the layout area.

Click Open to change a selected local field. Click New to create a new local field.

Click New; the Local Field Definition window will appear.

Specify the name of the local field here.

A local field can use its own local data type, or it can use a global data type.

2. Name the field and set the field characteristics.

Name the field and select the characteristics for the local field. A local field can use a data type created with the Data Type Definition window or a local data type.

To use a local data type, mark the Local radio button and select the control type, keyable length and static values for the field. To use a global data type, mark the Global radio button and use the Data Type lookup button to open a list of global data types defined for the application. Select a data type and click OK. You can also choose to create a new global data type.



In most cases, local data types are used for local fields.

After you've set up the field, click OK to save the changes. The local field will appear in the fields list in the Toolbox.

3. Drag the field to the layout area.

Select the appropriate local field name and drag it into the layout area. The size of the box is the default size based upon the field's data type and current display properties.

4. Set properties for the field.

Setting field properties is described in the next section.

Refer to [The Toolbox](#) on page 129 for a list of the prototyping tools.

You can also use the prototyping tools from the Toolbox to add local fields to the window layout. To use a prototyping tool, simply select it in the Toolbox and click in the layout area where you want the local field to appear. A local field will be created and placed in the layout window where you indicated.

The default name of each local field is based upon its control type and the order in which it was placed on the window. For example, the local field PushButton2 field was the third local push button field placed in the window. The numbering starts at zero.



You may want to edit the local fields you created with the prototyping tools to change the names and add static items. To do this, choose Local Fields in the drop-down list in the Toolbox, select the local field, and click Open.

The Properties window

The Properties window is used to set several characteristics of the window and the objects in the window. To display the properties for the window, select Properties from the Layout menu. If it isn't already open, the Properties window will appear.

Select an object in the layout area and click the tab indicating which type of property you want to view. Select from Object, Visual, or Script. To set a property, select it in the list and then change its value in the settings box.

Click a tab to display the corresponding list of properties.

To set the value of a property, select it in the list. Then set its value in the settings box.

Properties

Buyer ID

Object | Visual | Script

DataType

Buyer_ID

...

AutoCopy

True

DataType

Buyer_ID

DisableLookup

False

Editable

True

Field

Buyer ID

FieldID

22024

FieldType

Global

HelpContextID

25

LinkedFormat

LinkedLookup

Lookup Button

LinkedPrompt

Buyer ID:

Password

False

Required

True

SavedOnRestart

False

SetChangeFlag

True

TabStop

True

Settings box

Setting the property value involves choosing a value from a drop-down list, typing a value, or using a lookup. Some properties listed can't have their values changed.

Property type	Description
<div>Editable</div> <div>True</div>	Some properties are set using a drop-down list.
<div>Size-Height</div> <div>21</div>	Some properties have the value entered directly.
<div>DataType</div> <div>Buyer_ID</div> <div>...</div>	Some properties use a lookup to retrieve values.
<div>FieldID</div>	Some properties can't be set.



Double-clicking a property in the properties list is a shortcut to setting its value. For instance, double-clicking a property whose value is set with a drop-down list will set the property to the next value in the list.

Window properties

*Script properties are described in [Chapter 3, “Attaching Scripts,”](#) in Volume 2 of the *Dexterity Programmer’s Guide*.*

*Scrolling window properties are described in [Chapter 10, “Scrolling Windows,”](#) of Volume 2 of the *Dexterity Programmer’s Guide*.*

To view window properties, be sure the Properties window is open. Select the arrow tool from the Toolbox and click in the layout window outside of the window’s area to select the window. The window is selected when resize handles appear on its perimeter. The following table lists the window properties.

Object property	Description
AutoLinkTable	Indicates the table auto-linked to the window. Specify an auto-linked table if you want users of your application to be able to use the Modifier to add fields from the auto-linked table to the window. The best table to auto-link to a window is the source or destination of most of the information displayed in the window. The information from the auto-linked table should be copied to the window fields using the copy from table statement. It should be copied from the windows to the auto-linked table using the copy to table statement. Otherwise, when Modifier users drag fields from the auto-linked table to the window, the contents of the fields won’t be updated or saved.
AutoOpen	If set to true, the window opens automatically when the form opens. By default, this value is set to true.
CloseBox	If set to true, the close box on the window will be active.
HelpContextID	Lists the help context ID value for the window.
Name	The name of the window. This name is used to refer to the window in scripts.
Resizable	If set to false, the window cannot be resized by the user. If set to true-automatic, the window can be resized and the appropriate fields will be resized automatically. If set to true-per field, the window can be resized, but the resize characteristics must be specified for each individual field.
Title	The name that is displayed in the title bar of the window at runtime.
WindowID	Lists the resource ID of the window.
WindowType	Indicates what type the window is. It can be one of the following: Primary, Modal Dialog, Modeless Dialog, Lookup, Palette, Toolbar, Wizard, or Internal.

Visual property	Description
BackColor	If set to true, sets the background area of the window to light gray. If set to false, the background of the window will be white.
ControlArea	If set to true, a dark gray band called the control area will be displayed across the top of the window. Often, push buttons are placed in the control area.
Position-Left	Indicates the position of the left edge of the window, measured in pixels from the left edge of the screen.
Position-Top	Indicates the position of the top edge of the window, measured in pixels from the bottom edge of the toolbar. For toolbar windows, the position is relative to the bottom edge of the menu bar.
Size-Height	Indicates the window height, measured in pixels.
Size-Width	Indicates the window width, measured in pixels.

Field properties

The field properties control the characteristics of the fields in the window. If you are using the Themes service provided by the operating system, be aware that this service will override some of the properties you can set for a field. The following table lists the field properties.

Object property	Description
AutoComplete	For string fields, specifies whether the field will use the auto-complete capability. If this property is set to true, the contents of the field will automatically be completed as the user types, based on a list of values associated with the field.
AutoCopy	Specifies whether information can be automatically transferred between the field and table buffer when the copy to table , copy from table to window , copy from window to table , or copy to table script statements are used. If set to false, these statements don't affect the field.
Cancel	If set to true, the change script attached to this field is run when the user presses ESC while the window containing the field is active. This property is typically set to true for a Cancel push button field. This property is available only for push button fields.
DataType	Indicates the data type used for the field.
Default	If this property is set to true, pressing the ENTER key or double-clicking a list box or scrolling window with the DefaultDbClick property set to true causes the push button's change script to run. This property is typically set to true for an OK or Save push button field. This property is available only for push buttons.
DefaultDbClick	Setting this property to true allows the user to double-click a list box field to accomplish the action of selecting an item and clicking the push button whose Default property is set to true.

Object property	Description
DisableLookup	If set to true, prevents a lookup form associated with the field from opening when a user chooses the Lookup menu item.
Editable	Specifies whether the user can edit the contents of the field.
EditMode	For text fields, this property specifies how text fields can be edited. If set to DisplayOnly, the text field can't be edited. If set to Editable, the field can be edited. If set to SelectOnly, text can be selected, but not changed.
EndTransaction	When set to true, this property causes all pending change scripts to be run. This property is available only for push button fields.
Field	Indicates the field being used and allows you to edit characteristics of the field definition.
FieldID	Indicates the resource ID of the field.
FieldType	Indicates whether the field is a global field or a local field.
HelpContextID	Lists the help context ID value for the field.
Hyperspace	When set to true, the push button can be clicked without running the change script or post script for the currently-focused field. Only the change script for the push button will be run. Typically, this property is set to true for Cancel buttons. This property is available only for push button fields.
LinkedFormat	Lists the format field linked to the current field.
LinkedLookup	Lists the lookup field (typically a push button) linked to the current field.
LinkedPrompt	List the prompt linked to the field.
Password	If set to true for string fields, the field will display Xs instead of the data entered in the field. This is typically used for password fields.
Required	Specifies whether the field will be checked by the required() function. Set this property to true when you want to be sure the user enters a value in this field. Also, if this property is set to true, and the Show Required Fields menu item is selected, any prompt linked to the field will be displayed according to the color and style specified by the Field SetRequiredFormat() function.
SavedOnRestart	If set to true, causes the field to keep the value that was last entered in it, when the form or window is restarted with the restart form or restart window statements. If set to false, a restart will clear the field.
ScrollBars	If set to true, a text field will have scroll bars. If set to false, no scroll bars will be displayed.
SetChangeFlag	Specifies whether changes to a field's value will set the change flag for the form or window. If set to true, the change flag will be set when the contents of the field change. If set to false, the change flag won't be set.
TabStop	Specifies whether to include a window field in the tab sequence. If set to true, the window field will be in the tab sequence.
Tooltip	Specifies the tooltip that will be displayed for the field.

Object property	Description
UseUniqueHelp	If set to true, a unique HelpContextID will be generated for this instance of the field. If set to false, the same HelpContextID will be generated for all instances of this field in the application.
Visible	If set to true, the field will be visible. If set to false, the field will be hidden.
VisibleItems	For drop-down list and combo box fields, specifies the maximum number of items displayed when the list is displayed in the dropped position.
WordWrap	For text fields, specifies whether text will wrap.

Visual property	Description
Alignment	Specifies whether the item is left-, center- or right-aligned.
Appearance	For edit fields, specifies what type of border the field will have. For push buttons, button drop lists and push button-style check boxes, setting this property to 3D Highlight causes the button to have a flat appearance until the mouse pointer passes over the button.
BackColor	Specifies the background color of the field.
Border	If set to true, a border is drawn around the field.
Direction	For progress indicator fields, indicates the direction the bar will move in the progress indicator.
DropIndicator	For button drop list fields, specifies whether the drop indicator will be displayed.
DropPosX	For button drop list fields, specifies the distance in pixels the drop indicator will be drawn from the right edge of the control.
DropPosY	For button drop list fields, specifies the distance in pixels the drop indicator will be drawn from the bottom edge of the control.
Font	Specifies the font to use for the field.
FontColor	Specifies the color of the text for the field.
FontBold	If set to true, the field text will be displayed in bold type.
FontItalic	If set to true, the field text will be displayed in italic type.
FontUnderline	If set to true, the field text will be underlined.
IndicatorColor	Specifies the color of the progress indicator blocks or bar.
Pattern	Specifies the pattern to apply to the background.
PatternColor	Specifies the color of the pattern that is applied to the background.
PatternSelect	Specifies the pattern used when a push button-style check box is selected.
Position-Left	Indicates the position of the left edge of the field, measured in pixels from the left edge of the window.
Position-Top	Indicates the position of the top edge of the field, measured in pixels from the top of the window.
Resize-Horizontal	Specifies the horizontal resize behavior when per field resizing is used. Refer to Resizing windows on page 146 for more information.

Visual property	Description
Resize-Vertical	Specifies the vertical resize behavior when per field resizing is used. Refer to Resizing windows on page 146 for more information.
Scaling	For picture fields, specifies how the picture pasted into the field will be scaled. Proportional indicates the picture will maintain its original size and proportions. Stretch indicates the picture will be stretched to fill the picture field.
ShowPartialItems	For native list boxes, setting this property to true causes the list box to be drawn exactly the height indicated in the layout window. If there isn't enough room to display the last item, it will be only partially drawn. When this property is set to false, the list box will be resized vertically to include just the last complete item.
Size-Height	Indicates the field height, measured in pixels.
Size-Width	Indicates the field width, measured in pixels.
Style	<p>For push button and button drop list fields, specifies whether the button will display text only, graphics only, or both text and graphics. It also specifies the arrangement of the text and graphics.</p> <p>For check box fields, specifies whether the field displays as a standard check box or a push button.</p> <p>For progress indicator fields, specifies whether the progress indicator is composed of blocks or is a solid rectangle.</p>
Zoom	For push button fields, setting this property to true causes the pointer to become the zoom pointer (magnifying glass) when it's moved over the field. Typically, the Visible property is also set to false to make the push button invisible.

Drawn object properties

The following table lists the drawn object properties.

Property	Description
Alignment	For static text, indicates whether the text is left, center, or right-aligned.
Appearance	Specifies whether the drawn object is displayed with a 2-D or 3-D border.
Border	For static text, specifies whether a border is drawn around the object.
BackColor	Specifies the background color of the object.
Font	Specifies the type style and size to use for static text.
FontBold	If set to true, static text will be displayed in bold type.
FontColor	For static text, specifies the color of the text.
FontItalic	If set to true, static text will be displayed in italic type.
FontUnderline	If set to true, static text will be underlined.
LineColor	Specifies the color of the line used to draw the object.
LineSize	Specifies the width of the line used to draw an object.
Pattern	Specifies the pattern to apply to the background.
PatternColor	Specifies the color of the pattern that is applied to the background.
Position-Left	Indicates the position of the left edge of the object, measured in pixels from the left edge of the window.
Position-Top	Indicates the position of the top edge of the object, measured in pixels from the top of the window.
Resize-Horizontal	Specifies the horizontal resize behavior when per field resizing is used. Refer to Resizing windows on page 146 for more information.
Resize-Vertical	Specifies the vertical resize behavior when per field resizing is used. Refer to Resizing windows on page 146 for more information.
Shape	Specifies the shape of an item drawn with the shape tool.
Size-Height	Indicates the object height, measured in pixels.
Size-Width	Indicates the object width, measured in pixels.
Zoom	For static text items, setting this property to true causes the text to be displayed as specified by the Field SetZoomFormat() function.

Previewing a window

Choose Preview from the Layout menu to see how the window will appear at runtime. A preview of the current window will be displayed. This is only a preview of the visual characteristics of the window; the window will not be functional. When you have finished, close the preview window to return to the layout window.

Tab sequence

A tab sequence is the order in which the focus moves through fields in a window when the TAB key is pressed. You may want to change a tab sequence to change the order in which users enter information, ensuring they enter information you consider most important first.

Setting a tab sequence is typically one of the last tasks you need to complete when designing a window. After you've added all the fields to a window and arranged them in their proper order, you can use the following procedure to set a tab sequence.

1. Choose Set Tab Sequence from the Layout menu.

2. Select the field you want first in the sequence by double-clicking the field.

The focus will appear in this field when the window is opened initially. The field you double-click will appear shaded, meaning that it's now the first field in the tab sequence.

3. Press the TAB key.

4. Double-click the field you want next in the sequence.

The field will appear shaded, indicating it's the next field in the tab sequence.

5. Press the TAB key again and double-click the field you want third in the tab sequence.

Continue to mark fields in the window in the order you want, using the mouse to double-click the field and the TAB key to move the focus to the next object in the sequence.

6. Test the tab sequence.

When the tab sequence for all the fields in the window has been set, press the TAB key several times to move through the tab sequence. When you've finished setting the tab sequence, choose Set Tab Sequence from the Layout menu.



You can use the TabStop property to exclude fields from the tab sequence.

You can stop the process of setting a tab sequence at any time by closing the layout window and choosing not to save the changes to the window.



*Some fields, such as radio buttons, have special characteristics that must be considered when setting the tab sequence. This is described in [Chapter 8, “Standard Controls,”](#) of Volume 2 of the *Dexterity Programmer’s Guide*.*

Adding pictures

Graphics, such as a company logo, can be added to windows using standard cut-copy-and-paste techniques. If you wish to use several custom graphics throughout your application, you can store these graphics in the Dexterity picture library. This library allows you to add, name and store specific graphics in your application dictionary. Use the following checklist to add a new picture to the library, and then add it to a window.

1. Copy the graphic.

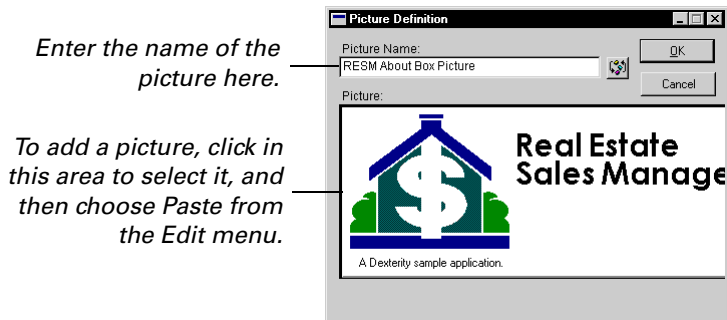
Select a graphic in your graphics application and choose Copy from the Edit menu.

2. Open the Pictures window.

Switch to Dexterity and choose Pictures from the Resources menu. The Pictures window will appear.

3. Create a new picture definition.

Click New; the Picture Definition window will appear as shown in the following illustration.



4. Name the picture.

Type a name for the picture in the Picture Name field.

5. Paste the picture.

Click the picture layout area beneath the Picture Name field, and then choose Paste from the Edit menu. The graphic you copied will appear in the picture area. Click OK to save the picture.

6. Add the picture to the window using the picture tool.

Use the picture tool to add the pictures in the library to the layout window. Select the picture tool and click where you want the picture to appear in the window. The Pictures window will open, allowing you to select which picture you'd like to place in the window.

Pictures you add to the library are treated like other resources in Dexterity; if you change the graphic in the library (for example, if you replace a picture), every occurrence of that picture will change in your dictionary.



If you paste a picture directly into the layout area, you'll be prompted to name the picture. The picture will appear in the layout and be added to the picture library automatically.

Linking fields to prompts

You can link a field to the text prompt associated with it by using the Link Prompt item in the Tools menu. By linking a field to its prompt, the prompt can indicate the status of the field, such as if the field is disabled or required.

To link a field to a prompt, choose Link Prompt from the Tools menu. Click on the field and drag the mouse pointer to its prompt (the text describing the field). A flashing black line will indicate that the link was made. Linking a prompt is shown in the following illustration.



When you've finished, be sure to choose Link Prompt again to turn prompt linking off. To unlink a prompt, switch to the prompt linking mode. Click on the field you want to unlink, drag to an unused area of the window, and release the mouse button. The prompt will be unlinked.

Positioning the window

You can also specify the position of the window using the Position properties for the window.

You can specify the opening position of windows when they're opened in your application. This allows you to arrange where a group of windows will appear on the screen when your application is being used, reducing the time users spend organizing windows on their screen.

You should be aware of the other objects, such as palettes or other windows, that will be on the screen at the same time as the window you're positioning. Use the following procedure to position a window in your application.

- 1. Choose Position Window from the Layout menu.**

The Position Window window will appear in the location where the window will be positioned at runtime. The default position and size coordinates are displayed in the window.

- 2. Drag the window to the location you want it to appear when opened.**

Note that the coordinates change each time you move the window. Use the coordinates in the window to more accurately position this window. When the window is positioned in the proper location, click OK.

Resizing windows

You can allow windows you create to be resized by the user when they are displayed. This is controlled by the `Resizable` property for the window. If set to `True`, the window can be resized. Dexterity supports two types of resizing: automatic and per field.

Automatic resizing

With automatic resizing, fields such as list boxes, text fields, tree views and list views will resize automatically when the window is resized. To use automatic resizing, set the `Resizable` property for the window to `True-Automatic`. No additional work is required.



Automatic resizing was the type supported in earlier versions of Dexterity.

As you add more fields to a window, the rules necessary to properly resize the window and maintain the proper appearance become very complex. When a window layout becomes too complex for Dexterity to automatically resize the window, Dexterity will no longer try to resize the controls in the window. When this occurs, you need to use per field resizing.

Per field resizing

To use per field resizing, set the `Resizable` property for the window to `True-Per Field`. You must then specify the `Resize Horizontal` and `Resize Vertical` properties for each field in the window. These two properties specify the resize characteristics of the field.



You can specify resize characteristics for static text and drawn objects, as well.

The following tables describe the possible values for these properties:

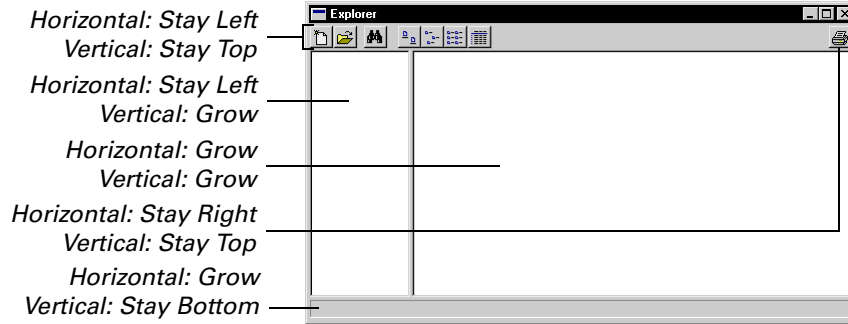
Resize Horizontal

Value	Description
Grow	The field will keep its current horizontal position and grow horizontally by the same amount the window grows. Example: When the window is resized 100 pixels wider, the field will maintain its horizontal position and be 100 pixels wider.
Stay Centered	The field will keep its current horizontal size, but will slide horizontally by half the amount the window grows. Example. When the window is resized 100 pixels wider, the field will maintain its horizontal size and slide 50 pixels to the right.
Stay Left	The field will keep its current horizontal size and horizontal position. Example: When the window is resized 100 pixels wider, the field will maintain its horizontal size and position.
Stay Left - Grow	The field will keep its current horizontal position and grow horizontally by half the amount the window grows. Example: When the window is resized 100 pixels wider, the field will maintain its horizontal position and be 50 pixels wider.
Stay Right	The field will keep its current horizontal size and move horizontally to maintain a constant distance from the right edge of the window. Example: When the window is resized 100 pixels wider, the field will slide 100 pixels to the right.
Stay Right - Grow	The field will grow horizontally by half the amount the window grows and move horizontally to maintain a constant distance from the right edge of the window. Example: When the window is resized 100 pixels wider, the field will slide 100 pixels to the right and be 50 pixels wider.

Resize Vertical

Value	Description
Grow	The field will keep its current vertical position and grow vertically by the same amount the window grows. Example: When the window is resized 100 pixels taller, the field will maintain its vertical position and be 100 pixels taller.
Stay Bottom	The field will keep its current vertical size and move vertically to maintain a constant distance from the bottom edge of the window. Example: When the window is resized 100 pixels taller, the field will slide 100 pixels down.
Stay Bottom - Grow	The field will grow vertically by half the amount the window grows and move vertically to maintain a constant distance from the bottom edge of the window. Example: When the window is resized 100 pixels taller, the field will slide 100 pixels down and be 50 pixels taller.
Stay Centered	The field will keep its current vertical size but will slide vertically by half the amount the window grows. Example: When the window is resized 100 pixels taller, the field will maintain its vertical size and slide 50 pixels down.
Stay Top	The field will keep its current vertical size and vertical position. Example: When the window is resized 100 pixels taller, the field will maintain its vertical size and position.
Stay Top - Grow	The field will keep its current vertical position and grow vertically by half the amount the window grows. Example: When the window is resized 100 pixels taller, the field will maintain its vertical position and be 50 pixels taller.

The following illustration shows a window that is too complex to be resized automatically. The horizontal and vertical resize characteristics of each field are shown.



Setting required fields

For your application to work properly, you may require that certain window fields contain information before processing can continue. You can use the Required property for fields to indicate which fields must contain data for processing to continue. Later, you will use the [required\(\)](#) function in your scripts to find out whether all of the fields with the Required property set to true actually contain data.

System color support - automatic

You can set the color properties for the various window elements to use system colors. When you use this setting, Dexterity automatically assigns colors for window items based on the current operating system settings.

Color properties

You can choose **System** as a color choice when you specify the FontColor, BackColor or LineColor visual properties for objects. If system colors are not supported for a particular color property or operating system, a default color is used.

The following chart lists the color properties and shows the operating system color setting that will be used when the color property is set to **System**.

Property	Windows operating system color
FontColor	Window font color
BackColor (For editable fields)	Window color when editable; 3D Object color when not editable
BackColor (For static items)	3D Object color
LineColor	Based upon 3D Object color



To give your application the best appearance when using system colors, we recommend that you set the BackColor property for windows to True and choose System as the FontColor and BackColor for all editable fields. We also recommend that you set the line color to System for any separator lines or group boxes.

Window background color

System colors are also used for the window background color. In the Windows operating system, the window background color used is determined by the operating system’s 3D Object color setting.

If you use pictures or native pictures in your application, you may want the picture’s background to blend automatically with whatever window background color a user selects. For example, the lookup button shown in the following illustration uses native pictures for the up and down states. If the user changes the operating system’s background color to a color other than light gray, the lookup button won’t blend with the new background color.

Before changing the operating system’s background color setting.

After changing the operating system’s background color setting.



The picture for the lookup button won’t blend properly.

To make the lookup button blend automatically with the background color, you must do the following:

- Set the BackColor visual property for the push button to System.
- Use a special color for the background of the pictures that appear on the push button. This special color is the light gray shade that appears in the 16-color palette for the Windows Paint application.

The RGB values for this color are Red=192, Green=192 and Blue=192 on a scale of 255. If you use a paint or draw application that specifies RGB colors as percentages, use the following percentages: Red=75.294%, Green=75.294% and Blue=75.294%.

When the BackColor property has been set to System, any part of the picture that uses the special light gray color will instead be displayed with the system's background color. If the lookup button shown in the previous example was drawn using the special light gray color, and the BackColor property for the lookup button was set to System, the lookup button would blend with the window background color.



Areas of the picture that use the special light gray color are changed to the window background color when the BackColor property is set to System.



Any pictures or native pictures can be made to blend with the window background color by using the special shade of gray in the picture and setting the BackColor property for the picture or control to System.

System color support - customizable

Dexterity maintains a list of system color categories that you can assign to window items. For instance, “System - Button Text” is one of the system color categories. You can set the color properties for a window element to use the color associated with a specific system color category.

The initial color used for each color category is derived from the operating system’s current color settings. If you don’t want to use the default color for a category, you can use sanScript code to specify a different color for the category.



Customizable system colors are the preferred way to support system colors in a Dexterity application.

System color categories

The following table lists the system color categories maintained by Dexterity.

System - Active Border	System - Highlight Text	System - Status Area*
System - Active Title Bar	System - Inactive Border	System - List Header1*
System - Active Title Bar Text	System - Inactive Title Bar	System - List Header1 Text*
System - Application Workspace	System - Inactive Title Bar Text	System - List Header2*
System - Button Face	System - Menu Bar	System - List Header2 Text*
System - Button Dark Shadow	System - Menu Text	System - Toolbar*
System - Button Light Shadow	System - Scrollbars	System - List Line1*
System - Button Shadow	System - Tooltip	System - List Line1 Text*
System - Button Highlight	System - Tooltip Text	System - List Line2*
System - Button Text	System - Window Background	System - List Line2 Text*
System - Desktop	System - Window Frame	System - Field Border*
System - Disabled Text	System - Window Text	System - Button Border*
System - Highlight		

The categories marked with an asterisk (*) are specific to Dexterity-based applications. These system color categories don’t correspond directly to colors defined by the Windows operating system. Their initial color values are derived from the other colors defined by Windows.



When you add new fields and static text to a window, they will be assigned colors from the appropriate system color category. You can change the color category assigned if you wish.

Overriding a system color category

The initial color used for a system color category is derived from the current color settings for the operating system. Use the [`Color_SetSystemColor\(\)`](#) function to specify a different color to use for a system color category. Use the [`Color_ResetSystemColor\(\)`](#) function to reset the system color category to its default value.

Custom color support

If you want users to be able to specify the colors used for prompts, zoom fields and scrolling window lines, you can use several functions from the Field function library. These functions, listed below, allow you to specify the colors used for these items in your application.

- [`Field_SetAltLineColor\(\)`](#)
- [`Field_SetCustomPromptFormat\(\)`](#)
- [`Field_SetZoomFormat\(\)`](#)

Refer to the description of these functions in the Function Library Reference manual for more information.

Procedure: Creating windows

You must create a form before attaching any windows to it. If you haven't created any forms, refer to [Chapter 10, "Forms,"](#) in this manual for more information about doing so.

To create a window, open the form definition the window will be attached to. Be sure the Windows tab is displayed. Click New; the Toolbox and Layout windows will appear.

1. Name the window.

If the Properties window isn't already displayed, choose Properties from the Layout menu. Select the arrow tool from the Toolbox and click in the layout window outside of the window's area to select the window. The resize handles that appear on the window's perimeter indicate it is selected. Use the Object tab's Name property to set the window's name. This name will be used in scripts, so you may want to substitute underscores for spaces in the name.



Use caution when changing the name of a window. If the window name has already been used in scripts, the scripts won't compile properly until the window name is changed in the scripts as well.

Use the Title property to enter the name that will be displayed in the window's title bar at runtime. This name can have spaces and can be changed at any time because it isn't used in scripts.

2. Select the appropriate window type.

Use the WindowType property to select the window type. Several types of windows are available: primary, modal dialog, modeless dialog, lookup, wizard, palette and toolbar. The following four are the most common window types:

- Primary windows have the standard title bar, minimize/maximize controls and scrolling controls. Most of the windows you'll create will be primary windows.
- Modal windows are typically used for alert or warning dialog boxes. They require the user to take a specific action before the application can proceed.

- Palettes are small windows that typically have a series of buttons arranged vertically. Palettes are described in detail in [Chapter 3, “Navigation,”](#) in the Dexterity Stand-alone Application Guide.
- Toolbar windows display across the top or down the left side of the screen. Toolbars always remain open while the application is running. An application should have only one toolbar open at a time.

3. Set other window properties.

Use the Properties window to specify other characteristics of the window, such as the background color and whether it will have resize controls. Refer to the [Window properties](#) on page 137 for a complete list of window properties.

4. Select the table to auto-link to the window (optional).

Use the AutoLinkTable window property to specify a table to link to the window. An auto-link table should be specified if you want users of your application to be able to add fields from the auto-linked table using the Modifier. The best table to auto-link to a window is the source or destination of most of the information displayed in the window.

5. Design the window.

To design the window, complete the following tasks.

Size the window Drag the resize handles around the perimeter of the window to set its size. The window will be this size when it's first opened in the application.

Position the fields in the window Drag global and local fields to the layout area from the Toolbox. Be sure to include all necessary fields, such as push buttons. You can use the drop-down list in the Toolbox to select one of the tables attached to the form. This narrows the list of global fields to those from a particular table, making it easier to find a specific field.

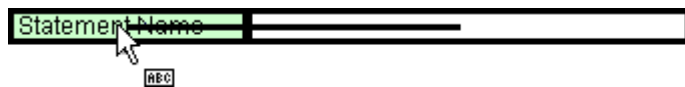
Add any graphics or prompts These include pictures, lines, boxes, prompts or text that help describe the purpose of the window.

Specify field and drawn object properties Use the Properties window to specify characteristics of the window fields and drawn objects. Refer to the [Field properties](#) on page 138 and [Drawn object properties](#) on page 142 for a complete list of properties.

For a description of the layout window and the drawing tools, refer to [Creating a window layout](#) on page 128.

Set the tab sequence Choose Set Tab Sequence from the Layout menu. Double-click the field you want to be first in the tab sequence, and then press the TAB key. Double-click the field you want to be second in the sequence. Continue this procedure until the sequence is in the desired order, and then choose Set Tab Sequence from the Layout menu again.

Link fields to their associated prompt Choose Link Prompt from the Tools menu. Select each field and drag the mouse pointer to its prompt (the text describing the field). A flashing black line will indicate that the link was made. The link is used to indicate required and disabled fields. When a field is required or disabled, its prompt shows the status of the field. Typically, the linked text is shown in bold for required fields and light gray for disabled fields. Linking a field to a prompt is shown in the following illustration.



When you've finished, be sure to choose Link Prompt again to turn prompt linking off.

You can also specify the position of the window using the Position properties for the window.

Position the window To set the position in which the window will appear when opened, choose Position Window from the Layout menu. A window the size of the window you're designing will appear. Move this window to the position where you want the window you're designing to appear. The actual size and position of the window are displayed in the window. When the window is positioned correctly, click OK.

At any time you can choose Preview from the Layout menu to display a preview of the window. The preview shows only the visual characteristics of the window and will not be functional.

When you've finished designing the window, close the layout window and click Save to save the new window. Click OK in the Form Definition window to save the form the window is part of.



Windows are saved only when the layout window is closed. Save your work often by closing the layout window and then reopening it to continue designing the window. Be sure the window and form have been saved before attaching any scripts.

Chapter 12: Using Test Mode

Once you have begun developing your application, you will want to test it. Dexterity has a special mode that allows you to view and operate your application as it will appear when used with the runtime engine. This special mode is called *test mode*. Information about test mode is divided into the following sections:

- [*Starting and stopping test mode*](#)
- [*Test mode tools*](#)
- [*Script lookup in test mode*](#)
- [*Limitations*](#)

Starting and stopping test mode

Before you can preview your application in test mode, it must have a Main Menu form. Also, you should have provided some method of navigation in your application. Typically, this would be push buttons or menus with scripts attached that open other forms in the dictionary. For information about using scripts to open forms and windows, refer to [*Controlling forms and windows*](#) on page 53 of Volume 2 of the Dexterity Programmer's Guide.

Refer to [*Chapter 14, "Form-based Menus,"*](#) for more information about menus.

To start test mode, choose Test Mode from the Debug menu. After a moment, your application will start. The Dexterity tool bar and menus will disappear. Any menus you have defined for your application will appear, along with the Debug menu. The Debug menu will appear as the rightmost menu.

When you have finished working with your application, simply choose Test Mode from the Debug menu. After a moment, you will return to Dexterity.

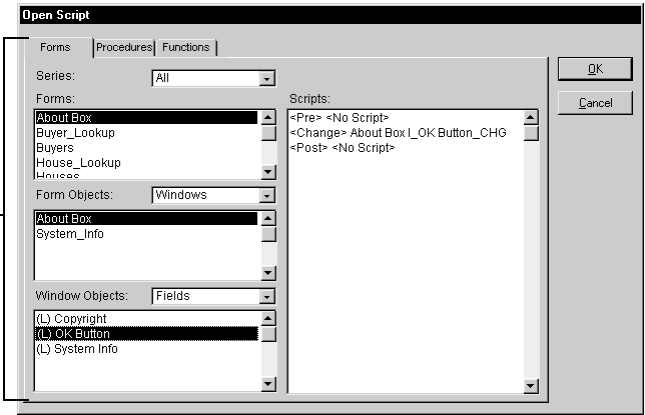
Test mode tools

Several tools are available for use in test mode: the Runtime Script Editor, the Script Debugger, the Script Profiler and the Script Logger.

Runtime Script Editor

The Runtime Script Editor allows you to open and edit any existing script in the dictionary. You can make changes in your application and see the results immediately. To use the Runtime Script Editor, choose Open Script from the Debug menu. The Open Script window will appear, as shown in the following illustration.

Click the appropriate tab indicating the type of script to open. Then use the list fields to navigate to the script.



Click the appropriate tab to indicate whether you are opening a form script, global procedure or global user-defined function. Then use the list boxes and drop-down lists to navigate to the script. When the script appears in the Scripts list, select it and click OK. The Script Editor will open, allowing you to edit the script.



As you become more familiar with attaching scripts, you will find it easier to select a script using the Open Script window.

Script Debugger

You have access to the same Script Debugger commands in test mode that you do in tools mode. Refer to [Chapter 27, “Script Debugger,”](#) in Volume 2 of the Dexterity Programmer’s Guide for information about the debugger.

Script Profiler

Use the Script Profiler in test mode to analyze the performance of your application. Refer to [Chapter 28, “Script Profiler,”](#) in Volume 2 of the Dexterity Programmer’s Guide for more information.

Script Logger

Use the Script Logger to keep track of which scripts have run in your application. Refer to [Chapter 29, “Script Logger,”](#) in Volume 2 of the Dexterity Programmer’s Guide for more information.

Script lookup in test mode

Dexterity provides easy access to scripts while an application is running in test mode. You can also use *script lookup mode* to select a script and display it in either the Script Editor or the Script Debugger. You invoke script lookup mode by holding down the SHIFT key and choosing Open Script or Script Debugger from the Debug menu. You can also use the keyboard. The following keyboard equivalents are used to invoke script lookup mode.

Script Debugger	CONTROL-F1
Script Editor	CONTROL-SHIFT-F1

When you invoke script lookup mode, the pointer will change into a script icon. You then click on the field, window or menu item whose script you want to access. The Open Script window will appear with the appropriate selections made to indicate the script you specified. Then click OK to display the script in the Script Editor or Script Debugger, depending upon how you invoked script lookup mode.

Limitations

Every effort has been made to ensure your application operates the same way in test mode and with the runtime engine. However, there are two limitations you should be aware of.

- Test mode has no support for the Runtime Report Writer or the Modifier. They can't be used in test mode.
- Test mode does not support multidictionary operation. This affects developers who make products that integrate with Microsoft Dynamics GP. It means you can't use test mode to run your application in a multidictionary environment. Instead, you must do your testing before you extract your application from the Dynamics.dic dictionary, or you must run your application with the runtime engine.

To allow running in test mode, you may need to write two versions of specific sections of sanScript code. One version will be designed to run in test mode, and the other will be designed to work with the runtime engine. You can use the following code to determine whether code is being executed in test mode or with the runtime engine:

```
if Launch_GetFileName() <> "" then
    {Runtime engine is being used}
else
    {Test mode is being used}
end if;
```


Part 4: Navigation

This part of the documentation provides detailed information about how to add navigation elements, such as menus and toolbars to your application. Following is a list of the topics that will be discussed, with a brief explanation of each:

- [Chapter 13, “Main Menu,”](#) describes the special Main Menu form. This serves as the base for navigation in your application.
- [Chapter 14, “Form-based Menus,”](#) explains how to use form-based menus for an application.
- [Chapter 15, “Commands,”](#) explains how to implement commands in an application, and how they are used.
- [Chapter 16, “Command-based Menus,”](#) describes how to implement command-based menus for an application.
- [Chapter 17, “Context Menus,”](#) describes how to use context menus in an application.
- [Chapter 18, “Toolbars,”](#) explains how to implement toolbars for an application.

Chapter 13: Main Menu

Each application dictionary you create must have a form named Main Menu that will be the first form opened when the application starts. The Main Menu form serves as a starting point for the application, allowing the user to navigate to other parts of the application. Information about the main menu is divided into the following sections:

- [*The main window*](#)
- [*The menu bar*](#)

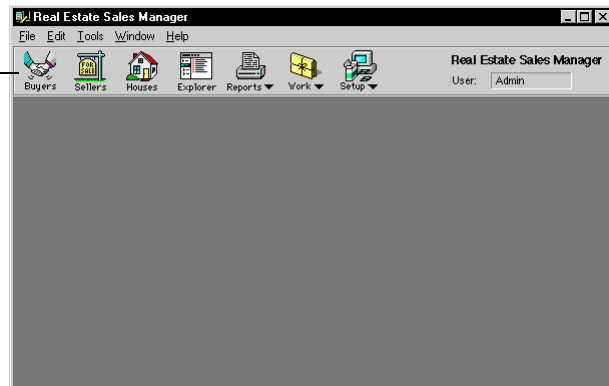
The main window

The window associated with the Main Menu form is usually a standard window or a toolbar window. Another option is using command-based toolbars to display navigation.

Traditional toolbar

A traditional toolbar used as a main window often contains push buttons or button drop lists used to open other forms in the application. The toolbar main menu for an earlier version of the Real Estate Sales Manager sample application is shown in the following illustration.

This toolbar is the main window for the application.

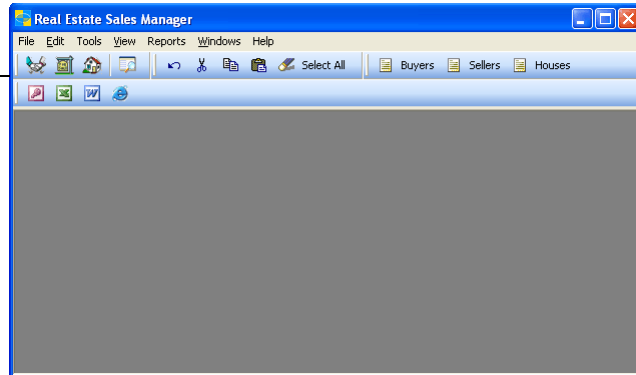


A window isn't required in the Main Menu form. If an application doesn't have any windows in the Main Menu form, only the menu bar is displayed when the application is first started.

Command-based toolbar

Instead of using a traditional toolbar window, you may want to define commands for your application and use command-based toolbars. The command-based toolbars for the Real Estate Sales Manager sample application are shown in the following illustration.

These are command-based toolbars for the application.



The commands used for the toolbars are defined on the Main Menu form for the application.

The menu bar

When you use Dexterity to create a new application dictionary, the Main Menu form is created automatically in the dictionary. No window is added to the form, but a set of default form-based menus is. These include the File, Edit, Macro, Windows and Help menus.

Any menus that you want continuously available in the application must be part of the Main Menu form. Such menus are referred to as *application-level menus*. For more information about menus, refer to [Chapter 14, "Form-based Menus."](#)

You might also choose to use command-based menus for your application. Command-based menus provide additional flexibility, and the commands displayed in the menus can also be used on command-based toolbars. For more information, refer to [Chapter 16, "Command-based Menus."](#)

Chapter 14: Form-based Menus

Menus are found across the top of the screen in your application's menu bar. With form-based menus, the content of the menus is defined in the individual forms for the application. Prior to Dexterity 8.0, these were the only type of menus supported. Menus that were defined as part of the Main Menu form were always accessible. Menus that were defined as part of other forms were displayed only when windows attached to those forms were displayed.

Microsoft Dynamics GP uses command-based menus. If you create applications that integrate with Microsoft Dynamics GP, most of the menus and menu items you create will be command-based menus. You can use form-based menus for your integration. Form-based menus will appear as submenus on the "Additional" menu in Microsoft Dynamics GP.



All references to menus and menu items in this portion of the documentation refer to form-based menus.

Information about menus is divided into the following sections:

- [Menu types](#)
- [Menu elements](#)
- [Cascading menus](#)
- [Default menu set](#)
- [Using menus](#)
- [Application-level menus and sanScript](#)
- [Displaying form-level menus](#)
- [Procedure: Creating form-based menus](#)

Menu types

You create form-based menus in Dexterity by defining them as part of a form definition. Form-based menus in Dexterity can be divided into two categories, depending on the form where the menus are defined: application-level menus and form-level menus.

Application-level menus

Menus that are part of the Main Menu form are always available for any form in the application. Application-level menus can inherit sets of predefined menu items, such as the items for Dexterity's macro system.

Refer to [Displaying form-level menus](#) on page 176 for more information about how form-level menus are displayed.

Form-level menus

Form-level menus are used to add menu items that are specific to a single form. The menus defined for forms other than the Main Menu form are visible only when a window attached to the form is active. Form-level menus can't inherit sets of predefined menu items.

If you use command-based menus for your application, any form-level menu you create will appear as a submenu in the “Additional” menu for your application.

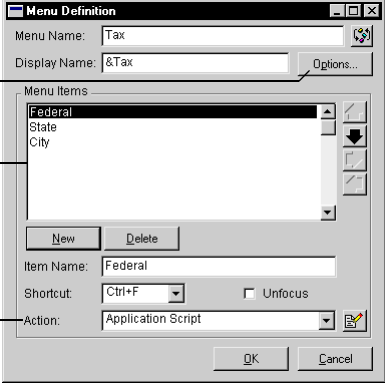
Menu elements

Each menu attached to a form has a menu name, display name and a list of items defined for the menu. Application-level menus can also have a set of inherited items that perform specific operations, such as clipboard functions. The Menu Definition window, shown in the following illustration, is used to create menus.

Application-level menus can inherit sets of predefined actions.

The items in the menu are listed here.

Each menu item can have a script attached or inherit a predefined action.



Name

Two names are required for each menu. The *menu name* is the name that is used in scripts to refer to the menu. To make the name easier to use in scripts, we suggest you use underscores (_) instead of spaces between the words in the name.

The *display name* is the name that appears when the menu is displayed in the menu bar. The menu bar area is limited, so menu names should be kept as short as possible.



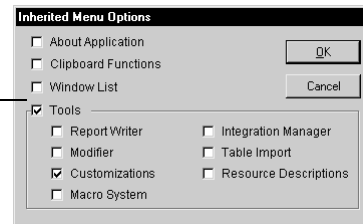
Dexterity can display a maximum of 15 menus at one time, regardless of whether space is available on the menu bar.

You can define an access key for the menu by placing an ampersand (&) in the menu display name. The ampersand won't be displayed in the menu name at runtime. Instead, the character immediately following the ampersand will be underlined. To choose the menu, a user can press the ALT key and the underlined character at the same time.

Inherited Options

Application-level menus can inherit sets of predefined items that perform special actions in your application. Click the Options button in the Menu Definition window to display the Inherited Menu Options window, shown in the following illustration.

Mark these options for an application-level menu to inherit groups of items.



The following table lists the groups of menu items that can be inherited and where the items will appear in the menu.

Option	Menu items	Location	Description
About Application	About...	At the bottom	In Windows, contains the item allowing the application's About Box to be accessed. This item won't appear on the Macintosh.
Clipboard Functions	Undo Cut Copy Paste Clear Select All	At the top	The standard cut/copy/paste items commonly found in graphical applications.
Window List	List of open windows	At the bottom	List of the standard windows currently open. Palette and toolbar windows are not included in this list.
Tools - Report Writer	Report Writer Modify Current Report	Subitems of the Customize menu item	Accesses the Runtime Report Writer or attempts to open the current report.
Tools - Modifier	Modifier Modify Current Window	Subitems of the Customize menu item	Accesses the Modifier. If VBA is installed, several VBA menu items are also displayed.
Tools - Customizations	Customization Status Customization Maintenance	Subitems of the Customize menu item	Displays the Customization Status window and the Customization Maintenance window.

Option	Menu items	Location	Description
Tools - Macro System	Play/Stop Play Record/Stop Record Pause/Continue Suspend Recording Step ----- Status Window Dump Field Dump (All) Field Dump (Selected) Menu Dump Menu Dump All Table Dump Open Form Insert Header Insert Comment	At the top	The items to access Dexterity's built-in macro system. By default, the first group of items is included in the menu. The second group of commands are accessed by adding the ShowAdvancedMacroMenu=TRUE setting to the defaults file.
Tools - Integration Manager	Integration Manager Table Import	Subitems of the Integrate menu item	These items are used only by Microsoft Development.
Tools - Table Import	Table Import	A subitem of the Integrate menu item	This item opens the Table Import Definition window.
Tools - Resource Descriptions	Tables Fields Windows	Subitems of the Resource Descriptions menu item	These items open the Table Descriptions, Field Descriptions and Window Descriptions windows.

Menu items

Use the Menu Items list to define additional items for the menu. You can specify the menu item's name, shortcut and action. You also indicate whether focus is pulled from the current window when the menu item is chosen.

Name

After clicking New to create a new item, enter the name of the menu item in the Item name field. The &, #, - and ~ characters have special meaning in menu item names. These characters are described in the following table.

Character	Effect
&	Makes the next character underlined and act as the access key..
#	Draws an ellipsis (...) after the item.
-	If used as the item name, causes a separator line to appear in the menu. Use separator lines to group items in the menu.
~	Draws a forward slash (/).

Shortcut

You can define a shortcut for the menu item by selecting a value from the Shortcut field. The shortcut is CTRL + the specified character. To avoid conflicts, a shortcut key should be used only once in an application.

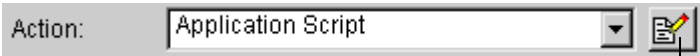
Action

The Action drop-down list specifies whether the menu item will perform a predefined action or run an application script. You can choose a predefined action or application script.

Predefined actions The following table lists the predefined actions that can be assigned to the menu item. These predefined actions can be assigned to form-level or application-level menus. A specific predefined action should be assigned to only one menu item in the application.

Action	Description
Application Script	Runs the script attached to the menu item.
Delete Row	Runs the line delete script for a scrolling window.
Help Contents	Displays the help contents topic.
Help on Help	Displays help for the Windows help system.
Help on Item	Displays help for the currently focused item.
Help on Window	Displays help for the current window.
Help Search	Displays the search window for the help system.
Insert Row	Runs the line insert script for a scrolling window.
Lookup	Opens the lookup form for the current field or runs the change script linked to the current field using Link Lookup.
Modifier	Provides access to the Modifier.
Print	Runs the print script for the current window.
Print Setup	Displays the Print Setup window for the current operating system.
Process Monitor	Opens the Process Monitor window.
Quit Application	Closes the application.
Report Writer	Provides access to the Runtime Report Writer.
Show Required	Displays the required fields for the application.

Scripts You can attach a script to the menu item. Be sure Application Script is selected in the Action drop-down list and click the Script button. The script will be run each time the menu item is chosen.



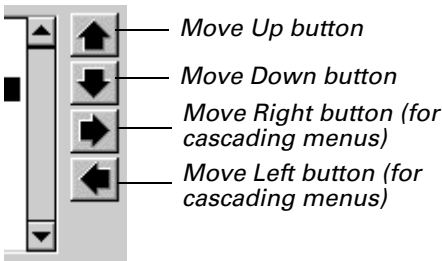
Click the Script button to attach a script to a menu item.

Unfocus

Mark the Unfocus option to cause the focus to be pulled from the current window when the menu item is chosen. This causes any pending field scripts to be run before the menu item’s action is carried out. If the application is in a state such that the menu action can’t be performed, the field’s scripts can then prevent the action from being performed. If you don’t mark the Unfocus check box, the focus will remain in the current field when the menu item is chosen.

Menu item position

You can use the positioning buttons to the right of the Menu Items list to change the position of items in the menu. These buttons are shown in the following illustration.



Move Up button

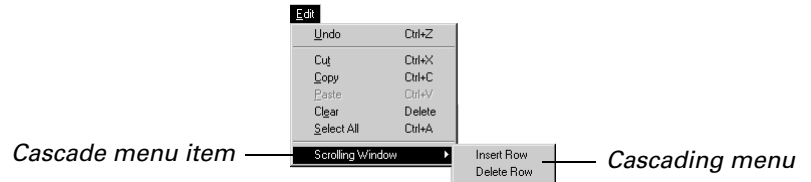
Move Down button

Move Right button (for cascading menus)

Move Left button (for cascading menus)

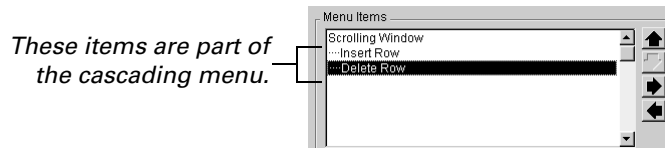
Cascading menus

Cascading menus, or “submenus,” appear to the right of menu items as shown in the following illustration:



A cascading menu is useful for displaying related menu items without cluttering the menu.

To create a cascading menu item, first add the cascade menu item and all of the items that will cascade from it. Don't specify an action for the cascade menu item, since it simply provides navigation to the cascading menu. Then use the Move Left and Move Right buttons to position the menu items beneath the cascade menu item, as shown in the following illustration.



The dashed lines in front of the menu items indicate they are part of a cascading menu.



For maximum usability, we recommend that you don't create cascading menus more than one level deep.

Default menu set

When you create a new application dictionary with a Main Menu form, it has a basic set of application-level menus. The following table lists the menus that are added to a new application dictionary.

Menu	Item	Description
File	Print	Runs the print script for the current window.
	Print Setup	Displays the Print Setup window for the current operating system.
	Quit/Exit	Closes the application.
Edit	Undo Cut Copy Paste Clear Select All	The standard cut/copy/paste items commonly found in graphical applications.
	Insert Row	Runs the line insert script for a scrolling window.
	Delete Row	Runs the line delete script for a scrolling window.
Temporary	(Form-level menus)	The placeholder that indicates where form-level menus will be displayed. No “Temporary” menu will appear on the menu bar.
Tools	Macro	The items to access Dexterity’s built-in macro system.
	Resource Descriptions	The resource description windows.
Window	List of open windows	List of the standard windows currently open. Palette and toolbar windows are not included in this list.
Help	Lookup	Opens the lookup form for the current field or runs the change script linked to the current field using Set Lookup.
	Show Required Fields	Displays the required fields for the application.

These menus are intended to serve as a starting point. Feel free to customize the menus any way you like.

Using menus

Menus have two basic uses: initiating actions and indicating options.

Initiating actions

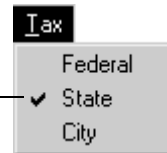
You can use menus to initiate actions, such as opening a specific form or performing a calculation. To do this, assign the menu item for one of the predefined actions or attach a script to carry out the action.

Indicating options

You also can use menus to indicate and set options by using the [check menu](#) and [uncheck menu](#) statements to add or remove a check mark next to a menu item. For example, a menu can be used to select which taxes apply to a purchase. When the menu is checked, the corresponding tax will be included.

Refer to the SanScript Reference manual for information about the statements used with menu items.

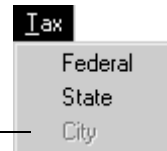
*The **check menu** and **uncheck menu** statements are used to add or remove the check mark from an item.*



Enabling and disabling menu items

You can enable and disable menu items by using the [enable menu](#) and [disable menu](#) statements. These statements allow you to control when a user can select a particular menu item.

*The **disable menu** and **enable menu** statements are used to enable or disable an item.*



Controlling menu content

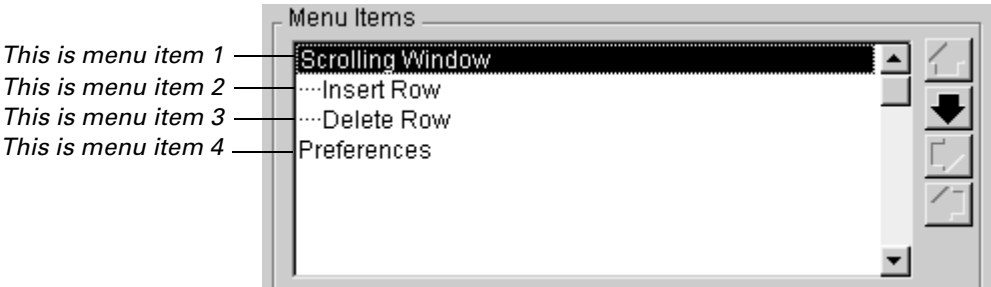
You can control the content of menus by using the [change item](#) statement, which allows you to change contents of a menu item. The following commands can also be used with menus.

Function	Description
countitems()	Returns the number of items in a menu.
finditem()	Returns the location of a specific item in a menu.
itemname()	Returns the name of the item in a specific location in the menu.
hide menu	Hides the specified menu item.
show menu	Shows the specified menu item.

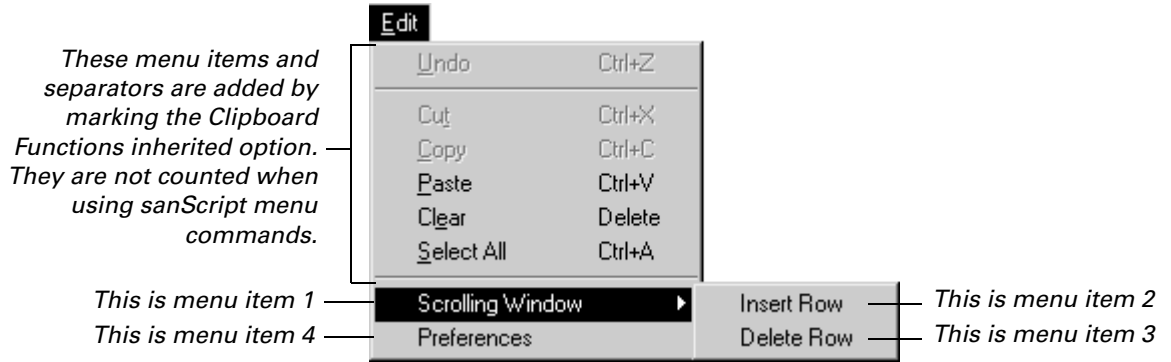
Application-level menus and sanScript

If your application-level menus inherit groups of menu items and use sanScript commands to manipulate items in those menus, you need to be aware of how the groups of inherited menu items are handled. The sanScript commands to manipulate menus, such as [enable menu](#), [disable menu](#), [check menu](#), [uncheck menu](#), [hide menu](#), and [show menu](#) use a number to specify which menu item to alter. The menu items added by the groups of inheritable options are *not* counted when you specify which menu item to alter.

For example, inheriting the clipboard functions adds six menu items and two separators to a menu. None of these items are counted when specifying a menu item for a sanScript command. In the following illustration, the clipboard functions were inherited. Four additional items were added to the menu.



The four additional menu items are numbered 1 through 4, even though they are in positions 9 through 12 in the menu. This is because the group of inherited clipboard functions isn't counted.



The following sanScript statement would disable the Preferences menu item.

```
disable menu Edit 4;
```



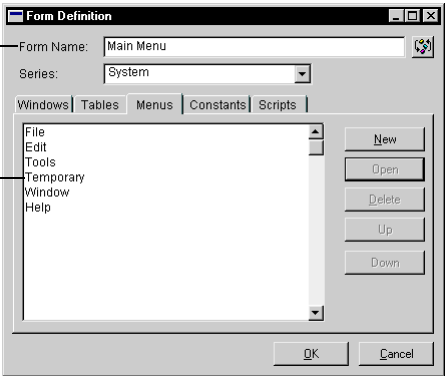
*Only the **groups** of inherited menu items such as the clipboard functions or window list are ignored. Individual inherited actions, such as Show Required and Lookup, are counted when using the sanScript menu commands.*

Displaying form-level menus

Form-level menus are displayed only when a window attached to the form is open. To indicate where the form-level menus will appear, add a special menu with the name Temporary to the Main Menu form. The Temporary menu won't be displayed. It simply acts as a placeholder. If you don't add a Temporary menu, form-level menus will appear after the last menu displayed.

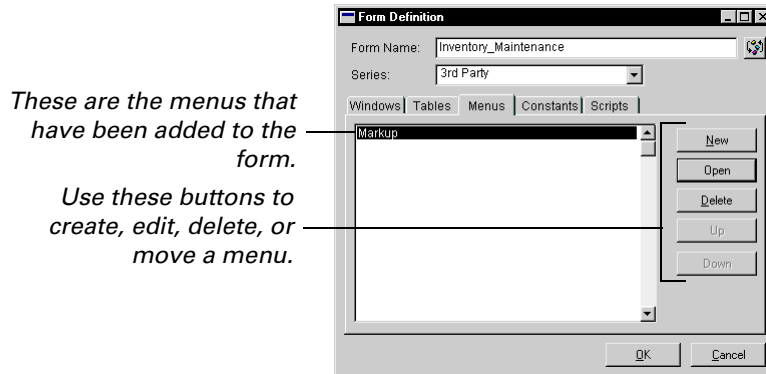
The Temporary menu must be part of the Main Menu form.

The Temporary menu indicates where form-level menus will be displayed.

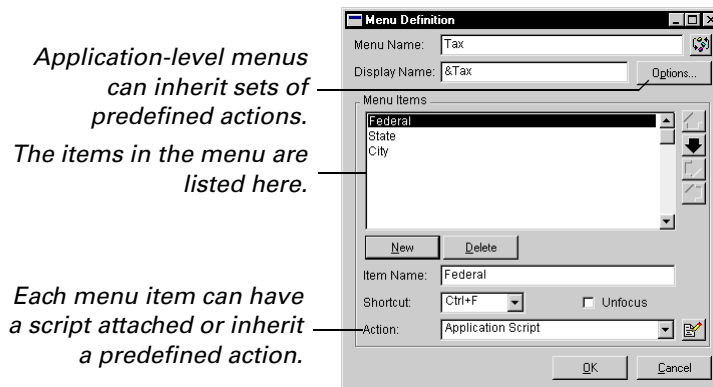


Procedure: Creating form-based menus

To create or edit a menu, open the form definition window for the form where the menu will be attached. Click the Menus tab to display the list of menus that have been defined for the form.



Click New to create a new menu, or select a menu and click Open to edit an existing menu definition. The Menu Definition window will appear.



1. Name the menu.

Two names are required for a menu. The Menu Name is the name used in scripts, while the Display Name is the name displayed in the menu bar.

You can specify the access key for the menu by placing an ampersand (&) in the display name. The character immediately following the ampersand will be underlined. To choose the menu at runtime, press the ALT key and the underlined character at the same time.

2. Add menu items.

If you're creating an application-level menu (one attached to the Main Menu form), choose which groups of menu items you want the menu to inherit, if any. Do this by clicking the Options button to display the Inheritable Menu Options window and marking the appropriate check boxes.

To add additional menu items, click New and type the menu item name in the Item Name field. Include an ampersand (&) character in the name to define a Windows access key for the menu item. Use the pound sign (#) to include an ellipsis in the name. Use the tilde (~) to include a slash in the name.

Select an item in the Shortcut field if you want to specify a shortcut key for the menu item. The shortcut key is CTRL + the selected character.

You can use a separator line to group the items in the menu. To add a separator, add a menu item between the items to be separated. Enter a hyphen (-) as the menu item's name. At runtime, the item will appear as a separator line in the menu.

To remove an item from the menu, select the item and click Delete. The item will be deleted.

To rearrange items in the menu, select an item and click the Move Up or Move Down buttons to move the menu item. Use the Move Left and Move Right buttons to create cascading menu items.

Mark the Unfocus option to indicate that the menu items should pull the focus from the current window field when it is chosen. Pulling the focus causes any pending field scripts to be run before the menu item's action is carried out. If the application is in a state such that the menu action can't be performed, the field's scripts can then prevent the action from being performed.

3. Specify menu actions.

You can specify an action for a menu item by selecting one of the predefined actions or by attaching an application script. To specify a predefined action, select one from the Action drop-down list. The following table lists the actions.

Action	Description
Application Script	Runs the script attached to the menu item.
Delete Row	Runs the line delete script for a scrolling window.
Help Contents	Displays the help contents topic.
Help on Help	Displays help for the Windows help system.
Help on Item	Displays help for the currently focused item.
Help on Window	Displays help for the current window.
Help Search	Displays the search window for the help system.
Insert Row	Runs the line insert script for a scrolling window.
Lookup	Opens the lookup form for the current field or runs the change script linked to the current field using Link Lookup.
Modifier	Provides access to the Modifier.
Print	Runs the print script for the current window.
Print Setup	Displays the Print Setup window for the current operating system.
Process Monitor	Opens the Process Monitor window.
Quit Application	Closes the application.
Report Writer	Provides access to the Runtime Report Writer.
Show Required	Displays the required fields for the application.



A specific predefined action should be assigned to only one menu item in the application. If an action is assigned to more than one menu item, only the last menu item assigned the action will actually perform it.

To attach a script to the menu item, choose Application Script in the Action drop-down list, and then click the Script button. The script editor will appear, allowing you to enter a script that will be run each time the menu item is chosen.

4. Click OK to save the menu definition.

The menu will appear in the list of menus in the Form Definition window.

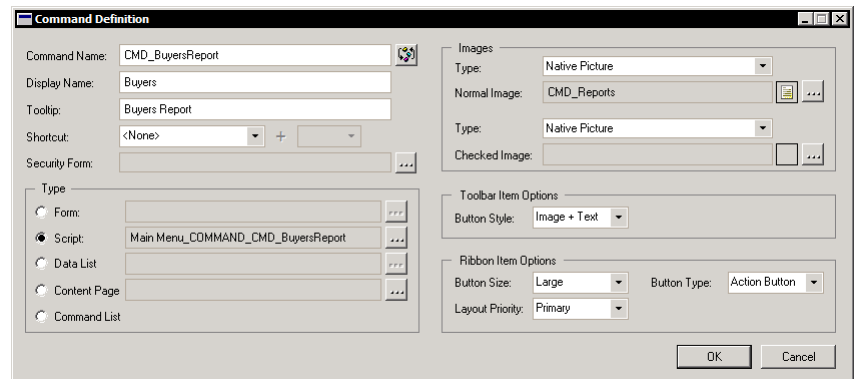
Chapter 15: Commands

Commands are used to encapsulate a small piece of functionality for an application, and are typically used for navigation. They are an essential part of command-based menus, toolbars, and the ribbon for data lists. Information about commands is divided into the following sections:

- [*Command elements*](#)
- [*Using commands*](#)
- [*Controlling command access*](#)
- [*Built-in commands*](#)
- [*Procedure: Creating a command*](#)

Command elements

Each command has a name, display name, tooltip, shortcut, security form, type, images, toolbar options, and ribbon options. The Command Definition window, shown in the following illustration, is used to create commands and specify their characteristics.



Name

Each command has a name that is used in scripts to refer to the command. To make the name easier to use in script, you can use underscores (_) between parts of the name instead of spaces.

Display name

The display name is the name that appears when the command is displayed in a menu or on a toolbar in the application.

You can define an access key for the command by placing an ampersand (&) in the display name. The ampersand won't be displayed in the command name at runtime. Instead, the character immediately following the ampersand will be underlined. To choose the menu item, a user can press the ALT key and the underlined character at the same time.

To display special characters in the display name, such as ampersands or forward slashes, you must precede them with a backslash. For example, the following display name will display an ampersand character:

Display Name	Time \& Billing
--------------	----------------------------

Tooltip

When a command is used on a toolbar, it will display a tooltip if one is provided.

Shortcut

The shortcut for the command is a combination of a modifier key (Control, Alt, or Shift) and an alphanumeric or function key. When the command is enabled, the user can execute the command by using the specified key combination.

Security Form

This is the form that the user must have security privileges to access for the command to be accessible.

Type

A command can be one of the following types:

Form A form command opens the specified form. Use the lookup button to select the form to open.

Script A script command runs the supplied sanScript code.

Data List A command that will open a data list form and display data in a list view.

Content Page A command that will display content in the main application area for the Dynamics runtime. A content page is used for content such as the area pages in Microsoft Dynamics GP.

Command List A command list command is a container for other commands. A command list can be used as a menu, submenu, or a toolbar. Commands are added to a command list through sanScript code.

Images

Icons or Native pictures are used as the images for commands. For proper display on toolbars, the images should be 20 pixels by 20 pixels. Two images can be supplied:

Normal Image This is the image displayed when the command is used on a toolbar, and is in the unchecked state.

Checked Image This is the image displayed when the command is used on a toolbar, and is in the checked state. Use the [check command](#) and the [uncheck command](#) to toggle the command between the checked and unchecked state.

Toolbar item options

You can specify whether the command will appear as text, as an image, or with both the text and image.

Ribbon Item Options

The following options apply when the command is used on a ribbon for a data list.

Button Size Indicates the size of the button on the ribbon. It can be large or small.

Layout Priority Indicates the preferred position of a command when it appears in a group on the ribbon. The Primary position is a large icon in the group. The Secondary position is a smaller icon vertically stacked on the right side of the group. The Force Overflow position is in a pull-right menu accessible on the right side of the group.

Button Type For script commands used in the ribbon for a data list, specifies the type of behavior the command will initiate. Choose Action Button to specify that a standard sanScript processing will be performed by the command. Choose Drop Dialog to specify that the command will open a drop dialog window to perform the action for the command.

Using commands

Commands are primarily used for navigation in an application. They are the basis of command-based menus and toolbars.

Command form

Commands are a form-based resource. For a command to be accessed, the form it is part of must be open at the time the command is used. For stand-alone applications, commands that should always be available could be defined for the Main Menu form.

You might also want to create separate command forms that contain groups of related commands. This gives added flexibility when controlling access to commands. Opening or closing the command form will control whether the commands are available. This is the approach used by applications that integrate with Microsoft Dynamics GP. Each integrating application defines its own command form.



If a command is part of an active form, it is available to be used, even if it does not appear on a menu or a toolbar.

Tags

When a command is made available, it is automatically assigned a tag. A tag is an integer that uniquely identifies that command for the current application session. A tag for a command is retrieved using the [Command.GetTag\(\)](#) function. The tag is required when running other commands such as [Command.GetType\(\)](#).



Don't store the tag for a command. A new tag is assigned to the command each time the application is run.

Command lists

A command list is a special type of command used to group related commands for use in toolbars or command-based menus. Commands are added to a command list using functions from the [Command list function library](#). You will also use functions from this library to manage the contents of command lists.

Menus

Commands and command lists form the basis of command-based menus. Refer to [Chapter 16, "Command-based Menus,"](#) for more information.

Toolbars

Commands and command lists are also used for toolbars that can be displayed in an application. Refer to [Chapter 18, “Toolbars,”](#) for more information.

Running commands

Form and script commands can be executed directly by the user when the commands appear on menus or toolbars. They can also be run through scripts. You can run a command directly using the [run command](#) statement. If only the tag for the command is available, use the [Command.Execute\(\)](#) function.

Controlling command access

By default, commands are accessible whenever the form they are defined for is open. You can control whether a command is accessible by using the [enable command](#) and [disable command](#) statements. When command is disabled, it cannot be executed. It will also appear disabled on any menu or toolbar that displays it.

You can use the [hide command](#) and [show command](#) statements to control whether a command is visible. When a command is hidden, it is automatically removed from any menus or toolbars that display it.

Built-in commands

Several built-in commands are already defined, and can be used in your application. These commands provide access to system-level features, such as the clipboard operations cut, copy, and paste.



Built-in commands have the product ID value 1, and the form ID value -1.

The following table lists the built-in commands that are provided.

Category	Command	Type	Display name	Description
System	cmdLookup	Command	Lookup	
	cmdProcessMonitor	Command	Process Monitor	
	cmdQuitApplication	Command	Exit	
	cmdShowRequired	Command	Show Required Fields	
	cmdListFormTriggers	Command list	Additional	Lists form trigger items
	cmdListFormMenus	Command list	Extras	Lists form-level menu items
Context menu	cmdListContextMenu	Command list	N/A	Used for the contents of context menus
Separator	cmdSeparator	Command	-	Used for all separators on menus and toolbars
Editing	cmdListEdit	Command list	Edit	Edit menu command list
	cmdEditUndo	Command	Undo	
	cmdEditCut	Command	Cut	
	cmdEditCopy	Command	Copy	
	cmdEditPaste	Command	Paste	
	cmdEditClear	Command	Clear	
	cmdEditSelectAll	Command	Select All	
Windows	cmdWindowList	Command list	Windows	List of all open windows
Customization	cmdListCustomize	Command list	Customize	Customize menu command list
	cmdListReportWriter	Command list	Report Writer	Report Writer command list
	cmdReportWriter	Command	Report Writer	Starts the Report Writer
	cmdModifyCurrentReport	Command	Modify Current Report	
	cmdListModifier	Command list	Modifier	Modifier command list
	cmdModifier	Command	Modifier	Starts the Modifier
	cmdModifyCurrentWindow	Command	Modify Current Window	
	cmdCustomizationMaintenance	Command	Customization Maintenance	
	cmdImportExportReports	Command	Import/Export Reports	
	cmdCustomizationStatus	Command	Customization Status	
VBA	cmdVBAAEditor	Command	Visual Basic Editor	
	cmdVBAAAddCurrentWindow	Command	Add Current Window to Visual Basic...	
	cmdVBAAAddFields	Command	Add Fields to Visual Basic...	
	cmdVBARemoveWindow	Command	Remove Current Window from Visual Basic	

Category	Command	Type	Display name	Description
Macro	cmdListMacro	Command list	Macro	Macro command list
	cmdMacroPlay	Command	Play/Stop Play	
	cmdMacroRecord	Command	Record/Stop Record	
	cmdMacroPause	Command	Pause/Continue	
	cmdMacroSuspendRecord	Command	Suspend Recording/ Resume Recording	
	cmdMacroStep	Command	Step	
	cmdMacroInsertPause	Command	Insert Pause	
Advanced Macro	cmdListMacroAdvanced	Command list	Advanced	Advanced macro command list
	cmdMacroStatus	Command	Status...	
	cmdMacroWindowDump	Command	Window Dump...	
	cmdMacroFieldDumpAll	Command	Field Dump (All)...	
	cmdMacroFieldDumpSelected	Command	Field Dump (Selected)...	
	cmdMacroStopSelecting	Command	Stop Selecting	
	cmdMacroMenuDump	Command	Menu Dump	
	cmdMacroMenuDumpAll	Command	Menu Dump All	
	cmdMacroTableDump	Command	Table Dump...	
	cmdMacroOpenForm	Command	Open Form...	
	cmdMacroInsertHeader	Command	Insert Header...	
	cmdMacroInsertComment	Command	Insert Comment	
Import	cmdTableImport	Command	Table Import	
Resource Descriptions	cmdListResourceDescriptions	Command list	Resource Descriptions	Resource Descriptions command list
	cmdTableDescriptions	Command	Tables	
	cmdFieldDescriptions	Command	Fields	
	cmdWindowDescriptions	Command	Windows	
Scrolling Windows	cmdInsertRow	Command	Insert Row	
	cmdDeleteRow	Command	Delete Row	
Help	cmdHelpContents	Command	Contents	
	cmdHelpOnHelp	Command	Help on Help	Help on Windows Help
	cmdHelpOnItem	Command	Help on Item	
	cmdHelpOnWindow	Command	Help on Window	
	cmdHelpSearch	Command	Index	
	cmdAboutBox	Command	About...	
Print	cmdPrint	Command	Print	
	cmdPrintSetup	Command	Print Setup	

Procedure: Creating a command

To create or edit a command, open the form definition window for the form where the command will be attached. Click the Commands tab to display the list of commands that have been defined for the form. Click New to create a new command, or select a command and click Open to edit an existing command definition. The Command Definition window will appear.

The screenshot shows the 'Command Definition' dialog box with the following fields and options:

- Command Name:** CMD_BuyersReport
- Display Name:** Buyers
- Tooltip:** Buyers Report
- Shortcut:** <None>
- Security Form:** (empty)
- Type:**
 - ☐ Form
 - ☒ Script: Main Menu_COMMAND_CMD_BuyersReport
 - ☐ Data List
 - ☐ Content Page
 - ☐ Command List
- Images:**
 - Type: Native Picture
 - Normal Image: CMD_Reports
 - Type: Native Picture
 - Checked Image: (empty)
- Toolbar Item Options:**
 - Button Style: Image + Text
- Ribbon Item Options:**
 - Button Size: Large
 - Button Type: Action Button
 - Layout Priority: Primary

Buttons: OK, Cancel

1. Name the command.

Two names are required for a command. The Command Name is the name used in scripts, while the Display Name is the name displayed when the command is used in a menu or on a toolbar.

You can define an access key for the command by placing an ampersand (&) in the display name. The ampersand won't be displayed in the command name at runtime. Instead, the character immediately following the ampersand will be underlined.

2. Add a tooltip (optional).

When a command is used on a toolbar, it will display a tooltip if one is provided.

3. Add a shortcut (optional).

The shortcut for the command is a combination of a modifier key (Control, Alt, or Shift) and an alphanumeric or function key.

4. Specify the security form (optional).

5. Specify the command type.

A command can open a form, execute a script, open a data list, open a content page, or act as a list of other commands.

6. Specify the command images (optional).

If the command will be used on a toolbar, you may want to specify native pictures or icons to use. One image will be displayed in the normal state, and the other will be displayed when the command is in the checked state. For best display, images should be 20 pixels by 20 pixels.

7. Specify toolbar options.

You can specify whether the command will appear as text, as an image, or with both the text and image.

8. Specify ribbon options.

You can specify the default size and location of a command used in the ribbon for a data list. You can also specify the type of action a script command displayed on a ribbon will be performing so the command will be processed correctly.

9. Click OK to save the command definition.

The command will appear in the list of commands in the Form Definition window.

Chapter 16: Command-based Menus

Menus are found across the top of the screen in your application's menu bar. With command-based menus, the entire menu structure is defined through a script that you add for your application. Commands you have previously defined are assembled to create the menus and submenus displayed. Information about command-based menus is divided into the following sections:

- [*Implementing command-based menus*](#)
- [*Sample menu script*](#)
- [*Automatic menu clean-up*](#)

Implementing command-based menus

By default, an application uses form-based menus. To enable command-based menus, you must add the [`Tools EnableCommandMenus\(\)`](#) function to the Startup script for the application. Once this function has run, any form-based menus for the Main Menu form will be removed. You must then add menus using commands from the [*Menu bar function library*](#).

Before you write the script to implement the menus, you must create the commands and command lists that you will use for them. Refer to [*Chapter 15, "Commands,"*](#) for information about creating commands and command lists.

Each menu is created by the [`MenuBar AddMenu\(\)`](#) function, which adds a command list to the menu bar. Once the menu is added, you can use the other functions from the [*Command function library*](#) and the [*Command list function library*](#) to add items to the menu.

Sample menu script

The following procedure from the Real Estate Sales Manager sample application creates command-based menu. It creates a set of menus that use command lists defined in the sample dictionary, as well as built-in command lists.

```
local integer result;  
local integer menu_tag;  
local integer submenu_tag;  
  
{Add the top-level menus}
```

```

{File menu}
menu_tag = Command_GetTag(command CL_File of form 'Main Menu');
result = MenuBar_AddMenu(menu_tag);

CommandList_Add(menu_tag, Command_GetTag(command cmdPrint));
CommandList_Add(menu_tag, Command_GetTag(command cmdPrintSetup));
CommandList_Add(menu_tag, Command_GetTag(command cmdSeparator));
CommandList_Add(menu_tag, Command_GetTag(command
cmdQuitApplication));

{Edit menu}
result = MenuBar_AddMenu(Command_GetTag(command cmdListEdit));

{Tools menu}
menu_tag = Command_GetTag(command CL_Tools of form 'Main Menu');
result = MenuBar_AddMenu(menu_tag);

CommandList_Add(menu_tag, Command_GetTag(command CL_Export of form
➤ 'Main Menu'));
submenu_tag = Command_GetTag(command CL_Export of form 'Main Menu');
CommandList_Add(submenu_tag, Command_GetTag(command CMD_ExportAccess
➤ of form 'Main Menu'));
CommandList_Add(submenu_tag, Command_GetTag(command CMD_ExportExcel
➤ of form 'Main Menu'));
CommandList_Add(submenu_tag, Command_GetTag(command CMD_ExportWord of
➤ form 'Main Menu'));
CommandList_Add(submenu_tag, Command_GetTag(command CMD_ExportXML of
➤ form 'Main Menu'));

CommandList_Add(menu_tag, Command_GetTag(command cmdListMacro));
CommandList_Add(menu_tag, Command_GetTag(command
➤ cmdListResourceDescriptions));

{View menu}
menu_tag = Command_GetTag(command CL_View of form 'Main Menu');
result = MenuBar_AddMenu(menu_tag);

CommandList_Add(menu_tag, Command_GetTag(command CMD_Buyers of form
➤ 'Main Menu'));
CommandList_Add(menu_tag, Command_GetTag(command CMD_Sellers of form
➤ 'Main Menu'));
CommandList_Add(menu_tag, Command_GetTag(command CMD_Houses of form
➤ 'Main Menu'));

```

```

CommandList_Add(menu_tag, Command_GetTag(command cmdSeparator));
CommandList_Add(menu_tag, Command_GetTag(command CMD_Explorer of form
➤ 'Main Menu'));

{Reports menu}
menu_tag = Command_GetTag(command CL_Reports of form 'Main Menu');
result = MenuBar_AddMenu(menu_tag);

CommandList_Add(menu_tag, Command_GetTag(command CMD_BuyersReport of
➤ form 'Main Menu'));
CommandList_Add(menu_tag, Command_GetTag(command CMD_SellersReport of
➤ form 'Main Menu'));
CommandList_Add(menu_tag, Command_GetTag(command CMD_HousesReport of
➤ form 'Main Menu'));

{Window menu}
result = MenuBar_AddMenu(Command_GetTag(command cmdWindowList));

{Help menu}
menu_tag = Command_GetTag(command CL_Help of form 'Main Menu');
result = MenuBar_AddMenu(menu_tag);

CommandList_Add(menu_tag, Command_GetTag(command cmdLookup));
CommandList_Add(menu_tag, Command_GetTag(command cmdShowRequired));
CommandList_Add(menu_tag, Command_GetTag(command cmdSeparator));
CommandList_Add(menu_tag, Command_GetTag(command cmdHelpContents));
CommandList_Add(menu_tag, Command_GetTag(command cmdHelpSearch));
CommandList_Add(menu_tag, Command_GetTag(command cmdSeparator));
CommandList_Add(menu_tag, Command_GetTag(command cmdHelpOnWindow));
CommandList_Add(menu_tag, Command_GetTag(command cmdSeparator));
CommandList_Add(menu_tag, Command_GetTag(command cmdAboutBox));

```

Automatic menu clean-up

When you add command lists to the menu structure, the menu actually displays a visual representation of the command list. It automatically adjusts for hidden or disabled commands in the menu. For example, if a command is hidden by the `hide command` statement, it won't appear in any menus based on command lists that contain the command.

The runtime engine automatically adjusts for any visual issues that can result from hiding commands. For example, assume a command is surrounded by two separators. If the command is hidden, the two separators would be next to each other in the menu. However, the runtime will detect this condition and display only one separator. This only affects the display. The content of the command lists remains unchanged.

Chapter 17: Context Menus

A context menu appears when the user right-clicks a field, window, or scrolling window line in an application. It also appears when the user presses the Property key. When the user chooses to display a context menu, a script is run that allows commands to be added to the context menu.



You can use context menus only when command-based menus are being used for the application. Context menus are not supported when form-based menus are used.

For applications that integrate with Microsoft Dynamics GP, you can use the context script directly for the forms you create. You can also register focus triggers for context menu scripts defined in the core application. In the trigger processing procedure you can add your own commands to the context menu. Refer to [Chapter 9, “Focus Triggers,”](#) in the Microsoft Dynamics GP Integration Guide for more information.

Information about context menus is divided into the following sections:

- [Using context menus](#)
- [Displaying a context menu](#)
- [Context menus and macros](#)

Using context menus

A script for the context menu allows the contents of the menu to be defined. You can add context menus to the following places in an application:

- Window fields
- Windows
- Scrolling window fields
- Scrolling window lines

For browse-only scrolling windows, the context menu applies to the entire active line. For editable and adds-allowed scrolling windows, the line-level context menu items are added after any items for the field-level context menus.

The **cmdListContextMenu** command list is the built-in command list that contains the commands and command lists that define the contents of the context menu.

Automatically-added items

For many editable field types, the context menu is automatically populated with several standard editing commands. The following table lists the commands that are automatically added.

Field type	Context menu commands
Integer Long integer Tiny integer String Text Currency Variable currency Date Time Combo Box	Undo Cut Copy Paste Clear Select All
Picture	Undo Cut Copy Paste Clear

No commands are automatically added to context menus for windows or scrolling windows.

Adding context menu items

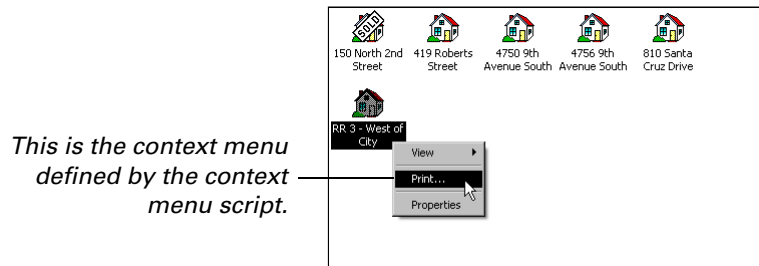
Use the context menu script to add items to the `cmdListContextMenu` built-in command list that defines the contents of the context menu. Keep in mind the following when adding context menu items:

- Add commands and command lists to the `cmdListContextMenu` just like you would add them to any other command list.
- Items in the context menu are not persistent. They must be added by the context menu script each time the context menu is displayed.
- If the context menu can have items added automatically, these items will have already been added to the `cmdListContextMenu` command list when the context menu script is run.



You can use the [*CommandList RemoveAll\(\)*](#) function to remove the commands that are automatically added to the `cmdListContextMenu` command list used for the context menu.

The following example is the context menu script for the Explorer_List field in the Real Estate Sales Manager sample application. Commands were added to the RESM_Explorer form that perform the view actions, print the current item, and display the properties for the current item. The context menu script defines the View submenu and adds the view commands to it. Then the script adds the View submenu, separators, and additional commands to the context menu. The following illustration shows the context menu when it is displayed.



```

local integer result;
local integer command_tag;
local integer view_menu_command_list;
local integer context_menu_command_list;

{Build the View submenu}
view_menu_command_list = Command_GetTag(command View of form
RESM_Explorer);

{Only need to build this once. Unlike the context menu, it is not
cleared
automatically when the context menu action is complete.}
if CommandList_NumCommands(view_menu_command_list) = 0 then
    {Large Icon}
    command_tag = Command_GetTag(command Large_Icon of form
    ► RESM_Explorer);
    result = CommandList_Add(view_menu_command_list, command_tag);

    {Small Icon}
    command_tag = Command_GetTag(command Small_Icon of form
    ► RESM_Explorer);
    result = CommandList_Add(view_menu_command_list, command_tag);

```

```

{List}
command_tag = Command_GetTag(command List of form RESM_Explorer);
result = CommandList_Add(view_menu_command_list, command_tag);

{Report}
command_tag = Command_GetTag(command Report of form
➡ RESM_Explorer);
result = CommandList_Add(view_menu_command_list, command_tag);

end if;

{Build the context menu}
context_menu_command_list = Command_GetTag(command
cmdListContextMenu);

{Add the View sub-menu}
result = CommandList_Add(context_menu_command_list,
view_menu_command_list);

{Separator}
command_tag = Command_GetTag(command cmdSeparator);
result = CommandList_Add(context_menu_command_list, command_tag);

{Print}
command_tag = Command_GetTag(command Print of form RESM_Explorer);
result = CommandList_Add(context_menu_command_list, command_tag);

{Separator}
command_tag = Command_GetTag(command cmdSeparator);
result = CommandList_Add(context_menu_command_list, command_tag);

{Properties}
command_tag = Command_GetTag(command Properties of form
➡ RESM_Explorer);
result = CommandList_Add(context_menu_command_list, command_tag);

```

Displaying a context menu

The context menu appears only when it has items defined for it. These can be the commands that are automatically added or the commands added by the context menu script.

The context menu automatically closes when an item is chosen. You can also close the context menu by pressing the Esc key or clicking somewhere outside of the context menu.

When a user right-clicks a field that is editable, the runtime engine will attempt to move the focus to that field. The insertion point will be placed where the user clicked. If text in the field is already selected, and the user clicks within the selected text, the selection will remain. If the user clicks outside of the selected text, the selection will be cleared and the insertion point will be placed where the user clicked.

If no field in the window has focus when the user chooses to display the context menu, such as by pressing the property key, the context menu script for the window will be run.

Context menus and macros

Context menu actions are recorded when macros are recorded. When macros are played back, the context menus are not displayed, but the menu actions are still performed.

The contents of the context menu can be saved to a file. To do this, you must be displaying the Advanced Macro menu. When recording a macro with the Advanced Macro menu displayed, a Dump Context Menu command will be added to the context menu. When you choose Dump Context Menu, a file dialog opens that allows you to save the contents of the context menu to a file.

Chapter 18: Toolbars

Toolbars are typically used to provide shortcuts to items that are displayed on menus. Each toolbar is based on a command list, and displays the commands that are defined for the list. Information about toolbars is divided into the following sections:

- [Implementing toolbars](#)
- [Sample toolbar script](#)
- [Re-creating toolbar arrangement](#)
- [Automatic toolbar clean-up](#)

Implementing toolbars

To implement toolbars, you must first create commands and command lists for the toolbar. Then you will use a script to add the toolbars to your application.

Toolbar commands

You must create the commands and command lists that you will use for the toolbars in your application. Refer to [Chapter 15, “Commands,”](#) for information about creating commands and command lists.

For the best appearance, each command that you plan to use on a toolbar should have a 20 pixel by 20 pixel native picture defined for it. Each command should also have an appropriate tooltip.



Remember that you can use the built-in commands on toolbars as well.

Creating toolbars

The [CommandBar Create\(\)](#) function is used to add toolbars to the application. This function creates a toolbar from the specified command list. If the command list for the toolbar contains another command list, that list will appear as a drop-down menu on the toolbar.

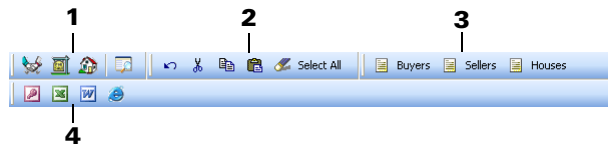


By default, the toolbars will be displayed on a single row. If the toolbars contain too many items to fit on that row, they will wrap to the next row. You can use the optional *sequence* and *row* parameters of **CommandBar_Create()** function to specify where each toolbar is created.

The *sequence* value for a toolbar indicates its relative position.

For applications using the multiple-document interface (MDI), the first toolbar in the upper-left corner has the sequence value 1. The toolbar next to it has the value 2, and so on for that row. The toolbars on the next row simply continue the sequence.

The following illustration shows the sequence numbers for 4 toolbars displayed for the Real Estate Sales Manager sample application.



For applications using the single-document interface (SDI), the sequence is the horizontal distance of the left edge of the toolbar, measured in pixels from the left edge of the main window.

How you use the *sequence* parameter when adding toolbars depends on whether your application is running in multiple-document or single-document mode. For MDI applications, start the value 1 for the first toolbar you add, the value 2 for the second, and so on. A sequence number shouldn't repeat. For SDI applications, supply the actual offset from the left edge of the window as the sequence value.

The *row* parameter specifies the row you want the toolbar to appear in. The value 1 specifies being the first row. When adding toolbars, begin by adding those in the first row (row value 1), then those in the second row (row value 2), and so on.

Removing toolbars

Toolbars can be removed by using the **CommandBar_Destroy()** function. Removing a toolbar leaves the underlying command list unchanged.

Sample toolbar script

The following is the procedure from the Real Estate Sales Manager sample application that creates the toolbars. Four toolbars are created.

```
local integer result;

{View items}
result = CommandBar_Create(Command_GetTag(command CL_View of form
➤ 'Main Menu'),1, 1);

{Edit items}
result = CommandBar_Create(Command_GetTag(command cmdListEdit), 2,1);

{Report items}
result = CommandBar_Create(Command_GetTag(command CL_Reports of form
➤ 'Main Menu'),3,1);

{Export items}
result = CommandBar_Create(Command_GetTag(command CL_Export of form
➤ 'Main Menu'), 4, 2);
```

Notice that the sequence number is incremented as each toolbar is added. Also notice that the first three toolbars are added to the first row, so the *row* parameter is set to 1. The last toolbar is added to row 2, so its *row* parameter is set to 2.

Re-creating toolbar arrangement

The user can arrange the toolbars any way they like. You may want your application to remember the user's toolbar arrangement, and then re-create that arrangement when the application is used again. To do this, use the [CommandBar_GetPosition\(\)](#) function.

Keep track of the toolbars (command lists) that the user has displayed. Use the **CommandBar_GetPosition()** function to retrieve the sequence and row for each toolbar, and save these values in a table or some other permanent storage. Typically this is done when the application is shut down.

To re-create the toolbar arrangement, use the [CommandBar_Create\(\)](#) function to add the toolbars in the order indicated by the sequence values you saved. Be sure to use the corresponding row value you saved for each toolbar. The toolbars will appear in the same order the user had specified.

Automatic toolbar clean-up

When you display a command list as a toolbar, the toolbar actually displays a visual representation of the command list. It automatically adjusts for hidden or disabled commands in the toolbar. For example, if a command is hidden by the [hide command](#) statement, it won't appear in any toolbars that contain the command.

The runtime engine automatically adjusts for any visual issues that can result from hiding commands. For example, assume a command is surrounded by two separators. If the command is hidden, the two separators would be next to each other in the toolbar. However, the runtime will detect this condition and display only one separator. This only affects the display. The content of the command lists remains unchanged.

Part 5: Additional Resources

This part of the documentation provides detailed information about additional resources that can be part of Dexterity applications. Each chapter contains detailed information about the resource and a step-by-step procedure describing how to create and use the resource. Refer to the procedural information when creating your own applications. Following is a list of the topics that will be discussed, with a brief explanation of each:

- [Chapter 19, "Messages,"](#) describes how to use messages in your application.
- [Chapter 20, "Pictures and Native Pictures,"](#) explains how to use pictures in your application.
- [Chapter 21, "Constants,"](#) explains how to create and use constants.
- [Chapter 22, "Global Variables,"](#) describes global variables and how to use them in an application.
- [Chapter 23, "Strings,"](#) explains how to change string resources in a Dexterity application.
- [Chapter 24, "Libraries,"](#) describes how to use and create libraries in a Dexterity application.
- [Chapter 25, "Icons,"](#) describes how to use icon resources in an application.
- [Chapter 26, "Table Groups,"](#) explains how to use table groups in your application.
- [Chapter 27, "Virtual Tables,"](#) describes virtual tables and how to use them in your application.

Chapter 19: Messages

Messages are statements or questions that can be used to provide information or prompt a user to make a selection. Information about messages is divided into the following sections:

- [Messages overview](#)
- [Messages examples](#)
- [Using product names](#)
- [Procedure: Creating messages](#)

Messages overview

Messages are entered using the Messages window and can be referenced in scripts using the [getmsg\(\)](#) function. Messages can also contain replacement markers, allowing you to substitute text for the replacement markers using the [substitute](#) statement.

Each message has a message ID, which is used when referencing the message in a script. To view a list of messages and IDs in an application dictionary, use the Global Resources report in Dexterity Utilities.



If you are adding messages to an application that integrates with Microsoft Dynamics GP, be sure the message IDs for messages you create are 22,000 or greater. Otherwise, your messages won't be extracted when you package your application.

The use of messages allows you to easily update messages in an application dictionary or translate them for a multi-lingual application. All the messages can be accessed through the Messages window so you don't need to change or recompile any scripts in your application.

Messages examples

The following example show how messages can be used in an application.

Example 1

The following message has the indicated value.

Message ID	100
Message	You must enter your password.

The following script uses the `getmsg()` function to retrieve the message and the `warning` statement to display a warning for the user. The message “You must enter your password.” will be displayed in a warning dialog box.

```
local string Message;

{Retrieve the message using the message ID.}
Message = getmsg(100);
warning Message;
```

Example 2

Additional strings can be added to messages by inserting replacement markers in the message indicating where the items will be substituted. A replacement marker is a percent symbol (%) followed by a number.

The following message has the indicated value.

Message ID	101
Message	The module %1 could not be opened because %2.

The following script uses the `getmsg()` statement to retrieve the message. The `substitute` statement substitutes the two text items for the replacement markers within the message. The `error` statement displays the message “The module Inventory could not be opened because you don’t have access privileges.” in an error dialog.

```
local string Message;

{Retrieve the message using the ID for the desired message.}
Message = getmsg(101);
{Substitute the items for replacement markers in the message.}
substitute Message, "Inventory", "you don't have access privileges";
error Message;
```

Using product names

In some cases, you may find it necessary to use a product name in a message. This is especially true when you create products that integrate with Microsoft Dynamics GP. Rather than hard-coding the product name in your messages, we recommend that you use a token instead.

The token to indicate the product name in a message or static string is **@PROD*prodID*@** where *prodID* is the product ID of the product whose name you want to include in the message. The product name will be read automatically from the launch file. For example, the following message includes the name of the currently-active product, which has a product ID 0:

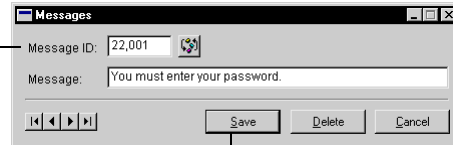
Message ID	22001
Message	The current product is @PROD0@.

If the token specifies a product for which a product name cannot be found, the string [unknown] is substituted.

Procedure: Creating messages

To create a message, point to New in the Explorer menu and choose Message. The Messages window will appear, as shown in the following illustration.

Enter the Message ID here. If the message exists, it will be retrieved.



Click Save to save the message.

1. Enter or edit the message.

Enter the message ID and the message. If the ID is already used, the message associated with it will be displayed and can be edited. To delete an existing message, click Delete.



If you are adding messages to an application that integrates with Microsoft Dynamics GP, be sure the message IDs for messages you create are 22,000 or greater. Otherwise, your messages won't be extracted when you package your application.

2. Save the message.

Click Save to save the message.

Related items

Use the information listed to learn more about creating and using messages.

Commands

[ask\(\)](#), [error](#), [getmsg\(\)](#), [substitute](#), [warning](#), [Utility_SubstituteTokens\(\)](#)

Additional information

[Transferring strings and messages](#) in [Chapter 3, "Transfer Utilities,"](#) of the Dexterity Utilities manual.

Chapter 20: Pictures and Native Pictures

Dexterity applications use two types of pictures: pictures and native pictures. Pictures are stored in a generic format and can be displayed on any platform. They are typically used to display large graphics and logos in an application. Pictures are also used for some controls, such as tree views, list views, push buttons and button drop lists. Native pictures are pictures that are used only on a particular platform. They are used to display pictures on push buttons and visual switches.

Information about pictures and native pictures is divided into the following sections:

- [Pictures](#)
- [Procedure: Adding a picture to the picture library](#)
- [Procedure: Using a picture from the picture library](#)
- [Native pictures](#)
- [Procedure: Creating native pictures](#)
- [Procedure: Synchronizing native pictures](#)

Pictures

Pictures are stored in a generic format and can be displayed on any platform. Currently, Dexterity can convert pictures up to 32K in size to a form that can be stored in the application dictionary's *picture library*. Pictures are stored only once, but can be placed in several windows of an application or on a report, using the picture tool from the Toolbox window.

A company or product logo can be pasted into the picture library and used in application windows and reports. The following picture is used in the Real Estate Sales Manager application.

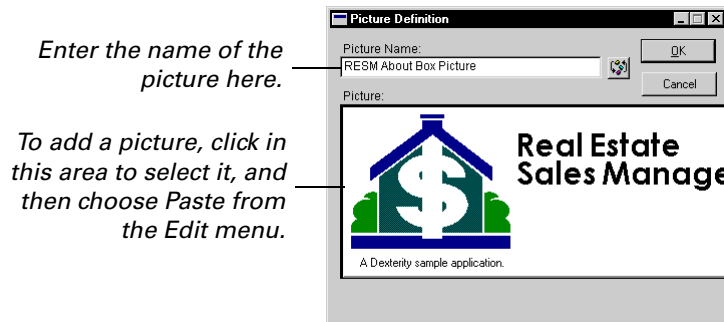


Procedure: Adding a picture to the picture library

To create a new picture, point to New in the Explorer menu and choose Picture. The Picture Definition window will appear.

1. Create a new picture or edit an existing one.

If you are editing an existing picture, it will be displayed in the Picture Definition window.



2. Name the picture.

In the Picture Definition window, name the picture.

3. Add the picture.

Be sure the picture you want to add is in the clipboard. Click in the area below the Picture Name field to select it, and then choose paste from the Edit menu to paste the picture into that area.

4. Click OK to add the picture to the library.

Procedure: Using a picture from the picture library

Open the window layout or report layout window where you want to paste a picture.

1. Select the picture tool.

2. Click in the layout area where you want the picture to appear.

After you click in the layout area, the Pictures window will appear, allowing you to select a picture to paste into the layout.

3. Select the picture to paste.

From the Pictures window, select the name of the picture to paste into the window and click OK.



You can paste a picture directly into the layout window, bypassing the step of adding the picture to the picture library. Simply copy the picture to the clipboard and choose Paste from the Edit menu to paste it into a window layout. You'll be asked to name the picture. The picture will appear in the layout window and will be added to the picture library automatically.

Native pictures

Metafiles and bitmap images can be used as native pictures. Native pictures are used as the picture static values for push buttons and visual switches.

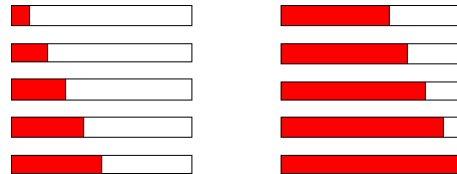
Push buttons

Two native pictures are required for push buttons. The “button up” picture is displayed when the button is not being clicked. The “button down” picture is displayed when the button is being clicked. Often, the button up and button down pictures are drawn to produce a three-dimensional effect (offsetting the button down picture one pixel down and to the right) when the button is pushed. To have the best appearance, the native pictures used should be the same physical size.

Visual switches

Visual switches can be used to display a series of native pictures. For each image displayed, only one native picture is required. To have the best appearance, all native pictures used for the visual switch should be the same physical size.

The following example shows the ten native pictures used for a visual switch in a progress control window. Based upon the value of the visual switch, the corresponding picture is displayed, indicating progress.

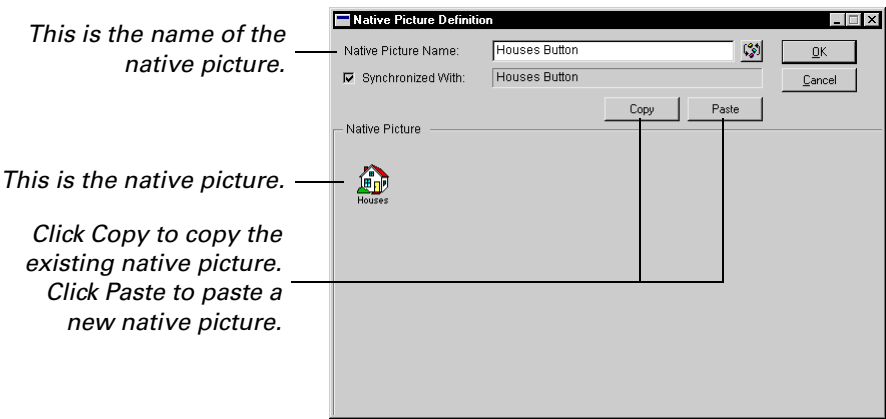


Procedure: Creating native pictures

To create a native picture, point to New in the Explorer menu and then choose Native Picture. The Native Picture Definition window will appear.

1. Create a new native picture or edit an existing one.

If you are editing an existing native picture, it will appear in the Native Picture Definition window.



2. Name the native picture.

In the Native Picture Name field, enter the name of the native picture.

3. Add the native picture.

Be sure the picture you want to add is in the clipboard. Click Paste to paste the picture into the Native Picture Definition window.

4. Click OK to save the native picture.

Procedure: Synchronizing native pictures

If your application will be used on more than one platform, you must create corresponding native pictures on both platforms. These pictures must be synchronized (given the same internal ID) so the correct picture will be displayed on each platform.

- 1. Create the native picture on the first platform**

When you create the first native picture, be sure the Synchronize With option in the Native Picture Definition window is unmarked.

- 2. Open the dictionary on the next platform.**

- 3. Create the corresponding native picture for the new platform.**

Before you save the native picture, mark the Synchronize With option. A list of pictures from the first platform will be displayed.

- 4. Select the name of the appropriate picture from the first platform and click OK.**

The pictures will automatically be given the same internal ID.

Chapter 21: Constants

A constant is a fixed numeric or string value used in scripts, where the constant name is used in place of the value associated with it. You can define constants for commonly-used values in an application; for instance, for the number of billing periods in a year or common return values from a user-defined function. The constant is used the same way its value would be, with the advantage that the constant name is easy to remember.

Dexterity supports global constants and form-level constants. Dexterity also has predefined constants used for actions such as handling errors, controlling dialog boxes, applying formats and generating sounds.



If you need to learn the value of a constant, the easiest way is to paste the constant into the Expressions window in Dexterity and click Evaluate. You can do this in both tools mode and test mode.

Information about constants is divided into the following sections:

- [Global constants](#)
- [Procedure: Creating global constants](#)
- [Form-level constants](#)
- [Procedure: Creating form-level constants](#)
- [Predefined constants](#)
- [Constant examples](#)

Global constants

Global constants are accessible by every script in an application. You should use global constants when you anticipate a constant will be used by several scripts throughout your application.

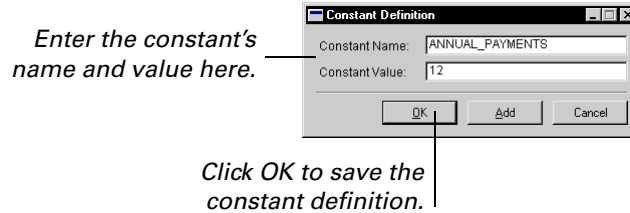
If you change the value of the constant, you must recompile all of the scripts that use the constant. To compile several scripts at once, choose Compile or Compile All from the Explorer menu in Dexterity.



If a constant is already used in scripts and you change its value, you must recompile all the scripts where the constant is used. If you create applications that integrate with Microsoft Dynamics GP, you shouldn't change any constants for this reason.

Procedure: Creating global constants

To create a global constant, point to New in the Explorer menu and choose Constant. The Constant Definition window will appear.



1. Name the constant and enter a value.

If you are creating a string constant that has trailing spaces, you must enclose the string in quotation marks. If you want to include quotation marks in the constant, the entire constant value must be enclosed in quotation marks.

2. Click OK to save the constant definition.

Form-level constants

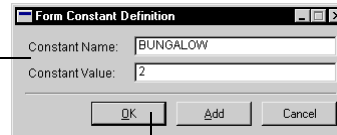
Form-level constants are designed to be used by a specific form in the application. You should use a form-level constant when several scripts on a form use a constant, but there is little need to use the constant elsewhere in the application.

Within the scope of a form, you can reference a form-level constant directly. To reference a form-level constant from outside the scope of a form, use the qualifier **of form** *form_name*. If a form-level constant and a global constant have the same name, by default, scripts attached to the form will use the form-level constant. To refer to the global constant instead, use the qualifier **of globals** following the name of the constant.

Procedure: Creating form-level constants

To create a form-level constant, open the Form Definition window for the form where the constant will be created. Select the Constants tab. Click New to create a new constant, or select a constant and click Open to modify it; the Form Constant Definition window will appear.

Enter the constant's name and value here.


 A screenshot of the 'Form Constant Definition' dialog box. It has two text input fields: 'Constant Name:' with the value 'BUNGALOW' and 'Constant Value:' with the value '2'. At the bottom are three buttons: 'OK', 'Add', and 'Cancel'.

Click OK to save the constant definition.

1. Name the constant and enter a value.

If you are creating a string constant that has trailing spaces, you must enclose the string in quotation marks. If you want to include quotation marks in the constant, the entire constant value must be enclosed in quotation marks.

2. Click OK to save the constant definition.

Predefined constants

Dexterity has predefined constants for several maximum and minimum numeric values. These constants are listed in the following table.

Constant	Value	Description
MINTINY	0	The smallest tiny integer
MAXTINY	255	The largest tiny integer
MININT	-32768	The smallest integer
MAXINT	32767	The largest integer
MINLONG	-214748348	The smallest long integer
MAXLONG	2147483647	The largest long integer

Constants are available for individual characters. For example, CH_D is a constant representing the string “D”. The constant CH_h is a constant representing the string “h”. These individual character constants are available for all alphanumeric US characters, 0-9, a-z, and A-Z.

Dexterity also has several predefined constants that are used for various functions, statements and procedures. These constants are listed with each statement or function they are used with. The constants used with the [ask\(\)](#) function are shown in the examples in the next section.

Constant examples

The following examples show how constants can be used in an application.

Example 1

The following constant represents the number of payments in a year.

Constant Name	ANNUAL_PAYMENTS
Constant Value	12

The following script uses the constant to calculate the amount of the monthly payment.

```
'Monthly Payment' = 'Total Cost' / ANNUAL_PAYMENTS;
```

The constant is used the same way its value would be, with the advantage that the constant name is easy to remember.

Example 2

The following constant represents a specific style of house. The constant is defined for the Houses form.

Constant Name	BUNGALOW
Constant Value	2

The following script is part of the Buyers form. It uses the constant from the Houses form to set house type. Notice that the qualifier **of form** *form_name* is used to indicate where the constant is defined.

```
'Type' of window Buyers = BUNGALOW of form Houses;
```

Example 3

The following example shows how the ASKBUTTON constants are used with the `ask()` function to display a message in a dialog box. The ASKBUTTON constants are always used in conjunction with the `ask()` function.

The `ask()` function returns the appropriate ASKBUTTON value, corresponding to the button the user clicks. The `if then...end if` statement is used to ascertain which button was clicked so the appropriate action can be taken.

```
local integer Choice;

Choice = ask("Which option do you want?", "First","Second","Third");
if Choice = ASKBUTTON1 then
    run script First_Choice;
elseif Choice = ASKBUTTON2 then
    run script Second_Choice;
else
    run script Third_Choice;
end if;
```


Chapter 22: Global Variables

Global variables are available to any script in the application dictionary at any time. You can create them from global fields in your application dictionary using the Resource Explorer.

Information about global variables is divided into the following sections:

- [*Global variables overview*](#)
- [*Global variables example*](#)
- [*Procedure: Creating global variables*](#)

Gobal variables overview

Refer to [*Chapter 1, "Syntax,"*](#) in Volume 2 of the *Dexterity Programmer's Guide* for more information about using global variables in scripts.

Global variables remain active until you exit the application. Memory space for global variables is allocated when the application starts. Numeric global variables are initialized to zero, string global variables to empty, and boolean global variables to false. Typically, the form pre script on the main menu sets the initial values of the global variables.

You can reference global variables from any script by using the qualifier **of globals** after the name of the variable. The values of global variables can't be stored directly in a file, but their values can be copied to fields that are stored in a table.

Global variables example

In this example, the Total and Warning Flag fields are global variables. The following two script statements add the value in the Current Amount field to Total and set Warning Flag to false.

```
Total of globals = Total of globals + 'Current Amount';  
'Warning Flag' of globals = false;
```

Procedure: Creating global variables

Use the following procedure to create global variables.

- 1. Create global field definitions (if necessary).**

If you haven't already done so, use the Field Definition window to define the global fields that will be used for global variables.

- 2. View the global variables that have been defined for the application.**

In the Resource Explorer, choose Globals in the Resources list.

- 3. Create new global variables.**

Click New in the Resource Explorer or point to New in the Explorer menu and choose Global. The Field Lookup window will be displayed, listing all of the global fields. Select a field in the list and click OK. A global variable will be created based on the field you selected.

Related items

Use the information listed to learn more about creating and using global variables.

Additional information

[Chapter 8, "Global Fields"](#) in Volume 1 of the Dexterity Programmer's Guide

[Chapter 1, "Syntax"](#) in Volume 2 of the Dexterity Programmer's Guide

Chapter 23: Strings

A string is a sequence of up to 79 characters that doesn't contain carriage returns. Strings are used throughout an application dictionary for window names, field prompts, static text values and report text.

Information about strings is divided into the following sections:

- [*Strings overview*](#)
- [*Procedure: Modifying a string*](#)

Strings overview

Using the strings resource, every occurrence of a string used in a dictionary can be viewed and updated throughout the application, wherever it occurs. This allows you to update all occurrences of a string in one step; for instance, if you're updating certain terms or translating a dictionary to another language, you can change a string resource once instead of changing the same string in each place it occurs.

String resources are useful for translating an application dictionary to a different language or for accommodating different terms used in other regions or businesses. For instance, to change all occurrences of the words "Customer Name" to "Client Name," you could select the Customer Name string and change it to Client Name instead of changing each individual prompt, text value or window name.

Procedure: Modifying a string

Use the following procedure to modify strings.

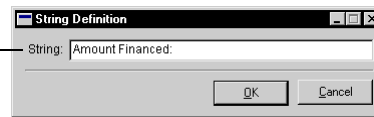
1. View the strings for the current dictionary.

In the Resource Explorer, choose Strings in the Resources list.

2. Select the string to edit.

Select the string you wish to edit and click Open; the String Definition window will appear, as shown in the following illustration.

*Edit the string in this field
and click OK to save the
changes.*



3. Edit the string.

Edit the string in the String Definition window and click OK to save the modified text string.

Related items

Use the information listed to learn more about creating and using string resources.

Additional information

[Transferring strings and messages](#) in [Chapter 3, "Transfer Utilities."](#) in the Dexterity Utilities manual

Chapter 24: Libraries

Libraries are separate files that contain resources or functionality that can be used within a Dexterity-based application. To use a library, you must create a reference to it. Dexterity supports two types of library files: COM type libraries and resource libraries. Information about libraries is divided into the following sections:

- [*Library types*](#)
- [*Procedure: Adding a library reference*](#)
- [*Creating a resource library*](#)

Library types

Dexterity-based applications support two types of libraries: COM type libraries and resource libraries.

COM type libraries

A COM type library is a special type of file that uses a standard format to describe the interfaces, classes, methods, properties, and events an application makes available through COM. Most applications that implement COM will provide a type library that describes the implementation. A Dexterity-based application can have references to COM type libraries so that code can be written to access the functionality the COM interface is providing. Refer to [*Chapter 37, "COM Libraries,"*](#) in Volume 2 of the Dexterity Programmer's Guide for details about using COM type libraries.

Resource libraries

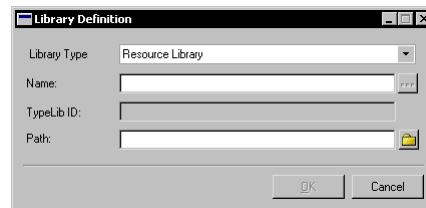
Resource libraries contain applications resources, such as icons, that can be used within a Dexterity-based application. To use the resources in the resource library, you first create a reference to it. Then you create references to the specific resources within resource library.

Procedure: Adding a library reference

Use the following procedure to add a library reference.

1. Open the Library Definition window.

In the Resource Explorer, choose Libraries in the Base group for the current dictionary. Click the New button in the Resource Explorer to create a new library reference.



2. Select the library type.

Choose one of the following types:

COM Type Library The library being referenced provides functionality through COM.

Resource Library The library being referenced contains resources that will be used in the application.

3. Name the library reference.

If you are creating a reference to a resource library, supply a name for the reference. The name should indicate what resources are available in the library.

4. Select a registered type library or type library file.

If you are creating a reference to a COM type library, click the Name lookup button to open the COM Type Libraries window, which lists all of the type libraries that have been registered on the current machine. Select the appropriate library and click Select.

If the type library file hasn't been registered, click the Path lookup button in the Library Definition window. Use the File dialog to select the type library file for the application you want to reference.

5. Select the path to the library file.

If you are creating a reference to a resource library, click the Path lookup button in the Library Definition window. Use the file dialog to select the resource library file that contains the resources you want to reference.

6. Save the library reference.

Click OK to save the library reference.

Creating a resource library

You can create a resource library (a DLL) that contains resources used for your Dexterity-based application. A resource library is a standard Win32 DLL created with a development tool such as Microsoft Visual Studio®.

The following procedure describes how to use Visual Studio 2005 to create a resource library file that can be used by a Dexterity-based application.

1. Start Visual Studio.

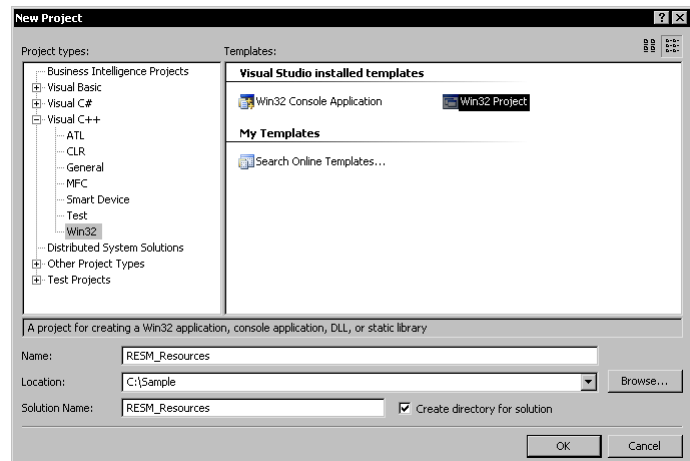
If it isn't running, start Visual Studio.

2. Create a new project.

In the File menu, choose to create a new project.

3. Select the project type.

In the New Project window, expand the C++ project types. Select Win32 as the type of project to create. Choose the Win32 Project template.

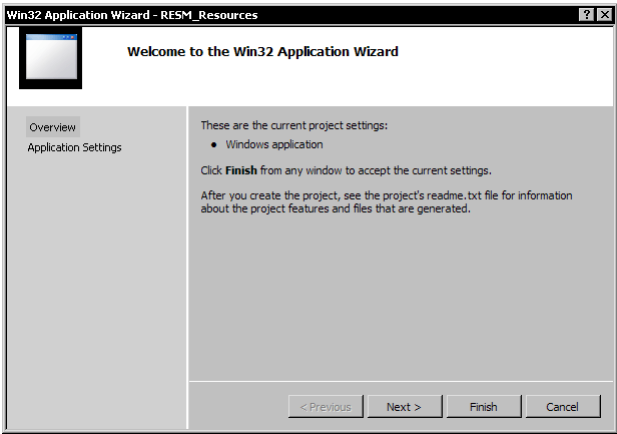


4. **Name the project.**

Specify the name and location for the new project.

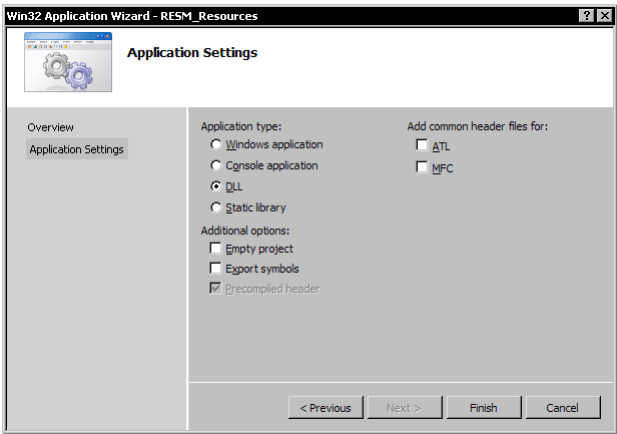
5. **Create the project.**

Click OK to create the project. The Win32 Application Wizard will be displayed.



6. **Specify the application settings.**

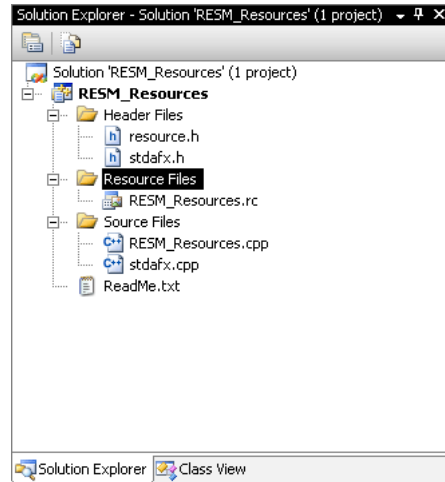
Click the Application Settings link to display the settings for the new project.



Specify DLL as the type of application to create. Click Finish to continue.

7. Add resource files.

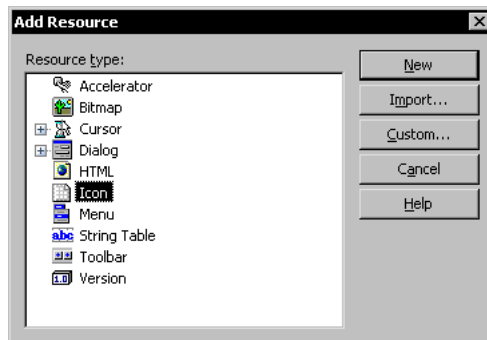
Select the Resource Files group in the Solution Explorer.



In the Project menu, choose Add Resource. The Add Resource window will be displayed.

8. Choose the resource type.

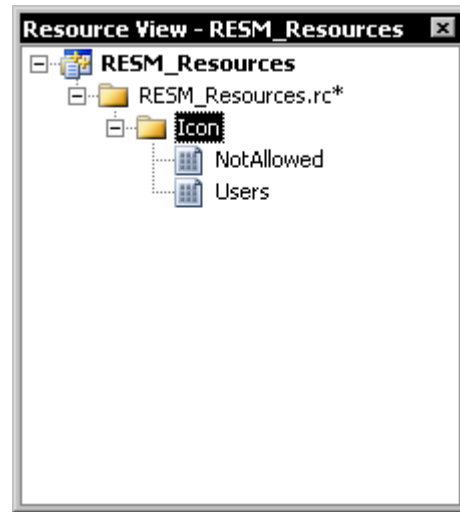
In the Add Resource window, choose the type of resource you want to add. In the following example, Icon resources are being added.



9. Create or import resources.

Click New to create new resources of the specified type. Click Import to add resources of the specified type. For example, if you chose Icon as the resource type, you could import files of with the .ico extension.

Once a resource file has been added to the project, you can use the resource editor within Visual Studio to add additional resources to it. For example, the following illustration shows icons added to a resource file.

**10. Build the solution.**

After the resources have been added, choose Build Solution from the Build menu. The resource library file contains the resources you added will be created.

Chapter 25: Icons

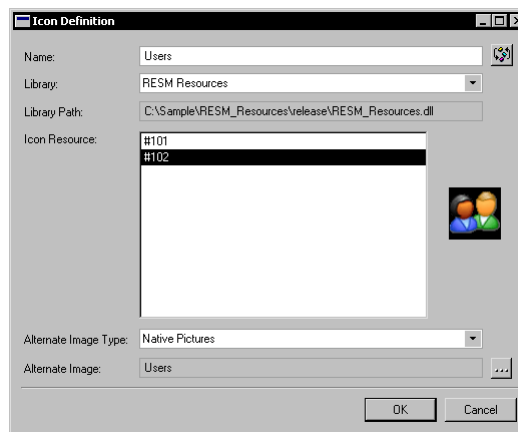
Icon resources from a resource library can be used for command images. Because they support great color depth and multiple image sizes, it can be preferable to use icon resources for commands. Refer to [Chapter 24, “Libraries,”](#) for information about creating a resource library that contains icon resources.

Information about icons is divided into the following sections:

- [Icon elements](#)
- [Procedure: Creating an icon resource](#)

Icon elements

An icon resource has a name, resource library location, icon resource, and alternate image. The Icon Definition window, shown in the following illustration, is used to set the characteristics of the icon resource.



Name

Each icon resource must have a name. The name should indicate what image is used for the icon.

Library

The library drop-down list specifies what resource library file the icon resource is being read from. When you use an icon resource from a resource library file, you must ship that resource library file with your application.

Icon resource

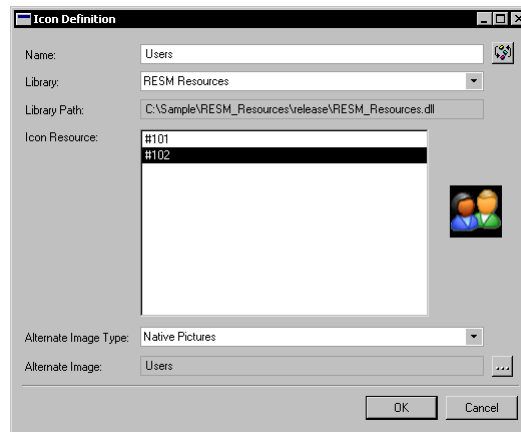
The available icon resources in the specified resource library file will be listed. Choose the icon resource you want to use. The preview image shows the icon selected.

Alternate image

Since the resource library file is separate from the dictionary that uses the icon, it is possible the dictionary could be used without the resource library being available. If the resource library cannot be found, the alternate image will be used in place of the icon. Specify whether a picture or native picture will be used for the alternate image, and then use the alternate image lookup to choose the image.

Procedure: Creating an icon resource

To create an icon resource, point to New in the Explorer menu and choose Icon. The Icon Definition window will appear as shown in the following illustration.



1. Name the icon resource.

Enter a name for the icon resource that identifies the icon being referenced.

2. Select the resource library that contains the icon.

Use the Library drop-down list to select one of the resource libraries that has been added to the current dictionary. Refer to [Chapter 24, "Libraries,"](#) for details about adding a resource library to an application dictionary.

3. Select the icon image to use.

Choose one of the icons in the list of icon resources from the selected resource library. The preview shows the icon selected.

4. Specify the alternate image type.

Specify whether pictures or native pictures will be used for the alternate image. The alternate image will be displayed in cases where the resource library file cannot be accessed by the application.

5. Choose the alternate image.

Use the Alternate Image lookup to choose which image will be used as the alternate image for the icon.

6. Save the icon reference.

Click OK to save the icon reference.

Chapter 26: Table Groups

Some tables in your application may have a logical relationship to each other. For example, invoice information is often divided into two tables. One table contains the header information for each invoice, while the other table contains the line items for the invoice. These two tables have an inherent relationship; each table requires the other. Dexterity allows you to create a *table group*. A table group allows you to group logically related tables under a single name.

Information about table groups is divided into the following sections:

- [Using table groups](#)
- [Procedure: Defining a table group](#)

Using table groups

Refer to [Chapter 5](#),
["Security"](#) in the
Stand-alone
Application Guide for
information about
security in Dexterity
applications.

Table groups have two main purposes in Dexterity: security and table maintenance. In Dexterity applications, table security is based on table groups. Each time a table is accessed, Dexterity examines which table group the table is part of. Based on how security is set for the table group, access to the table is either granted or denied.

In some implementations of table maintenance, the user selects a table group on which maintenance will be performed. The application then determines which tables are part of that table group and performs the maintenance on those tables.

Procedure: Defining a table group

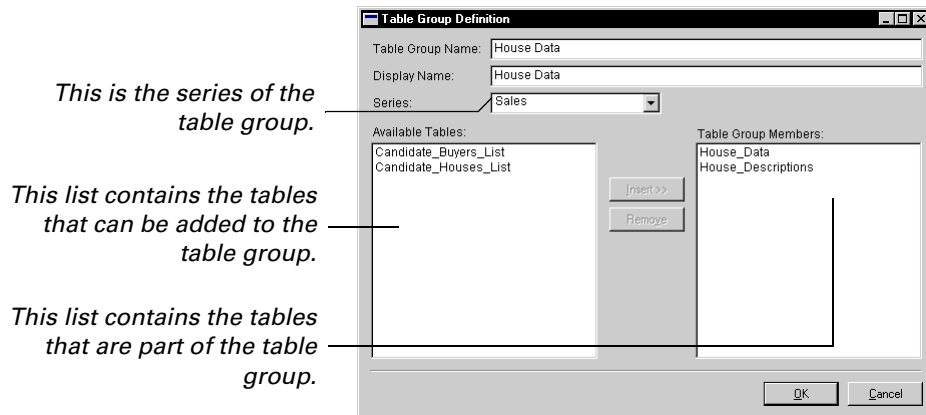
Use the following procedure to create a table group.

1. View the table groups for the current dictionary.

In the Resource Explorer, choose Table Groups in the Base group for the current dictionary.

2. Open a table group or create a new table group.

To view an existing table group, select one in the list and click the Open button in the Resource Explorer. To create a new table group, click the New button in the Resource explorer. The Table Group Definition window will appear.



3. Enter the table group name and display name.

The table group name is used internally by Dexterity to reference and store the table group. In the Table Group Name field, enter the appropriate name for the new table group you are creating or modify the existing name.

In the Display Name field, enter the display name you want to use for the table group. The display name will be used any time the name of the table group appears in your application. The name can be the same as the table group name, or you can enter a different name.

4. Select a series.

Select the series of the table group being created. This series is used to group table groups together, similar to how tables are grouped into series. Typically, the series selected is the same as the series of the tables in the table group.

5. Select tables to be included in the table group.

From the Available Tables list, select the tables you want to include in the table group. Select each table in the Available Tables list and click Insert to add it to the Table Group Members list.

6. Save the table group.

Click OK to save the table group.

Chapter 27: Virtual Tables

Grouping and creating relationships between standard tables can have significant benefits. There are several methods to group and relate tables: table groups, table relationships, and virtual tables. Table groups create a logical grouping of tables and are solely used for maintenance and security purposes. Table relationships allow data stored in several tables to be accessed by the Report Writer. Virtual tables build upon the idea of relating tables. Virtual tables allow data that is stored in separate tables to be read as if it has been stored in a single table.

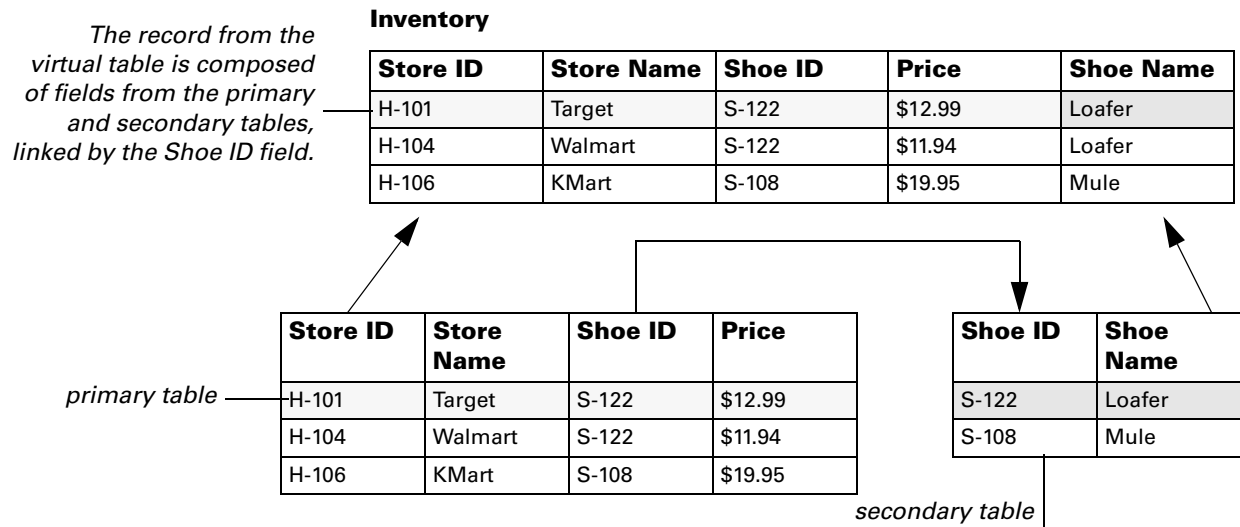
Information about virtual tables is divided into the following sections:

- [*Virtual table concepts*](#)
- [*Virtual table elements*](#)
- [*Member table relationships*](#)
- [*Virtual tables and SQL views*](#)
- [*Using virtual tables*](#)
- [*Procedure: Creating a virtual table*](#)

Virtual table concepts

A virtual table acts as a single, read-only, non-physical table. It contains member tables, member fields, member keys and detailed relationship information. This information allows access to data in multiple related tables through the linking of one table's keys to another table's fields.

A virtual table accesses member tables by utilizing a related field from a primary table and using it as a key in a secondary table. In the following illustration, the Inventory virtual table has stored the related field, Shoe ID, from the main member table, Prices. The key for the secondary table, Shoes, is Shoe ID, which allows the data to be retrieved. At runtime, Dexterity identifies the related fields and completes the record in the virtual table by combining the records from the main and secondary tables.



Virtual table elements

Virtual tables are constructed of member tables, member fields, member keys and detailed relationship information to relate the member tables.

Member tables

Member tables are the standard tables that are joined together to form a virtual table. A virtual table must contain at least one and no more than sixteen member tables.

A virtual table contains a primary table. All other member tables are joined to the primary table, whether directly or through another member table.

The virtual table must be granted read-access to all member tables. If a virtual table's member table has been opened for exclusive use, and the virtual table is opened, an open table error will occur.

Member fields

Member fields come from the member tables and become the virtual table fields.

For more information about keys and tables, refer to [Keys](#) on page 105.

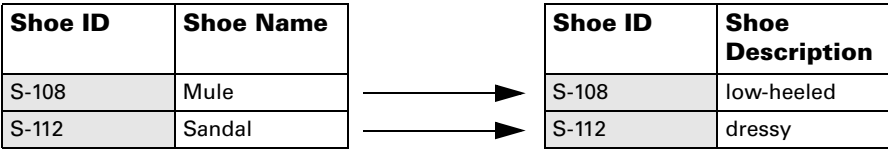
Member keys

Member keys provide access to the data in the virtual table. The member keys are selected from the primary table’s keys. The virtual table can add at most sixteen keys from the primary table. If a key contains a field that is not a member field of the table, the field will be added when the key is added.

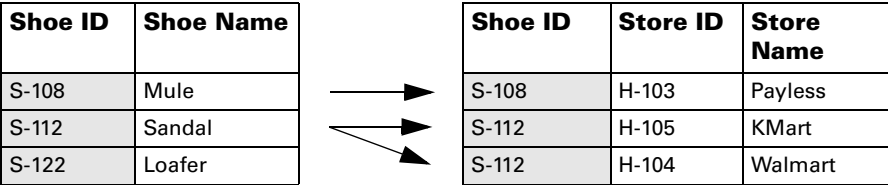
Detailed relationships

By creating relationships between the fields, you relate the fields from one member table to another. A virtual table allows two relationship types: one-to-one and one-to-many.

A one-to-one relationship exists when a member table contains a record that corresponds to only one record in another member table. This type of relationship is created when the key used to join the two member tables is unique.



A one-to-many relationship exists when a member table contains a record that corresponds to more than one record in another member table. This type of relationship is created when the key used to join the two member tables is not unique.



Member table relationships

All of the member tables are related to the primary table, either directly or indirectly. The member tables that are directly related to the primary table are located on a main branch, and their main table is the primary table. Indirectly related tables are those that are linked to the primary table through the related member tables. Indirectly related tables are located on subbranches. Their main tables are the member tables that they have been joined to.

While the key for the primary table provides access to the virtual table, the keys of the non-primary member tables join the tables together. To join the table columns, a field or fields from the primary table are related to the key in the secondary table. To relate fields and keys, the storage type and the storage size must be identical. At runtime, the relationships provide access to data in all member tables. The database will pass the record (row) formed from the relationships stored in the virtual table to the user when all of the data has been collected.

Member tables on main and subbranches can be related through one-to-one and one-to-many relationships. There are rules that govern the combinations of one-to-many relationships that exist. When member tables have been related so that more than one main branch exists, only one main branch can contain one-to-many relationships. If the relationship that you are trying to create is illegal, the key will not appear in the Relationship Key field in the Relationship Definition window.

The lower main branch contains a one-to-many relationship; therefore, the upper main branch can't contain a one-to-many relationship.



The filled in boxes show that the lower main branch has been joined with a left outer join.

Along with one-to-one and one-to-many relationships, joins also determine how relationships are created. Joins regulate how data is gathered and how the columns of data are related. Virtual tables allow two types of joins: equal and left outer.

Equal joins require that only complete records are included in the virtual table's result set. If any member table doesn't contain a corresponding record, the entire record won't be included in the virtual table.

Left outer joins do not require all fields to be complete when added to the virtual table. A left outer join will add all of the fields specified by the primary table's key, whether data exists in the secondary table or not. When a secondary table does not contain data in the related fields, a null value will be inserted instead, based on the field's storage type. If a record does not exist in the primary table, no data from the secondary table will be returned.

For example, the following report was generated from a virtual table that uses a left outer join. The dash signifies that there wasn't corresponding data in the joined secondary table.

S-108	Sneaker Mule	
H-103	Payless ShoeSource	Neenah, Wisconsin
S-109	Aragon Dress Pump	
H-109	Nine West	Buffalo, New York
S-110	Classic Wing Tip	-
S-111	Dress Mocassin	
H-105	KMart	Council Bluffs, Iowa

The dash signifies that there wasn't corresponding data in the joined secondary table.

When using either type of join, the join column is not required to be a member column of the virtual table.

Virtual tables and SQL views

A virtual table requires a physical name even though it is a non-physical table. The reason for this is that when used on Microsoft SQL Server, the virtual table may create a native view.

A view is a SQL object that exists on a SQL Server but does not contain data. It contains pointers to standard SQL tables. A virtual table may create a view on the SQL Server and use it to access the data and return it to the virtual table.



A virtual table will only create a view to optimize performance. When the relationships between member tables are too complex, a virtual table will not create a view.

Once a virtual table creates a view, it remains on the Microsoft SQL Server. You can use these views to access data with third-party applications, such as Crystal Reports, Visual Basic, and Microsoft Query.

Using virtual tables

Virtual tables combine data from more than one table or filter data that resides in a single table. Because it can only be used in read-only situations, a virtual table is ideal when creating a report or a scrolling window.

To test a virtual table, you can create a report or a scrolling window.

At least one and at most sixteen member tables can be contained within a virtual table. When a virtual table contains only one member table, column filtering, or excluding data from the member table's record, occurs. If more than one standard table belongs to a virtual table, the virtual table combines data from member tables to create a record from multiple standard tables.

Scripting

Using virtual tables has several scripting advantages. Most importantly, the use of virtual tables reduces the code needed to access data stored in multiple tables. As an example, the following sanScript script links two tables together without using a virtual table:

```
set SHOE_ID of table SHOES to ShoeID;
get table SHOES;
if err() = OKAY then
    set SHOE_ID of table MANUFACTURER to ShoeID of table SHOES;
    set MAN_ID of table MANUFACTURER to manufacturerID of table SHOES;
    get table MANUFACTURER;
end if;
```

As data is needed from more tables, the script grows. By contrast, retrieving the data using virtual tables decreases the code to two lines:

```
set MAN_ID of table MANUFACTURERS_OF_SHOES to manufacturerID;
get table MANUFACTURERS_OF_SHOES;
```

This simplification of script allows for less code maintenance and quicker retrieval of data, improving the performance of an application.

When scripting for virtual tables, the commands **set**, **get**, **fill**, and **clear** are supported.



You can only read from the virtual table. If you try to use a statement like “change” that writes to a table, you’ll get a compile error.

Scrolling Windows

Because virtual tables are read-only, a scrolling window that accesses a virtual table must be browse-only. An open table error occurs at runtime if the scrolling window `WindowType` property is set to `Editable` or `AddsAllowed`.

Reports

Reports frequently need to combine data from multiple tables. While table relationships can be used for this purpose, virtual tables provide not only a performance optimization but also a simpler method to combine data.

Virtual tables do have limits. If you need to combine more than sixteen tables for a report, it is possible to create table relationships between a standard table and a virtual table with no noticeable decrease in performance; however, a virtual table cannot be a member table of another virtual table.



If the primary table does not exist, an open table error will occur when the virtual table is accessed.

Procedure: Creating a virtual table

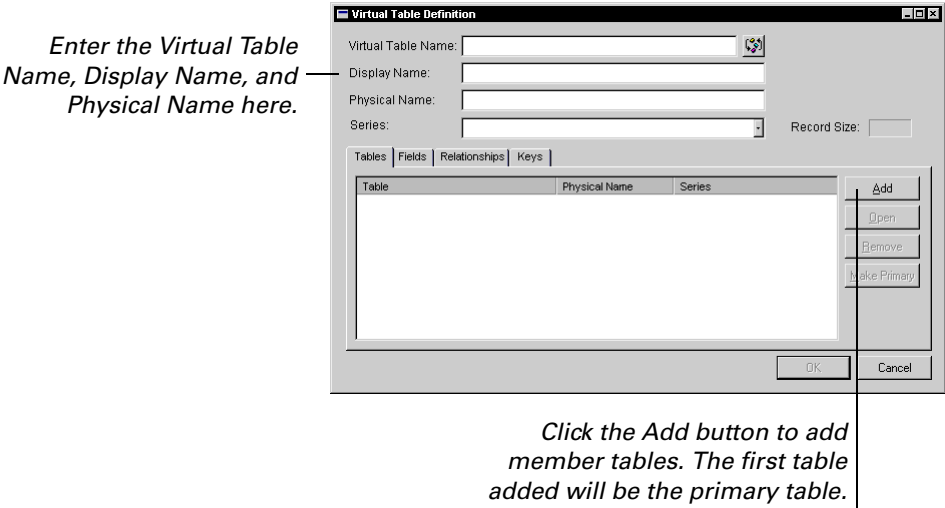
To create a virtual table, point to New on the Explorer menu item and choose Table (Virtual).

1. Name the table.

A virtual table is required to have a Virtual Table Name, a Display Name, and a Physical Name. The Virtual Table Name is used in scripts to identify the table. We suggest using underscores rather than spaces in the name. The Display Name is the name shown on screen to the user, and the Physical Name is used when creating a view on a SQL Server.

2. Specify the series.

The table series allows you to group related tables in a Dexterity application together using series categories like Sales, Financial and System. You will use these series groupings when you use the Report Writer and when you set up pathnames.



3. Add member tables.

Select the Tables tab and click the Add button. The Table Lookup window will appear. This window lists the tables available to add to the virtual table. A table can be included in the virtual table only once. The first table is the primary table.



Selecting a member table and clicking the Make Primary button on the Tables tab can change the primary table; however, changing the primary table removes all of the member keys as well as the relationships built upon the original primary table.

Close the Table Lookup window.

To remove a member table from the list, select the member table on the Tables tab and click the Remove button.



Removing a table removes all of the relationships based on that table. Removing the primary table causes the member table listed below the primary table in the Tables tab to become the primary table.

4. Add member fields.

Select the Fields tab in the Virtual Table Definition window. Click the Add button. The Field Lookup window will open and display a drop-down list containing the member tables. The table's corresponding fields that are available to add are listed in the Fields list. As you add the fields, they are removed from the available list for all tables that contain them. The virtual table can contain only one instance of a field.

Close the Field Lookup window.

To remove a field from the virtual table, select the field in the Fields tab and click the Remove button.

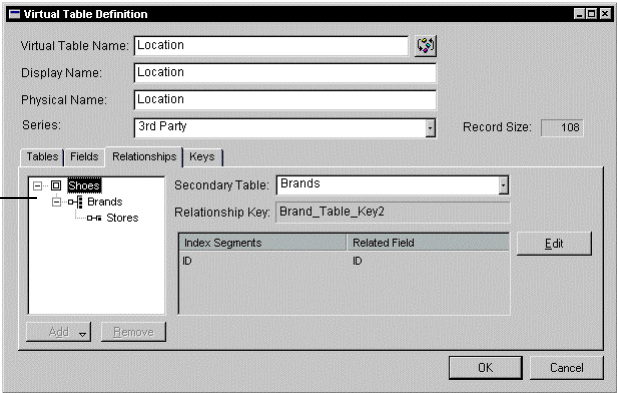


Removing a member field from the primary table also removes the member keys based on that field.

5. Create relationships between member tables.

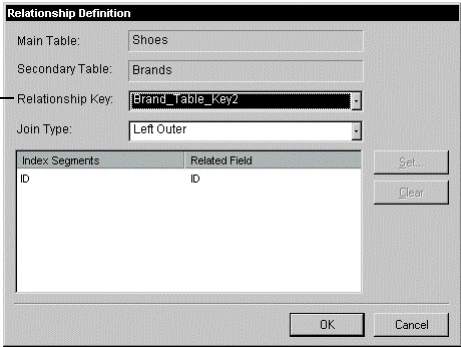
To create relationships between the member tables, select the Relationships tab.

Shoes, the primary table, has been joined to the Brands table with a left outer join in a one-to-many relationship. It uses the field from the primary table, ID, and the key from the secondary table, ID.







The Relationships tree view will already contain the primary table. Select the primary table and click the Add button. A drop-down list of the member tables available to relate to the primary table will appear. Select a member table to add. The Relationship Definition window will open. To relate the member table to the primary table, you must complete the fields in the Relationship Definition window.

Determine the key for the secondary table.



Specify the relationship key and the join type. Equal joins require that only complete records are included in the virtual table's result set. Left outer joins do not require all fields to be complete when added to the virtual table. A left outer join will add all of the fields specified by the primary table's key, whether data exists in the secondary table or not.

The following table lists the icons used to indicate the various types of joins.

Icon	Join type
	1-1 Equal Join
	1-Many Equal Join
	1-1 Left Outer Join
	1-Many Left Outer Join



If a table is linked using a left outer join, all tables linked to that table must use a left outer join as well.

Select the Index Segments and Related Field lines. The Index Segment displays all key segments from the secondary table's key. Click Set. The Related Field Lookup window opens and displays the fields from the primary table that can be joined to the secondary tables' key. These fields have the same storage size and storage type as the Related Field field. Select the appropriate field and click OK. If there is not a matching field, click Cancel and select another key in the Relationship Key field.



Only one main branch can contain a one-to-many relationship. If the table you wish to add violates this rule, the key will be disabled in the Relationship Key field in the Relationship Definition window.

6. Add member keys.

To add member keys, click the Add button on the Keys tab. A drop-down list will appear with available keys to use for the table. All of the keys available to add to the virtual table are keys from the primary table. All of the fields in the keys you select are automatically added to the virtual table if they are not already member fields.

7. Save the virtual table definition.

Click okay to save the virtual table and return to the Resource Explorer. The virtual table will appear with the standard tables.



When creating a virtual table, you must enter all required information and relate all member tables before the OK button is enabled.

Part 6: Working with Dictionaries

This portion of the documentation describes tools available in Dexterity that you will use to interact with dictionaries. The following topics are discussed:

- [Chapter 28, “Resource Explorer,”](#) describes the Resource Explorer, which is the primary interface for working with a dictionary.
- [Chapter 29, “Worksets,”](#) explains a feature of the Resource Explorer that allows you to work with groups of related resources.
- [Chapter 30, “Search and Replace,”](#) describes the search and replace capabilities in Dexterity.
- [Chapter 31, “Importing and Exporting,”](#) explains how to import and export resources from a dictionary in textual form.

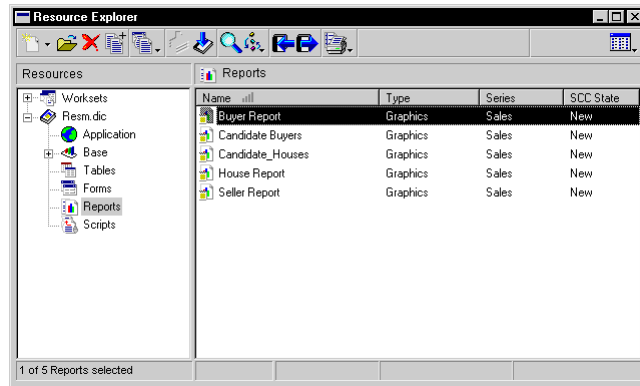
Chapter 28: Resource Explorer

The Resource Explorer is the primary means for working with resources in a dictionary. You will use it for virtually all tasks dealing with resources. Information about the Resource Explorer is divided into the following sections:

- [*Resource Explorer window*](#)
- [*Resource Explorer toolbar*](#)
- [*Explorer menu*](#)

Resource Explorer window

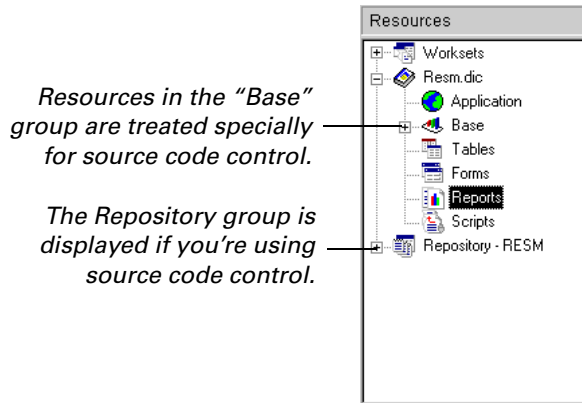
The Resource Explorer window is the primary means of navigation in Dexterity. You will use it create, modify and delete resources in a dictionary. The Resource Explorer window is shown in the following illustration:



The Resource Explorer window has four main areas: the resources tree, resources list, toolbar, and status area.

Resources tree

The resources tree, the left pane of the Resource Explorer window, lists the categories of resources you can view for the current dictionary.



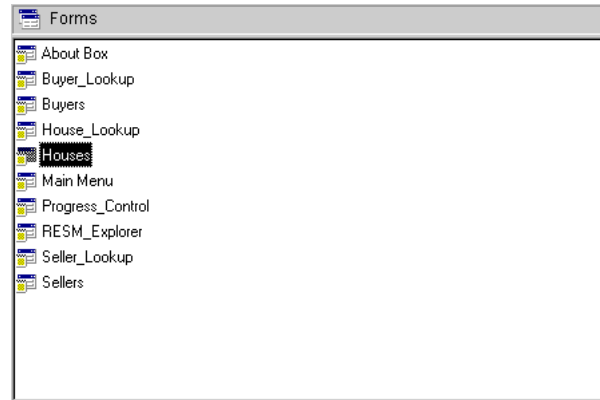
Notice the “Base” group in the resources tree. Resources in this group are handled in a special manner for source code control. If you aren’t using source code control with Dexterity, consider this just an additional way of categorizing resources.

The Worksets group lists all of the worksets that have been created for the current dictionary. Worksets allow you to create groups of related resources for viewing in the Resource Explorer. They are especially useful if you will be working in a dictionary with many resources, or if you’re creating an application that integrates with Microsoft Dynamics GP. Refer to [Chapter 29, “Worksets,”](#) for more information.

If you have source code control enabled, you will also see the Repository group in the resources tree. The items in this group allow you to work with the source code control repository. Refer to [Chapter 50, “Introduction to Source Code Control,”](#) to learn more about source code control.

Resources list

The resources list, the right pane of the Resource Explorer window, lists the resources in the category selected in the resources tree. For instance, if you select Forms in the resources tree, all of the forms in the current dictionary will be listed in the resources list.



You can select single or multiple items in this list. To select multiple items, use the SHIFT key. To select non-contiguous items, use the CTRL key.

The resources list can display information in four modes: Large Icon, Small Icon, List and Report. Use the View button drop list on the Resource Explorer toolbar to specify the view mode.

Large Icon In large icon view, each item is displayed with a large version of its associated icon.

Small Icon In small icon view, each item is displayed with the small version of its associated icon.

List In list view, items are displayed as a list, along with the small version of the icon.

Report In report view, items are displayed as a multicolumn list, along with additional information about the resource, such as the series or type of resource.



When resources are displayed in report view, additional information is retrieved from the dictionary. If you will be displaying a large number of resources, you may want to use one of the other view modes.

Toolbar

The Resource Explorer toolbar provides access to all of the actions you can perform on resources. Some of the buttons on the toolbar are displayed only if you are using source code control. Each of the toolbar items is described in detail in [Resource Explorer toolbar](#) on page 256.

Status area

The Resource Explorer status area, located at the bottom of the window, displays information about the quantity of resources displayed and selected. It also displays status information for various operations performed on resources, such as compiling.

1 of 10 Forms selected







0 Errors, 14 Warnings 








The quantity of resources displayed and selected is shown here.

Compiling status information is shown here.

Resource Explorer toolbar

The Resource Explorer toolbar provides access to all of the actions you can perform on resources. Some buttons are displayed only when source code control is enabled. Those are described in [Chapter 53, “Using Source Code Control.”](#) The following table lists the buttons on the toolbar and a short description of each.

Button	Name	Description
	New/New Resource	Creates a new resource of the selected type.
	Open	Opens the resource selected in the resources list.
	Delete	Deletes the resource or resources selected in the resources list.
	Duplicate Resource	Opens the Duplicate window, allowing you to duplicate the selected resource.
	Worksets	Allows you to create and select worksets.
	Compile	Compiles the scripts for the select forms, global procedures or global functions.

Button	Name	Description
	Compile Dictionary	Compiles all scripts in the current dictionary.
	Find	Opens the Resource Find window, allowing you to search scripts, messages and strings.
	Reference Information	Opens the Reference Information window, allowing you to see how the selected resource is used in the dictionary.
	Import from Text File	Allows importing resources into a dictionary from a text file.
	Export to Text File	Allows selected resources to be exported from the dictionary to text files.
	Report Destination	Allows you to preview a report and print its report definition.
	View	Sets the view mode for the Resource Explorer.

Explorer menu

The Explorer menu contains items that allow you to work with resources selected in the Resource Explorer. Many of the same actions can be performed using the buttons in the Resource Explorer toolbar. The following is a list of the menu items.

New

These menu items create new resources of the selected type.

Open

This menu item opens a resource selected in the Resource Explorer.

Delete

This menu item deletes the resources selected in the Resource Explorer.

View

These menu items display resources of the selected type in the Resource Explorer.

Compile

This menu item compiles the scripts for the selected forms, global procedures or global functions.

Compile All

This menu item compiles all of the scripts for the current dictionary.

Source Control

These menu items allow you to perform source code control operations. They are described in [Chapter 53, "Using Source Code Control."](#)

Import from Text File

This menu item allows importing resources into a dictionary from a text file.

Export to Text File

This menu item allows selected resources to be exported from the dictionary to text files.

Export Dictionary to Text Files

This menu item allows you to export selected resources in the current dictionary to text files.

Refers To

Opens the Reference Information window, displaying the resources the selected resource refers to.

Referenced By

Opens the Reference Information window, displaying the resources that reference the selected resource.

Refresh

Refreshes the current view in the Resource Explorer.

Chapter 29: Worksets

A workset is a group of resources in the Resource Explorer. Each workset is typically composed of related resources. For instance, you may be working with several forms, some global procedures, and a table. Creating a workset that contains these resources allows you to view them all at once in the Resource Explorer. Worksets are especially useful in large dictionaries, and when you're creating applications that integrate with Microsoft Dynamics GP. Information about worksets is divided into the following sections:

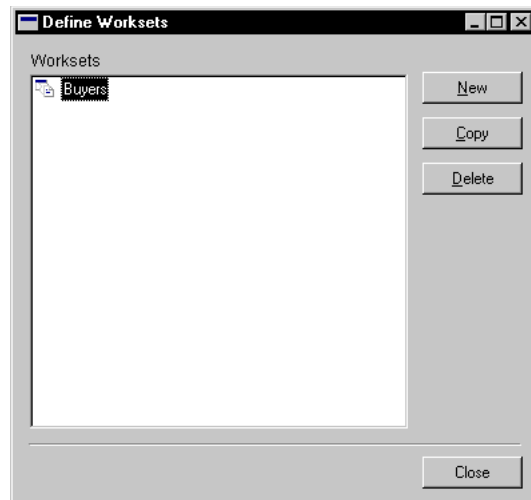
- [*Creating worksets*](#)
- [*Adding resources to a workset*](#)
- [*Viewing worksets*](#)
- [*Removing resources from a workset*](#)
- [*Deleting worksets*](#)

Creating worksets

Worksets are created directly within the Resource Explorer. To create a workset, complete the following procedure.

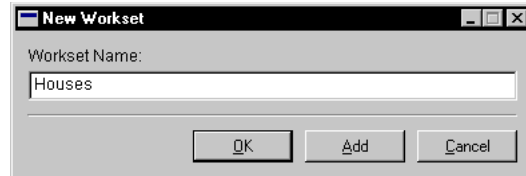
1. Open the Define Worksets window.

Choose Define Worksets from the Worksets button drop list on the Resource Explorer toolbar. The Define Worksets window will appear.



2. Create a new workset or copy an existing one.

In the Define Worksets window, click New to open the New Workset window. Enter the name of the new workset and click OK.



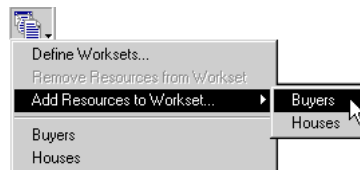
If you want to copy an existing workset, select the workset in the list and click Copy. In the New Workset window, supply the name of the new workset and click OK.

3. Close the Define Worksets window.

When you have finished creating worksets, click Close to close the Define Worksets window.

Adding resources to a workset

To add items to a workset, select the resource or resources to add in the Resource Explorer. In the Worksets button drop list, point to Add Resources to Workset. All of the worksets defined will be listed. Choose the workset you want to add the resources to.



If you are viewing the contents of a workset and then create a new resource, by default, that resource will be added to the workset. If you don't want new resources you create to automatically be added to the current workset, unmark the Add New Resource to Selected Workset option in the General tab of the Options window.



A resource can be part of multiple worksets.

Viewing worksets

There are several ways that you can view the contents of a workset in the Resource Explorer. To display the contents of a workset, do one of the following:

- All of the worksets you have defined are listed in the Worksets button drop list in the Resource Explorer toolbar. Select the workset from the list to display its contents.
- All worksets are listed in the Worksets group in the left pane of the Resource Explorer window. You can display the contents of a workset by selecting it in this group.
- To list all of the worksets that have been defined, click the View Worksets button on Dexterity's toolbar. All worksets will be listed in the Resource Explorer. Select a workset and click the Open button in the Resource Explorer to display the items in the workset.

When displaying the resources in a workset, you can work them much the same way you would in any other view in the Resource Explorer. One difference is that you can't delete resources from the dictionary while viewing them in a workset. You can delete resources only in the standard resource view.

Removing resources from a workset

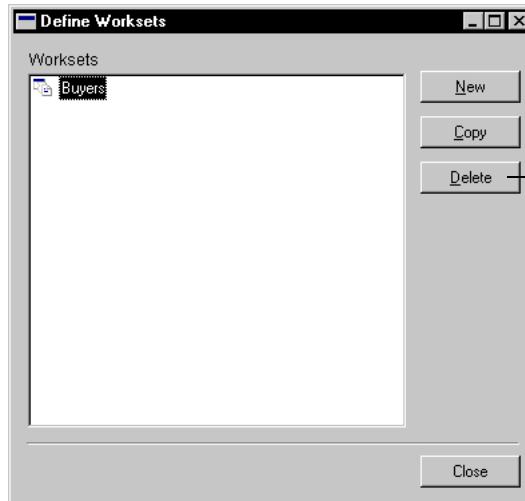
To remove resources from a workset, display the contents of the workset in the Resource Explorer. Select the resource or resources you want to remove from the workset, and then choose Remove Resources from Workset from the Worksets button drop list in the Resource Explorer toolbar. The resources will be removed from the selected workset. No other worksets will be affected.



When you delete a resource from the dictionary, it is automatically removed from any worksets it is part of.

Deleting worksets

To delete a workset, choose Define Worksets from the Worksets button drop list in the Resource Explorer toolbar. In the Define Worksets window, select the workset you want to delete, and then click Delete.



To delete a workset, select it in the list and click Delete.

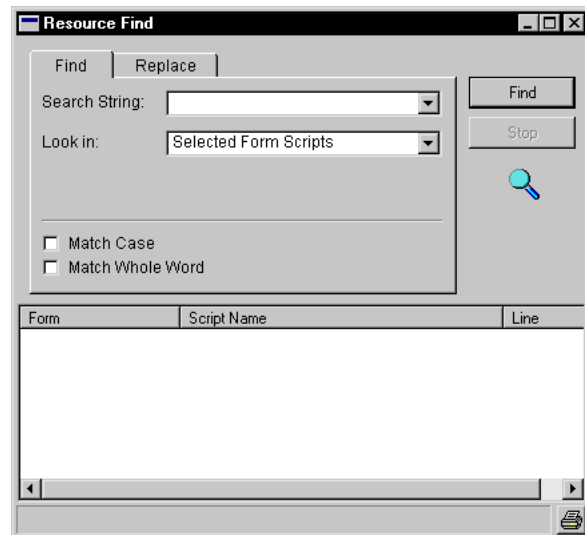
Chapter 30: Search and Replace

As you develop your application, you may need to make a specific change throughout the entire dictionary. You can use the search and replace capability in Dexterity to make these types of changes. Information about search and replace is divided into the following sections:

- [*Resource Find window*](#)
- [*Procedure: Using Find*](#)
- [*Procedure: Using Replace*](#)

Resource Find window

Use the Resource Find window to search for and replace items in the dictionary. You can access this window by choosing Find from the Edit menu or clicking the Find button in the Resource Explorer. The window is shown in the following illustration:



The Resource Find window allows you to search the following items in a dictionary:

- Scripts
- Messages
- Strings

Several options are available to help refine the search and replace operations. You can also print a report that contains the results of the search and replace operation.

Procedure: Using Find

Use the following procedure to search for items in the dictionary.

- 1. Open the Resource Find window.**

Choose Find from the Edit menu or click the Find button in the Resource Explorer.

- 2. Specify the item to search for.**

In the Search String field, enter the text of the item you want to search for.

- 3. Indicate where to search in the dictionary.**

In the Look in drop-down list, choose one of the following items, indicating where you want to search:

All Dictionary Scripts Searches all scripts in the dictionary.

All Global Scripts Searches all global procedures and global functions in the dictionary.

All Form Scripts Searches the scripts attached to and contained within the forms in the dictionary.

All Messages Searches all of the message resources in the dictionary.

All Strings Searches all of the string resources in the dictionary.

Selected Form Scripts Searches the scripts attached to and contained within the forms you have selected in the Resource Explorer.

Selected Global Scripts Searches the global procedures and functions you have selected in the Resource Explorer.

- 4. Specify search options.**

You can specify the following options for the search:

Match Case Mark this option to have case considered when performing the search. The case must match exactly for an item to be found.

Match Whole Word Mark this option to find only whole words that match the Search String.

5. Perform the Find operation.

Click Find to perform the find operation. The results will appear at the bottom of the Resource Find window.



You can stop the Find operation any time by clicking Stop in the Resource Find window.

6. Print the results (optional).

To print the results of the Find operation, click the Print button in the Resource Find window.

Procedure: Using Replace

Use the following procedure to search for and replace items in the dictionary. Since the replace operation changes the contents of your dictionary, you may want to perform a find operation first to verify the changes that will be made.

1. Open the Resource Find window.

Choose Find from the Edit menu or click the Find button in the Resource Explorer.

2. Specify the item to search for.

In the Search String field, enter the text of the item you want to search for.

3. Specify the replacement.

In the Replace String field, enter the text you want to replace each item that is found.

4. Indicate where to search in the dictionary.

In the Look in drop-down list, choose one of the following items, indicating where you want to search:

All Dictionary Scripts Searches all scripts in the dictionary.

All Global Scripts Searches all global procedures and global functions in the dictionary.

All Form Scripts Searches the scripts attached to and contained within the forms in the dictionary.

All Messages Searches all of the message resources in the dictionary.

All Strings Searches all of the string resources in the dictionary.

Selected Form Scripts Searches the scripts attached to and contained within the forms you have selected in the Resource Explorer.

Selected Global Scripts Searches the global procedures and functions you have selected in the Resource Explorer.

5. Specify search options.

You can specify the following options for the search:

Match Case Mark this option to have case considered when performing the search. The case must match exactly for an item to be found.

Match Whole Word Mark this option to find only whole words that match the Search String.

Compile After Replace Mark this option to have scripts automatically compiled after items in them have been replaced. This field is available only when you are performing a replace operation for scripts.

6. Perform the Replace operation.

Click Replace to perform the replace operation. The results will appear at the bottom of the Resource Find window.

If you chose to compile scripts, and any compiling errors occurred, the Compile Messages window will be displayed.

7. Print the results (optional).

To print the results of the Replace operation, click the Print button in the Resource Find window.

Chapter 31: Importing and Exporting

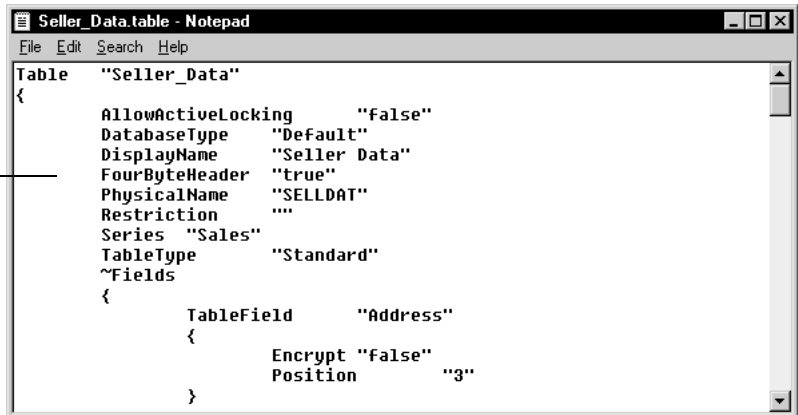
Dexterity has the ability to import and export textual representations of resources. This capability is useful when moving resources from one dictionary to another, and for making large-scale modifications to a dictionary. Information about importing and exporting is divided into the following sections:

- [*Textual representation of resources*](#)
- [*Importing*](#)
- [*Exporting*](#)

Textual representation of resources

Because they are text files, you can view the exported resources with any standard text editor such as Windows Notepad. The content of the files is designed to be human-readable. For example, the following illustration shows a portion of the exported file that represents the Seller Data table.

The text is designed to be human-readable.



```
Table "Seller_Data"
{
    AllowActiveLocking    "false"
    DatabaseType          "Default"
    DisplayName            "Seller Data"
    FourByteHeader        "true"
    PhysicalName           "SELLDAT"
    Restriction            ""
    Series                 "Sales"
    TableType              "Standard"
    ~Fields
    {
        TableField        "Address"
        {
            Encrypt        "false"
            Position        "3"
        }
    }
}
```

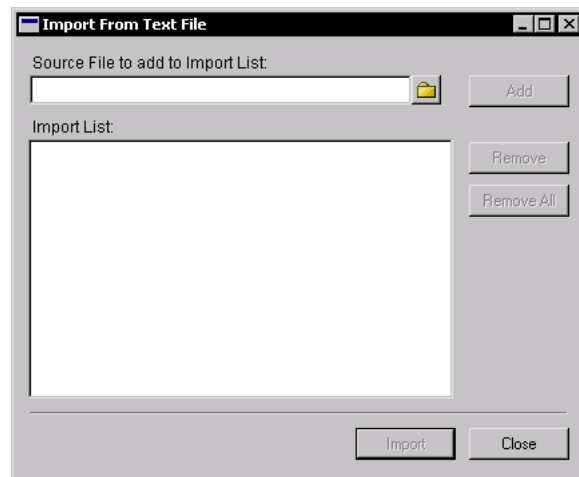
When you examine the textual representation of a resource, you will see the terminology used closely follows that found in the Dexterity user interface. This makes the files easy to understand, without requiring additional documentation.

Importing

To import text files into the current dictionary, complete the following procedure.

- 1. Open the Import From Text File window.**

Choose Import from Text File from the Explorer menu, or click the Import from Text File button in the Resource Explorer. The Import From Text File window will appear.



- 2. Select the file or files to import.**

Supply the complete path to each text file to import and click Add. The file will be added to the Import List. You can also click the lookup button to display a dialog that allows you to select a file and add it to the Import List.

- 3. Import the file.**

Click Import to import the contents of the files in the Import List field.



Any resources that have the same names as those in the text file you're importing will be overwritten.

- 4. View any errors that occurred.**

If any errors occurred while importing the contents of the text file, a message will be displayed. To view detailed information about the errors that occurred, click the Resource Conversion Errors button in the Resource Explorer.

5. Compile scripts (optional).

If any of the resources you imported contained scripts, you will need to compile them. Select the resources in the Resource Explorer and click the Compile button.

Exporting

When exporting resources to text files, you can export resources selected in the Resource Explorer. You can also export the entire contents of the current dictionary.

Exporting selected resources

To export selected resources from the dictionary, complete the following procedure.

1. Select the resources to export.

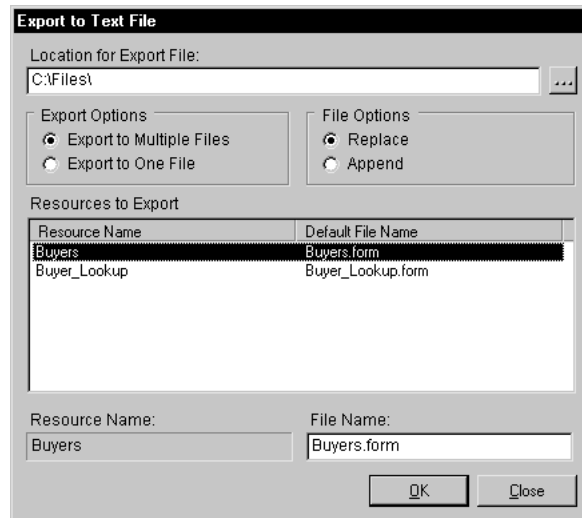
In the Resource Explorer, select the resources you want to export.



You can select single or multiple items in this list. To select multiple items, use the SHIFT key. To select non-contiguous items, use the CTRL key.

2. Export the resources.

Choose Export to Text File from the Explorer menu or click the Export to Text File button in the Resource Explorer. The Export to Text File window will be displayed.



3. Specify the location of the export file or files.

Supply the location where the export file or files will be placed, or click the lookup button to display a dialog that allows you to select a location.

4. Specify export options.

You can specify the following options:

Export Options If you have selected more than one resource in the Resource Explorer, you can indicate whether the resources will be exported to multiple text files or to a single text file.

File Options If text files already exist in the export location specified, this option indicates whether they will be replaced or appended to.

5. Name the export file or files (optional).

The Resources to Export list view displays the resources that will be exported, along with the file name that will be used for each resource. To change the file name that will be used for the resource, select the resource in the list and edit the name that appears in the File Name field.

If you mark the Export to One File option, all resources will be exported to a single text file. The text file's name will appear in the File Name field, allowing you to change it.

6. Export the resources.

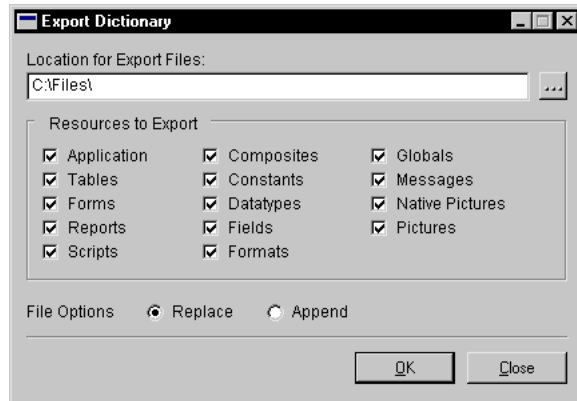
Click Export to export the selected resources.

Exporting the entire dictionary

You can export all of the resources of a specific type from the dictionary to a text file. To do this, complete the following procedure.

1. Open the Export Dictionary window.

Choose Export Dictionary to Text Files in the Explorer menu to open the Export Dictionary window.



2. Specify the location of the export file or files.

Supply the location where the export files will be placed, or click the lookup button to display a dialog that allows you to select a location.

3. Specify the resources to export.

Mark each of the resources. Each form, report, table, global procedure and global function will be exported to a separate file. The following table lists the resource types and the extension that is used for the export files.

Resource type	Extension
Forms	.form
Reports	.report
Tables	.table
Global procedures	.procedure
Global functions	.function

Each of the other resource types, such as fields and data types will be exported as a group. The following table lists the resource types and the names of the corresponding output files.

Resource type	Output File
Constants	constant.constant
Composites	composite.composite
Data types	datatype.datatype
Fields	field.field
Formats	format.format
Global variables	global.global
Pictures	pictures.picture
Native pictures	nativepictures.picture
Messages	message.message
Table groups	tablegroups.tablegroup
Product information	product.product
Install information	install.install

4. **Specify export options.**

You can specify the following option:

File Options If text files already exist in the export location specified, this option indicates whether they will be replaced or appended to.

5. **Export the resources.**

Click Export to export the specified resources.

Part 7: Reports

This portion of the manual provides information on using the Report Writer in Dexterity. The following is a list of topics discussed, with a brief explanation of each:

- [Chapter 32, "Report Writer Overview,"](#) provides basic information about the Report Writer and describes how to plan a report.
- [Chapter 33, "Table relationships,"](#) explains how to create and use table relationships to gather data to use in a report.
- [Chapter 34, "Report Definition,"](#) describes how to use the Report Definition windows to define a report.
- [Chapter 35, "Report Layout,"](#) provides information on using the Report Layout window to design the appearance of a report.
- [Chapter 36, "Sorting,"](#) explains how to sort the data used in a report.
- [Chapter 37, "Restrictions,"](#) describes how to restrict what data appears in a report.
- [Chapter 38, "Calculated Fields,"](#) explains how to create and use calculated fields in a report.
- [Chapter 39, "Additional Headers and Footers,"](#) describes additional headers and footers, and provides information about how to use them.
- [Chapter 40, "Legends,"](#) explains how to use legends in reports.
- [Chapter 41, "Modifying Fields,"](#) describes how to modify how fields appear in a report.
- [Chapter 42, "Using Reports in Applications,"](#) provides information on using the **run report** and **run report with name** statements to print reports you've created using the Report Writer. It also describes how reports can be mailed to other users.

Chapter 32: Report Writer Overview

Reports allow you to retrieve data from your application's tables and present it to users in an organized manner. A report can be as simple as a mailing list or as complex as a year-end account reconciliation. It can be based on data in a single table or in multiple tables.

Information about reports is divided into the following sections:

- [*Reports and the Report Writer*](#)
- [*Report types*](#)
- [*The Dexterity Report Writer and the Runtime Report Writer*](#)
- [*Planning a report*](#)

Reports and the Report Writer

Reports can serve a wide variety of purposes, from printing checks to calculating the total sales to all customers in a particular state. They can be as formal as a department's quarterly sales report, or as informal as a tally of users with access to a specified form. You can use the Report Writer to create reports that are as general or as specific as you need them to be. You can include a company logo or any other picture in a particular report. You can send reports to printers, to files or to the screen.

You can divide a report into several sections: report header, page header, additional headers, body, additional footers, page footer and report footer. Additional headers and footers allow you to create groupings on your report, starting a new section whenever the value in a particular field changes. For example, if your report contains the key field "Customer State," you could have a section for each state, showing the state name in the header, and the total of customers in that state in the footer.

Report types

Dexterity allows you create two different types of reports: *graphics reports* and *text reports*. Unless otherwise specified in the Report Definition window, all reports created with the Report Writer are graphics reports.

Graphics reports can contain a variety of non-text objects, such as pictures, underlines and boxes. You also can use colors and multiple fonts in graphical reports. Reports that take advantage of graphical features can look more professional than text reports. However, as a result of the extra formatting, graphical reports take longer to print than text reports.

Text reports can use only the default font. They can't contain graphical items such as lines or boxes. As a result, text reports print faster than graphical reports.



When either type of report is printed to a file rather than to a printer or to the screen, only the text of that report is sent to the file. Therefore, if you are creating a report specifically to be printed to a file, you should create it as a text report.

The Dexterity Report Writer and the Runtime Report Writer

The version of the Report Writer included in Dexterity is referred to as the Dexterity Report Writer. You can use it to create standard reports to be shipped with your application. The version of the Report Writer that is available with Dexterity applications is referred to as the Runtime Report Writer. It allows users to modify the reports provided with the Dexterity application, or to create new reports to address their specific needs. The following table explains the major differences between the Dexterity and Runtime Report Writer versions.

Characteristic	Dexterity	Runtime
Resource ID	Resources created are numbered between 22,000 and 32,767.	Resources created are numbered between 20,000 and 21,999.
Windows	The Reports button on the toolbar opens the Reports window, which lists each report.	The Reports button on the toolbar opens the Report Writer window, which contains two separate lists. The first list shows the "original reports" that came with the application. The second list contains the "modified reports." These are copies of original reports that have been modified using the Runtime Report Writer. New reports created with the Runtime Report Writer are also included in this list.
Location	All reports are stored in the application dictionary.	New or modified reports are stored in a reports dictionary separate from the application dictionary. This reports dictionary is associated with the application dictionary. Use the Product Information window in Dexterity Utilities to specify a name for the reports dictionary.
Accessibility	All reports can be modified or deleted.	Users can copy a report and modify the copy; however, this allows only limited modifications. For example, to change the tables related to the report, users must first make a <i>secondary copy</i> of a modified report. This in effect creates a new report with a different name. Users can then modify that report as though it was one they created. Only newly-created or modified reports can be deleted. Original reports can't be deleted.

You can make the Runtime Report Writer available with your stand-alone application, provided your customers pay the necessary license fee. Refer to your Dexterity License Agreement for more information about the licensing procedures for the Runtime Report Writer.

Both versions of the Report Writer offer the same basic functionality for creating new reports. This functionality is discussed in detail throughout this part of the manual.

Planning a report

Before you create a report, it's a good idea to plan the report completely. The following list outlines the major steps you should take when planning a report.

1. Define the purpose of the report.

Knowing exactly what the report you are creating will be used for will help keep you focused as you define and design the report.

2. Sketch the report layout.

Include all the different types of information that you wish to appear on the report. Consider the size of the report's headings, whether you want to use graphics, and any other options that would affect the layout.

3. Decide which fields and tables to use.

Once you've decided upon the information to be included in the report, deciding which fields to use should be fairly straightforward.

If a single table doesn't contain all the fields that store the information you need, you'll need to choose one table as the main table, and link additional tables to it so that you can access the desired information. Most often, you should select the table that contains the majority of the fields for your report as your main table, and link other tables to that table as needed.

Refer to [Chapter 33, "Table relationships,"](#) for more information about table relationships.

A table can be linked to the main table only if a relationship between the two tables has been defined, using the Table Relationship Definition window. Relationships are based upon fields that exist in both tables, and are key segments in the secondary table.

4. **Decide whether additional items are required for the report.**

These include items such as additional sorting options, restrictions, calculated fields or conditional expressions.

Reports usually are sorted based upon the sorting criteria associated with the main table key selected in the Report Definition window, or any key used in a one-to-many relationship. You can sort your report by any field in any of the report's tables. Sorting is described in [Chapter 36](#).

Restrictions limit the scope and amount of the information included on a report. For example, you could use a restriction to limit the scope of a report to list only cash accounts rather than all accounts. Restrictions are described in [Chapter 37](#).

Calculated fields allow you to perform calculations using fields in the tables associated with the report, such as calculating the number of line items in an invoice. By including these calculated fields in your report, you can include information in your report that isn't stored anywhere in your application.

Calculated fields can also contain conditional expressions, which allow the field to display a particular value if a specified condition is present, and another value if the condition is absent. Calculated fields are discussed in detail in [Chapter 38](#).

Once you have completely planned your report, you are ready to create the report using the Dexterity Report Writer. The remaining chapters in this part provide information about doing so.

Chapter 33: Table relationships

Table relationships allow the Report Writer to gather data from related tables and use the data in a single report. Information about table relationships is divided into the following sections:

- [Table relationship overview](#)
- [Types of table relationships](#)
- [Defining a table relationship](#)

Table relationship overview

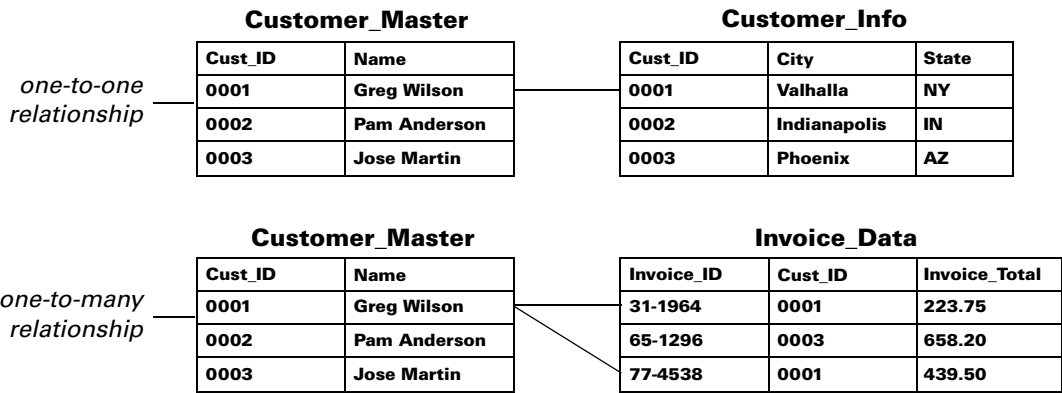
You can define a relationship between two tables if any of the fields in one of the secondary table's keys are also present in the primary table. The fields the relationship can be based on don't need to be part of a key in the primary table, nor do they have to use the same global field. However, they must be based on compatible data types.

For example, in a Real Estate tracking application, a relationship could be defined between the Sales table and the Customer List table, based upon the Buyer Name field in the Sales table and the Customer Name field in the Customer List table.

Types of table relationships

Table relationships can be categorized as *one-to-one* or *one-to-many* relationships. A one-to-one relationship means that for every record in the primary table, there is at most one and only one corresponding record in the secondary table. A one-to-many relationship means that for every record in the primary table, there can be any number of records in the secondary table.

The following illustration shows the differences between one-to-one and one-to-many relationships.



The Customer_Master table is the primary table in both examples. In the one-to-one relationship, there can be only one record in both the Customer_Master and Customer_Info tables with the Cust_ID 0001. In the one-to-many relationship, there can be only one record in the Customer_Master table with the Cust_ID 0001, but there can be multiple invoices associated with that Cust_ID in the Invoice_Data table.

You do not need to specify the type of relationship when you use the Table Relationship Definition window to define a relationship between two tables. The Report Writer automatically chooses the appropriate type of relationship based upon the fields you use to link the tables. It is still important to understand relationship types because the type of relationship between tables will affect the type and quantity of data that you can use in your report.

Defining a table relationship

To access the Table Definition window, click Tables on the tool bar. The Tables window will open. Select the name of the table you wish to use as the primary table, and then click Open. The Table Definition window for the desired table will open.

Follow these steps to access the Table Relationship Definition window and define a new relationship:

1. Open the Table Relationship window.

From the table definition of the table you wish to use as the primary table, click the Relationships button.

2. Open the Table Relationship Definition window.

Click New to define a new relationship. The Table Relationship Definition window will open, and the name of the current table will appear in the Primary Table field.

Click this lookup button to select the secondary table.

The primary table name automatically appears.

3. Select a secondary table.

Click the Secondary Table lookup button to open the Relationship Table Lookup window. Select the name of the table you wish to use as the secondary table. Click OK. That table's name will appear at the top of the Secondary Table portion of the Relationship scrolling window.

4. Select a secondary table key.

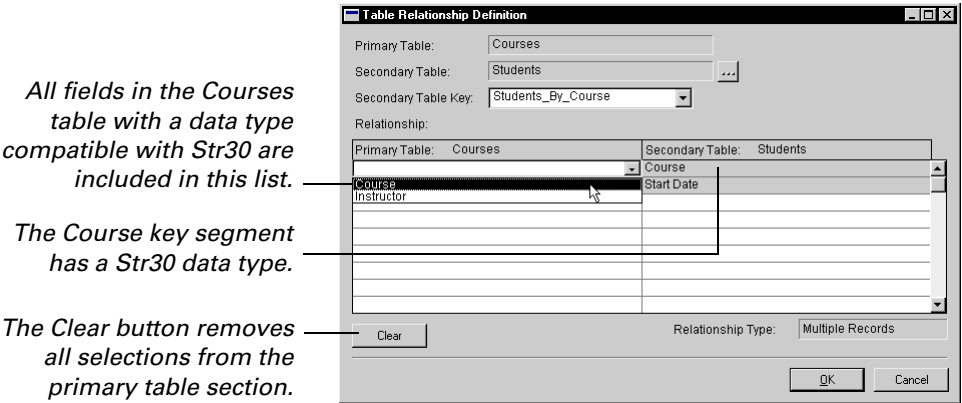
Choose a key containing the field or fields that you wish this table relationship to be based upon from the Secondary Table Key drop-down list. Once the key is selected, each of its key segments is listed in the Secondary Table portion of the Relationship scrolling window.



If you are defining a one-to-many relationship, selecting a key that places records from the secondary table in the desired order will optimize performance.

5. **Select related fields.**

For at least the first of the secondary table’s key segments, select a related field from the corresponding drop-down list in the Primary Table portion of the Relationships scrolling window. The drop-down list contains only fields from the primary table that have a data type compatible with the corresponding key segment in the secondary table. For example, if the key segment Course is a STR30 data type, only fields in the primary table that are compatible with the STR30 data type will appear in the drop-down list, as shown in the following illustration:



If the key contains multiple segments, you need not select a corresponding primary table field for each key segment. However, the key segments you do select corresponding fields for must be consecutive key segments. For example, if a key has four segments, you could select corresponding fields for the first two segments, but not for the first and third segments only.

If the secondary table’s key upon which this relationship is based allows duplicates, the relationship will be one-to-many, and Multiple Records will appear in the Relationship Type field.

If the selected secondary table’s key doesn’t allow duplicates, and you select a corresponding field for each key segment, the relationship type will be one-to-one, and One Record will appear in the Relationship Type field. If you select a corresponding field for only some of the key segments, the relationship type will be one-to-many, and Multiple Records will appear in the Relationship Type field.

To change the primary table field you've selected, simply choose a different field from the drop-down list. To deselect fields you've selected and *not* replace them with different selections, click Clear.



Clicking Clear removes all primary table field selections.

6. Save the relationship.

When you've finished defining the relationship, click OK; you will be returned to the Table Relationship window. You can define another new relationship or edit an existing relationship. To edit an existing relationship, select the name of the secondary table from the Table Relationships list and click Open.

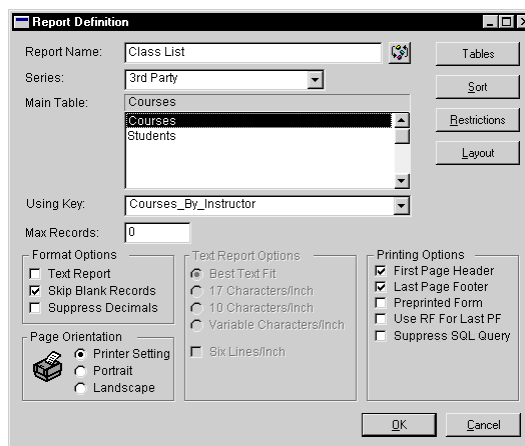
Chapter 34: Report Definition

Before you can design the layout of a report, you must create the report definition, which specifies certain characteristics of the report. Many of these characteristics can be adjusted after you begin designing the report layout. Information about the report definition is divided into the following sections:

- [Report elements](#)
- [Report options](#)
- [Printing a report definition](#)

Report elements

Some of the items specified in the report definition include the name of the report, whether any restrictions will be used to control the scope of the data included when the report is generated, and which tables data presented in the report will come from. Once you've defined necessary table relationships for a report, you can create the report. From the Resource Explorer, choose Reports for the current dictionary. To create a new report, click the New button in the Resource Explorer. The Report Definition window will open, as shown in the following illustration.



1. Name the report.

Enter a report name. A report requires only one name. This is the name used in scripts when the report is run from a Dexterity application, and the name that will appear in the Report Destination window.

2. Select a report series.

Series assignments allow you to group related reports in a Dexterity application using categories like Sales, Financial and Inventory. To ensure the consistent use of series in your application, it's a good idea to select the series of the primary table as the report series.



If you're creating a Dexterity application that integrates with Microsoft Dynamics GP, be sure you select the series your application is integrating with. This will allow security and pathname support to function properly in your application.

3. Select a main table for the report.

The name of each table in the current dictionary is displayed in the list box below the Main Table field. From this list, select the table you want to use as the main table for the report. The main table is typically the table that contains the majority of information used by the report.

4. Add additional tables to the report.

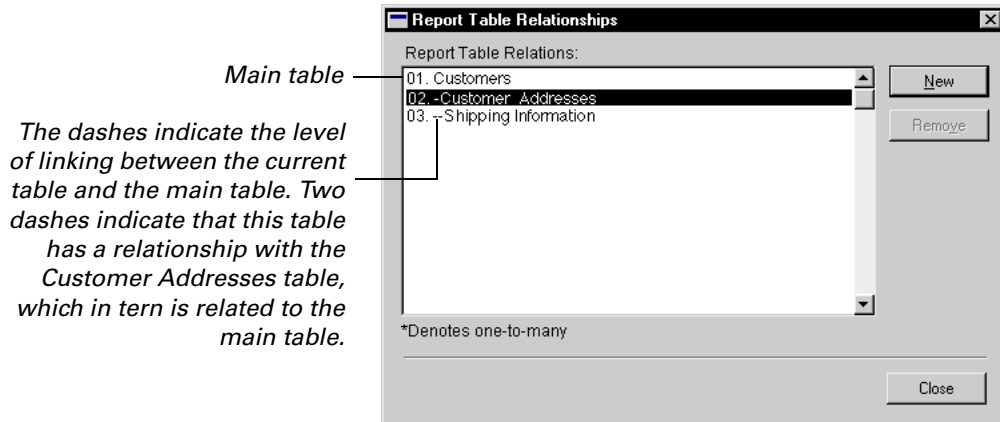
Click the Tables button to open the Report Table Relationships window. The report's main table will be the only table listed in this window. The number 01 will appear to the left of the table name, denoting its status as the report's main table.

Refer to [Chapter 33, "Table relationships,"](#) for more information about creating table relationships.

Click New to add another table to the report. The Report Tables window will open. This window contains a list of all tables that have a relationship with the main table already defined using the Table Relationship Definition window. An asterisk appearing next to the table name denotes a one-to-many relationship. Select the desired table and click OK. Each table that is added to the report and is directly linked to the main table will be listed below the main table in the Report Table Relationships window, with a single dash to the left of the table name.

To add a table that is linked to a table other than the main table, select the secondary table's name in the Report Table Relationships window and click New. Select the name of the table to be added and click OK. This table will appear in the Report Table Relationships window below the table that it is linked to, and will have two dashes to the left of its name.

The following illustration of the Report Table Relationships window shows the dashes that denote levels of linking.



You can have only one one-to-many relationship for each table used for the report. For example, if the Customers table is related to the Address, Invoice and Shipping tables, only one of the relationships can be one-to-many. The other relationships must be one-to-one.

Once you have added all the necessary additional tables, click Close to close the Report Table Relationships window.

5. Set the maximum number of records for the report.

Use the Max Records field to restrict the number of records that will be printed on the report. The default value for this field is 0, which indicates that all records will be printed.

If you enter the number 10 in the Max Records field, for example, only the first 10 records in the report will be printed, unless a restriction has been created to specify another number. Any limit set using a restriction overrides the limit set using the Max Records field.



You may want to enter a relatively small number while you're creating the report, so that test reports will be generated more quickly. Be sure to change the number to 0 or another appropriate number when you are finished testing.

What is an appropriate number depends upon the type of report and how it will be used. For example, you could generate a report that lists all sales statistics by salesperson. If the user will only want to view the top ten salespersons, you could set the Max Records field to 10.

Report options

Several options are available to control characteristics of a report. Use the following procedure to set options for the report.

1. Set the report format options.

These settings specify various characteristics of the report. The Format Options are described in detail below.

Text Report Mark this option if you want to create a text report. Text reports do not allow you to specify fonts or use graphical items in your report, such as lines and pictures. However, text reports are less likely to be adversely affected by different printer configurations.



We suggest you create graphics reports only when creating custom reports for specific clients, where you can be sure of the type of printer the report will be printed to. If you create graphics reports for general distribution, be sure to test the reports with a wide variety of printers.

Skip Blank Records Mark this option to include only records for which there is corresponding data in the main and related tables. If there isn't a corresponding record in each of the related tables, the entire record won't be included in the report.

Suppress Decimals Mark this option to round currency values to nearest whole currency unit. This option is primarily used for international versions of applications where inflation makes the fractional portions of currency amounts insignificant.

2. Set page orientation options.

You can specify the page orientation that will be used for the report.

Printer Setting Choose this option to use the page orientation as specified for the printer currently selected.

Portrait Choose this option to have the report printed in portrait mode, regardless of the current printer setting.

Landscape Choose this option to have the report printed in landscape mode, regardless of the current printer setting.

Refer to [Chapter 35, “Report Layout,”](#) for more information about the Report Layout window.

3. Set text report options.

If you marked the Text Report option, you will activate the Text Report Options portion of the Report Definition window. Marking this option also changes the appearance of the Report Layout window, adding vertical guides at the 80 and 132 character marks. These marks allow you to count characters and more precisely place your field if you are using a fixed text pitch. The text report options are described in detail below.

Best Text Fit Choose this option to have the report printed using the largest printer font that will allow all of the information to be printed on the paper size and paper orientation specified for the printer to which the report will be sent.



If you're creating a report in a dictionary that will be distributed to multiple locations, selecting Best Text Fit will help ensure that the report will print properly on the widest variety of printers.

17 Characters/Inch Choose this option to have the report print in compressed text format.

10 Characters/Inch Choose this option to have the report print in uncompressed text format.

Variable Characters/Inch Choose this option to be able to specify the the characters per inch on a line-by-line basis for the report.

Six Lines/Inch Choose this option to ensure that six lines of the report will print in each inch of report height, regardless of which other text option is selected.



This option is most useful when Best Text Fit is selected. It will prevent the font from becoming too small when the text is compressed. The font will shrink in width so that the entire report can be printed on the available paper, but no more than six lines of text will be printed per vertical inch.

To access the Report Section Options window, open the Report Layout window and select Report Section Options from the Tools menu.

4. Set printing options.

Four printing option are available:

First Page Header Choose this option if you want the page header to be printed on the first page of the report. If you don't want to print page headers on any page of the report, use the Report Section Options window to inactivate page headers. If page headers have been inactivated, the Page Header check box will appear dimmed.

Last Page Footer Choose this option if you want the page footer to print on the last page of the report. If you don't want to print page footers on any page of the report, use the Report Section Options window to inactivate page footers. If page footers have been inactivated, the Page Footer check box will appear dimmed.

Preprinted Form Select this option to remove the built-in margin from the layout area when designing a text report. With this option selected, you can place fields anywhere in the layout area. However, if you place fields outside of the printer's predefined margins, data outside of the margin will not print.

If Preprinted Form is not selected, a margin will appear in the report layout area; the margin is drawn to show the default printer margins of the currently-selected printer. Dexterity will not allow you to place fields outside of this margin.

If Preprinted Form is not selected and you lay out a report while you have one printer selected, then save the report layout, select a different printer and reopen the report layout, the margins may shift, depending on the default margins of the new printer. If the margins shift, fields in the layout area will shift with the left and top margins. This shift may force fields outside of the area bounded by the right margin.

The Preprinted Form selection isn't available if the Best Text Fit option is selected. The use of preprinted forms isn't an option when Best Text Fit is selected because the size and spacing of the font used for Best Text Fit reports varies based on the data in the report. Therefore, you can never be certain that text will appear in the appropriate position on a preprinted form.

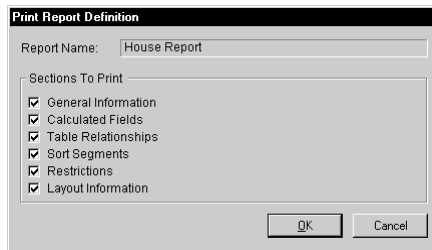
Use RF for Last PF Mark this option to replace the last page footer on the report with the report footer. The report footer will be printed in the area designated for the page footer. If you mark this selection, the page footer and report footer must be the same size, and both sections must be active.

To access the Footer Options window, open the Report Layout window, select Report Options from the Tools menu, and click Add or Open in the Additional Footers section.

Mark this option if the report will contain totals in the report footer. For instance, an invoice report containing invoice numbers, invoice items and an additional footer could have a Sum type field in the additional footer (which breaks on the invoice number field) to display the sum of the invoice items. If you wish to display an overall sum at the bottom of the report as well, you must place it in the report footer, then select Use RF For Last PF, as well as Suppress Last Record's Footer in the Footer Options window for the additional footer.

Printing a report definition

To keep track of the numerous options for a report, you may want to print the report definition. To do this, select the report in the Resource Explorer, then choose Print Report Destination from the Report Destination button drop list in the Resource Explorer toolbar. The Print Report Definition window will appear.



Mark the options indicating what type of information you want included on the report, then click OK. The report will be generated.



Printing report definitions for existing reports is a good way to learn about the report.

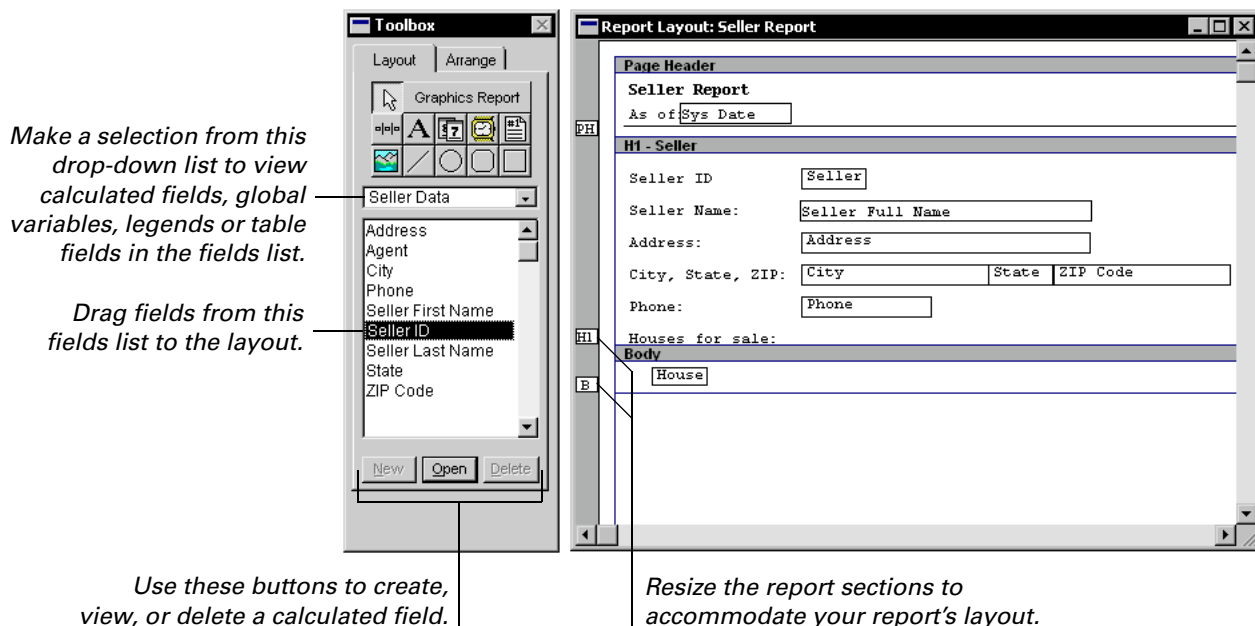
Chapter 35: Report Layout

Once you've planned and defined your report, you're ready to design the report layout. Use the information provided here, along with the sketches from the planning stage, to create your report layout. Information about the report layout is divided into the following sections:

- [Creating a report layout](#)
- [Layout sections](#)
- [The Toolbox](#)
- [Adding fields to a report layout](#)
- [The Properties window](#)
- [Report properties](#)
- [Field properties](#)
- [Drawn object properties](#)
- [Report field characteristics](#)
- [Applying drawing options](#)
- [Viewing your report](#)

Creating a report layout

Click the Layout button in the Report Definition window to access the Report Layout window and Toolbox.



The Report Layout window will be displayed differently depending on whether you are creating a graphics or text report. Changes affect the tools shown in the Toolbox and the width of layout area.

Toolbox

The type of report being created is displayed in the Toolbox, next to the arrow tool. Also, several tools are available only when creating graphics reports, such as the line and picture tools.

Layout area

For a text report, the layout area includes a left border, a right border and two intermediate vertical guides. These guides mark the positions 80 characters and 132 characters from the left margin. They can help you more precisely place fields in the layout area and are especially useful if you're using a preprinted form and have specified a fixed pitch text option. Then, regardless of the printer printed to, you are assured of proper placement of fields.

For graphics reports, the layout area simply includes the left and right borders.

Layout sections

Before you can add fields and data to the report layout, you must decide where to place that information in the layout area. The Report Layout window for a new report contains four evenly-spaced blank sections, separated by border lines. Each border has a handle in the left margin that is labeled using a one- or two-letter abbreviation. Each section can be resized by dragging the handle up or down. Once fields have been placed in a section, you can't drag the lower border of a section above the lowest field in the section.

Additional headers and additional footers are described in [Chapter 39, "Additional Headers and Footers."](#)

Seven types of sections can appear in the Report Layout window, depending on which options are selected in the Report Section Options window. Once the Report Layout window has been opened, you can access the Report Section Options window by choosing Report Section Options from the Tools menu.

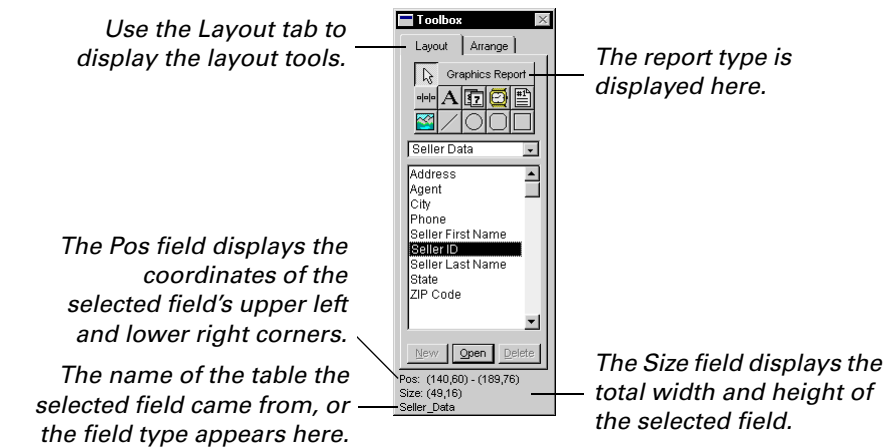
Each section is described in the following table.

Section	Use
Page Header (PH)	<p>Items in this section are placed at the top of every report page. Page number, date and time fields are commonly placed in this section of a report. You can prevent the Page Header from being included on the first page of a report by deselecting the First Page Header option in the Report Definition window.</p> <p>To exclude page headers from a report, deselect the Page Header option in the Report Section Options window.</p>
Report Header (RH)	<p>Items in this section appear only on the first page of a report. The title of the report and introductory information are often included in this section. If a page header is also included on the first page, the report header will appear after the page header.</p> <p>To exclude a report header from a report, deselect the Report Header option in the Report Section Options window.</p>
Additional Headers (H1, H2, H3...)	<p>Additional headers and footers allow you to create groupings in your report. Each header will print when the data in the field it is based on changes. Therefore, the sorting order used will affect the order in which the headers appear on the report. For example, if the field related to header 2 is sorted before the field for header 1, header 2 will print before header 1.</p> <p>Use the Report Section Options window to add additional headers. You can have up to 15 additional headers in a report.</p>
Body (B)	<p>The report body normally contains the bulk of the report. Depending on the number of additional headers and the sorting order used, there could be a body section for each additional header section.</p>
Additional Footers (F1, F2, F3...)	<p>Additional headers and footers allow you to create groupings in your report. Footers should correspond to headers and break on the same fields. They are often used to display summary data, such as a total of all records in the report's body under the footer's related header.</p> <p>Use the Report Section Options window to add additional footers. You can have up to 15 additional footers in a report.</p>
Report Footer (RF)	<p>Items in this section appear only on the last page of a report. Summary information is often included in this section. If a page footer is also included on the last page, the report footer will appear before the page footer.</p> <p>To exclude a report footer from a report, deselect the Report Footer option in the Report Section Options window.</p>
Page Footer (PF)	<p>Items in this section are placed at the bottom of every report page. This section often includes administrative information, such as the name of the person running the report. You can prevent this section from being included on the last page of a report by deselecting the Last Page Footer option in the Report Definition window.</p> <p>To exclude page footers from a report, deselect the Page Footer option in the Report Section Options window.</p>

The Toolbox


Use the Toolbox to place the fields and other items in the report layout area. The Toolbox window's Layout and Arrange tabs let you toggle between two different sets of tools. Tools in the Layout tab help you place information in the layout area, while tools in the Arrange tab are used to arrange selected items in the layout area.











To activate the Toolbox window, simply move the pointer from the Layout window to the Toolbox window. The following illustration shows the Toolbox for a graphical report when the Layout tab is displayed and a field in the Layout area is selected. Each of the Layout and Arrange tools, as well as the Pos and Size fields, are described in the following sections.



The layout tools












To use a layout tool, click the tool's icon, then click in the layout area.

Tool	Name	Description
	Arrow tool	Use the arrow tool to select objects in the layout area. When you select an object, handles will appear; dragging a handle allows you to resize the object. To select more than one object at once, hold down the SHIFT key as you select the object, or drag the arrow tool to draw a rectangle around the set of objects you want to select. You can also use the arrow tool to drag objects to a new position in the layout area.

Tool	Name	Description
	Divider tool	Use the divider tool to divide the report into columns, such as for printing labels. To mark a division on the report, click the divider tool, then click in the body of the report where you want to place the division. You must add the division to the right of any existing fields; the division won't appear if there are fields or other objects to the right of where you've clicked. Once you've placed a division on the report, you can't add any objects to the right of the division. You can reposition a division by dragging it with the arrow tool.
	Text tool	Use the text tool to place text in the layout area, such as a report title or column headings. You can enter up to 79 characters in a text field. You can't use the ENTER key to make the text wrap; this can only be accomplished by resizing the text field.
	Date tool	Use the date tool to add fields that display the current date. The date is determined by the operating system setting.
	Time tool	Use the time tool to add fields that display the current time. The time is determined by the operating system setting.
	Page number tool	Use the page number tool to add fields that display the current page number.
	Picture tool	<p>The picture tool is available for use only with graphics reports. It allows you to add pictures from your application's picture library to a report. To add a picture to the report, click the position in the layout area where the picture's upper left corner should appear. The Pictures window will appear. Select a picture and click OK; the picture will appear in the layout area, where it can be moved or resized like any other object. Refer to Chapter 20 for more information about the picture library.</p> <p>To add a picture to your application's picture library, select Pictures from the Resources menu. Click New in the Pictures window, and paste the desired picture in the Picture field of the Picture Definition window. Name the picture and click OK to save it.</p>
	Line tool	The line tool is available for use only with graphics reports. Use it to draw lines in the report layout.
	Circle tool	The circle tool is available for use only with graphics reports. Use it to draw circles in the report layout.
	Rounded rectangle tool	The rounded rectangle tool is available for use only with graphics reports. Use it to draw rounded rectangles in the report layout.
	Rectangle tool	The rectangle tool is available for use only with graphics reports. Use it to draw rectangles in the report layout.

The arrange tools

Clicking the Arrange tab in the Toolbox displays an additional set of tools used to align, resize or tile fields in the window. These tools are described in the following table.

Category	Tool	Name	Description
Align		Align to Top	Aligns the selected objects with the top object in the group.
		Align to Left	Aligns the selected objects with the leftmost object in the group.
		Align to Right	Aligns the selected objects with the rightmost object in the group.
		Align to Bottom	Aligns the selected objects with the bottom object in the group.
Size		Size to Shortest	Shrinks the selected objects to the height of the shortest object in the group.
		Size to Narrowest	Shrinks the selected objects to the width of the narrowest object in the group.
		Size to Widest	Enlarges the selected objects to the width of the widest object in the group.
		Size to Tallest	Enlarges the selected objects to the height of the tallest object in the group.
		Size to Default	Resizes the selected text object or field. If the layout grid is active, resizes the object to the nearest horizontal and vertical grid lines. If the grid isn't active, resizes the object to its default size based on the drawing options for the item.
Tile		Tile Horizontally	Tiles the selected objects horizontally. The value in the Space field specifies the space between objects.
		Tile Vertically	Tiles the selected objects vertically. The value in the Space field specifies the space between objects.



Arranging objects can't be undone. Be sure to save your report layout before arranging objects. If you aren't satisfied with the result of an arrangement, you can close the window without saving the changes.

The Pos and Size fields

These fields appear at the bottom of the Toolbox when a field is selected in the layout area. The Pos field displays, in pixels, the coordinates of the selected field's upper left and lower right corners. The Size field displays the total width and height, in pixels, of the selected field. These fields are automatically updated as you resize the selected field in the layout area, allowing you to use the information displayed to exactly size a field.

If a table field is selected, the name of the table from which the field was selected will appear below the Size field. If the selected field is a global variable, calculated field or legend, the resource type (Global, Calculated Field or Legend) will appear below the Size field.

Adding fields to a report layout

For more information about legends, refer to [Chapter 40](#), "[Legends](#)."

When you open the Report Layout window, the name of the report's main table will appear under the layout tools in a drop-down list, and the name of each field in that table will appear in the fields list. You can change which fields appear in the fields list by selecting a different entry from the list. The choices in the drop-down list are Globals, Calculated Fields, Legends and the name of each table associated with the report.

The arrow tool is selected by default when you open the window. Use it to select each desired field from the fields list and drag it to the report layout area. If you inadvertently drag the wrong field to the layout area, select the field using the arrow key, then press DELETE or BACKSPACE. This will remove the field from the layout area without affecting the field in the table.

If you add an array field to the layout, the Report Field Options window will appear, allowing you to specify the element of the field being placed on the report. In the Array Index field, enter the number of the array element, and then click OK. If necessary, you can use the Report Field Options window to change the array index to a different element of the array.

Fields placed in the layout area will automatically appear sized according to the maximum keyable length of the field. You can resize each field as you wish; however, if you shorten a field, the information in that field may not be displayed in its entirety when the report is printed. If you resize a field by vertically enlarging it, the field's text will be centered. If you reduce a field's height, it will not display properly.

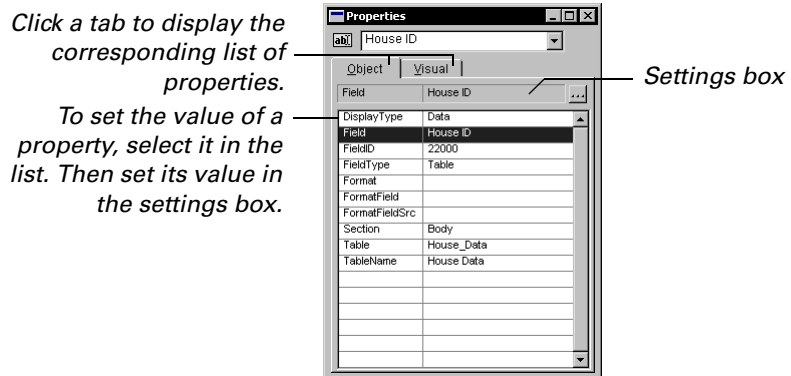
You can click and drag with the Arrow tool to select multiple fields in the layout. By default, the selection rectangle will be limited to the current group. Hold down the CTRL key to allow the selection rectangle to expand beyond the current group.

In general, fields placed in the body of the report should be placed at the very top of the body section, since any white space between the top of the body and the field below it will be repeated for every record that's printed. Similarly, once you've finished placing fields in the report body, you should resize the body section so that excess space is removed from the bottom of the section.

The Properties window

The Properties window is used to display and set several characteristics of a report and the objects in the report. To display the properties for the report, select Properties from the Layout menu. If it isn't already open, the Properties window will appear.

Select an object in the layout area and click the tab indicating which type of property you want to view. Select Object or Visual. To set a property, select it in the list and then change its value in the settings box.



Setting the property value involves choosing a value from a drop-down list, typing a value, or using a lookup. Some properties listed can't have their values changed.

Property type	Description
DisplayType Data	Some properties are set using a drop-down list.
Size-Height 14	Some properties have the value entered directly.
Format	Some properties use a lookup to retrieve values.
FieldType	Some properties can't be set.



Double-clicking a property in the properties list is a shortcut to setting its value. For instance, double-clicking a property whose value is set with a drop-down list will set the property to the next value in the list.

Report properties

To view report properties, be sure the Properties window is open. Select the arrow tool from the Toolbox and click anywhere in the background of the layout area to select the report. The following table lists the report properties.

Object property	Description
DisplayName	The name displayed for the report at runtime.
MainTable	The primary table from which data will be read for the report.
MainTableKey	The key from the main table that will be used to determine the default sorting order for the report. An alternative sorting order can be defined using the Sorting Definition window.
MainTableName	The display name for the main table selected for the report.
MaxRecords	Specifies the number of records that will be printed on the report. For example, the value 10 indicates only the first 10 records will be printed. The default value is 0, which indicates that all records will be printed.
Name	The name of the report. This name is used to refer to the report in scripts.
Orientation	Specifies the page orientation that will be used for the report. Printer Setting will use the orientation for the printer currently selected. Portrait will always use portrait orientation. Landscape will always use landscape orientation.
ReportID	Lists the resource ID of the report.
Series	Indicates the series the report is assigned to.
Type	Indicates what type the report is. It can be one of the following: Graphics, Text, or Text (Variable CPI).

Field properties

The following table lists the field properties.

Object property	Description
ArrayIndex	Specifies which element of an array field or legend to display on the report.
DisplayType	Specifies how the data in a field will be displayed on a report. Most fields on a report use the Data display type. Refer to Changing display types on page 357 for more information.
Field	Indicates the table field or calculated field being used on the report, and allows you to view or edit the characteristics of the field definition.
FieldID	Indicates the resource ID of the field.
FieldType	Indicates whether the field is a table field, calculated field, global field, or a legend.
Format	Displays the name of the format applied to the field.
FormatField	Displays the name of the field that contains the integer value specifying the format to use. Refer to Field formatting on page 355 for more information.
FormatFieldSrc	Indicates whether the format field is a table field or a calculated field. For table fields, the name of the table containing the field is displayed.
Section	Indicates which section of the report the selected field is located in.
Table	For table fields, displays the technical name of the table the selected field is part of.
TableName	For table fields, displays the display name of the table the selected field is part of.

Visual property	Description
Alignment	Specifies whether the item is left-, center-, or right-aligned.
BackColor	Specifies the background color of the field.
Font	Specifies the font to use for the field.
FontBold	If set to true, the field text will be displayed in bold type.
FontColor	Specifies the color of the text for the field.
FontItalic	If set to true, the field text will be displayed in italic type.
FontSize	Specifies the size of the text, in points, for the text displayed in the field.
FontUnderline	If set to true, the field text will be underlined.
Pattern	Specifies the pattern to apply to the background.
PatternColor	Specifies the color of the pattern that is applied to the background.
Position-Left	Indicates the position of the left edge of the field, measured in points from the left edge of the report.

Visual property	Description
Position-Top	Indicates the position of the top edge of the field, measured in points from the top edge of the report.
Size-Height	Indicates the field height, measured in points.
Size-Width	Indicates the field width, measured in points.
Visibility	Indicates conditions when the field will be printed on the report. It can be one of the following: Visible, Invisible, or Hide When Empty.

Drawn object properties

The following table lists the drawn object properties.

Object property	Description
Section	Indicates which section of the report the selected drawn object is located in.

Visual property	Description
Alignment	For static text, indicates whether the text is left, center, or right-aligned.
BackColor	Specifies the background color of the object.
Font	Specifies the font used for static text.
FontBold	If set to true, static text will be displayed in bold type.
FontColor	For static text, specifies the color of the text.
FontItalic	If set to true, static text will be displayed in italic type.
FontSize	For static text, specifies the size of the text, in points.
FontUnderline	If set to true, static text will be underlined.
LineColor	Specifies the color of the line used to draw the object.
LineSize	Specifies the width of the line (in pixels) used to draw the object.
Pattern	Specifies the pattern to apply to the background.
PatternColor	Specifies the color of the pattern that is applied to the background.
Position-Left	Indicates the position of the left edge of the object, measured in points from the left edge of the report.
Position-Top	Indicates the position of the top edge of the object, measured in points from the top edge of the report.
Shape	Specifies the shape of an item drawn with the shape tool.
Size-Height	Indicates the object height, measured in points.
Size-Width	Indicates the object width, measured in points.

Report field characteristics

When printed on a report, most fields you add to a report layout simply display the data values stored by them. However, some fields have special display characteristics you need to be aware of when you use them in a report.

Booleans

When a boolean is added to a report, the string “Yes” is printed on the report if the field contains the value true. The string “No” is printed on the report if the field contains the value false.

Check boxes

When a check box is added to a report, an X is printed if the check box is marked. Nothing is printed on the report if the check box is not marked.

Combo boxes

When a combo box field is added to a report, the text string selected or entered in the field is printed on the report.

Dates

When a date field is added to a report, the short version of the date value is printed on the report. The date value is formatted based on the regional settings for the operating system.

Drop-down lists

When a drop-down list field is added to a report, the static text value corresponding to the field’s value is printed on the report. If the drop-down list field doesn’t have any static text items defined for its data type, no text is displayed on the report. In this case, you must create a calculated field to display a value based on the value of the drop-down list field.

List boxes

When a list box field is added to a report, the static text value corresponding to the field’s value is printed on the report. If the list box field doesn’t have any static text items defined for its data type, no text is printed on the report. In this case, you must create a calculated field to display a value based on the value of the list box field.

Multi-select list boxes

When a multi-select list field is added to a report, the static text values corresponding to the items selected in the field are printed on the report. If the multi-select list box field doesn’t have any static text items defined for its data type, no text is printed on the report.

Pictures

Picture fields should not be added to the report layout.

Radio groups

When a radio group is added to a report, the integer value of the field is printed on the report. If you want to display some other value in the report, you must create a calculated field to display a value based on the value of the radio group field.

Text fields

When you add text fields to a report layout, you should be aware that they can display no more than 10K of data, even though text fields can hold up to 32K of data. Text fields should only be placed in the body of a report.

Resizing a text field will have different results, depending on whether the report is a graphics or text report. If you resize a text field in a graphics report, data will be printed to fill the resized area. However, if the text field is larger than the amount of data in that field for a given record, the unused spaced for that field will still be included in the report. If you resize a text field in a text report, the field width you specify will be used, but the field height will be adjusted automatically to accommodate the text in the text field, up to the 10K limit.

For example, you could resize a text field to be two inches wide and tall enough to include 18 rows of data. If the data in that field for a given record will fill only 12 rows, then only those 12 rows will be included if it's a text report, while those 12 rows plus 6 blank rows will be included if it's a graphics report. Similarly, if the data in that same field for a different record contains 20K of data, up to 10K will be included if it's a text report, while only the first 18 rows will be included if it's a graphics report.

Times

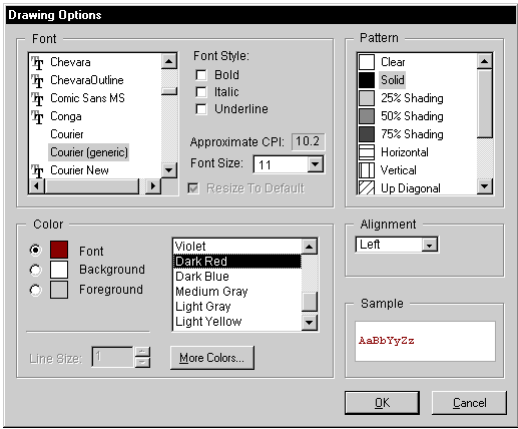
When a time field is added to a report, the time value is printed on the report. The time value is formatted based on the regional settings for the operating system.

Visual switches

If the visual switch field has static text values defined, the static text value corresponding to the field's value is printed on the report. If the visual switch field has picture or native picture static values defined, the integer value of the field is printed on the report. If the visual switch field has no static values defined, it can't be used on the report. Instead, you must create a calculated field to display a value based on the value of the visual switch field.

Applying drawing options

Use the Drawing Options window to select a variety of fonts, font styles, colors and fill patterns for items in graphics reports. To open the Drawing Options window, select an item in the layout and choosing Report Drawing Options from the Tools menu.



If several items are selected in the layout area when you open the Drawing Options window, the options you specify will apply to all of them. If no items are selected when you open this window, the options defined will be applied to all new objects placed in the layout area.



For many objects, such as text, lines and rectangles, you can open the Drawing Options window by double-clicking the object.

Fonts

Use items in the Font section to specify font characteristics for items in the report. For graphics reports, you can choose any font installed on the current system.



Keep in mind that if the report is used on another system that doesn't have the appropriate fonts installed, the missing fonts will be substituted.

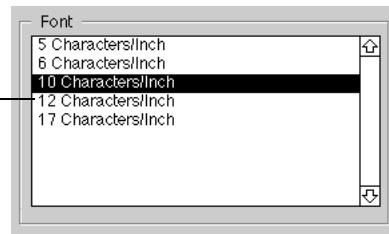
Three “generic” fonts are always available for every report. These are:

- Courier (generic)
- Helvetic (generic)
- Times (generic)

These fonts correspond to the fonts that were available in earlier versions of Dexterity and the Report Writer. If you want your reports to be independent of the fonts installed on a particular system, use these generic fonts.

If you are creating a text report and have marked the Variable Characters/Inch option in the report definition, the Font section in the Drawing Options window allows you to specify the characters per inch to use for each line. Simply select a field in the line and display the Drawing Options window. You can choose 5, 6, 10, 12 or 17 characters per inch.

If you mark the Variable Characters/Inch option for a text report, you can specify the characters per inch for each line of the report.



Once you set the characters per inch for a field on the report, any other field you move to that line will also be printed in that size. To see how your text report will appear when printed, be sure the Show Field Names item in the Layout menu is not marked. The report will be displayed as it will appear when printed.

Patterns

Use the Patterns section to select the fill pattern for fields, text items or shapes in the layout.

Color

Use the Color section to select the font/line, background and foreground colors of items you've added to the layout. The line color is used for lines and field borders. The font color is used for text. The foreground color is used to draw the pattern selected in the Patterns section. The background color is used to fill in the remaining area of the object. The line size specifies the size of the line used to draw lines, rectangles, circles and other shapes.

When specifying colors, you can select one of the predefined colors in the list. To use a custom color, select User Defined as the color and then click More Colors to display the Color dialog. Select a color and click OK. The color you selected will be applied to the current color selection.

Alignment

Use the Alignment setting to specify the horizontal alignment of text in fields or of text items you've added using the text tool.

Viewing your report

Once you've completed adding fields and objects to your report layout, you can view your report in one of several ways. You can send the report to a printer or a text file, or you can view it on your screen. To quickly view your report without leaving tools mode, follow these steps:

- 1. Close the Report Layout window.**

Be sure to save any changes you may have made.

- 2. Click OK to close the Report Definition window.**

- 3. Select the report in the Resource Explorer.**

Once a report is selected, choose Print Report from the Report Destination button drop list in the Resource Explorer toolbar. The Report Destination window will appear.

- 4. Select the desired print destination and click OK.**

If any errors were encountered in the processing of the report, the appropriate error messages will be displayed at this point. If no errors were encountered, the report will print to the specified destination.



Reports that access temporary tables won't print properly in tools mode. To view such reports, you must run them in test mode or with the runtime engine. Similarly, reports in applications that support pathnames and store the tables in locations other than in the same location as the application dictionary won't print properly in tools mode. The tables accessed by a report must be in the same location as the application dictionary for the report to run in tools mode.

For more information about the [run report](#) and [run report with name](#) statements, refer to the *SanScript Reference manual*.

Reports viewed in this manner may not necessarily appear as they would had you run the report from your application using the [run report](#) or [run report with name](#) statement. For example, if your report includes legends, those fields will not print, because the values for legend fields aren't set until runtime. Also, whether your report contains any data depends on whether the application's tables contain data. If there is no data to be included in the report, either because no data tables exist or because none of the data meets the report criteria, table fields in the layout will be blank. However, text and other objects added with tools from the Toolbox will appear.

Chapter 36: Sorting

There are two methods for specifying how the data in a report is sorted. In the first method, the report is sorted based on a key you select for the main table used for the report. In the second method, you create a sorting definition based on the fields in the tables used for the report. Information about sorting is divided into the following sections:

- [Using a main table key](#)
- [Creating a sorting definition](#)

Using a main table key

The primary method of sorting a report is to select a key from the main table for the report. You specify the main table and a key from that table when you create the definition for a report. The data in the report is sorted based on the key you select.

*The report data is sorted
based on this key selected
from the main table.*

Main Table:	Buyer Data
	<div>Buyer Data</div> <div>Candidate Buyers List</div> <div>Candidate Houses List</div> <div>House Data</div> <div>House Descriptions</div>
Using Key:	Buyer_Data_By_ID

This is the preferred method for sorting the report, because an index already exists for each table key. In effect, the order of the data is already known. Dexterity simply retrieves data from the primary table using the key you specified.



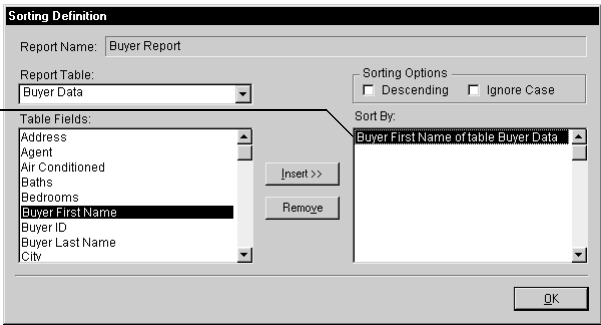
Keep sorting in mind when you create keys for a table that will be the main table for a report. If your table has the appropriate key, the report can be printed much faster.

Creating a sorting definition

In some cases, there isn't an appropriate key available to sort the report data the way you would like. In this situation, you must create a *sorting definition*. A sorting definition defines how you want to sort the report data. It is based on fields in the tables associated with the report. Using a sorting definition is a less-efficient method for sorting the report, because Dexterity must read through each table used by the report to find the appropriate records to use. For optimal performance, you should use a sorting definition only when necessary.

To create a sorting definition, click the Sort button to open the Sorting Definition window, shown in the following illustration. Use this window to define a sorting order that includes fields from any of the tables associated with the report.

The report will be sorted based on the fields in this list.



From the Report Table drop-down list, select the table containing the field you wish to use to specify the sorting order. The name of each field in the selected table will appear in the Table Fields list. Select the desired field, and click Insert. The field name will be added to the Sort By list. Subsequently-added fields will be added to the *top* of the Sort By list, unless an item in the list is selected. If an item is selected, newly-added fields will be added *below* the selected item.

The sorting order will be based on the fields listed in the Sort By list. The data will be sorted based on the first field, then, if a secondary sorting order is needed, the second field, and so on. Be sure to include all fields necessary to fully define the desired order.

For each field selected, you can specify whether you want the data to be sorted in ascending or descending order (ascending is the default order) and whether to ignore the case of any string fields used as sorting criteria. If you ignore case, the strings will be sorted in strict alphabetical order. If you don't ignore case and you are sorting in ascending order, all strings beginning with lowercase letters will appear in alphabetical order, followed by all strings beginning with uppercase letters, also in alphabetical order. In descending order, all of the uppercase strings will appear before the lowercase strings.

To select the Descending and/or Ignore Case options for a given field, select the desired field in the Sort By list and check the appropriate sorting options. Once you have specified all the sorting options, click OK to close the Sorting Definition window.



If none of the fields used to define the sorting order are from a secondary table, the report information drawn from that table will be sorted according to the key used to define that secondary table's relationship with the report's main table.

Once you have created a sorting definition for a report, it will override the sort order specified by the key selected for the report's main table. To use the main table's key as the sorting method, you must remove all of the items from the Sort By list in the Sorting Definition window.

Chapter 37: Restrictions

Refer to [Chapter 42, “Using Reports in Applications,”](#) for more information about adding restrictions at runtime.

Use restrictions to limit the amount and scope of data included in your report. For example, you could create a report with a restriction based upon the ‘Region’ field, so that the report will contain only sales data for a specific region.

Restrictions added to a report using the Report Writer become part of the report definition. Restrictions added at runtime using the **restriction** clause of the [run report](#) or [run report with name](#) statements are used *in addition to* the restrictions defined in the report.

Information about restrictions is divided into the following sections:

- [Defining report restrictions](#)
- [Restriction functions](#)

Defining report restrictions

To add restrictions to an existing report definition, use the following procedure. Select Reports from the tool bar to open the Reports window. Select the name of the desired report and click Open to open the Report Definition window. Click the Restrictions button to open the Report Restrictions window. Click New to open the Report Restriction Definition window and define a new report restriction. The Report Restriction Definition window is shown in the following illustration.



Before defining a restriction, it's a good idea to plan the restriction and write it out completely. Decide which fields, operators, constants and functions you'll need to express the restriction properly. Some restrictions may require that the steps defined below be performed in a slightly different order.

1. Name the restriction.

Enter a name in the Restriction Name field. The name can be up to 79 characters long.

2. Select the appropriate function, if any.

If the restriction you are defining uses a system-defined function, the function must be included in the expression before the field it is applied to. For example, to restrict the Customer Order report so that it lists only customers that ordered more than five items, you could use the following restriction on the Item_Number field:

```
FREQUENCY (Item_Number) > 5
```

For a detailed description of each function available for use when defining a restriction, refer to [Restriction functions](#) on page 315.

3. Specify the field to restrict on.

From the Report Table drop-down list, select the table containing the field on which you wish to restrict the report. Once the appropriate table is selected, select the desired field from the Table Fields drop-down list. Click Add Field to add the selected field to the Restriction Expression.

4. Select the appropriate operator.

From the Operators section, select the appropriate operator. It will automatically be added to the Restriction Expression field. These are the same operators described in [Operators](#) on page 325 with the exception that the concatenation operator is not available for use with restrictions.

5. Specify the constant to be used in the restriction.

From the Type drop-down list in the Constants section, select a control type for the constant you are defining. The five supported control types for constants in restriction expressions are integer, currency, string, date and time. Once you've selected the constant's control type, enter the desired value in the Constant field and click Add Constant.

6. Verify the restriction expression and save it.

Review the contents of the Restriction Expression field. If any part of the expression is incorrect, highlight the incorrect value and click Remove. Replace the removed value, if necessary. Once the restriction expression is correct, click OK to save the restriction and close the Report Restrictions window.

Restriction functions

This section describes the functions available when defining a restriction for a report. It includes a definition of each function, an example of how the function is used in a restriction expression, and an explanation of how you can use such an expression in a report. Also included is information about the storage types that can be used with each function, since some functions can be used only in expressions with fields of a certain storage type.

SUBSTRING and WILDCARD are the only functions that can be used with table fields from all report tables.

Most of these functions can be used only with fields from a report table that has a one-to-many relationship with another report table, such as a secondary table with a one-to-many relationship with the report's main table. Records in secondary tables with a one-to-many relationship with the main table are sometimes referred to as detail records. Using one of these functions with any other fields will cause an error at runtime. Only two functions can be used with fields from any of the report tables: SUBSTRING and WILDCARD.

AVERAGE

The AVERAGE function calculates the average of an expression for all detail records. It can be used with expressions containing fields of all storage types except strings.

Example

You can use the AVERAGE function to create a restriction on an invoice list so that it will print only invoices for which the invoice's items average a markup of at least 20% over cost.

```
AVERAGE ((Item Price) - (Item Cost) / (Item Cost)) > .2
```

EXISTS

The EXISTS function ascertains whether at least one record in a group of detail records makes the expression true. The expression the EXISTS function evaluates should be a comparison as shown in the following example. This function can be used with all boolean expressions.

You can use the NOT operator with this function to increase its scope and create expressions restricting the information on your reports according to expressions that aren't true.

Example

You can use the EXISTS function to create a restriction on an order list to list customers whose orders include at least one item over \$100.

```
EXISTS (Item Price > 100)
```



Unlike other functions, EXISTS and FORALL must be used to evaluate an expression where the comparison operator is inside parentheses, as shown. This causes the function to evaluate the entire expression, rather than only the item following it.

FIRST

The FIRST function finds the first occurrence of an expression in a group of detail records. It can be used with expressions containing fields of any storage type.

Restrictions are evaluated before the report is sorted, so the item designated by the FIRST function may not be the same as the first item in a group on the report once it's been sorted and printed. The first item will be the first value in the table, sorted by the key used to link the secondary table to the report's main table.

For example, you could print a report that lists invoices and corresponding line items, then create a restriction to print only invoices for which the first line item's price was more than \$1,000. If you then sort the report in ascending order by line item price, the line item that the FIRST function finds won't necessarily be the line item listed first on the report. It will be the first line item in the table, which depends on the key used to create the table relationship. If the key is composed of the invoice number and the line item number, the first line item will be the item with the smallest (or largest, depending upon the key's sorting method) line item number.

The following example shows an invoice and its line items as they're sorted within the tables, by invoice number and by item number.

Records in the table are sorted by the Invoice Number and Item.

This is the record the FIRST function will evaluate.

Invoice Number	Item	Price
5005	1201	\$5,300.00
5005	1205	\$3,900.00
5005	1211	\$5,250.00

Since the line items are sorted by item and not by price, the record identified by the FIRST function is item 1201. If you sort the report by the price of each line item, that's how the information will appear on the report, but the FIRST function will still identify item 1201 as the first item.

*Records on the report are
sorted by the Price.*

Invoice 5005

<u>Item</u>	<u>Price</u>
1205	\$3,900.00
1211	\$5,250.00
1201	\$5,300.00

Example

The following example shows how to use the function to create the expression explained in the previous paragraphs and illustrations.

```
FIRST (Item) > 1000
```

FORALL

The FORALL function prints only those records for which all of the detail records meet the expression's requirements. This function can be used with all boolean expressions.

You can use the NOT operator with this function to increase its scope and create expressions restricting the information on your reports according to expressions that aren't true.

Example

You can use the FORALL function to create a restriction on an invoice list to display only invoices containing items that all cost more than \$500.

```
FORALL ((Item Price) > 500)
```



Unlike other functions, EXISTS and FORALL must be used to evaluate an expression where the comparison operator is inside parentheses, as shown. This causes the function to evaluate the entire expression, rather than only the item following it.

FREQUENCY

The FREQUENCY function counts the number of detail records. The FREQUENCY function can be used with expressions containing fields of any storage type.

Example

You can create a restriction on a customer order list to print only orders containing more than five items.

```
FREQUENCY (Item Number) > 5
```

LAST

The LAST function finds the last occurrence of an expression in a group of detail records. The LAST function can be used with expressions containing fields of any storage type except string.

Restrictions are evaluated before the report is sorted, so the item designated by the LAST function may not be the same as the last item in a group on the report once it's been sorted and printed. The last item will be the last value in the table, sorted by the key used to link the secondary table to the report's main table.

The LAST function performs much like the FIRST function. For more information, refer to the description of the FIRST function earlier in this section.

For example, you could print a report that lists Salesperson IDs and corresponding line items containing dates upon which they've made sales and the total amount of each sale. You could then create a restriction to print only the records for salespeople whose most recent sale totaled more than \$1,000. If you then sort the report in ascending order by the amount of the sale, the record that the LAST function finds won't necessarily be the one listed last on the report. The last record on the report will be the item with the largest total sale amount.

The following example shows the records for Salesperson ID 001 as they're sorted within the tables, by salesperson ID and sale date.

Records in the table are sorted by the Salesperson ID and Sale Date.

Salesperson ID	Sale Date	Sale Total
001	1/2/97	\$1,200.00
001	2/4/97	\$2,300.00
001	2/6/97	\$1,400.00
001	4/12/97	\$1,000.01

This is the record the LAST function will evaluate.

Since the salesperson records are sorted by sale date and not by sale total, the record identified by the LAST function is the 4/12/97 sale. If you sort the report by the sale total, that is how the information will appear on the report, but the LAST function will still identify the 4/12/97 sale as the last item.

Records on the report are sorted by the Sale Total.

Salesperson ID 001

<u>Sale Date</u>	<u>Sale Total</u>
4/12/97	\$1,000.01
1/2/97	\$1,200.00
2/6/97	\$1,400.00
2/4/97	\$2,300.00

Example

The following example shows how to use the function to create the expression explained in the previous paragraphs and illustrations.

```
LAST (Sale Total) > 1000
```

MAXIMUM

The MAXIMUM function finds the detail record with the highest value that satisfies the given expression. The MAXIMUM function can be used with fields of any storage type except strings.

Example

You could create a restriction on an invoice list that prints only invoices that include items priced over \$100.

```
MAXIMUM (Item Price) > 100
```

MINIMUM

The MINIMUM function finds the detail record with the lowest value that satisfies the given expression. The MINIMUM function can be used with fields of any storage type except strings.

Example

You could create a restriction on an invoice list that prints only invoices that include items priced under \$100.

```
MINIMUM (Item Price) < 100
```

SUBSTRING

The SUBSTRING function is used to search fields with a string storage type for a match to a specified pattern. The SUBSTRING function will find a match if the specified pattern appears anywhere within the specified field. It is case-sensitive.

Only the equality (=) and inequality (<>) operators can be used with the SUBSTRING function. The data in the specified field either matches or doesn't match the pattern. No other operators can be used.

This function is used to restrict the data on the report to information that matches a set of characters specified as the search pattern. Three special characters can be used in the search pattern to indicate the type of match you're searching for: the asterisk (*), the question mark (?) and the backslash (\), as indicated in the following table.

Character	Matching search results
*	Any character A group of characters No characters
?	Any single character
\	Used in front of special characters so they can be treated as part of the search pattern

The SUBSTRING function implicitly adds an asterisk to the beginning and end of the pattern you specify, so that a match will be found even if the pattern occurs within the information in the field. For example, the SUBSTRING function would find a match for the pattern P*L in the word APPLE, while the WILDCARD function wouldn't.

The backslash is used in front of special characters, such as quotation marks, that have specific meaning in Report Writer. A quotation mark indicates the beginning or end of a field. If you wish to search for fields containing, among other characters, a quotation mark, you'll need to place a backslash in front of it to indicate that the quotation mark is simply a character, and not the sign for the end of the field. Most symbols and non-alphanumeric characters are special characters and should be used with backslashes.

Example

You could use this function to print a list of companies in a particular sales region whose names contain the string John's. The following example shows an expression for which "John's Fish Market" and "Mabel and John's Deli" would both be valid matches.

```
SUBSTRING ("John\'s") = Company_Name
```

SUMMATION

The SUMMATION function calculates the sum of a specified expression for each record in a set of detail records. The SUMMATION function can be used with fields of any storage type except strings.

Example

You can use the SUMMATION function on a report listing invoices and the items on each invoice to print only the invoices where the total of all item extended prices is greater than \$1,000. The extended price is the price of each item multiplied by the number of items that were purchased. The SUMMATION function would first calculate the extended price for each item on the invoice, then sum those extended prices and compare the sum to the specified \$1,000 value. The following expression demonstrates how this restriction would be written.

```
SUMMATION (Quantity Ordered * Item Price) > 1,000
```

WILDCARD

The WILDCARD function is used to search fields with a string storage type for a match to a specified pattern. The WILDCARD function will find a match only if the specified pattern appears in the specified field exactly as stated. The asterisk, question mark and backslash special characters, described in the explanation of the SUBSTRING function, can also be used with this function.

By using these special characters, you can expand the scope of the search conducted by the WILDCARD function. Without the special characters, the specified field must contain exactly the specified pattern, and only the specified pattern, for a match to occur. For example, the expression WILDCARD (Apple) = Snack Type would not find a match if the value in the Snack Type field was Apples.

Only the equality (=) and inequality (<>) operators can be used with the WILDCARD function. The data in the specified field either matches or doesn't match the pattern. No other operators can be used.

The WILDCARD function is case-sensitive.

Example

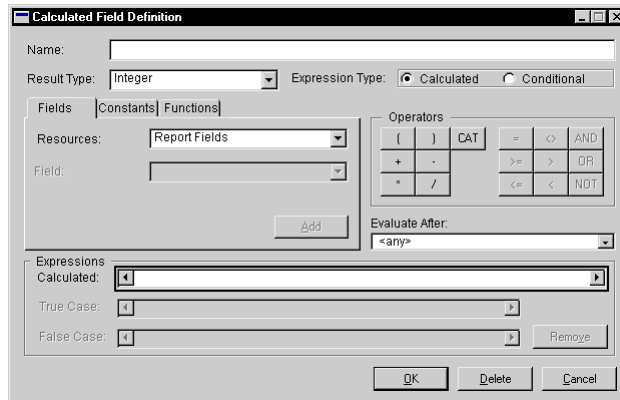
You could use this function to print a list of users whose IDs begin with the letter A.

```
WILDCARD ("A*") = User ID
```


Chapter 38: Calculated Fields

Refer to [Chapter 21, “User-defined Functions,”](#) for more information on user-defined functions.

Calculated fields are a powerful tool you can use to manipulate data in your report. Calculated fields are defined using fields, constants and operators, as well as system-defined or user-defined functions. The following illustration shows the Calculated Field Definition window.



A calculated field is based on either a conditional expression or a calculated expression. A *conditional expression* allows you to print one value or another, depending on the result of a boolean expression. A *calculated expression* allows you to print the result of an expression, such as the sum of several currency fields, or the concatenation of several string fields.

The result of a calculated field must be an integer, currency, variable currency, string or date value. You can use time fields or constants in a conditional expression, but the result printed based on that conditional expression must be an integer, currency, string or date. For example, you could compare the value of a time field to a time constant to determine whether to print the string “Morning” or the string “Afternoon.”

When defining an expression for a calculated field, the expression can’t be typed in directly. Instead, the expression must be built by adding the elements to the expression from the Fields, Constants or Functions tabs and the Operators keypads.

Information about calculated fields is divided into the following sections:

- [*Creating a calculated field*](#)
- [*Operators*](#)
- [*Fields tab*](#)
- [*Constants tab*](#)
- [*Functions tab*](#)
- [*System-defined functions*](#)
- [*User-defined functions*](#)
- [*Evaluation order*](#)

Creating a calculated field

Follow these steps to create a new calculated field for your report:

Click the Reports button on the tool bar. From the Reports window, select the name of an existing report and click Open. Click Layout in the Report Definition window. The Report Layout and Toolbox window will open. From the Layout tab in the Toolbox window select Calculated Fields from the drop-down list, then click New.

1. Name the calculated field.

The calculated field name can be up to 79 characters long.

2. Select the result type.

This is the storage type that will be used for the value the calculated field produces. For example, if the calculated field is a calculated expression that concatenates two string fields, the result type must be a string.

If the calculated field is based upon a conditional expression, both the True Case and False Case values must be of the same storage type as the result type.

3. Select the type of expression.

Calculated is selected by default. If you select Conditional, nine additional boolean operators will be enabled in the Operators section; the True Case and False Case fields will also be enabled.

The options under each tab, and the usage for each operator are described in the following sections.

4. Define the expression.

Select each element to be included in the expression from the appropriate tab (Fields, Constants or Functions). Click Add to place the selected element in the currently-highlighted position within the expression. Add the appropriate operators to the expression as needed.

5. Define conditional values.

If this calculated field is based on a conditional expression, define the True Case and False Case values. One of these values will be printed in the calculated field on the report, depending on whether the expression defined is true or false.

6. Save your entry.

Click OK to save the calculated field and return to the Report Layout window.

Operators

Operators allow you to specify the relationships between the various components of expressions. Some operators are only enabled if you are creating a conditional expression.

*Refer to [Chapter 1, "Syntax,"](#) in Volume 2 of the *Dexterity Programmer's Guide* for more information about the default order of evaluation in Dexterity expressions.*

Parentheses ()

Indicate the beginning and end of an expression that should be evaluated as a unit, separate from the rest of the expression. In general, you should use parentheses to enclose parts of the whole expression when you're using two or more operators and you wish to control the order of evaluation.

Parentheses are automatically added to an expression when you enter a function. The item following it, usually a field, is automatically placed within the parentheses.

Addition (+)

Adds the value to the left of the addition operator to the value to the right of the operator.

Subtraction (-)

Subtracts the value to the right of the operator from the value to the left of the operator.

Multiplication (*)

Multiplies the value to the left of the operator by the value to the right of the operator.

Division (/)

Divides the value to the left of the operator by the value to the right of the operator.

CAT

Joins the data in two or more fields with a string storage type into one string of characters. CAT stands for “concatenate,” and is symbolized by the pound (#) sign when displayed in an expression. You may want to use the CAT operator in conjunction with the STRIP function so that only the data in each field is displayed, and not empty blanks following the data.

The following operators are enabled only when you’re creating conditional expressions.

Equality (=)

Indicates that if the two values to either side of the operator are equal, the expression is true.

Inequality (<>)

Indicates that if the values to either side of the operator aren’t equal, the expression is true.

AND

Joins a series of expressions within a conditional expression. All of the expressions joined by an AND operator must be true for the entire expression to be true.

Greater than or equal to (>=)

Indicates that if the value to the left of the operator is greater than or equal to the value to the right of the operator, the expression is true.

Greater than (>)

Indicates that if the value to the left of the operator is greater than the value to the right of the operator, the expression is true.

OR

Joins a series of expressions within a whole condition or restriction. Only one of these expressions must be true for the entire expression to be true.

Less than or equal to (<=)

Indicates that if the value to the left of the operator is less than or equal to the value to the right of the operator, the expression is true.

Less than (<)

Indicates that if the value to the left of the operator is less than the value to the right of the operator, the expression is true.

NOT

Reverses the meaning of the expression following it. For example, the following expressions are written differently but have the same result:

```
NOT(Account Balance < 5000)
Account Balance >= 5000
```

Fields tab

The Fields tab contains two drop-down lists: Resources and Field. Once a resource is selected, the Field drop-down list is enabled, displaying all available choices of the selected resource type. If there are no values for the selected resource type, the Field drop-down list will remain disabled.

The Resources drop-down list always begins with four system entries: Report Fields, Calculated Fields, RW Legends and System Variables. These are followed by the name of each table associated with the current report. If a table name is selected, each field in that table will be listed in the Field drop-down list. The following describes each system entry, and what appears in the Field drop-down list if that entry is selected.

Refer to [Chapter 41, “Modifying Fields,”](#) for information about using the Report Field Options window to change a field’s display type.

Report Fields

Report Fields fills the Field drop-down list with all fields on the report that have modified display types (fields for which the Report Field Options window has been used to change the field’s display type from Data to some other type, such as Count). The display type, or an abbreviation of the type, is listed first, followed by the name of the field to which it was applied. Use the Report Field Options window to change a field’s display type.

Calculated Fields

Calculated Fields fills the Field drop-down list with all calculated fields the current calculated field can reference. Under special circumstances, dependencies can be created between calculated fields that prevent them from being referenced by the current calculated field. Refer to [Evaluation order](#) on page 338 for more information.

Refer to [Chapter 40, “Legends,”](#) for more information about using legends in reports.

RW Legends

RW Legends fills the Field drop-down list with one selection: Legend. Selecting this value allows you to use one of the values passed to the report through the **legends** clause of the [run report](#) or [run report with name](#) statement. When you click Add, a dialog box will appear, prompting you to enter an index value indicating which legend value to use. The legend will be added to the expression you’re creating.



A legend is passed into the calculated field as a string. If you need the legend value to have some other storage type, you will need to use one of the system-defined functions to convert the legend value. System-defined functions are described in [System-defined functions](#) on page 330.

Often, legends are used in a calculated field to specify the array index of an array field. The following procedure describes how to use a legend to specify the array index in a calculated field.

- Select an array field to add to the expression and click Add. A dialog box will appear, prompting you to enter the array index. Enter any number, then click OK.
- Highlight the newly-entered index number in the expression, then click Remove.
- Click the Functions tab, and select the STR_LNG system-defined function. Click Add. This function will convert the legend (a string value) passed from the [run report](#) or [run report with name](#) statement into a long integer so it can be used as the array index.
- Click the Fields tab. Select RW Legends from the Resources drop-down list, and Legend from the Field drop-down list.
- Click Add; you will be prompted to enter the array index. Enter the number corresponding to the position of the desired legend value from the **legends** clause. For example, if you passed four legend values to the report, enter an index of 1 to specify the first legend, 2 to specify the second, and so on.

For example, Quarterly Bonuses in an array field that has four array elements. The bonus amount for each quarter is stored in an element of the array. To show this value on your report, you would need to create a calculated field similar to the following:

The screenshot shows the 'Expressions' dialog box. The 'Calculated:' field contains the expression: `[Seller_Data.Quarterly Bonus] STR_LNG(Legend: 1)]`. The 'True Case:' and 'False Case:' fields are empty. A 'Remove' button is located at the bottom right.

The value you passed to the first legend will specify which array element value to display. If the value 1 is passed to the first legend, the first quarter's bonus value is displayed. If the value 2 is passed to the legend, the second quarter's value is displayed, and so on.

Global Variables

Globals populates the Field drop-down list with all of the fields declared as global variables in the current dictionary.

Constants tab

Constants are values you set. Typically, they are used in the True Case and False Case fields, or as a value against which fields are compared in the expression. This tab contains two fields: the Type drop-down list and the Constant field.

The Type drop-down list contains five choices: Integer, Currency, String, Date and Time. The value you are allowed to enter in the Constant field depends on the constant type selected. For example, if you select Integer as the constant type, you will be able to enter only numeric values in the Constant field.

The following expression shows the use of two different constant types. The 12:00:00 PM value is a constant of the Time type, while the True Case and False Case fields' values of Morning and Afternoon are String type constants.

The screenshot shows the 'Expressions' dialog box. The 'Conditional:' field contains the expression: `Time Of Day < 12:00:00 PM`. The 'True Case:' field contains the string constant `"Morning"`. The 'False Case:' field contains the string constant `"Afternoon"`. A 'Remove' button is located at the bottom right.

Functions tab

Within the Functions tab, you must select the type of function you want to use, either System-Defined or User-Defined. System-Defined is the default.

Depending on which type of function is selected, one or two drop-down lists will appear below the radio group. If System-Defined is selected, the Function drop-down list will appear, allowing you to choose one of 14 predefined functions. If User-Defined is selected, two drop-down lists will appear, Core and Function. To select a function you've created, you must first select the core with which the function is associated. You can then select the desired user-defined function from the Function drop-down list.



Many of the system-defined functions convert a field's storage type. For example, the CUR_STR function converts a currency control type field to a string control type field. The conversion doesn't affect the way the field is stored in the table, only how its data is evaluated in the calculated or conditional expression.

System-defined functions

To use system-defined functions in expressions, select a function from the drop-down list and click Add. Then, from the Fields tab, select the field you wish to use with the function and click Add. The field will appear in the expression, enclosed in parentheses.

You can't apply one system-defined function to another, or "nest" one function within another. For example, if you're using the CUR_STR function to convert a currency field to a string, you can't use the STRIP function to remove extra spaces from the same field. For example, the following is not permitted:

```
STRIP (CUR_STR ( Beginning Balance ) )
```

If the field you wish to create requires using two system-defined functions, first create a calculated field using the first function, then create another calculated field that applies a second function to the first calculated field, as in the following example:

Calculated field 1	CUR_STR (Beginning Balance)
Calculated field 2	STRIP (Calculated field 1)

Each system-defined function takes only one parameter. The following briefly explains each of these functions:

CUR_STR

Converts a field with a currency storage type to a string type.

DAY

Takes a date value and returns the day portion of the date. The integer value returned will range from 1 to the maximum number of days in the month.

DAY_NAME

Takes a date value and returns a string containing the name the day of the week for the date value.

DOW

Takes a date value and returns an integer representing the day of the week the specified date falls on. The following table lists the integer values and the corresponding day:

Integer value	Day
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

INT_STR

Converts a field with an integer storage type to a string type.

LFT_STR

Prints only the data that lies to the left of the caret (^) in a field with a string storage type. If there is more than one caret in the field, the function will print only the characters to the left of the leftmost caret. The caret must be part of the field, not part of a format applied to the field's data type. If there is no caret in the string, the entire string will be printed, provided enough space is allocated in the layout area.

LNG_STR

Converts a field with an integer or long integer storage type to a string type.

MONTH

Takes a date value and returns the month portion of the date. The integer value returned will range from 1 to 12.

MONTH_NAME

Takes a date value and returns a string containing the name of the month for the date value.

POWER_10

Multiplies 10 to the power specified as this function's parameter. For example, the value of POWER_10(2) is 10 to the power 2, or 100. This function is useful for dealing with currency fields and decimal placement.

RGT_STR

Prints only the data that lies to the right of the caret (^) in a field with a string storage type. If there is more than one caret in the field, the function will print all the characters to the right of the *leftmost* caret. The caret must be part of the field, not part of a format applied to the field's data type. If there is no caret in the string, no data will be printed.

STR_DAT

Converts a field with a string storage type to a date. The string of characters you convert to a date must be in MM/DD/YY format. If the year listed is 35 or less, the year will be preceded by "20," so that the year will be 2035, for example, instead of 1935.

STR_CUR

Converts a field with a string storage type to a currency type. The current control panel settings for the currency symbol, thousands separator, decimal symbol, and negative currency format will be used as the basis for analyzing the string. If the string cannot be properly interpreted based on the current control panel settings, it will be interpreted using the following rules:

- No currency symbol
- The current control panel setting for the decimal separator
- No thousands separator
- A minus sign as the negative indicator at the beginning of the value

STR_LEN

Counts the number of characters, excluding blanks at the end of the string of characters, in a field with a string storage type. Blanks within a string are counted.

STR_LNG

Converts a field with a string storage type to a long integer. This function can be used to compare a string field to a long integer field. Since Report Writer can't compare two fields with different storage types, using the STR_LNG function to convert the string field to a long integer field allows you to compare it to other long integer fields.

STR_VCUR

Converts a field with a string storage type to a variable currency type. The string will be interpreted using the following rules:

- No currency symbol
- The current control panel setting for the decimal separator
- No thousands separator
- A minus sign as the negative indicator at the beginning of the value

STRIP

Removes trailing blank spaces from fields with string storage types. The STRIP function can be used with the CAT operator, but can't be applied to an expression containing the CAT operator.

For example, the following expression is valid:

```
STRIP (Cust_Name) # " has this address: " # STRIP (Address)
```

The following expression is invalid:

```
STRIP (Customer Name # Address)
```

SUBSTRING

Searches fields of the string storage type for a match of the specified string pattern, regardless of whether that match occurs at the beginning of or within the string field. Searches are case-sensitive.

Three special characters can be used in the search pattern: the asterisk (*), the question mark (?) and the backslash (\), as indicated in the following table.

Character	Matching search results
*	Any character Any group of characters No characters
?	Any single character
\	That the following special character is to be treated as part of the search pattern

The backslash is used in front of special characters that have specific meaning in Report Writer, such as quotation marks that indicate the beginning or end of a field. If you want to search for fields containing a quotation mark, you'll need to place a backslash in front of it to indicate that the quotation mark is simply a character, and not the sign for the end of the field. Most symbols, such as the pound sign (#) or the at sign (@), are special characters and should be preceded by backslashes if you want them to be evaluated literally in an expression.

The difference between this function and the WILDCARD function is that this function implicitly adds an asterisk to the beginning and end of the pattern you specify. For example, the SUBSTRING function would find a match for the pattern P*L in the word APPLE, while the WILDCARD function wouldn't.

You must use the SUBSTRING and WILDCARD functions with the equality (=) and inequality (<>) operators in conditional expressions, to ascertain whether the data in the field matches or doesn't match the pattern.

In an expression, the SUBSTRING and WILDCARD functions should always precede the name of the field you want to match to the pattern. For example, the following example would check whether the values for the Name field contains the string Ann. Each of the following names would be considered matches: Anne Jones, MaryAnn Magaldi and Jose Annez.

SUBSTRING ("Ann") = Name

The following example checks the `First_Name` field for all names containing the letter “t” separated from the letter “n” by one other character. Names that would match this search include “Astin”, “Barton” and “Antonette”.

```
SUBSTRING ("t?n") = First_Name
```

UCASE

Prints the alphabetic characters in the specified string field in uppercase. Numeric characters won’t be affected.

WILDCARD

Searches fields of the string storage type for an exact match of the specified string pattern. Searches are case-sensitive.

You can use the `*`, `?` or `\` special characters to indicate the type of match you’re searching for. These special characters have the same effect as they do with the `SUBSTRING` function.

Unlike the `SUBSTRING` function, this function does not implicitly add an asterisk to the beginning and end of the pattern you specify. Therefore, this function is useful when you want to search for strings that begin with a certain character or set of characters.

The following example checks the `First_Name` field for all names beginning with the letters “To”. Names that would match this search include “Tony”, “Tonya” and “Tomas”.

```
WILDCARD ("To*") = First_Name
```

The following example checks the `Local_Businesses` field for all stores beginning with the string “Tommy’s”. Note that this search uses both the `\` and `*` special characters. Business names that would match this search include “Tommy’s Pizzeria”, and “Tommy’s Bait and Tackle Shop”.

```
WILDCARD ("Tommy\'s*") = Local_Businesses
```

YEAR

Takes a date value and returns the year portion of the date. The integer value returned will be a four-digit year, such as 2001.

User-defined functions

In addition to the system-defined functions, you can define your own functions and use them in calculated fields. The names of user-defined functions intended for use in calculated fields must begin with the letters “RW” to be recognized by the Dexterity and Runtime Report Writers. The “RW” can be uppercase, lowercase or a mixture of cases, and can be followed by any character.



Only user-defined functions with names that begin with the letters “RW” will be displayed when User-Defined is selected in the Functions tab of the Calculated Fields window.

To add a user-defined function to an expression, select the core the function is part of from the Core drop-down list. Once a core is selected, the Function drop-down list will become active. You can then select the desired user-defined function from the Function drop-down list. Click Add to add the function to the expression.

Dexterity automatically adds the selected function to the expression enclosed in parentheses and preceded by the system-defined function `FUNCTION_SCRIPT`. This system-defined function is used only to process user-defined functions. It is added to the expression by Dexterity, and can't be added to the expression from the System-Defined Function drop-down list.

Once the user-defined function is added to the expression, add any needed parameters using the fields and constants tabs. Table fields, calculated fields and constants can be used as parameters for a user-defined function included in a calculated field. Be sure the parameters are placed before the closing parenthesis.



Be sure to add the appropriate number and type of parameters for the selected function. Dexterity will not check the compatibility of the specified parameters until runtime. Also note that the function should not have any optional parameters. All parameters must always be supplied to the function.



To use a report field value as parameter for a user-defined function, create a calculated field with that report field as the only element of the calculated expression. Then, use that calculated field as the parameter for the user-defined function in another calculated field.

When a user-defined function in a calculated field is evaluated depends on the parameters that are passed to the function. If no “in” parameters are passed to the user-defined function, the function will be evaluated one time when the report is run. If a table field or value based on a table field is passed as a parameter to the user-defined function, the function will be evaluated each time a new record is read from that table.

Example 1

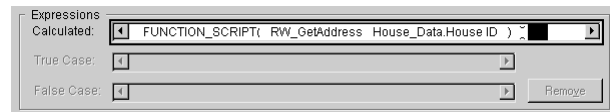
The following user-defined function is used to retrieve the address of a house, based on the House ID passed to the function.

Function: **RW_GetAddress**

```
function returns string address;
in string house_ID;

'House ID' of table House_Data = house_ID;
get table House_Data;
if err() = OKAY then
    address = Address of table House_Data;
end if;
```

The expression for the calculated field that uses this user-defined function is shown in the following illustration. Notice that a field from the House_Data table is passed to the function.



Example 2

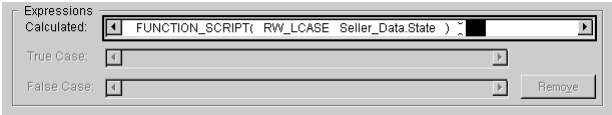
The following user-defined function is similar to the UCASE system-defined function, except that it converts the specified value to lower case.

Function: **RW_LCASE**

```
function returns string lower_case_value;
in string input_string;

{Use the built-in function to convert the string to lower case.}
lower_case_value = lower(input_string);
```

The expression for a calculated field that uses this function is shown in the following illustration.



Evaluation order

In versions of Dexterity prior to 5.5, calculated fields were evaluated in the order they were created. This mean that a calculated field could refer to only the calculated fields that were created before it. This restriction has been removed. As long as there aren't any dependencies that prevent it, a calculated field can refer to any other calculated field.

Dependencies

The order in which calculated fields are evaluated is important when dependencies exist between them. For example, if calculated field B refers to calculated field A, A must be evaluated before B can be. A dependency exists between these two calculated fields.

Dexterity can recognize dependencies among calculated fields, and evaluates them in the appropriate order to ensure they have correct values. Dexterity won't allow you to create invalid situations such as circular dependencies. The following example is one situation that creates a circular dependency: calculated field A refers to calculated field B, which refers back to calculated field A.

If you use user-defined functions in your calculated fields, you can create dependencies that Dexterity can't recognize. These "hidden" dependencies occur in the following cases:

- A calculated field uses a user-defined function to write to a table. Then another calculated field reads the value from the table and uses it.
- A calculated field uses a user-defined function to set the value of a global variable. Then another calculated field uses that global variable.

In both of these cases, the order in which the calculated fields are evaluated is crucial. If they are evaluated in the incorrect order, the report will produce incorrect results.

Specifying the evaluation order

You can use the Evaluate After field in the Calculated Field Definition window to specify when a calculated field will be evaluated. The calculated field being defined will be evaluated after the calculated field you select in the Evaluate After drop-down list. This drop-down list will contain the names of the other calculated fields in the report. Some calculated fields won't appear in this list if known dependencies exclude them as valid choices. When you specify a field with the Evaluate After drop-down list, you are creating a dependency between the two calculated fields.

The Evaluate After drop-down list also contains the choice <any>. This is the default for a new calculated field. Choose <any> when the order in which the calculated field is evaluated isn't significant. Choosing <any> also makes the calculated field available for use in as many other calculated fields as possible.



In virtually all cases, you will leave the Evaluate After field set to <any>. If calculated fields for a report have hidden dependencies, use the Evaluate After field to ensure the calculated fields are evaluated in the proper order.

Updating existing reports

In versions of Dexterity prior to 5.5, calculated fields were evaluated in resource ID order. To ensure backward compatibility, the calculated fields for existing reports created with these earlier versions have the Evaluate After drop-down list set to duplicate the resource ID evaluation order.

We recommend that you set the Evaluate After field for all calculated fields to <any> for each of your reports. Then identify any hidden dependencies among the calculated fields. If any hidden dependencies exist, use the Evaluate After field to ensure the fields involved in the dependency are evaluated in the proper order. You can use the Evaluate After utility, found in Dexterity Utilities, to help you update existing reports.

Chapter 39: Additional Headers and Footers

Additional headers and footers are used primarily to create *groups* on your report, grouping all records that contain the same value for a particular field. Information about additional headers and footers is divided into the following sections:

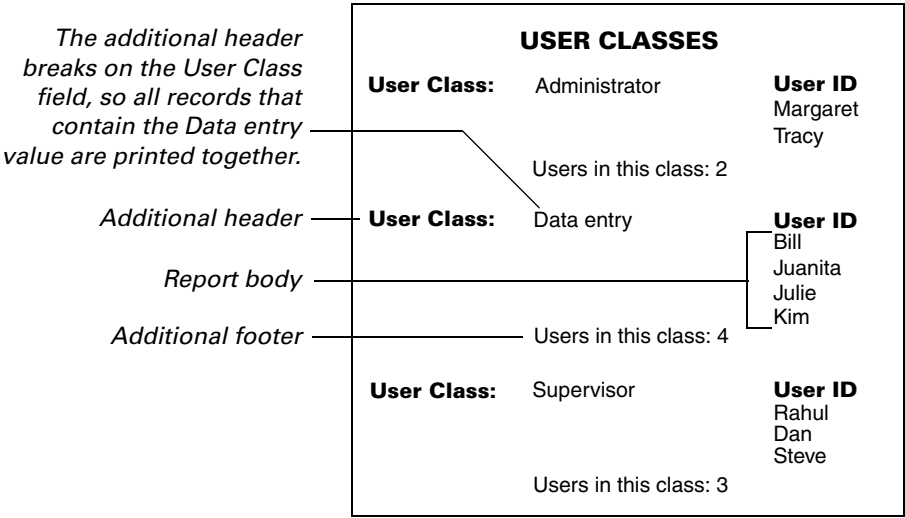
- [Overview of headers and footers](#)
- [The order of the headers and footers](#)
- [Creating additional headers or footers](#)
- [Group overview](#)
- [Sorting for groups](#)
- [Group headers](#)
- [Group footers](#)
- [Counting items in a group](#)
- [Counting groups](#)
- [Totaling and subtotaling](#)

Overview of headers and footers

If your report includes a field that contains the same value for several records, and that field is part of a key or a defined sorting order – for instance, the same invoice number for several line items – you can use an additional header to print that field once, then include its related fields from each record in the body of the report. When all records related to the current value of the common field have been printed, the next value of the common field will be printed, followed by its related fields, and so on. If an additional footer is based on this same common field, the additional footer will print *before* the next iteration of the additional header.

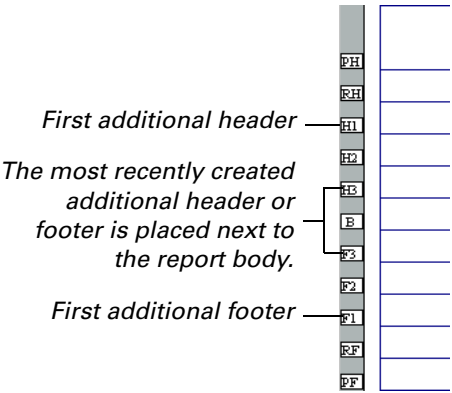
For instance, if you want to create a report listing each of your user classes and the users in each class, you could use an additional header to print the name of each class, followed by a list of the users in that class. Once all the records for the first user class are printed, the additional footer will be printed. Next, the additional header will be printed again, containing the name of the second user class, and so on.

The following illustration displays an example report that uses additional headers and footers.



The order of the headers and footers

If you’re creating two or more additional headers, or two or more additional footers, be sure to create them in the correct order. They will be evaluated in the order they appear on the report layout. For example, if you create three additional footers, the first will appear in the report layout toward the bottom, next to the report footer. The second will appear above the first, and the third will appear between the second additional footer and the report body, as shown in the following illustration.



You can specify where an additional header or footer should appear. In the Report Section Options window, highlight the additional header or footer that should appear directly *above* the one you're adding, then create the section as usual.



If you aren't satisfied with the order of your additional headers or footers, you must delete the header or footer you want to move, then re-create it in the desired location.

Be sure to consider the report's sorting order when determining the placement of your additional headers and footers. Remember that an additional header or footer prints when the data in the field it's based on changes. For example, if the field that Header 2 is based on changes before the field that Header 1 is based on, Header 2 will print first. Therefore, if you always want Header 1 to appear before Header 2, be sure that the field you base Header 1 upon appears in the sorting order before the field Header 2 is based on.

Creating additional headers or footers

To create an additional header or additional footer for a report, open the Report Layout window for the report. Choose Report Section Options from the Tools menu to open the Report Section Options window.

1. Open the appropriate window.

Click the New button next to the Additional Headers list or the Additional Footers list, depending upon which you are adding. The Header Options or Footer Options window will open. If an existing header or footer is highlighted in the list, the one you add will appear after the selected one.

2. Name the resource.

Enter the header name or footer name. This name will appear in the appropriate section in the Report Section Options window.

3. Limit the number of records in the report body.

If you are defining an additional footer and wish to limit the number of records that appear in the body of the report before the additional footer is printed, enter that number in the Records Per Report Body field. If you have multiple additional footers, this field should be specified only in the first additional footer.

Typically, this feature is used when the report is to be printed on a preprinted form. For example, consider a preprinted form with a stub on top and a check blank on the bottom. If the stub can list up to four separate invoices that are being paid by the check, you should include the invoice number and invoice amount fields in the body of the report, and the date, and payment amount (both in numbers and in words), appropriately spaced in the additional footer.

4. **Select the field the resource will be based upon.**

From the Report Table drop-down list, select the name of the table that contains the field the additional header or additional footer should be based on. The report's main table will appear in this list by default.

If you haven't created a sorting definition for the report, only fields that are part of the key for the report table you've selected will be displayed in the Table Field list. If you've created a sorting definition for the report, only fields from the selected table that are part of that sorting definition will appear.



If an additional header or footer is based on a field used in a sorting definition, and that sorting definition is later deleted, the additional header or footer must be deleted as well.

Refer to [Chapter 38, "Calculated Fields,"](#) for more information about calculated fields.

5. **Specify the resource's suppression option.**

Select the Suppress When Field Is Empty option if you don't want this additional header or footer to be printed if a specific calculated field is empty. Once this option is selected, the Calculated Field drop-down list will be enabled. Select the name of the desired calculated field.

6. **Set remaining additional footer options.**

If you are defining an additional footer, several other options are available. Select each option that is appropriate for the current additional footer:

Page Break Starts a new page started after the additional footer is printed. You can use this option even if you don't display any data in this additional footer.

Reset Report Begins a new report each time this additional footer completes printing. The page footer and report footer, if active, will print after this additional footer, before the new report begins printing. Once the new report begins, the page numbers will begin again at 1, and the report header, if active, will be printed.

Suppress Last Record's Footer Suppresses the last occurrence of the additional footer. If you mark this option, you may want to mark Reset Report as well. If you don't, only the last occurrence of the additional footer on the entire report will be suppressed.

For example, you may want to mark this option if you're creating a layout for checks with a stub both on the top and bottom. The stubs would be placed in the body, while the check would be placed in an additional footer. Since only one check should be printed, the second additional footer should be suppressed.

No Break At Record Count Mark this option if you have specified a number in the Records Per Report Body field, but want the additional footer to print only when the value in the field the footer is based on changes. If a value is specified in the Records Per Report Body field and this option isn't marked, the footer will print even if the field the footer is based on hasn't changed.

This option shouldn't be marked if you're printing checks on preprinted forms, but you may want to mark it if you're printing invoices on forms with more than one page.

7. **Save the resource.**

Click OK to save the additional header or additional footer, and return to the Report Section Options window.

Group overview

One way a report can transform data into useful information is by grouping it into meaningful categories. For example, customer classes, credit limits, or transaction amounts are just some of the ways you could group information. To group information on a report, you will do the following:

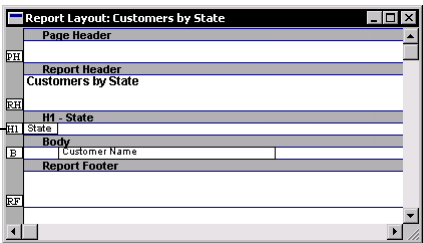
- Determine the items to appear in the group. Determine which table or tables contain the information you want to appear in the group. List the fields that you want to appear in the group, and the fields that will be used to determine the members of each group.
- Sort the report data in an order appropriate for creating the group. You need to sort the data for the report so that the report items appear in an order that allows them to be grouped.

- Create additional headers for each group category. Additional headers for the report will contain the information that describes each group.
- Create additional footers to summarize the group information. Additional footers for the report can perform summary actions such as totaling or subtotaling for each group.

Sorting for groups

To create groups for a report, the information in the report must be sorted in a way that allows the groups to be formed. This means the information for the report must be sorted first by the primary field that identifies each group, then by the items in the group. For example, if you are creating a report that groups customers by state or province, the report must first be sorted by the State field, and then by the Customer Name.

To group the report by State or Province, the data for the report must be sorted based on the State field.



For optimal performance when printing the report, you should use an existing key from the primary table to specify how the data for the report is sorted. However, when you are creating groups for a report, there often isn't a key available for the primary table that sorts the data appropriately. This is especially true when you want to create groups based on data in secondary tables for the report. In this case, you must create a sort order to indicate how to sort the data for the report. This is described in [Chapter 36, "Sorting."](#)

Group headers

To add header information to a group, create an additional header for the report. The additional header contains the field or fields that each group is based on, and will be printed once for each group on the report. It also typically contains descriptive column headers for the information that appears in each group.

For example, the following illustration shows the layout for the User Classes report. Notice that user information is grouped according to the User Class, and then Account Access, and so on. An additional header is used for each group category for the report.

The screenshot shows a window titled "Report Layout: User Classes Report". It displays a hierarchical report structure with the following sections and fields:

- Page Header:**
 - System: Sys Date Sys Time
 - User Date: User Date
- Report Header:**
 - System: Sys Date Sys Time
 - User Date: User Date
- H1 - ClassHeader:**
 - Class: User Class
- H2 - GroupHeader:**
 - Access To All Accounts: YesNo
- H3 - Product Name:**
 - Product: Product Name
- H4 - Series:**
 - Series: Form Series WSM
- H5 - Security Type:**
 - Security Type: Topic
 - Resource Name
- Body:**
 - Resname
- F1 - FooterSpace:**

Group footers

Use footers for groups to add summary information for the group, such as totals, subtotals, or average values. To add footer information to a group, create an additional footer for the report.

Any additional footers for a group should typically “break” on the same field as any additional headers for the group. This keeps the headers and footers with the group, and allows the items in the group to be counted.

When you add a field to an additional footer, it is automatically assigned the Last Occurrence display type. If you don’t use the Last Occurrence display type, the field will appear to be looking ahead to the next record for the report. This occurs because the next record for the report has already been read to determine whether a group should be considered complete.

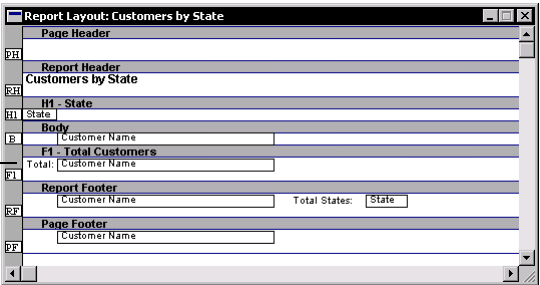
If you place calculated fields in footers, you change their display type from Last Occurrence to Data, so that the calculations are performed properly.

Counting items in a group

It's often useful to count the number of items that appear in each group on a report. Counting the items in a group is done in an additional footer for the group. The additional footer should be set up to print each time the main category for the group changes. Place the field for the item you want to count in the additional footer and set the field's type to Count.

For example, the Customers by State report lists all of the customers by each state. To count the number of customers in each state, an additional footer is added that is printed each time the State changes. A field that occurs for each customer, such as the Customer Name is added to the additional footer, and the display type is set to Count.

This additional footer is printed each time the State field changes. The Customer Name display type is set to Count to count the number of customers per state.



Counting groups

If you have several groups on a report, you may find it useful to count the individual types of groups. To count the number of groups in a report, you need to use a Control Count field.

A Control Count field is designed to count the number of times an additional footer appears in a report. This means that the group you are counting must have an additional footer that indicates the end of the group. Next, create an additional footer immediately below the additional footer that indicates the end of the group. This additional footer must contain an instance of the field on which the group you are counting is based, and you must set the field's type to Control Count.

For example, the Customers by City and State report groups all of the customers by State, and then further groups them by City. To count the number of cities in each state, an additional footer is used that appears at the end of each City group. Another additional footer is added that will contain the City field, but with the display type set to Control Count. This field will count the number of times the additional footer at the end of the City group appears, which is the same as the number of cities for each state.

The report is grouped by State, and then by City.

This additional footer is at the end of each City group.

This display type is set to Control Count, to count the number of City groups.

The screenshot shows a report layout window titled "Report Layout: Customers by City and State". The report structure is as follows:

- Page Header:** Page [Page]
- Report Header:** Customers by State
- H1 - State:** State
- H2 - City:** City
- Body:** Customer Name
- F2 - Total per City:** Total for city: [Customer Name]
- F1 - Total per State:** Total customers for state/province: [Customer Name]
Cities per state/province: [City]
- Report Footer:** Total Customers: [Customer Name] Total States: [State]
- Page Footer:** Customers on this page: [Customer]



The field to which you apply the Control Count display type must be the same field that the footer immediately above the footer containing the Control Count field breaks on.

Totaling and subtotaling

Another way to extract useful information from a report is by totaling or subtotaling numeric values. You do this by adding fields that have the Sum or Running Sum display type applied.

Sum fields

Fields with the Sum display type print the total of the field's values within a group, or within the entire report. If you add a Sum field to an additional footer for a group, it will total the items for the group. Adding Sum fields to the report footer will total all values for the entire report.

Running sum fields

Fields with the Running Sum display type print the total of the field's values at the point where they are placed in the entire report. They are often placed in additional footers that occur throughout a report, or in the page footer to allow running sum values to appear on each page.

Chapter 40: Legends

Refer to [Chapter 42, “Using Reports in Applications.”](#) for more information about using legends with the [run report](#) and [run report with name](#) statements.

Legends are fields you can add to your report to display information that is passed to the report at runtime. Typically, legends are used to display string values representing the sorting and restriction options selected by the user running the report. Legend values can also be used as elements in calculated fields. The legend values passed to a report are specified using the legends clause of the [run report](#) or [run report with name](#) statement used to print the report.

Information about legends is divided into the following sections:

- [Creating legends](#)
- [Legends example](#)

Creating legends

To add legends to the Report Layout area, follow these steps:

- 1. Select Legends from the drop-down list in the Layout tab of the Toolbox.**

The scrolling window below the list will be filled with the word Legend followed by an array index (such as Legend[1] and Legend[2]).

- 2. Drag a legend to the layout area.**

Add legends to the report layout area sequentially, beginning with Legend[1]. You can add the same legend to report multiple times if you want that legend's value to appear in multiple locations on the report.

- 3. Add additional legends to the layout area.**

You can add up to 255 legends the layout area. Add the first 30 legends by selecting a legend from the scrolling window in the toolbox and dragging it to the layout area. To add additional legends, drag a legend from the toolbox to the layout area, and double-click it to open the Report Field Options window. Use this window to change the array index of the legend. To change the array index, change the value of the Array Index field to appropriate index number, and click OK.



Be sure to resize each of your legends to accommodate the largest amount of information that could be included in it. If legends aren't sized properly, information will be truncated.

Legends example

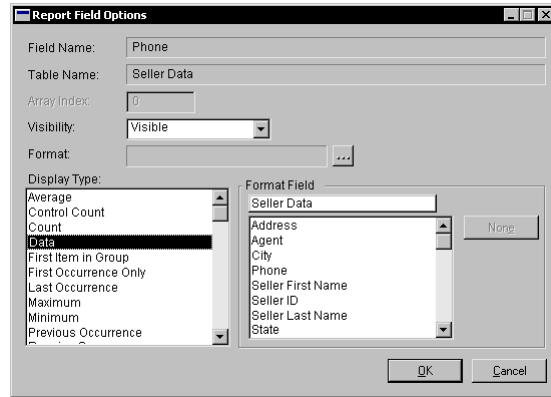
Refer to [Chapter 42](#), [“Using Reports in Applications.”](#) for information about the **run report** statement used to print this report and its legends.

For example, you could include a window in your application requiring the user to select a sorting method and range for the Customer List report shown earlier. This window could include the sorting options of “By State” and “By Customer,” and fields for the user to enter the starting and ending points for the range. Using legends, you could include the sorting and restriction information on the report. The layout for the Customer List report might then appear as shown in the following illustration.

The diagram illustrates the layout of a report titled "Customer List". The report is structured into four main sections: Page Header, Report Header, Body, and Report Footer. The Page Header section contains a "Page No." field. The Report Header section contains the title "Customer List" and three legend fields: "Sorted: Legend[1]", "From: Legend[2]", and "To: Legend[3]". The Body section contains a list of customer fields: "Customer Name", "address", "city", "state", "zipcode", and "phonenumber". The Report Footer section is empty. A vertical bar on the left side of the report contains labels for each section: PH, RH, BH, B, and RF.

Chapter 41: Modifying Fields

Fields in the report layout can be modified in three ways: a format can be applied to them, their visibility setting can be altered, and their display type can be changed. Use the Report Field Options window, shown in the following illustration, to make these modifications.



Information about modifying fields is divided into the following sections:

- [Field visibility](#)
- [Field formatting](#)
- [Changing display types](#)
- [Display type summary](#)

Field visibility

Fields in the report layout area can have their visibility set to one of three options: visible, invisible and hide when empty. Fields added to the layout area are automatically set to visible.

Visible fields are always displayed on the report, even if the field is blank for a given record. If a visible field is blank but has a format applied to it, the formatting will be displayed on the report without any data. For example, a ZIP code field could be formatted to include a dash between the fifth and sixth characters. If this field is included on a report and is visible, and a given record has no ZIP code data, only a dash will appear in the ZIP code field for that record on the report.

A hide when empty field will not be included on a report for a specific record if the field is blank. For example, if a customer record doesn't include a telephone number, and the phone number field is marked as hide when empty, then for that customer record, the empty phone number field and any related formatting won't be included on the report.



Selecting hide when empty prevents formatting from appearing on a report when the field contains no data.

An invisible field will never be shown on a report. Commonly, fields are marked as invisible when their values are needed for use in calculated fields, but are not needed for display on the report. For example, you may want to display a field on the report showing the difference between the highest and lowest values of specific field. To accomplish this, add the field to a footer in the report layout twice. Change the display type of one instance to Minimum, and the display type of the other to Maximum. Then, create a calculated field that subtracts the minimum value from the maximum value. Since only the calculated field needs to be displayed on the report, set the visibility of the minimum value and maximum value fields to invisible.

Field formatting

Applying a format to a field changes the way in which the data stored in the field is displayed. For example, the phone number field may contain the value 7015551234. By applying a format that adds parentheses around the first three digits and adds a dash between the sixth and seventh digits, the value will be readily recognizable as the telephone number (701) 555-1234.

Formats

You can apply formats created with the Format Definition window to string, currency, composite and numeric fields that appear in the report layout. To apply a format to a field, click the Format lookup button in the Report Field Options window to display a list of formats that can be applied to the field. Select a format in this list, or click New to create a new format. When you have finished, the name of the format applied will appear in the Format field in the Report Field Options window.

Refer to [Chapter 7, “Formats.”](#) for more information about creating formats.

Multiple format selector

For fields with the currency or string data types, you can use the Multiple Format Selector to format the data in the field. The Multiple Format Selector is described in [Chapter 7](#). Just as the Multiple Format Selector allows you to use a format field so specify the format to apply to a currency or string field at runtime, you can use the same format field to apply the selected format when the field is printed on a report.

Refer to [Chapter 7, “Formats.”](#) for more information about the Multiple Format Selector.

For example, the Multiple Format Selector is used for the Account Number field shown in the following illustration. It allows the user to specify the format to apply to the Account Number field. In this example, the user chose to display the Account Number with dashes.



This drop-down list is the format field for the Account Number. It specifies which format string to use for the Account Number field.

Both the Account Number field and the Format field are stored for each record in the table, allowing a user to specify which format to use for each record. When the Account Number field is printed on a report, you will want to use the same format that was used when the record was saved. The Format Field in the Report Field options window is used to specify the format field to use. The following illustration shows how the Account Number field is displayed on a report.

Account Number: 10-50-86



Don't use the Format field in the Report Field Options window to apply a format to a report field and use the Multiple Format Selector for the same field; the format results will be unpredictable.

For string fields, the value of the format field indicates which format string will be applied. For currency fields, the value of the format field indicates which predefined currency format will be used. The following table shows Dexterity's predefined currency formats and the integer value associated with them.

Integer	Format	Integer	Format
0	Control Panel Defaults	10	\$1,234.567
1	1,234	11	\$1,234.5678
2	1,234.5	12	\$1,234.56789
3	1,234.56	13	1,234%
4	1,234.567	14	1,234.5%
5	1,234.5678	15	1,234.56%
6	1,234.56789	16	1,234.567%
7	\$1,234.	17	1,234.5678%
8	\$1,234.5	18	1,234.56789%
9	\$1,234.56		

In addition, the following predefined formats and associated integer values are available for variable currency fields:

Integer	Format	Integer	Format	Integer	Format
100	1	200	\$1	300	1%
101	1.1	201	\$1.1	301	1.1%
102	1.12	202	\$1.12	302	1.12%
103	1.123	203	\$1.123	303	1.123%
104	1.1234	204	\$1.1234	304	1.1234%
105	1.12345	205	\$1.12345	305	1.12345%
106	1.123456	206	\$1.123456	306	1.123456%
107	1.1234567	207	\$1.1234567	307	1.1234567%
108	1.12345678	208	\$1.12345678	308	1.12345678%
109	1.123456789	209	\$1.123456789	309	1.123456789%
110	1.1234567890	210	\$1.1234567890	310	1.1234567890%
111	1.12345678901	211	\$1.12345678901	311	1.12345678901%
112	1.123456789012	212	\$1.123456789012	312	1.123456789012%
113	1.1234567890123	213	\$1.1234567890123	313	1.1234567890123%
114	1.12345678901234	214	\$1.12345678901234	314	1.12345678901234%
115	1.123456789012345	215	\$1.123456789012345	315	1.123456789012345%



You can use the functions in the Currency function library to define additional currency formats that can be applied at runtime. Refer to the [Currency function library](#) in the Function Library Reference manual for more information.

The format field must be an integer field. In Dexterity applications, it is common to use a drop-down list for this purpose.

Changing display types

Fields for which the display type has been changed are referred to as report fields.

A field in the report layout can have its display type changed to alter the way in which data is displayed in the field. For example, you can add the invoice_total field to the body of a report, then add it again to the report's footer. By changing the display type of the invoice_total field located in the report footer to Sum, that field will not display a specific invoice's total. Instead, it will display the sum of all instances of the invoice_total field displayed in the body.

To change a field's display type, select the desired type from the Display Type list in the Report Field Options window. The following describes each display type, including information about the storage types each type can or can't be used with:

Average

Prints the average value of the data printed in a group or on a report. This display type should not be used with fields having string, date or time storage types.

Control Count

Prints the number of times the additional footer immediately above the footer containing the Control Count field is printed on a report. The Control Count display type can be used only when you're printing information in groups using additional footers. This display type should not be used with fields having date or time storage types.

Count

Prints the number of times a field is printed in a group or on a report. This display type can be used with fields of all storage types. If you're printing a report with groups and counting categories, use the Control Count display type instead.

Data

Prints data without modifications, as it's stored in the table. All fields in the body of the report whose display types have not been modified have the display type Data. Calculated fields placed in footers should have their display type changed from Last Occurrence to Data, so that the calculations will perform properly. This display type can be used with fields of all storage types.



The Dexterity Report Writer allows you to assign the Data display type to fields other than calculated fields in a footer; however, calculated fields are the only fields that should have the Data display type in a footer. Dexterity sets the default display type for fields placed in footers to Last Occurrence. The Sum display type is another display type commonly used in footers.

First Item In Group

Prints the first value that was printed in a group or on a report. This display type should not be used with fields having date or time storage types.

First Occurrence Only

Prints a value only the first time it occurs, instead of each time it occurs. You can use the First Occurrence display type to organize your data in groups without the use of additional headers and footers. This display type can be used with fields of all storage types.

Last Occurrence

Prints the last value of the field that was printed in a group or on a report. It also can be thought of as the current occurrence. All fields placed in footers are automatically assigned a Last Occurrence display type. Calculated fields placed in footers should have their display type changed from Last Occurrence to Data, so that the calculations will perform properly. This display type should not be used with fields having date, time or texts storage types.

Maximum

Prints the greatest value of the field that was printed in a group or on a report. This display type should not be used with fields having string, date or time storage types.

Minimum

Prints the smallest value of the field that was printed in a group or on a report. This display type should not be used with fields having string, date or time storage types.

Previous Occurrence

Prints the value of the field printed immediately before the last occurrence in a group or on a report. This display type should not be used with fields having date or time storage types.

Running Sum

Prints the total of the field's values at the point where the field is placed within the entire report. This display type should not be used with fields having string, date or time storage types.

Sum

Prints the sum within a group or on a report. This display type should not be used with fields having string, date or time storage types.

Display type summary

The following table indicates which display types can be used in the various sections of a report.

	Report Header	Page Header	Additional Headers	Body	Additional Footers	Page Footer	Report Footer
Average		X			X	X	X
Control Count					X		X
Count		X			X	X	X
Data	X	X	X	X	X	X	X
First Item In Group					X		X
First Occurrence Only				X*			
Last Occurrence					X		X
Maximum					X		X
Minimum					X		X
Previous Occurrence					X		X
Running Sum		X			X	X	X
Sum		X			X	X	X

* Refer to notes in text

You can use a First Occurrence Only field in the body of a report if one of the following conditions is true:

- The field is in the selected key for the main table of the report.
- If you use an alternate sorting definition for the report, the field must be one of the segments in the sorting definition.

The value of any report fields you use in the page header or page footer, such as Average or Sum, will be based on records used for the body of the report. They won't take into account records used for additional headers or additional footers.

Chapter 42: Using Reports in Applications

Reports serve many purposes in an application. They can be used to print polished company-wide reports, or to display basic data and statistics on the user's computer screen. Reports can also be used to export data to files.

Information about using reports in applications is divided into the following sections:

- [Running reports](#)
- [Exporting data to a file](#)
- [Using temporary tables](#)
- [Mail connectivity](#)

Running reports

*For more information about the [run report](#) and [run report with name](#) statements, refer to the *SanScript Reference manual*.*

The [run report](#) and [run report with name](#) statements are the two sanScript statements that run a report created with the Report Writer. Each statement has a number of optional clauses that allow additional report criteria to be specified at runtime. The report criteria that can be specified include restrictions, sorting criteria, legends and report destinations.

Restrictions

The restrictions added at runtime are applied to the report *in addition to* the report restrictions defined as part of the report definition. The restrictions must be based on any fields in the report's main table, or any tables related to the main table.

Sorting criteria

Sorting criteria specified at runtime *supersede* the sorting criteria defined in the report definition. You can specify fields in the report to sort by, or you can choose to sort the report based on a key in the main table of the report.

Legends

The values for legend fields included in reports are also specified at runtime, using the **legends** clause. Most often, legends are used to display string values representing sorting and restriction options selected by the user.

The first value listed after the **legends** keyword will appear in the Legend[1] field, the second value listed will appear in the Legend[2] field, and so on.

Refer to the [Dexterity online help](#) for a description of the Report Destination window.

Refer to the [Printer function library](#) in the Function Library Reference manual for more information about named printer support.

Report destination

If you include the **destination** clause, the report will be sent to the specified destination when the report is run. If you don't specify a report destination when you use the [run report](#) or [run report with name](#) statement, the Report Destination window will be displayed automatically, allowing the user to specify a destination. This is useful when you want the user to be able to specify where the report is printed.

By default, the Printer check box will be marked and the Screen check box not marked when the Report Destination window is displayed. You can control the default values of these two check boxes when the Report Destination window is displayed. To do this, you must create two boolean global variables with the names **Print to Screen** and **Print to Printer**. If you set the global variable's value to true, the corresponding check box in the Report Destination window will be marked. If you set the system variable to false, the check box won't be marked. You must set the system variables before you execute the [run report](#) or [run report with name](#) statement.

You can use the **printer** clause to specify the printer to which the report will be printed. If you don't include the **printer** clause or named printers are not supported on the current platform, the report will be sent to the currently-selected printer.

Exporting data to a file

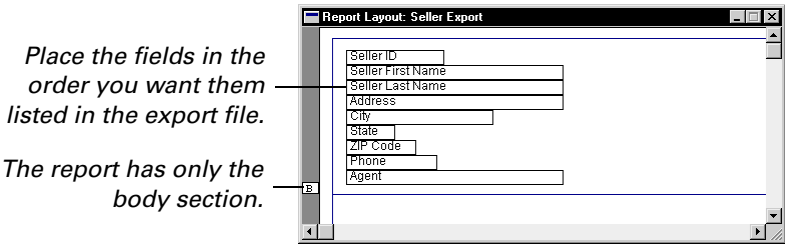
You may find it useful have a report that always prints to a file. For example, a list of customer names and addresses could be exported to a file for use with word processing software as a mailing list.

To export data to a file using the **destination** clause of the [run report](#) or [run report with name](#) statement, you must specify the file format to be used, as well as the name of the file. The file name must be enclosed in quotation marks. If the complete path (in generic format) for the file isn't specified, the file will be saved in the current directory.

Format	Description
Text file	The report will be saved as text without any formatting. This option should be used when the application to which you're converting the document is unable to read any of the other file formats.
Tab-delimited	This is the tab-separated ASCII character format used by programs such as Microsoft Excel®.

Format	Description
Comma-delimited	This is the standard comma-separated ASCII character format often used by database programs.
HTML	The report will be saved in HTML format to be viewed in a web browser.
XML	The report will be saved as an XML output file that contains a representation of the report layout and all of the report data.
PDF File	This format is available if you have the PDFWriter printer driver installed (included with Acrobat 5 and earlier), or Acrobat Distiller from Acrobat 6 or later. PDF (Portable Document Format) files can be read using the Acrobat Reader software available from Adobe.

If you will be using reports to export data to a tab-delimited or comma-delimited files, you must place fields in the report layout in a special way so the export file has the proper format. The fields to export must be placed in the body of the report, in the order they are to appear in the export file. The following illustration shows the layout of a report to export seller data from the Real Estate Sales Manager sample application.



You must also add the **ExportOneLineBody=TRUE** setting to the defaults file. This setting causes the items in the report body to be exported to a single line in the export file. The following illustration shows the output file when the report above is exported to a comma-delimited file.



Using temporary tables

You can create reports that use temporary tables, either as the main table or as a secondary table. Like other tables, temporary tables are defined in Dexterity using the Table Definition. Refer to [Chapter 9, “Tables,”](#) for more information about creating temporary tables.

You can treat temporary tables the same as any other Dexterity tables when creating reports. Their table names are included in the scrolling window from which you choose your report’s main table in the Report Definition window. They’re also included, when appropriate, in the Report Tables window from which you select secondary tables.

Temporary tables are useful for reports that need to access data from multiple tables with one-to-many relationships with other report tables. You can create a temporary table specifically for use by a report, and use sanScript to fill that table with the information you need. You can then access the temporary table to retrieve the desired data for your report.

You must use test mode or the runtime engine to test reports that use temporary tables.

One thing differentiates reports that use temporary tables from other reports created using the Report Writer. Reports that use temporary tables can’t be run in tools mode using the Print button on the Reports window. They must be run in either test mode or with the runtime engine, since, as their name implies, temporary tables are temporary and don’t exist outside of the runtime or test mode environments.

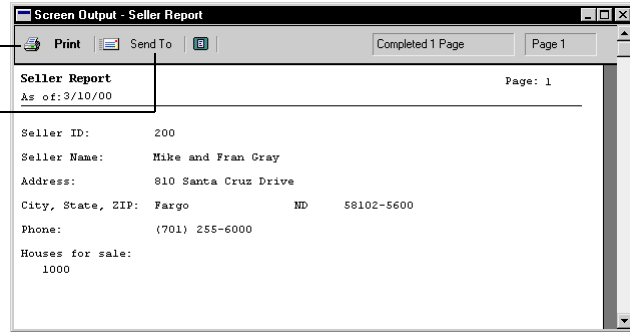
Mail connectivity

The Report Writer allows you to mail reports using mail connectivity applications. Dexterity supports Microsoft’s Mail Application Program Interface (MAPI), which allows you to send mail to other users with a MAPI mailbox. If your workstation has one of these mail systems enabled, you can mail reports from the Screen Output window.

To mail a report to another user, select the Screen option in the Report Destination window when you print a report. The Screen Output window will appear. If a user’s workstation has MAPI enabled, the Send To button drop list will be enabled.

You can mail the report as a text attachment. If you have the PDFWriter printer driver installed (included with Acrobat 5 and earlier), or Acrobat Distiller from Acrobat 6 or later, you can also mail the report as a PDF (Portable Document Format) attachment. Information about Adobe Acrobat is available at www.adobe.com.

Click Print to send the report to the printer.
Use Send To to mail the report to another user.



Enter the mail address in the dialog box that appears and click OK. The report's name will appear in the subject line of the mail message. If you chose a text attachment, the report will be sent with a .TXT extension. The user receiving the report can use the application that's associated with this extension to view the file. If you chose a PDF attachment, the report can be read with Adobe's Acrobat Reader.

Part 8: SQL Database Manager

This part includes information you'll need if you plan to use Dexterity to create applications for use with Microsoft SQL Server. It contains information about SQL-specific concepts, such as data sources, connections, cursors and stored procedures. Following is a list of topics that are discussed:

- [Chapter 43, "Data Sources,"](#) explains how to set up Microsoft SQL Server and create data sources necessary to access it.
- [Chapter 44, "Logins and Connections,"](#) explains how to log into a SQL data source from a Dexterity application. It also describes connections and how they are managed by Dexterity.
- [Chapter 45, "Cursors,"](#) explains the SQL-specific concept of a cursor, and provides information about manipulating the cursor to optimize your application's performance and provide users with the most current data from the application's tables.
- [Chapter 46, "Locking,"](#) provides information about the way in which active locking is implemented for use with SQL.
- [Chapter 47, "SQL Tables,"](#) explains how to create SQL tables, methods to add data to them, and how to grant privileges.
- [Chapter 48, "Stored Procedures,"](#) discusses SQL stored procedures and procedures you'll need to add to your Dexterity application to properly execute stored procedures.
- [Chapter 49, "Pass-through SQL,"](#) explains how to use pass-through SQL in an application.

Chapter 43: Data Sources

Before you can use Microsoft SQL Server with Dexterity, you must install and properly configure the Microsoft SQL Server software. Dexterity accesses SQL data through Open Database Connectivity (ODBC), so you must also configure ODBC data sources. Information about data sources is divided into the following sections:

- [*Installing Microsoft SQL Server*](#)
- [*Overview of data sources*](#)
- [*Configuring ODBC data sources*](#)

Installing Microsoft SQL Server

To use this release of Dexterity, you must use Microsoft SQL Server 2005 or later. Earlier versions of SQL Server aren't supported.

When you install Microsoft SQL Server, the installation process requires that you select a sorting order. To ensure that sorting functions properly in a Dexterity application, you must choose either **Binary Order** or **Dictionary Order, Case-insensitive**.

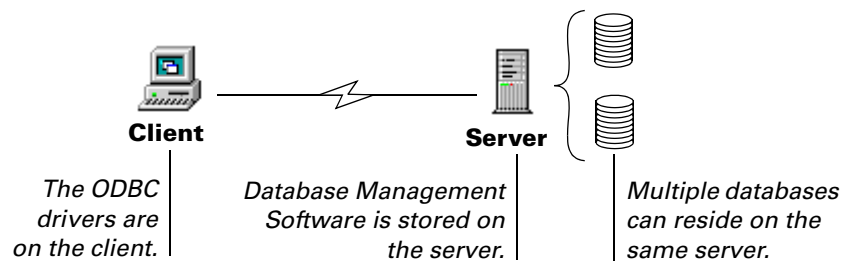


If you choose Binary Order, data in the SQL database will be sorted in exactly the same manner as it is for c-tree tables.

Overview of data sources

A data source is a database plus its associated Database Management System (DBMS) and any connections that are necessary for a client to access the data in a database. A data source is defined using the ODBC Administrator to ensure that the ODBC drivers used to connect to a given database have all the necessary information to successfully link to that database.

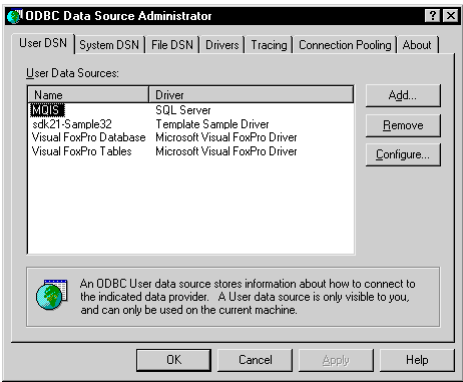
As shown in the following illustration, multiple databases can be stored on a single server.



Configuring ODBC data sources

To ensure that Dexterity-based applications store and retrieve SQL data properly, you must configure your ODBC data sources as described in the following procedure.

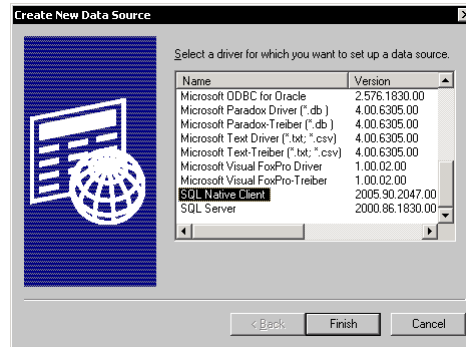
Launch the ODBC Administrator by double-clicking the ODBC icon in the Windows Control Panel. This will launch the ODBC Administrator application, and a Data Sources window similar to the one shown in the following illustration will appear.



- 1. Add a data source.**
Click **Add** to add a new data source.

2. Select the data source type.

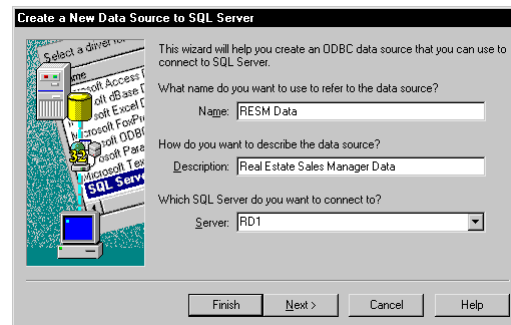
A Create new Data Source window will appear.



In this window, select the driver you want to use. For SQL Server 2005 or later, select SQL Native Client. Click Finish.

3. Use the wizard to specify information about the data source.

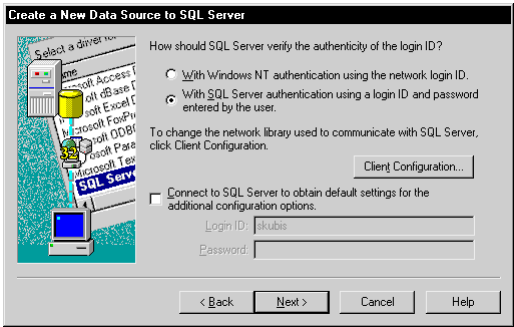
The next several windows are part of a wizard that will help you set up the ODBC data source. The first window for the wizard is shown in the following illustration.



Supply the name and description for the data source. Also specify the name of the server you want to connect to, then click Next to continue.

4. **Specify connection options.**

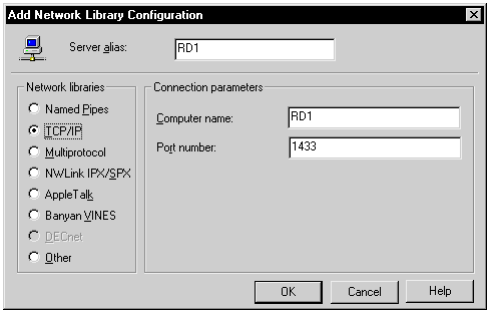
Be sure to select the option indicating you will use SQL Server authentication.



We don't recommend marking the option to connect to the SQL Server, because of security considerations.

5. **Specify the Client Configuration.**

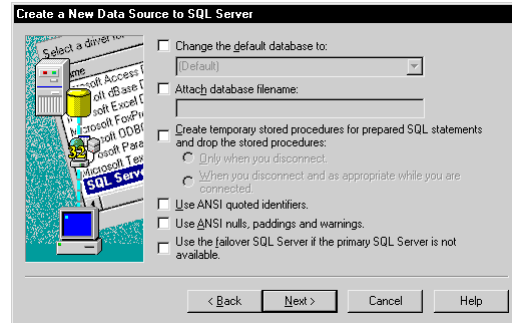
Click the Client Configuration button to specify the network library you want to use to communicate with the SQL Server.



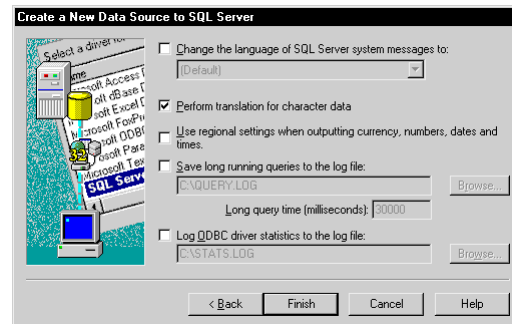
For optimal performance, we recommend choosing TCP/IP. Click OK when you have made your selection, and then click Next to continue.

6. Specify database options.

Next, specify the options for the database. Be sure that all of these options are **unmarked**.



Click Next to continue.



Specify additional options for the database. Be sure the **Perform translation for character data** option is marked. Click Finish to create the data source.

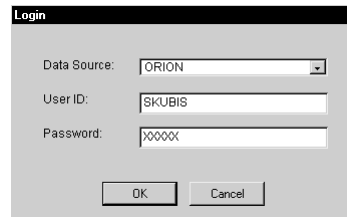
Chapter 44: Logins and Connections

To access SQL data, an application must log into a SQL data source. Once logged in, the application can create connections to the SQL server and perform database operations with the SQL data. Information about logins and connections is divided into the following sections:

- [Logins](#)
- [Connections](#)

Logins

Before users of your application can access SQL data, they must log into a SQL data source. When a user first attempts to perform a task on the data source, such as opening a table, a dialog will be displayed prompting them to log in. The Login window is shown in the following illustration.



The data source name is created using the administrative functions of the ODBC drivers. The user ID and passwords are created using your database management system (DBMS). Refer to your ODBC and DBMS documentation for information on valid syntax for the data source, user ID and password fields.

The login process verifies that the specified user has access to the selected data source, and confirms that the password entered is associated with that user. If the login is successful, the user can perform operations on the data source. The user doesn't have to log in again for subsequent database operations.

A login may fail for any of the following reasons:

- A user had already logged in during the current session.
- The user ID and password combination entered wasn't valid for the selected data source.
- The maximum number of connections to the server had been reached.
- The login attempt wasn't completed within the time allotted, as specified by the **SQLLoginTimeout** defaults file setting.

Displaying a login window

In some cases, you may want to control when the user logs into a SQL data source. You can use the **Login.OpenDexDialog()** function to open and control Dexterity's Login window, rather than have the Login window open automatically when user first performs a database operation.

Refer to the [Login function library](#) in the *Function Library Reference manual* for information about functions used to implement a SQL login window.



For security and consistency reasons, we recommend that your application require users to log in as soon as the application is started.

If you don't want to use Dexterity's Login window, you can use functions from the Login function library to incorporate the login process into one of your application's windows. Creating your own window will allow you to combine logging into your application with logging into the data source.

Login security

While you may want to create your own Login window to control user access to certain resources, you can rely on the data source security to provide a basic level of security for your application. To do this, you must require a user to log into the data source before your application starts.

The **SQLNumInitialConnections** defaults file setting can be used to force a user to log in before your application is started. If the number of initial connections required is one or more, the specified number of connections to the data source must be established before your application will start. The Dexterity Login window will automatically be used to prompt the user for login information. If the login fails, the user will be prohibited from opening and using your application.

Two other defaults file settings can affect the security of your application: **SQLNumLoginTries** and **SQLPassword**.

- The **SQLNumLoginTries** setting limits the number of consecutive failed login attempts your application allows before closing. However, closing the application doesn't prevent the user from restarting it and trying again.
- The **SQLPassword** setting can have significant impact on your application's security. If the defaults file includes valid values in the **SQLPassword**, **SQLUser** and **SQLDataSource** settings that allow access to the named data source, the login window will be bypassed. Anyone using that computer will be able to access your application and the related data source.

Not including the **SQLUser** and **SQLDataSource** settings in the defaults file won't reduce the security vulnerability if the Dexterity Login window is used. In the absence of those two settings, Dexterity will add the **SQLLastDataSource** and **SQLLastUser** settings to the defaults file once a user has accessed a data source from that workstation. Excluding the **SQLPassword** setting from the defaults file will help ensure the security of your application.



*The **SQLPassword** setting was created to allow developers to quickly and easily test their applications. To preserve client security, we recommend that it not be used in an application's release version.*

Connections

Connections are the links between the client and a data source. Connections allow the user at a client to issue commands to be executed on the server, and provide a pathway for the server's response to the client. A command sent to the server and the server's response is considered one statement. Multiple statements can be executed using the same connection.

Connection limits

The **SQLMaxConnections** setting in the defaults file allows you to define the maximum number of connections your application can maintain simultaneously between a client and server. The connectivity package used (such as TCP/IP) determines the maximum number of connections allowed between a client and a server. If this setting isn't used in the defaults file, the default value of 32 will be used. This is the maximum number of connections allowed by some TCP/IP packages.



*Be sure to consult the documentation of the connectivity package to be used with your application. If the package allows fewer than 32 connections, be sure to set the **SQLMaxConnections** defaults file setting to the number of connections allowed by that package, or lower. The **SQLMaxConnections** setting should never exceed the number of connections allowed by that package.*

If the maximum number of connections is reached before the application maximum is reached, the application may freeze or shut down. To avoid this, we recommend that you set the number of connections lower than the maximum number allowed by the connectivity package. If a user runs more than one application that uses connections to a server, the applications must share the limited number of connections offered by the connectivity package.



Applications you create will rarely require more than five simultaneous connections.

If a user reaches the maximum number of connections set for your application, a message will appear stating that the operation can't be executed because the maximum number of connections has been reached. The user will be returned to the point where he or she initiated the operation, and will be able to try again, in case a connection has become available.

Connection pools

Establishing a connection between a client and server at runtime can be time-consuming. Dexterity allows you to design your application to keep open connections in reserve, thus limiting the amount of time spent establishing connections. These open but inactive connections are stored in a *connection pool*, where they can be accessed and used immediately.

The size of the connection pool is specified by the **SQLMaxInactiveConnections** defaults file setting. A connection pool can be created when an application is started, using the **SQLNumInitialConnections** and **SQLMaxInactiveConnections** defaults file settings. If you want the connection pool to contain three connections at startup, you should set both defaults file settings to 3.



*If the **SQLNumInitialConnections** setting is greater than zero, the Dexterity Login window will be used to establish the connections to the data source, even if you have designed your own Login window. This is because your window is defined in your dictionary, and the connections are created before your dictionary is opened.*

If you choose to use your own Login window, you must set the **SQLNumInitialConnections** setting to 0 (zero). However, if the **SQLMaxInactiveConnections** setting is greater than 0 (zero), the connection established for use by your Login window will be added to the connection pool when the login has been successfully completed.

The default setting for **SQLNumInitialConnections** is zero, and the maximum setting is 5. For **SQLMaxInactiveConnections**, the default setting is 3, and the maximum setting is 5. If the number of initial connections is less than the number of inactive connections, then any new connection that is established will be stored in the connection pool when it's no longer needed.

When a connection is needed, one is taken from the connection pool and used. If no connections are available in the pool, a new connection will be established and used. As connections are no longer needed, they're returned to the connection pool until the maximum number of inactive connections is reached. When the maximum number is reached, any connections that are no longer needed will be released.

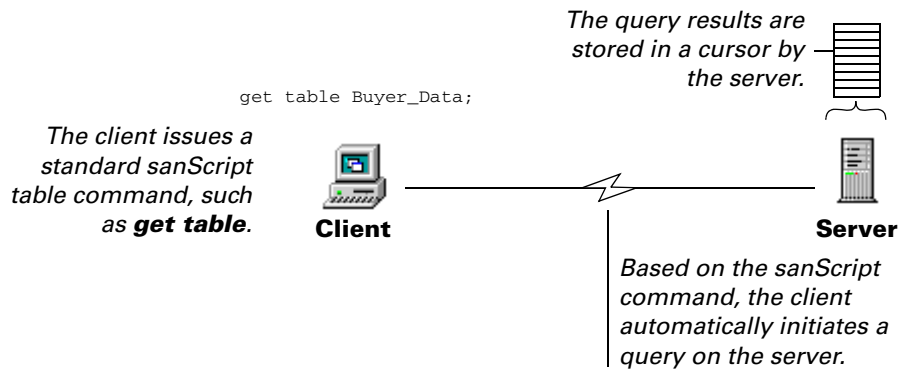
Chapter 45: Cursors

Dexterity uses *cursors* to bring information from the server to a buffer on the client. Information about cursors is divided into the following sections:

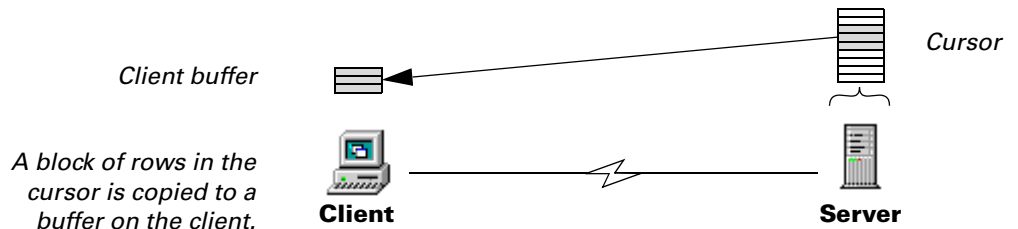
- [Cursor overview](#)
- [Cursor block size](#)
- [Stale data](#)
- [Cursor refreshing](#)

Cursor overview

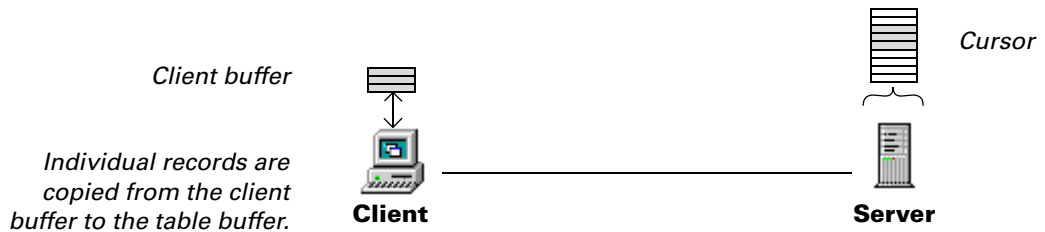
When you use standard table commands in your sanScript code such as [get](#) or [change](#), Dexterity automatically runs a query against the appropriate table on the server. The results of the query are gathered in a cursor.



Groups of records in the cursor are copied to a buffer on the client as they are needed. Each group of records is called a *cursor block*.



Then individual records are copied from the client buffer to the table buffer.



When the next block of data in the cursor is needed, the current data in the client buffer is released. The table on the server is then queried, and the next block of result data is retrieved and copied to the client buffer.

From sanScript, accessing data in a SQL data source is fundamentally no different than accessing data in c-tree or Pervasive.SQL tables. But, because of the use of cursors, there are some issues you must be aware of. These issues are described in the following sections.

Cursor block size

You can specify the number of records to be included in the block of records copied from the cursor to the client buffer. The maximum cursor block size allowed by Dexterity is 255 records. You can specify the cursor block size in one of three ways:

- Use the **SQLCursorBlockSize** setting in the defaults file.
- Specify the block size when using the sanScript **open table** statement.
- Select the Custom block size option in the Table Options window, and specify the number of records in the cursor block.



The default cursor block size is 25. As the block size increases, a degradation of your application's performance may occur.

What block size is reasonable depends on the size of the data set. The greater the size of the data set and the smaller the cursor block, the greater the number of queries made against the database, and the greater the impact on performance. We recommend that you use the default block size of 25 for the best overall performance.

Stale data

A side-effect of cursors is *stale data*. Stale data occurs when records have been changed in a table, but the corresponding records in client buffers have not been updated to reflect the changes. The following scenario illustrates how stale data occurs.

- Client A reads a record from a table. In this process, 25 records (the default block size) are copied to the client buffer.
- Client B reads and makes changes to a record from the same table as client A. The record that was changed happens to be one that was read into client A's client buffer.
- If client A accesses the record that client B changed, it will read the record from the client buffer. This record has become stale, because the old record in the client buffer will be used, not the updated record in the table.

You won't be able to eliminate stale data, but you can reduce the chances of it occurring. One solution is to reduce the cursor block size. By storing fewer records in the client buffer, you will reduce the number that could become stale. You can also use cursor refreshing, described in the next section, to help reduce the occurrence of stale data.

Cursor refreshing

Cursor refreshing is the act of replacing the data associated with a cursor with fresh data from the database. Cursors will be refreshed by any of the following instances:

- Changing the range used to access the table
- Using a different key to access the table
- Closing and reopening the table
- Using the **refresh** keyword when issuing a [get](#), [change](#) or [edit table](#) statement

In some of these instances, the record a user was last positioned on before the refresh occurred may not be included in the cursor block of data after the refresh. In the first two instances, the query parameters have changed, and the record may not fit the new criteria. There's a greater chance of that record still being included after the refresh in the last two instances, since the parameters haven't changed. However, that record was still subject to changes from other users, who may have deleted it from the table or changed its key values.

If the last record the user was positioned on is still included in the table after the refresh occurs, the cursor block containing that record will be copied to the buffer on the client. If that record isn't included in the table, the cursor block containing the record that followed that one in the table will be copied to the buffer on the client.



It's a good idea to refresh the cursor if your application runs in a multiuser environment, if your cursor block size is large, or if the data involved is subject to frequent changes.

Chapter 46: Locking

Active locking is implemented differently for the SQL database type than it is for the c-tree or Pervasive.SQL database types. The SQL implementation of active locking tracks locks on a per-client basis.



We recommend that you rely on passive locking instead of active locking whenever possible. Using active locking with the SQL database type can cause performance degradation.

This portion of the documentation explains the implementation of active locking for applications using the SQL database type, and outlines the steps you need to take to prepare your SQL Server databases to accommodate the Dexterity locking scheme. Information about locking is divided into the following sections:

- [Table definition requirements](#)
- [Session management](#)
- [The DEX_LOCK table](#)
- [Creating the DEX_LOCK and DEX_SESSION tables](#)
- [Clearing stranded active locks](#)

Table definition requirements



Active locking must be enabled on a per-table basis by marking the Allow Active Locking option in the table's Table Definition window.

For both active and passive locking to work properly in your application, every SQL table must include a column that is used to track the identity of individual records being locked.

If Dexterity creates your tables for you, this column will be added to each table automatically. You shouldn't include this column in the table definitions you create using the Table Definition window.

If you create your tables through a method other than allowing the Dexterity runtime engine to create them, you *must* include a column in each table that is defined with the following characteristics:

Column name	Type	Length	Nulls	Identity column
DEX_ROW_ID	integer	4	0	YES

Dexterity uses this column internally. You don't have to manipulate the column at all.



If you create your own tables outside of the Dexterity environment and do not include the DEX_ROW_ID column in each of your tables, active locking will not function properly for that table.

Session management

The time a client is connected to a server is called a *session*. Each time a client logs into a SQL database using a Dexterity-based application, a unique session ID is generated to identify that session. If multiple sessions are run from a single client computer simultaneously, each session is assigned its own unique session ID. If a client logs out and logs back into a database, the new session will be assigned a different session ID.

Dexterity uses a table named DEX_SESSION to generate the session ID. This table must be located in the standard SQL Server database named *tempdb*, and be defined to include the following columns with the specified attributes:

Column name	Type	Length	Nulls	Identity column
session_id	int	4	0	YES
sqlsvr_spid	small int	2	0	NO

This table's primary key consists of the session_id column. When a new session is established, the ID used by the server to track that session is added to the sqlsvr_spid column. Using this information, a session ID is generated, stored in the session_id column and returned to Dexterity.

Session tables

In addition to creating a session ID for each session, Dexterity creates a table in the standard SQL Server database named *tempdb*, and names the table DLxxxxxxxx, where xxxxxxxxx is the session ID. This table will remain open until the session associated with that session ID is closed.

This session table is used to verify whether a specific session is still active. If a session table for a given session ID doesn't exist, then that session is no longer connected to the server, and any locked records associated with that session ID can be unlocked.

For more information about removing locks on records held by disconnected clients, refer to [Clearing stranded active locks](#) on page 389.

The DEX_LOCK table



The DEX_LOCK table is used to track only active locks. If your application doesn't implement active locking, you can skip this section.

Dexterity uses the DEX_LOCK table to track active locks acquired on records. This table must be located in the standard SQL Server database named *tempdb*, and be defined to include the following columns with the specified attributes:

Column name	Type	Length	Nulls	Identity column
session_id	int	4	0	NO
row_id	int	4	0	NO
table_path_name	char	100	0	NO

This table's primary key is based on a combination of the row_id and table_path_name columns. The other column in this table, the session_id column, contains the session ID generated by the DEX_SESSION table, identifying the user who has the specified record locked.

Each time a user locks a record, a row is inserted in the DEX_LOCK table containing the following information:

- The session ID assigned to the user's current session
- The row ID of the record the user locked
- The pathname of the table containing the locked record

Since the DEX_LOCK table's primary key is based on the row ID and table pathname, and since primary keys do not allow duplicate entries with the same key values, a single record cannot be referenced in this table more than once at any given time. This prevents multiple users from actively locking a single record simultaneously.

When the lock on a given record is released, the row related to that record is deleted from the DEX_LOCK table. When the row is removed from the table, the record's row ID and the pathname of the table containing the record will no longer be key values in the DEX_LOCK table. Thus, another user may actively lock the record.

Creating the DEX_LOCK and DEX_SESSION tables

Since the DEX_LOCK and DEX_SESSION tables reside in the standard SQL Server database named *tempdb*, they must be created each time that database is created. Typically, this occurs each time the server is restarted.

If you are creating an application that integrates with Microsoft Dynamics GP, you will not have to create these tables. Stand-alone applications, however, must ensure these table is created each time the server is restarted.

If you use Dexterity or the runtime engine to log into a SQL data source as “sa”, and the DEX_LOCK or DEX_SESSION tables don’t exist, they will be created automatically in *tempdb*. Also, a stored procedure named *smDex_Build_Locks* that automatically re-creates these tables is added as a startup procedure. The following is the stored procedure that is added.

```
if exists (select * from sysobjects where id = object_id
          ('dbo.smDEX_Build_Locks') and sysstat & 0xf = 4)
drop procedure dbo.smDEX_Build_Locks
GO

create procedure dbo.smDEX_Build_Locks
as
if not exists (select * from tempdb..sysobjects where id =
              object_id('tempdb..DEX_LOCK') and sysstat & 0xf = 3)
begin
    exec ('create table tempdb..DEX_LOCK (session_id int, row_id int,
                                          table_path_name char(100))')
    exec ('create unique index PK_DEX_LOCK on
          tempdb..DEX_LOCK(row_id,table_path_name)')
end
if not exists (select * from tempdb..sysobjects where id =
              object_id('tempdb..DEX_SESSION') and sysstat & 0xf = 3)
begin
    exec ('create table tempdb..DEX_SESSION (session_id int identity,
                                             sqlsvr_spid smallint)')
    exec ('create unique index PK_DEX_SESSION on
          tempdb..DEX_SESSION(session_id)')
end
exec ('use tempdb grant insert,update,select,delete on DEX_LOCK to
      public')
exec ('use tempdb grant insert,update,select,delete on DEX_SESSION to
      public')
return
GO
```



The DEX_SESSION table must exist regardless of whether you implement active locking in your application. The DEX_SESSION table must exist for both active and passive locking to function properly. If you aren't implementing active locking, you can exclude the portion of the stored procedure that creates the and sets access to the DEX_LOCK table.

Clearing stranded active locks

A *stranded lock* is an active lock retained on a record, even though the session associated with that lock is no longer active. If a stranded lock exists on a record, no other user can actively lock that record, although they could still read the record, using a get statement, for example. Stranded locks are most often caused by a user abnormally terminating his or her session.

When the session for a given session ID is terminated, either normally or abnormally, the session's connection to the server is dropped. When this occurs, the session table associated with that session ID is deleted from *tempdb* by the SQL server.

If a session associated with a given session ID is concluded normally, Dexterity checks the DEX_LOCK table once the session table is deleted and removes all rows containing that session ID in the session_id column. Thus, it removes any remaining locks associated with that session ID.

When a session associated with a given session ID is concluded abnormally, the session table is removed but Dexterity does not automatically clear locks associated with that session ID. The locks must be cleared by some other means.

If your application integrates with Microsoft Dynamics GP, stranded locks will be cleared for you. Stand-alone applications, however, must include a means to clear these locks.

To ensure that stand-alone applications recognize and remove stranded locks on a regular basis, we suggest you include a stored procedure in your application similar to the one shown in the following script. You can call this stored procedure from your Dexterity login script, thereby checking for and removing stranded locks each time a user logs in.

```

create procedure Release_Stranded
as
declare @DEX_LOCK_ID char(15)
declare @iSession_ID int
declare T_cursor CURSOR for select session_id from tempdb..DEX_LOCK
set nocount on
OPEN T_cursor
FETCH NEXT FROM T_cursor INTO @iSession_ID
WHILE (@@FETCH_STATUS <> -1)
begin
    select @DEX_LOCK_ID = stuff( '##DL0000000000', 14 -
        datalength(rtrim(ltrim((convert(char(10),@iSession_ID)))))+1,
        datalength(rtrim(ltrim((convert(char(10),@iSession_ID ))))),
        convert(char (10), @iSession_ID))

    if not exists ( select name
        from
            tempdb..sysobjects
        where
            tempdb..sysobjects.name = @DEX_LOCK_ID)
    begin
        /*This is a stranded lock, so clear it.*/
        delete from tempdb..DEX_LOCK where
            tempdb..DEX_LOCK.session_id = @iSession_ID
    end
    FETCH NEXT FROM T_cursor INTO @iSession_ID
end
DEALLOCATE T_cursor
return
go

```

To create the stored procedure, copy the code shown here and paste it into a utility such as ISQL that can execute SQL statements. When you execute the SQL statements shown here, the Release_Stranded stored procedure is created. Then you can create a prototype procedure and use the [call sproc](#) statement to call this stored procedure from within your Dexterity application. Calling stored procedures is described in [Chapter 48, “Stored Procedures.”](#)

Chapter 47: SQL Tables

This portion of the documentation contains additional information about SQL tables created by Dexterity-based applications. The following topics are discussed:

- [*Creating SQL tables*](#)
- [*Auto-generated stored procedures*](#)
- [*Granting access to tables*](#)
- [*Paths for SQL tables*](#)
- [*SQL error handling*](#)
- [*Encrypted fields for SQL Server*](#)
- [*Accessing existing SQL data*](#)

Creating SQL tables

The easiest way to create the tables on the SQL data source for your application is to allow Dexterity to create them the first time they are accessed. This method ensures that the tables have the correct structure, and have the proper indexes and stored procedures.

You may choose to create SQL tables using some method external to Dexterity. If you do this, keep in mind that the tables must include the DEX_ROW_ID column, as described in [*Chapter 46, "Locking."*](#) The tables must also have the necessary indexes and auto-generated stored procedures for the tables to be accessed properly. You can use the [*Table CreateIndexes\(\)*](#) and [*Table CreateProcedures\(\)*](#) functions in Dexterity to create the indexes and auto-generated stored procedures for such tables. Refer to the Function Library Reference for more information about these two functions.

Auto-generated stored procedures

To optimize database performance, Dexterity automatically generates several stored procedures for each table in an application. Typically, Dexterity generates these auto-generated stored procedures the first time it opens a table. If you create a table by some means other than Dexterity, you must use the [*Table CreateProcedures\(\)*](#) function to generate the stored procedures for the table.

Stored procedures created by Dexterity have the form zDP_ followed by the table physical name and a code indicating the purpose of the stored procedure. The following table lists the various stored procedures created by Dexterity.

Stored procedure	Purpose code	Quantity	Comments
Insert	SI	One per table	Created only if the table doesn't contain text or picture fields.
Delete	SD	One per table	Created only if active locking is not allowed for the table.
Select	SS	One for each key	
First	F	One for each key	
Last	L	One for each key	
Next	N	One for each key	
Unpositioned next	UN	One for each non-unique key	

For example, the Seller_Data table has the physical name SELLDAT. The table has three keys, one of which is unique. Dexterity creates 13 stored procedures for this table. The stored procedures are listed in the following table.

Stored procedure	Description
zDP_SELLDATSI	Insert stored procedure
zDP_SELLDATSD	Delete stored procedure
zDP_SELLDATSS_1 zDP_SELLDATSS_2 zDP_SELLDATSS_3	Select stored procedures (one for each key)
zDP_SELLDATF_1 zDP_SELLDATF_2 zDP_SELLDATF_3	Select first stored procedure (one for each key)
zDP_SELLDATL_1 zDP_SELLDATL_2 zDP_SELLDATL_3	Select last stored procedure (one for each key)
zDP_SELLDATN_1 zDP_SELLDATN_2 zDP_SELLDATN_3	Select next stored procedure (one for each key)
zDP_SELLDATUN_2 zDP_SELLDATUN_3	Unpositioned next procedures (one for each non-unique key)

When optimizing the performance of your application, it is sometimes useful to not use some or all of the auto-generated stored procedures. Refer to [SQL Auto Procedure Disable Options](#) on page 109 for information about situations when disabling auto-generated stored procedures can improve application performance.

Granting access to tables

When you create SQL tables from within Dexterity, the user who created the tables will be the only person able to access them. The same situation occurs if you create tables outside of Dexterity using a utility such as ISQL. You must grant access privileges to the tables and associated stored procedures for other users to access data in your application.

Refer to [Chapter 48, "Stored Procedures,"](#) for more information about using stored procedures with Dexterity.

You can use a tool like ISQL to manually grant privileges to each table and its associated stored procedures, though this could be tedious for larger applications. A more automated method is to use a stored procedure to grant access privileges to a table. The SQL statements create a stored procedure that grants access privileges to the members of DEXGRP for the specified table.

To create the amAutoGrant stored procedure, copy the code shown here and paste it into a utility such as ISQL that can execute SQL statements. When you execute the SQL statements shown here, the amAutoGrant stored procedure is created. Then you can create a prototype procedure like the one shown following the SQL statements below and use the [call sproc](#) statement to call the amAutoGrant stored procedure from within your Dexterity application. Calling stored procedures is described in [Chapter 48, "Stored Procedures."](#)

```
/* amAutoGrant
   This procedure grants permissions on the passed table and its
   associated zDP_* stored procedures
*/
SET QUOTED_IDENTIFIER OFF
if exists (select * from sysobjects where id =
object_id('amAutoGrant'))
    drop procedure amAutoGrant
go

if exists (select * from sysobjects where id = object_id('autotemp'))
    drop table autotemp
go
```

```
create table autotemp(name char(150))
go

create procedure amAutoGrant
    @tablename char(150) output
as
set nocount on

DECLARE @command varchar(255)

/* Do the table grant*/
SELECT @command = 'grant SELECT,INSERT,DELETE,UPDATE on
'+rtrim(@tablename)+' to DEXGRP'
EXEC (@command)

/* Set up the procedure select */
DELETE FROM autotemp
SELECT @command = "insert into autotemp select name from sysobjects
where name like 'zDP_"+rtrim(@tablename)+"%"
EXEC (@command)

/* Declare the cursor */
DECLARE TheCursor CURSOR FOR
    select 'grant EXECUTE on '+ rtrim(name) + " to DEXGRP" from
autotemp

/* Open the cursor */
OPEN TheCursor

WHILE @@FETCH_STATUS = @@FETCH_STATUS      -- That is, 'WHILE TRUE'.
BEGIN
    FETCH NEXT FROM TheCursor INTO @command

    IF @@FETCH_STATUS = -2                  -- Row has been deleted.
        CONTINUE
    IF @@FETCH_STATUS = -1                  -- All rows processed.
        BREAK

    /* Execute the select for each proc */
    EXEC (@command)
END
```



```

/* Close and deallocate the cursor */
CLOSE TheCursor
DEALLOCATE TheCursor
GO

SET NOCOUNT off
GO

grant execute on amAutoGrant to DEXGRP
go

```

The following is the prototype procedure required to call this stored procedure.

```

sproc returns long Ret_Code;
inout string table_physical_name;

{This is the prototype procedure for the amAutoGrant stored procedure.
Note that you must supply the table physical name.}

call sproc "amAutoGrant", Ret_Code, table_physical_name;

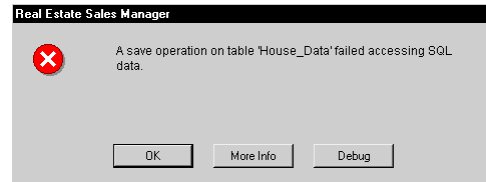
```

Paths for SQL tables

The SQLPath procedure is used to specify where SQL tables are located on a the data source. Refer to [Chapter 2, "Pathnames,"](#) in the Dexterity Stand-alone Application Guide for a complete description of the SQLPath procedure and pathnames for SQL tables.

SQL error handling

Dexterity allows you to control how SQL error messages will appear to the user. When a SQL error occurs, Dexterity's default behavior is to display a dialog like the one shown in the following illustration.



If your application integrates with a product that uses SQL, refer to the end of this section for information about using the SQLError procedure.

The user can click More Info to display specific information about the SQL error that occurred. For some SQL errors, you may want to suppress this dialog, display your own message, or log the message in a table. You can use the **SQLError** procedure to do this.

SQLError procedure

If you have added the SQLError procedure to your application, it will run each time a SQL error occurs. The SQLError procedure is automatically passed parameters containing information about the error that occurred. Based on this information, you specify a return value from the SQLError procedure indicating how the error message will be handled.

The SQLError procedure uses several parameters; six in parameters that are automatically passed into the script, and a single out parameter that specifies how the error message will be handled.

- *internal_table_ID* – A long integer containing the internal ID number of the table for which the SQL error occurred.
- *dict_ID* – An integer containing the ID of the dictionary that was active when the SQL error occurred.
- *table_series* – An integer containing the series of the table for which the SQL error occurred.
- *table_ID* – An integer containing the resource ID of the table for which the SQL error occurred.
- *operation* – An integer specifying the table operation that caused the SQL error. This value is provided for error logging purposes only. It is subject to change in future versions of Dexterity. The SQLError procedure should not perform conditional operations based on this value.
- *resource_name* – A string containing the name of the script or report that was being run when the SQL error occurred.

- *error_response* – An integer out parameter specifying how the SQL error message will be handled. The response corresponds to one of the following values:

Value	Description
0	Use Dexterity's default behavior. Display the SQL error message and stop the script or report that caused the error.
1	Don't display the SQL error message, and stop the script or report that caused the error.
2	Don't display the SQL error message, and continue running the script or report that caused the error.

Reading error information

Within the `SQLException` procedure, you will want to be able to read the error text that would normally be displayed to the user. To do this, use the [Table `GetSQLExceptionText\(\)`](#) function. The information retrieved can be displayed to the user or logged in a file. Refer to the description of the [Table `GetSQLExceptionText\(\)`](#) function later in this document for more information.

Sample `SQLException` procedure

The following `SQLException` procedure logs every SQL error that occurs in the application. The error information is stored in the c-tree table named `SQL_Error_Log`. The value 0 is returned to the *error_response* parameter so the default SQL error message is displayed to the user.

```
in long internal_table_ID;
in integer dict_ID;
in integer table_series;
in integer table_ID;
in integer operation;
in string resource_name;
inout integer error_response;

local integer status;
local string error_text;
local long native_error_number;

{Log the SQL error that has occurred.}
get last table SQL_Error_Log;
if err() = EOF then
    {This is the first error to log.}
    'Sequence Number' of table SQL_Error_Log = 0;
```

```

else
    {Log the next SQL error.}
    increment 'Sequence Number' of table SQL_Error_Log;
end if;

{Read the SQL error information.}
status = Table_GetSQLErrorText(internal_table_ID, error_text,
➡ native_error_number);
while status = STATUS_SUCCESS do
    {Successfully read an entry in the error text, so log the error.}
    'Log Date' of table SQL_Error_Log = sysdate();
    'Log Time' of table SQL_Error_Log = systime();
    'Resource' of table SQL_Error_Log = resource_name;
    'Native Error' of table SQL_Error_Log = native_error_number;
    'Error Text' of table SQL_Error_Log = error_text;
    save table SQL_Error_Log;

    {Read the next portion of the SQL error text.}
    status = Table_GetSQLErrorText(internal_table_ID, error_text,
➡ native_error_number);
end while;

{Use the default behavior that displays the SQL error to the user.}
error_response = 0;

```

Integrating applications

If your Dexterity application integrates with a product that uses SQL Server, you can create a procedure trigger for the `SQLError` procedure. Be sure to register the trigger to activate before the `SQLError` procedure runs. In the trigger processing procedure, check the `dict_ID` parameter from the `SQLError` procedure to find out whether it was your product that encountered the SQL error.



If you use the [Table_GetSQLErrorText\(\)](#) function in your trigger processing procedure, be sure to read all of the error messages. Otherwise, any other scripts that attempt to retrieve the error messages won't be able to retrieve all of the error text.

Encrypted fields for SQL Server

In versions of Dexterity prior to 5.5, encrypted strings were stored as character fields on SQL Server. We have become aware of situations that may cause data stored in these type of encrypted fields to become damaged. To prevent this problem, encrypted strings are now stored in binary fields, rather than character fields.

This section describes a procedure you can use to convert existing tables. It also provides recommendations for using encrypted fields in SQL tables.

Converting existing tables

If your application stores encrypted strings, and you have data stored on Microsoft SQL Server 7.0, you must perform a conversion procedure to convert the encrypted strings from character fields to binary fields. The following procedure describes how to convert a table that contains encrypted string fields.

1. Locate tables with encrypted string fields.

Find any tables that contain encrypted string fields. As an example, assume that the Agent field of the Seller_Data table is an encrypted string field.

2. Duplicate the tables that contain encrypted string fields.

Use Dexterity Utilities to duplicate each table that contains encrypted string fields. Continuing the example, the Seller_Data table was duplicated, and the new table was named Seller_Data_Temp.

3. Change the physical name of the new table.

In Dexterity, open the table definition for duplicate table that was created. Change the physical name of the table to something that uniquely identifies the table. In this example, the Seller_Data_Temp table was opened and its physical name was changed to SELLDATTEMP.



Verify that the new physical name you entered was saved correctly. You may need to open the Options window in Dexterity and mark the Allow Long Physical Table Names option.

4. Unmark the Encrypt option for each encrypted field.

In the table definition for the duplicate table, unmark the Encrypt option for each table field that is encrypted. In this example, the Encrypt option for the Agent field was unmarked.

5. Save the table.

Save the changes you made to the table definition.

6. Add the conversion procedure.

Add a procedure to your application to perform the conversion. The following procedure converts the Agent field in the Seller_Data table. This procedure requires the user to be logged into the appropriate data source before it can be run. The procedure performs the following actions:

- Drops the stored procedures for the current Seller_Data table.
- Uses pass-through SQL to rename the Seller_Data table so that it can be accessed using the duplicate table definition.
- Re-creates the Seller_Data table.
- Copies each record from the duplicate table to the Seller_Data table. The Agent field is read exactly as it is stored, decrypted using the [Utility DecryptTableString\(\)](#) function, then copied to the Seller_Data table.
- Deletes the duplicate Seller_Data table.

```
local long SQL_connection, status;
local boolean result;
local text SQL_Statements;
local integer int_result;

{Connect to the SQL data source.}
status = SQL_Connect(SQL_connection);
if status = 0 then

    {Drop the auto-generated stored procedures for the Seller_Data
    ➤ table.}
    int_result = Table_DropProcedures(0, table Seller_Data);
    if int_result <> 0 then
        error "Stored procedures could not be dropped.";
    end if;

    {Close the Seller_Data table.}
    close table Seller_Data;
```

```

{Use pass-through SQL to rename the Seller_Data table.}
{Build the SQL statements.}
SQL_Statements = "sp_rename SELLDAT, SELLDATTEMP";

{Execute the SQL statements.}
status = SQL_Execute(SQL_connection, SQL_Statements);
if status <> 0 then
    warning "Unable to rename the Seller_Data table.";
else
    {Turn on table auto-create.}
    result = Table_SetCreateMode(true);

    {Create the "new" Seller_Data table by opening it.}
    open table Seller_Data;

    {Read each record from the Seller_Data_Temp table.}
    get first table Seller_Data_Temp;

    repeat
        {Copy all of the fields.}
        copy from table Seller_Data_Temp to table Seller_Data;

        {Decrypt the Agent field.}
        Agent of table Seller_Data=Utility_DecryptTableString
        ─ (Agent of table Seller_Data_Temp);

        {Save the new data.}
        save table Seller_Data;

        {Get the next record.}
        get next table Seller_Data_Temp;
    until err() <> OKAY;

    {Delete the Seller_Data_Temp table.}
    close table Seller_Data_Temp;
    open table Seller_Data_Temp, exclusive;
    delete table Seller_Data_Temp;

    {Set access privileges for the new Seller_Data table.}

end if;

```

```

        {Disconnect from the SQL data source.}
        status = SQL_Terminate(SQL_connection);
    else
        {An error occurred creating the pass-through SQL connection.}
        warning "An error occurred creating the pass-through SQL
        ➔ connection: " + str(status);
    end if;

```

Recommendations

We recommend that you use as few encrypted fields as possible in SQL tables. If you must have encrypted data in your tables, but want the fields to be stored as character fields rather than binary fields, use the [Utility EncodeString\(\)](#) and [Utility DecodeString\(\)](#) functions to manually encrypt the data to be stored. These two functions can encode and decode string values in a form that can't be easily read by users or other applications.

Accessing existing SQL data

If you have an existing SQL database, you can use Dexterity to create a dictionary to access the data in that database. However, you must be sure to correctly name and define your data types, fields, tables and paths. If you don't define your dictionary's resources properly, you won't be able to access the existing data. The following sections provide information you will need to define your dictionary's resources properly.

Tables

A Dexterity table must be defined for each existing table in the database. Use the existing table's name as the physical name in the Dexterity Table Definition window. Be sure to include a table field for every column in the existing table.

Fields

A field must be defined in Dexterity for each *unique* column in the existing database. Be sure to use the column name as the field's physical name in the Field Definition window. Dexterity doesn't allow you to enter an underscore in the physical name field. If an existing column name contains an underscore, use a space in its place when defining the column's related field. For example, "Last Name" should be entered as the physical name for the field defined to correspond to the "Last_Name" column. Dexterity automatically converts the space into an underscore.

Data types

The Dexterity data types defined for the fields must have control types that correspond to the SQL data types defined for the columns in the existing tables. For example, a column in an existing table that is defined as a SQL integer must be defined as a long integer field in Dexterity.

Be sure that the keyable length of the data type defined for use with a given field is the exact length of the field's corresponding SQL column. If a field's keyable length is less than the actual length of the related SQL column, existing data will be truncated when it's displayed and saved using Dexterity.

Special care must be taken when defining the keyable length of string and text data types. The keyable length of a SQL char field is always the same as its storage size. In contrast, Dexterity string and text fields have length bytes added to their keyable length. One byte is added for string fields; two bytes are added for text fields. Since all SQL databases allow odd-length fields, you may want to select the Allow Odd Length Strings option in the Data Type Definition window when defining string or text data types in Dexterity.

For example, a SQL char field of keyable length 50 has an actual storage size of 50 bytes, while a Dexterity string field with a keyable length of 50 will have a storage size of 51, if odd-length strings are allowed. If you don't select the Allow Odd Length Strings option, a Dexterity string field of keyable length 50 will have a storage size of 52 bytes. The extra two bytes consist of a length byte and a pad to ensure an even-numbered storage size.

For more information on Dexterity data types, refer to [Chapter 6, "Data Types."](#)

Dexterity offers several different control types for you to base your data types upon, such as string, integer and currency. When you define a data type, you choose a control type that specifies how information will be stored and displayed in fields using that data type.

When you write SQL stored procedures, you use SQL data types to define the types of fields being specified. It's important that the SQL data type defined for a field in the stored procedure corresponds to the Dexterity control type of that field in your application.

Dexterity control types and SQL data types

The following table lists the Dexterity control types and their corresponding SQL data types. Button drop lists, push buttons and radio buttons aren't included in this list. They aren't stored in your application's tables, and therefore don't have a corresponding SQL data type.

Dexterity control type	SQL data type
Boolean (2 bytes)	tinyint (1 byte)
Check box (2 bytes)	tinyint (1 byte)
Combo box (Keyable length + 1 + pad if not even)	char (Length + 2 bytes) (Length < 256 bytes)
Composite (Total of components)	Storage type of each component. Each component is stored as its own column in the table. The column name is the field physical name followed by an underscore (_) and then the number of the component.
Currency (10 bytes)	numeric(19,5) (9 bytes) The currency value can be up to 19 digits long, including five decimal digits.
Currency (variable) (8, 10 or 12 bytes)	numeric() The total number of digits and the number of decimal digits is specified by the data type. The total number of digits can be up to 23, with up to 15 decimal digits.
Date (4 bytes)	datetime (8 bytes) The time portion of the datetime is set to 12:00 AM.
Drop-down list (2 bytes)	smallint (2 bytes)
Horizontal list box (2 bytes)	smallint (2 bytes)
Integer (2 bytes)	smallint (2 bytes)
List box (2 bytes)	smallint (2 bytes)
Long integer (4 bytes)	int (4 bytes)

Dexterity control type	SQL data type
Multi-select list box (4 bytes)	binary(4) (4 bytes)
Non-native list box (2 bytes)	smallint (2 bytes)
Picture (Variable)	image (Variable up to 2,147,483,647 bytes)
Radio group (2 bytes)	smallint (2 bytes)
String (Length + 1 + pad if not even)	char (Dexterity storage size - 1) (Length < 256)
Text (Length + 2 + pad if not even)	char (Variable up to 2,147,483,647 bytes)
Time (4 bytes)	datetime (8 bytes) The date portion of the datetime is set to 1/1/1900.
Visual switch (2 bytes)	smallint (2 bytes)

Chapter 48: Stored Procedures

Stored procedures are collections of SQL statements that are compiled and stored in a database. You can use stored procedures to improve the performance of your application or extend its power by performing tasks that may be beyond the capability of the Dexterity development system.

Stored procedures can receive in parameters, perform server-based processing tasks, and return a status code and out parameters to the application. One stored procedure can call another stored procedure. Stored procedures are often used to:

- Move complex processing tasks from the client to the server.
- Streamline your code, since the same stored procedure can be used by different parts of your application or multiple applications.

Information about stored procedures is divided into the following sections:

- [Using stored procedures](#)
- [Creating a prototype procedure](#)
- [Stored procedure parameters](#)
- [Pathname support for stored procedures](#)
- [Stored procedure time limits](#)

Using stored procedures

Refer to the *SanScript Reference manual* for more information about the **call sproc** statement.

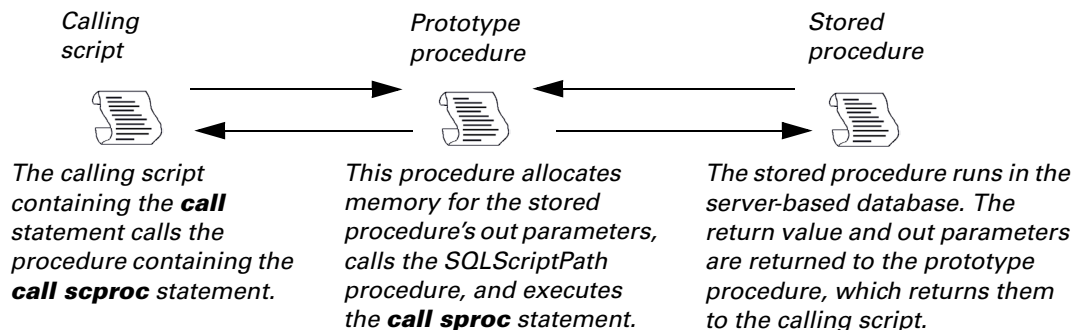
Use the **call sproc** statement to call stored procedures from sanScript. This function doesn't verify the existence of the named stored procedure or the items in the parameter list when it's compiled, so be sure to verify that information when writing your **call sproc** statement.

The application dictionary must contain a prototype procedure for the named stored procedure. This prototype procedure must have the same name as the stored procedure. The first non-comment line of the prototype procedure indicates what the stored procedure will return. It has the following format:

sproc returns long *variable*;

This line is followed by declarations of any additional in, out or inout parameters for the stored procedure. The prototype reserves the necessary space in the client's memory for any out parameters. The **call sproc** statement must be used in the prototype procedure to make the actual call to the SQL stored procedure. When you call the prototype procedure, the `SQLScriptPath` procedure will be called implicitly, which adds path information to the stored procedure named in the **call sproc** statement.

The following illustration shows the relationships between the calling script, the prototype procedure and the stored procedure. The calling script calls the prototype procedure, which contains the **call sproc** statement. This prototype procedure automatically calls the `SQLScriptPath` procedure to get pathname information for the specified stored procedure. It then sends the command to execute the stored procedure to the server-based database where the named stored procedure is located.



Creating a prototype procedure

When creating the prototype procedure for a stored procedure, you should follow these steps.

1. Name the procedure.

The name of the procedure must be exactly the same as the name of the stored procedure. The procedure name is case sensitive, and must be unqualified (not include the name of the database in which the stored procedure resides).

2. Indicate what the stored procedure will return.

The first non-comment line of the prototype procedure must be:

spc returns long *variable*;

where *variable* is a long integer that will contain the return value from the stored procedure.

3. List the remaining parameters.

The remaining parameters in the procedure must be listed in the order in which they are declared in the stored procedure. Their data types must be the Dexterity equivalents of the SQL data types used in the stored procedure. Depending upon the stored procedure's parameter set, you may need to specify several in and inout parameters.



Since you are passing parameters from script to script, you should be declaring in and inout parameters only. Out parameters can't be passed out of a script.

4. Add the remaining statements to the procedure.

Add the **call spc** statement to the script. Be sure that the name of the stored procedure is enclosed in double quotes.

5. Compile and close the procedure.

Once you've completed the procedure, click Compile to save the prototype procedure and check for errors. If no errors occurred, click Close to close the Script Editor.

Refer to [Accessing existing SQL data](#) on page 402 for more information about related SQL and Dexterity data types.

Stored procedure parameters

All stored procedures can receive in parameters and return a return value. The return value is most often used to return a status code from the stored procedure. If the stored procedure was successful, the value of the STATUS_SUCCESS constant, 0 (zero), is returned. If it failed, a value from -1 to -14 (native SQL error codes) or the SQL_SPROC_ERROR constant, -127, is returned.

The return value can be used as an out parameter instead, but it must be defined in Dexterity as a long integer. By using the return value as an out parameter, you give up the ability to verify the success or failure of the stored procedure.

Example 1

This example shows a stored procedure written using Microsoft SQL Server. The script used to call this stored procedure is also shown.

This stored procedure, named Count_Seniors doesn't require in parameters and returns only the return value, @Count. It queries the Employees table to find the number of employees over age 65.

```
CREATE PROC Count_Seniors
AS
    DECLARE @Count int
    /* Count the number of employees over age 65 */
    SELECT @Count = count(first_name) from EMPLOYEES where Age > 65
    RETURN @Count
```

The following Dexterity prototype procedure, also named Count_Seniors, calls the Count_Seniors stored procedure.

```
sproc returns long Ret_Code;

call sproc "Count_Seniors", Ret_Code;
```


The following script calls the `Count_Seniors` Dexterity prototype procedure and sets the value of the `Senior Employees` field. If the stored procedure is successful, `Ret_Code` will be set to the `@Count` value returned by the stored procedure. Otherwise, it will be set to `-127`, the value corresponding to the `SQL_SPROC_ERROR` constant.

```
local long Ret_Code;

call Count_Seniors, Ret_Code;
'Senior Employees' = Ret_Code;
```

Since the success of the stored procedure isn't verified in this example, the `Ret_Code` value will be displayed to a user, even if its value is `-127`. This potential for displaying erroneous data is why return values are rarely used as out parameters.

Returning out parameters

Stored procedures can return out parameters in addition to the return value. However, stored procedures must not generate a results set. Dexterity can handle only return values and parameters as stored procedure outputs.

Example 2

This example shows a stored procedure named `Profile`, which returns out parameters. The script syntax for the prototype procedure and the calling script are also shown.

The stored procedure returns a customer profile. It requires an in parameter and returns five out parameters. The return value isn't manipulated in this procedure; rather, it's used for its default functionality, status checking. If the stored procedure was successful, the value of the `STATUS_SUCCESS` constant, `0` (zero), is returned. If it failed, a value from `-1` to `-14` (native SQL error codes) or the `SQL_SPROC_ERROR` constant, `-127`, is returned.

```

CREATE PROCEDURE Profile(@Cust_ID smallint, @Name char out,
    @City char out, @State char out, @Tot_Purch numeric out,
    @Cust_Award char out)
AS
    SELECT @Name, @City, @State
    FROM Customer_Table
    WHERE Cust_ID = @Cust_ID
    SELECT @Tot_Purch=Q1_Sales + Q2_Sales + Q3_Sales + Q4_Sales
    IF @Tot_Purch >= 100000.00
        @Cust_Award = 'Statue'
    ELSE
    IF @Tot_Purch between 50000.00 and 99999.99
        @Cust_Award = 'Plaque'
    ELSE
        @Cust_Award = 'None'

```

The prototype procedure, also named Profile, is defined as follows. The Sproc_Status parameter is the return value, and must be declared using the **sproc returns** clause. The remaining parameters must be listed in the same order as in the stored procedure.

```

sproc returns long Sproc_Status;
in integer Cust_ID;
inout string Name;
inout string City;
inout string State;
inout currency Tot_Purch;
inout string Cust_Award;

call sproc "Profile", Sproc_Status, Cust_ID, Name, City, State,
➡ Tot_Purch, Cust_Award;

```

The following is a portion of a script using the **call** statement to run the Profile procedure.

```

local long Sproc_Status;

Cust_ID = Cust_ID of window Customer_Info;
call Profile, Sproc_Status, Cust_ID, Name, City, State, Tot_Purch,
➡ Cust_Award;

```

Pathname support for stored procedures

If your application includes any stored procedures, you should include a procedure named `SQLScriptPath` in your application. If any of your stored procedures are stored in a database other than your application's default database, you *must* include this procedure in your dictionary. The `SQLScriptPath` procedure adds pathname information to the stored procedure name.

The `call_sproc` statement passes a stored procedure name to the `SQLScriptPath` procedure, which then adds the appropriate pathname information to the beginning of the stored procedure name. This script ensures that the `call_sproc` statement sends the command to execute the stored procedure to the proper location. If this script isn't included in your application, Dexterity will search only the default database for all called stored procedures.

The `SQLScriptPath` procedure is much like the `SQLPath` procedure. It has similar syntax, and adds pathname information to the beginning of a resource name. However, this procedure is used to add that information to the names of stored procedures.

SQLScriptPath procedure parameters

The `SQLScriptPath` procedure uses four parameters; three in parameters and one out parameter. The out parameter specifies the pathname added to the stored procedure name.

- *product_ID* – An integer in parameter that contains the ID of the dictionary associated with the stored procedure.
- *core_ID* – An integer in parameter that contains the ID of the core associated with the stored procedure's prototype procedure. This is the core selected in the Procedures window and passed in when your application is running in test mode or with the runtime engine.

The cores and their corresponding integer values are listed in the following table:

Cores

1 – Financial	3 – Purchasing	5 – Payroll	7 – System
2 – Sales	4 – Inventory	6 – Project	

- *procedure_ID* – An integer in parameter that contains the resource ID of the stored procedure's prototype procedure.
- *data_path* – A string out parameter that specifies the location of the stored procedure. Use it to return a pathname in the following format, which is added to the beginning of the stored procedure name. Because the server name and owner name portions of the path are optional, they are shown in braces:

```
{:server_name;}database_name{/owner_name}/
```

Example 3

This example shows a `SQLScriptPath` procedure written for use in an application that accesses a data source in which the stored procedures for all databases associated with the data source are located in a single database, `StoredProcs`, created by user `JWILCOX`.

```
in integer product_ID;
in integer core_ID;
in integer procedure_ID;
out string data_path;

data_path = "StoredProcs/JWILCOX/";
```

If the owner name weren't included in this script, Dexterity would have used the current SQL user ID as the default owner name.

Stored procedure time limits

You can use the `SQLProcsTimeout` defaults file setting to limit the amount of time a stored procedure will be allowed to run before control is returned to the application. By default, Dexterity will wait 300 seconds (5 minutes) for a stored procedure to run to completion.

If you want Dexterity to wait indefinitely for a stored procedure to run, you must include this setting in the defaults file and set its value to 0.

While most stored procedures should run to completion within the 5 minutes allotted by default, you will need to include this setting in your defaults file and set it to a higher number of seconds if your stored procedures are consistently timing out.

Chapter 49: Pass-through SQL

Refer to the [SQL function library](#) in the Function Library Reference manual for information about functions used for pass-through SQL.

Dexterity is designed so that you don't need to know SQL when creating your application. Dexterity produces all the SQL statements required, based on your sanScript code. However, there may be instances when you want to use SQL to perform operations not possible with sanScript. With *pass-through SQL*, you can execute SQL statements directly from within sanScript code. Information about pass-through SQL is divided into the following sections:

- [Pass-through SQL connections](#)
- [Executing SQL statements](#)
- [Working with results sets](#)
- [Specifying a database](#)

Pass-through SQL connections

To use pass-through SQL from within sanScript, you must create a pass-through SQL connection. Before you can do this, your application must be logged in to a SQL data source. Then you will use the [SQL Connect\(\)](#) function to create the pass-through SQL connection. The login information for the current SQL login will be used for the new pass-through SQL connection.

A pass-through SQL connection is used only for executing SQL statements from sanScript. It is completely separate from the other connections used by Dexterity to access SQL data.

A pass-through SQL connection remains active until you use the [SQL Terminate\(\)](#) function to terminate it. Creating a pass-through SQL connection can take a significant amount of time, so to optimize performance, you may want to create a connection, leave it open until you have finished executing SQL statements, then terminate it.

Executing SQL statements

Refer to the example for the [SQL_Execute\(\)](#) function to see executing SQL statements from sanScript.

The SQL statements to execute must be placed in a text field in a window. This will typically be a hidden field, unless you want the user to be able to see or edit the statements. Use the [SQL_Execute\(\)](#) function to execute the SQL statements. After the statements are executed, use the [SQL_GetError\(\)](#) function to find out whether any errors occurred.

Working with results sets

If the SQL statements you execute generate one or more results sets, you will likely want to retrieve data from them. Several statements in the SQL function library allow you to do this.

Accessing a results set

If one or more results sets are generated, the first one will be active by default, allowing you to retrieve data from it. If there are multiple results sets, use the [SQL_GetNextResults\(\)](#) function to make the next results set active. Once you have moved to the next results set, you can't return to the previous results set. As an example, the following two SQL statements generate the two results sets shown below:

SQL statements

```
select Buyer_ID, Buyer_First_Name, Buyer_Last_Name from BUYERDAT where
City = 'Fargo'

select House_ID, Address from HOUSEDAT where City = 'Fargo'
```

Results sets

By default, the first results set is active.

Buyer_ID	Buyer_First_Name	Buyer_Last_Name
100	April	Berger
101	Candice	Peterson
102	Stan and Dawn	Maddock
105	Heide	Bertsch
108	Kyle	Anderson

Use the [SQL_GetNextResults\(\)](#) function to make the next results set active.

House_ID	Address
1000	810 Santa Cruz Drive
1006	4750 9th Avenue South
1007	4756 9th Avenue South

Retrieving data in a results set

You can retrieve data from the currently-active results set one row at a time. Use the [SQL FetchNext\(\)](#) function to access the first row in the current results set. The [SQL GetNumCols\(\)](#) function returns the number of columns in the results set. Use the [SQL GetData\(\)](#) function to retrieve a specific column from the current row. The [SQL DescribeColumn\(\)](#) function can be used to retrieve information about the contents of the column.

When you have finished retrieving data for the current row, use the [SQL FetchNext\(\)](#) function to move to the next row. Once you have moved to the next row, you can't return to the previous row.

There is no way to know how many rows a results set contains. To read all rows in a results set, use the [SQL FetchNext\(\)](#) function to retrieve rows until it returns an error 31, indicating that there are no more rows in the results set.

Clearing results sets

Once you have accessed all of the data resulting from the SQL statements that were executed, use the [SQL Clear\(\)](#) function. This function clears any results sets and error information, preparing the pass-through SQL connection to be used again.

Specifying a database

When you use pass-through SQL, by default you will be working in the default database for the current user. To switch to another database, execute the USE statement within your pass-through SQL code. You must use the [SQL Execute\(\)](#) function to execute the USE statement separately to switch to another database. Then you can execute additional SQL statements. Refer to the example for the [SQL Execute\(\)](#) function to see how to switch to a specific database.

Part 9: Source Code Control

This part includes information you will need if you plan to use a source code control (SCC) system to manage the development of your Dexterity-based application. Dexterity supports Microsoft Team Foundation Server 2010 and Microsoft Visual SourceSafe® version 6. If you want to use another source code control provider to manage your Dexterity development, the Generic provider is also included.

Following is a list of the topics that are discussed:

- [Chapter 50, "Introduction to Source Code Control,"](#) provides an introduction to using a source code control system with Dexterity.
- [Chapter 51, "Setting up Source Code Control,"](#) describes the steps required to install the source code control provider, and how to configure Dexterity to use source code control.
- [Chapter 52, "Source Files,"](#) describes source files, which are text files that represent the resources that make up a dictionary. Source files are stored by the source code control system.
- [Chapter 53, "Using Source Code Control,"](#) describes the common tasks performed when using a source code control system to manage development of a Dexterity application.
- [Chapter 54, "Maintaining the Repository,"](#) provides information about maintenance tasks you may need to perform on the source code control repository.
- [Chapter 55, "Source Code Control for Integrating Applications,"](#) explains how to use source code control if you are creating a Dexterity application that integrates with Microsoft Dynamics GP.
- [Chapter 56, "Generic Provider,"](#) describes the generic provider, which allows you to use other source code control providers with Dexterity.

Chapter 50: Introduction to Source Code Control

Dexterity has integrated support for using a source code control (SCC) system to manage application development. This chapter provides an introduction to using source code control with Dexterity. Information is divided into the following sections:

- [*Benefits of Source Code Control*](#)
- [*Source code control terms*](#)
- [*Configurations*](#)
- [*Providers*](#)

Benefits of Source Code Control

Using a source code control system to manage application development provides many benefits. These include:

- A central repository for all application resources. All resources for the application can be found in a central place. This provides enhanced security and makes creating backups easier.
- Concurrency control when multiple developers are developing a product. A source code control system can prevent users from trying to change the same resource at the same time.
- Revision management. The source code control system can store all of the revisions of a product, allowing you to easily re-create any version at any time.

Source code control terms

Several terms specific to source code control are used throughout this documentation. You may want to familiarize yourself with them before continuing.

Dexterity Source Code Control Server Software that allows Dexterity clients to communicate with the source code control provider. Depending on which source code control provider you are using, this software runs on the same computer as the source code control provider, or on the individual workstations.

Provider The software that performs the source code management tasks. For example, Microsoft Team Foundation Server is a source code control provider for which Dexterity has integrated support.

Repository The location where the source code is stored and maintained by the source code control system.

Revision management The process of tracking the versions of an application and the resources it is composed of.

Source files The physical files containing source code or other resources that are managed by the source code control system. Source files are stored in the repository.

Configurations

Two typical configurations are used when implementing source code control with Dexterity: single-developer and multiple-developer.

Single-developer

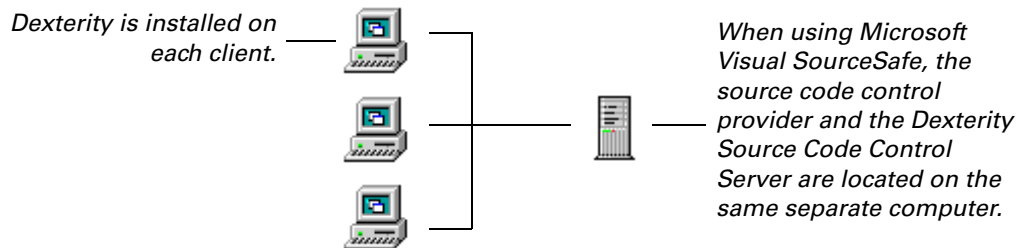
In this configuration, Dexterity, the Dexterity Source Code Control Server, and the source code control provider all reside on the same computer.



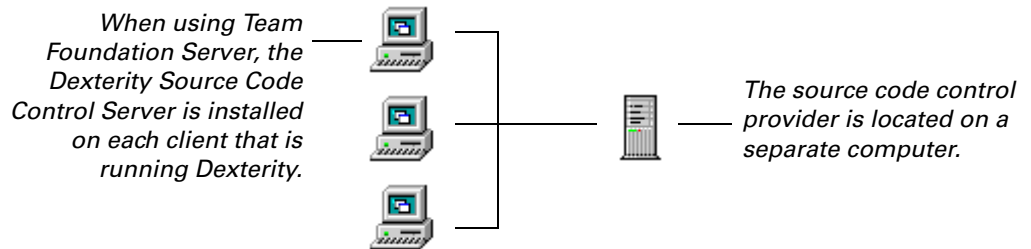
In the single-developer configuration, Dexterity, the Dexterity Source Code Control Server, and the source code control provider are located on the same computer.

Multiple-developer

In this configuration, each client computer has Dexterity installed. The source code control provider is located on a separate computer. The Dexterity clients use TCP/IP protocol to communicate with the computer that is running the source code control provider. Where the Dexterity Source Code Control Server is installed depends on the source code control provider you are using. When using Microsoft Visual SourceSafe, the Dexterity Source Code Control Server is installed on the same machine as Visual SourceSafe.



When using Microsoft Team Foundation Server, the Dexterity Source Code Control Server is installed on each client workstation.



Providers

The source code control provider is the actual software that manages the source code. Dexterity currently supports the following source code control providers:

- Microsoft Team Foundation Server 2010
- Microsoft Visual SourceSafe 6.0
- Generic Provider

When a provider is directly supported, such as Team Foundation Server, most source code control operations can be performed directly from within the Dexterity development environment. If a provider isn't directly supported, you can use the Generic Provider included with Dexterity. This is described in [Chapter 56, "Generic Provider."](#)



The information necessary for Dexterity to interact with a source code control provider is contained in a single DLL. To support a new provider, a provider DLL must be developed.

Chapter 51: Setting up Source Code Control

Before you can use source code control with Dexterity, you must install and configure several components. Information about setting up source code control is divided into the following sections:

- [Connectivity](#)
- [Installing components](#)
- [Configuring the repository](#)
- [Configuring the Source Code Control Server](#)
- [Configuring Dexterity](#)

Connectivity

Dexterity clients use TCP/IP protocol to communicate with the Dexterity Source Code Control Server, which is running on the machine that contains the source code control repository. The repository machine and each client machine must have valid IP addresses.

If you will be using source code control on a single machine, you must still have a valid IP address set up for your computer. Dexterity still uses TCP/IP protocol to communicate with the source code control repository, even when both are located on the same machine.

Installing components

It's important that you install the components of source code control in the proper order for the system to function properly.

Source code control provider

Complete the steps necessary to install Microsoft Team Foundation Server or any other source code control provider you will be using. Consult the documentation that comes with the source code control software for more information about installing it.

Dexterity Source Code Control Server

The Dexterity Source Code Control Server (DSCCS) is the software that manages communication between Dexterity clients and the source code control system. Where the DSCCS is installed depends on the source code control provider and configuration you are using.

Single developer In the single developer configuration, the DSCCS, source code provider, and Dexterity are all installed on the same machine.

Visual SourceSafe When Visual SourceSafe is being used, the DSCCS is installed onto the same machine that the Visual SourceSafe provider is installed on.

Team Foundation Server When Team Foundation Server is being used, the DSCCS is installed on each Dexterity client machine that will be accessing the repository.

Use the following procedure to install the Dexterity Source Code Control Server.

- 1. Locate the installer for the Dexterity Source Code Control Server (DSCCS).**

Find the installer (.msi) file, located in the DSCCS directory included with Dexterity. This file is the installer for the Dexterity Source Code Control Server. Double-click this file to begin the installation.

- 2. Review the license agreement.**

Indicate whether you accept the terms of the license agreement, and then click Next to continue.

- 3. Specify the location where you want the Dexterity Source Code Control Server installed.**

If you don't want the DSCCS installed in the default location, click Browse to specify a different location. Click Next to continue.

- 4. Specify the account to be used to run the service.**

You can choose to use the Local System Account, or you can choose to supply the credentials for another account.

Visual SourceSafe If you are using Visual SourceSafe as your source code control provider, you will typically choose the Local System Account.

Team Foundation Server If you are using Team Foundation Server as your source code control provider, you must use the credentials of the of the person that will be accessing the repository from the workstation on which you are installing DSCCS. The credentials you supply are used by the Team Foundation Server to identify each individual user of the source code repository. Each workstation on which Dexterity is installed should have a unique set of credentials for the DSCCS.

Click Install to install the service.

- 5. Specify the source code control provider you will be using.** The Dexterity Source Code Control Server control panel will be displayed. Choose one of the following providers:

Generic If you choose the Generic provider, you will need to specify a Root Directory. Refer to [Chapter 56, “Generic Provider,”](#) for more information.

Microsoft Team Foundation If you choose Microsoft Team Foundation, you will need to supply the Root Directory for the repository that you are connecting to. This must be the same folder location on the workstation that you specified when you used Team Explorer or Visual Studio to configure the local path for the Team Project you selected.

Microsoft Visual SourceSafe If you chose Visual SourceSafe, you will need to specify the location of the SRCSAFE.INI file.

- 6. Click Finish to complete the setup.**

The service for Dexterity Source Code Control Server will be started. If the service doesn't start, check the system event log for more information about the error that prevented the service from starting.

Dexterity

If you haven't already done so, install Dexterity on each of the machines that will be accessing the source code control system.

Configuring the repository

Refer to the documentation included with the source code control provider you are using for information about performing these tasks.

Before you can use source code control with Dexterity, you must configure the repository. The tasks required depend on the source code control provider you are using.

Team Foundation Server

The following basic tasks are required when you use Team Foundation Server as the source code control provider.

Create a team project In the Team Explorer, you will create a team project for your collection. Be sure to map the team project to a local path.

Create the folder structure Typically, you will create a folder for each dictionary that you want to include in the repository. Be sure you check in your changes so that the folders are committed to the repository.

Set access rights Team Foundation Server allows you to control what actions users can perform. You may want to set access rights for the individual users after you create your team project.

Visual SourceSafe

As an example, you must complete the following tasks when using Microsoft Visual SourceSafe as the source code control provider:

Add users In this process, you define the users that will be able to access the source code control system. Typically, this involves assigning a user ID and password for each user.

Create a project The project is a specific area in the source code control repository that will store the resources for your application. For consistency, the project should have a name similar to the name of the dictionary you will be storing in the repository.

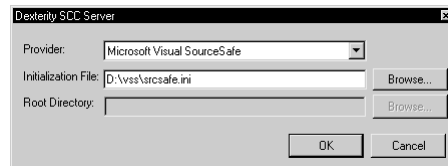
Set access rights Microsoft Visual SourceSafe allows you to control what actions users can perform with a specific project. You may want to set access rights for individual users after you create the project.

Configuring the Source Code Control Server

The Dexterity Source Code Control Server (DSCCS) is a Windows service that controls communication between the Dexterity clients and the source code control provider.

Control panel

Typically, all of the configuration for the DSCCS is performed when it is installed. If you need to change the configuration of this service, you can do so using the Dexterity Source Code Control Server control panel.



You can specify the following items using this control panel:

Provider Set this drop-down list to indicate the source code control provider you are using.

Initialization File If you are using Microsoft Visual SourceSafe, you must specify the initialization file (.INI file) used for the provider. For example, the initialization file for Microsoft Visual SourceSafe is SRCSAFE.INI. Specifying the location of the initialization file tells the Dexterity Source Code Control Server where the source code control provider is located.

Root Directory If you have selected Microsoft Team Foundation or the Generic provider, this field will be enabled. For the Microsoft Team Foundation provider, it specifies the folder on the local workstation that you specified when you used Team Explorer or Visual Studio to configure the local path for the Team Project you selected. For the generic provider, it indicates the directory where source files will be placed. Refer to [Chapter 56, “Generic Provider,”](#) for more information about the directory structure for the Generic provider.



After you make changes using the Dexterity Source Code Control Server control panel, you should restart the Microsoft Dexterity SCCS service so that the changes will take effect.

TCP/IP port

By default, the DSCCS service uses TCP/IP port 2725. Any other service or application using this port will cause a conflict. To use a different port for the DSCCS service, complete the following procedure:

- 1. Start the Registry Editor.**

On the system running the DSCCS service, start the Registry Editor.

- 2. Open the location for the DSCCS service.**

Open this location in the registry: `HKEY_LOCAL_MACHINE\SOFTWARE\Great Plains Software\Dexterity Source Code Control Server\DSCCSProvider`

- 3. Create a new DWORD value.**

In the location specified, choose to create a new DWORD value named "Port" (no quotation marks).

- 4. Specify the port number to use.**

The value for this new DWORD specifies the port number that the DSCCS service will use. Use the decimal format when specifying the port number.

- 5. Close the Registry Editor.**

After you have added the DWORD value, close the Registry Editor.

- 6. Restart the system.**

Restart the system running the DSCCS service so the new port number will be used.



If you change the port number, any Dexterity client machines that access the source code control server will need to have their connection information updated to use the new port.

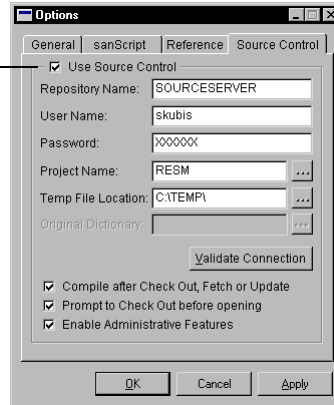
Firewall settings

If a firewall is being used on the system running the DSCCS service, you will have to open the port used by the DSCCS service so that the Dexterity client systems can connect.

Configuring Dexterity

Before you can use source code control with Dexterity, you must enable and configure it. To do this, choose Options from the Edit menu and display the Source Control tab.

Mark this option to use source code control.



Specifying options

Mark the Use Source Control option to enable source code control, then specify the following settings:

Repository Name This is the hostname or IP address of the machine that contains the repository and is running the Dexterity Source Code Control Server. If you are running Dexterity and the source code control system on a single machine, you can use the IP address 127.0.0.1 or the hostname **localhost** to indicate the current machine.

If you have changed the port used for the DSCCS server, you must specify the new port number at the end of the Repository Name string. Append a colon and the number of the port to use. For example, if you are using port 2726, the Repository Name string would look like this:

192.168.0.100:2726

User Name Enter the user name you were assigned when the source code control provider was set up. This is used only for Microsoft Visual SourceSafe.

Password Enter the password you were assigned when the source code control provider was set up. This is used only for Microsoft Visual SourceSafe.

Project Name Enter the name of the project you want to use in the repository. You can click the lookup button to view a list of projects available in the repository.

Temp File Location Many temporary files are created as resources are checked into and out of the repository. Specify the location where you want these temporary files placed. The location you choose should have at least twice as much disk space available as the size of the dictionary you will be storing in the repository.

Original Dictionary This setting is used only if you are creating an application that integrates with Microsoft Dynamics GP. For resources to be properly checked into the repository, you must specify the location of an unmodified original dictionary for the version of Dynamics.dic dictionary for which you are developing.

Options Several options can be specified when setting up source code control. Typically, you will mark the options to compile after source code control operations and to be prompted when opening items that are not checked out. If you will be performing builds of the dictionary, you should also mark the option to enable administrative features.

Validating the connection

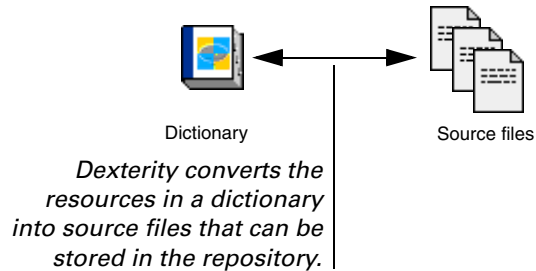
After you have specified the options for source code control, click Validate Connection to verify that a connection could be made to the repository. A dialog should be displayed indicating the connection was made. If the connection couldn't be validated, check the following:

- Be sure that both the Dexterity client and source code control repository have TCP/IP connectivity enabled. Verify that they have valid IP addresses and hostnames.
- Be sure the Dexterity Source Code Control Server is properly configured and that the service is running on the machine containing the source code control repository.
- Verify that you don't have a TCP/IP port conflict.

- If you have changed the TCP/IP port number used for the Dexterity Source Code Control Server, be sure you have specified that same port number in the Repository Name setting on each client workstation.
- Be sure that any firewall running on the Dexterity Source Code Control Server has opened the TCP/IP port used by the DSCCS service.
- Verify that the user has been set up in the source code control system.
- Verify that the user name and password are correct if you are using Microsoft Visual SourceSafe.
- Be sure the project specified has been created in the source code control repository.
- Verify that the source code control provider software is installed correctly.

Chapter 52: Source Files

Resources from a dictionary must be converted into textual form before they can be stored in the source code control repository. The textual representation of a resource is referred to as a *source file*. Dexterity can export all resources to source files. Dexterity can also read source files and convert them back into resources in the dictionary.



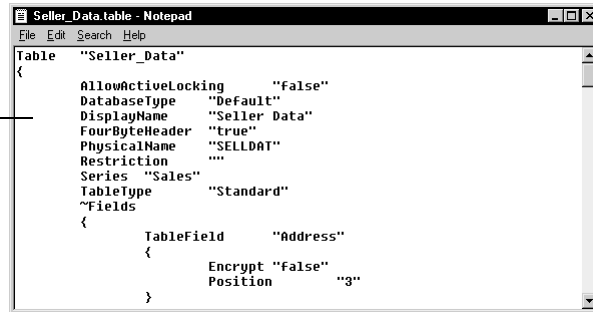
Information about source files is divided into the following sections:

- [Source file contents](#)
- [Source file structure](#)
- [Source file names](#)

Source file contents

Because they are text files, you can view source files with any standard text editor such as Windows Notepad. The contents of a source file is designed to be human-readable. For example, the following illustration shows a portion of the source file that represents the Seller Data table.

The text in a source file is designed to be human-readable.



```

Table "Seller_Data"
{
    AllowActiveLocking "false"
    DatabaseType "Default"
    DisplayName "Seller Data"
    FourByteHeader "true"
    PhysicalName "SELLDAT"
    Restriction ""
    Series "Sales"
    TableType "Standard"
    Fields
    {
        TableField "Address"
        {
            Encrypt "false"
            Position "3"
        }
    }
}
  
```

When you examine a source file, you will see the terminology used closely follows that found in the Dexterity user interface. This makes source files easy to understand, without requiring additional documentation.



You won't need to view or edit source files as you work with Dexterity. Advanced users may find situations where it is useful to change a resource by editing its source file.

Source file structure

The contents of a source file depends on the type of resource the source file contains. Some resources have their own source file. Other resources are stored as a group within a single source file. The following is a list of the resources that are each stored in a separate source file:

- Forms
- Reports
- Tables
- Global Procedures
- Global Functions

This means that there is one source file for every form, report, table, global procedure and global function in an application dictionary.

The following is a list of resources that are stored within a single source file:

- Constants
- Composites
- Data types
- Fields
- Formats
- Global variables
- Libraries
- Pictures
- Native pictures
- Messages
- Table groups
- Product information
- Install information

This means there is one source file that stores all constants, another source file that stores all composites, another that stores all data types, and so on. Resources that are stored this way are called *base resources*.

Source file names

Dexterity automatically assigns names to source files when they are created. The default name is based on the resource name, with an extension that indicates the type of resource contained in the source file. The following table lists the extension used for each type of source file.

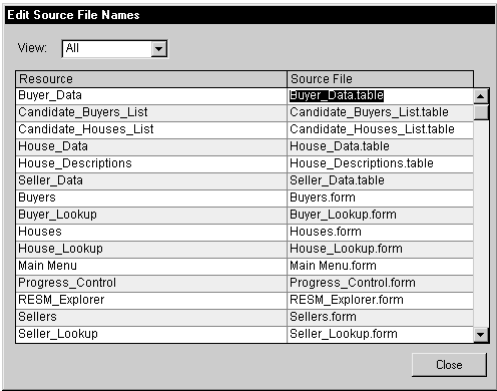
Resource type	Extension
Forms	.form
Reports	.report
Tables	.table
Global procedures	.procedure
Global functions	.function
Constants	.constant
Composites	.composite
Data types	.datatype
Fields	.field
Formats	.format
Global variables	.global
Libraries	.library
Pictures	.picture

Resource type	Extension
Native pictures	.nativepicture
Messages	.message
Table groups	.tablegroup
Product information	.product
Install information	.install



The extension used for the source file name is significant and should not be changed. Dexterity uses the extensions to determine where the source file will be located in the source code control repository.

At some point in the development of your application, you may need to change the name of a source file. For example, if you change the name of a resource, you may want to change the name of its source file. You can do this using the Edit Source File Names window, shown in the following illustration.



To open this window, point to Source Control in the Explorer menu and then choose Source Files. Refer to [Chapter 54, “Maintaining the Repository,”](#) for more information about situations where you may need to change source file names.

Chapter 53: Using Source Code Control

When you enable source code control for Dexterity, several buttons in the Resource Explorer and the Source Control items in the Explorer menu are enabled. Most actions relating to source code control are performed from these two locations. The following sections describe all of the basic operations you can perform when using source code control:

- [*Checking in a new dictionary*](#)
- [*Examining the repository*](#)
- [*Checking out source files*](#)
- [*Making changes to resources*](#)
- [*Creating new resources*](#)
- [*Checking in source files*](#)
- [*Locking and unlocking source files*](#)
- [*Fetching source files*](#)
- [*Updating a dictionary*](#)
- [*Creating and updating the index file*](#)
- [*Building a dictionary*](#)
- [*Viewing version information*](#)
- [*Source code control errors*](#)

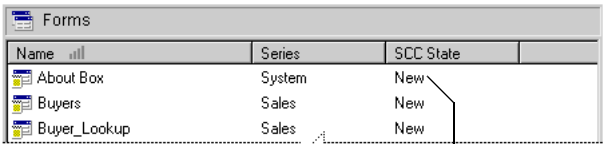
Most source code control operations can be performed by selecting resources in the “dictionary” view of the Resource Explorer. They can also be performed by selecting source files in the “repository” view. In this documentation, the repository view is considered the preferred method. The procedures included describe how to perform actions using the repository view.

Checking in a new dictionary

The first action you must perform when using source code control is to check your dictionary into the repository. You will perform this process only one time for each dictionary you are placing into the repository. During this process, source files are created for all of the resources in the dictionary, then checked into the repository. To check in a new dictionary, complete the following procedure:

1. Open the dictionary to check in.

In Dexterity, open the dictionary you want to check in. Notice in the Resource Explorer that all resources have New as the source code control (SCC) state.

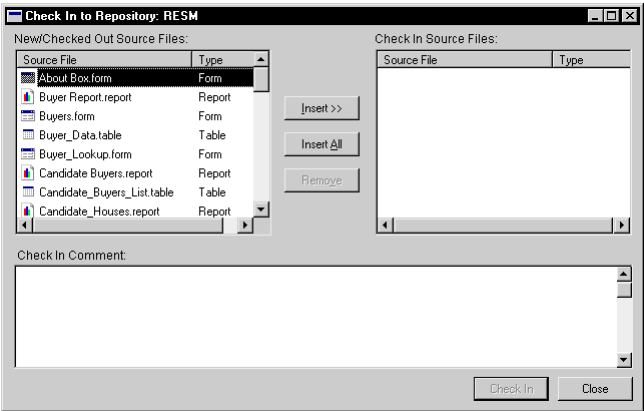


Name	Series	SCC State
About Box	System	New
Buyers	Sales	New
Buyer_Lookup	Sales	New

All resources will be in the New state.

2. Open the Check In to Repository window.

To open the Check In to Repository window, point to Source Control in the Explorer menu and choose Check In. This window is shown in the following illustration.



3. Select the source files to check in.

Because you're checking in the dictionary for the first time, click Insert All to indicate you want to check in all source files for the dictionary.

4. Add a Check In Comment.

Each time you check source files into the repository, you are required to supply a Check In Comment. This comment should supply information about why you are checking specific source files into the repository. In this case, the comment should indicate this is the initial check in for this dictionary.

5. Check in the source files.

Click Check In to begin checking source files into the repository. A dialog box will be displayed, indicating progress as source files are checked into the repository. When all source files are checked in, click Close to close the Check In To Repository window.

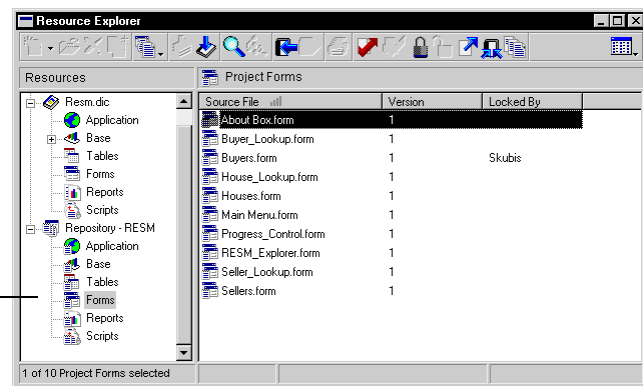
Examining the repository

Once you have checked source files into the repository, you can examine them using the Resource Explorer. You could also use the software included with the source code control provider.

Using the Resource Explorer

When you enable source code control in Dexterity, the Repository item is added to the tree view on the left side of the Resource Explorer.

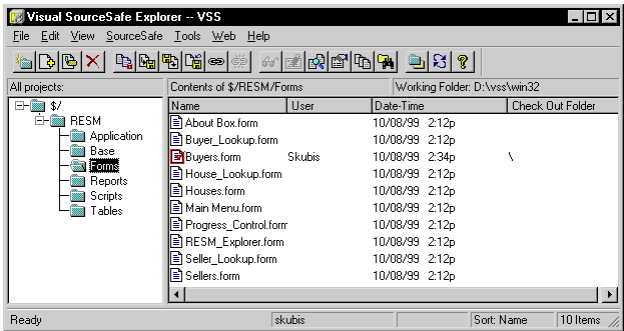
Selecting an item in the Repository tree allows you to view the contents of the source code control repository.



When you select an item in the Repository tree, information about the selected category is retrieved from the repository and displayed in the Resource Explorer. You can see the current version of each source file, as well as who may have the source file locked.

Using the source code control provider

You can also use software included with your source code control provider to view the contents of the repository. For example, the following illustration shows the Visual SourceSafe Explorer being used to view the contents of the RESM project in the source code control repository.



Typically, you will only need to work with the repository from within Dexterity. However, using a tool such as the Visual SourceSafe Explorer allows you to see how the source files are actually stored in the repository. Notice that the hierarchy created in the repository closes matches how source files are organized by Dexterity.

Checking out source files

Once you have checked a dictionary into the repository, you must check out a source file in order to make changes to any of the resources it contains. When you check out a source file, you lock it in the repository, preventing other developers from making changes to that same resource. You also copy the latest version of the source file from the repository, updating the resources in your dictionary.

Checking out one source file

You can use the Resource Explorer to check out one source file at a time. To check out one source file, complete the following procedure.

1. Select the source file you want to check out.

In the Resource Explorer, select the category of source file you want to check out in the Repository tree. The right side of the Resource Explorer will be filled with a list of source files of the selected type. Select the source file you want to check out.

If you're checking out one of the "base" resources, such as fields or data types, remember that you're checking out all of the resources of that type. If you created some new base resources that aren't in the source file in the repository, the Modified Resources window will appear, asking you which of the new resources you created you want to keep. By default, all new base resources you created will be kept. Mark the appropriate resources and click OK.

2. Click the Check Out button in the Resource Explorer.

To perform the check out operation, click the Check Out button in the Resource Explorer.



Click Check Out to check out the selected source file.

If the check out operation was successful, the source file will be copied into your dictionary and locked in the repository. The source code control state for the resource will change to show that it is checked out.



Sellers

Sales

Checked Out

This icon shows the resource is checked out.



You can also check out a source file by selecting its corresponding resource in the Resource Explorer and clicking Check Out. Dexterity will automatically check out the appropriate source file for the resource you have selected.

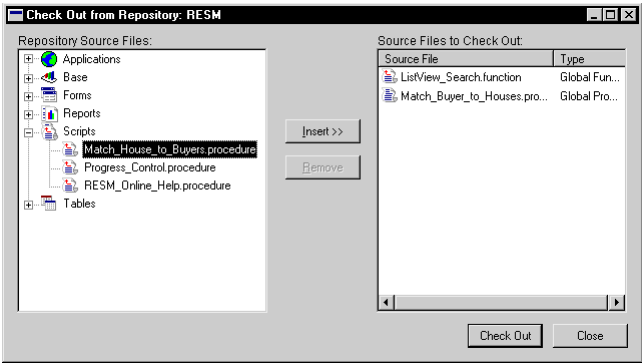
If any errors occurred during the check out process, refer to [Source code control errors](#) on page 460 for information about resolving them.

Checking out multiple source files

As you work with a dictionary, you often need to make modifications to several resources at one time. Rather than check out each resource individually, it is more efficient to check out multiple source files at one time. To do this, perform the following procedure:

1. Open the Check Out from Repository window.

To open the Check Out from Repository window, point to Source Control in the Explorer menu and choose Check Out. The window is shown in the following illustration.



2. Select the source files to check out.

In the Repository Source Files list, select the source files you want to check out and click Insert.

3. Check out the source files.

Click Check Out to begin checking source files out of the repository. A dialog box will be displayed, indicating progress as source files are checked out of the repository.

When all source files are checked out, click Close to close the Check Out from Repository window.

Making changes to resources

Once you have a resource checked out, you can make changes to it. If you try to make changes to resources that you don't have checked out, a warning is displayed asking whether you want to check out the resource. This prevents you from accidentally changing resources that you don't have checked out. You can disable this warning by unmarking the **Prompt to Check Out before opening** option in the Source Control tab in the Options window for Dexterity.



It's a good idea to keep track of which resources you have checked out and modified. That way, you can check in only those resources that you made modifications to.

Creating new resources

When you create new resources for a dictionary, you must ensure that the new resources are also added to the repository.

Standard resources

Standard resources are those that each have their own source file in the repository. When you create a new standard resource, such as a table or a form, it will have the source code control state New. To add it to the repository, simply check it into the repository.

Base resources

With base resources, all resources of a specific type are stored in a single source file. When you want to create a new base resource, such as a data type or field, you must first check out the source file that contains all of the resources of that type. If you try to create a new base resource, but don't have the appropriate source file checked out, a warning will be displayed that prompts you to check out the necessary source file.

Once you have checked out the source file, you can create new base resources. When you have finished, check the source file back into the repository.

Checking in source files

Once you have finished making changes to resources, you must check them into the repository. When you check in a source file, a new revision of it is added to the repository. The lock on the source file is also removed, allowing others to check out the source file.

Checking in one source file

You can use the Resource Explorer to check in one source file at a time. To check in one source file, complete the following procedure.

1. Select the source file you want to check in.

In the Repository tree of the Resource Explorer, select the category of source file you want to check in. The right side of the Resource Explorer will be filled with a list of source files of the selected type. Select the source file you want to check in.



If you're checking in one of the "base" resources, such as fields or data types, remember that you're checking in all of the resources of that type.

2. Click the Check In button in the Resource Explorer.

To perform the check in operation, click the Check In button in the Resource Explorer.



Click Check In to check in the selected source file.



You can also check in a source file by selecting its corresponding resource in the Resource Explorer and clicking Check In. Dexterity will automatically check in the appropriate source file for the resource you have selected.

3. Supply a check in comment.

In the Check In to Repository window, supply a check in comment. This comment typically describes what source files were modified and for what reason. Click Check In to continue.

If the check in operation was successful, a new revision of the source file will be added to the repository.

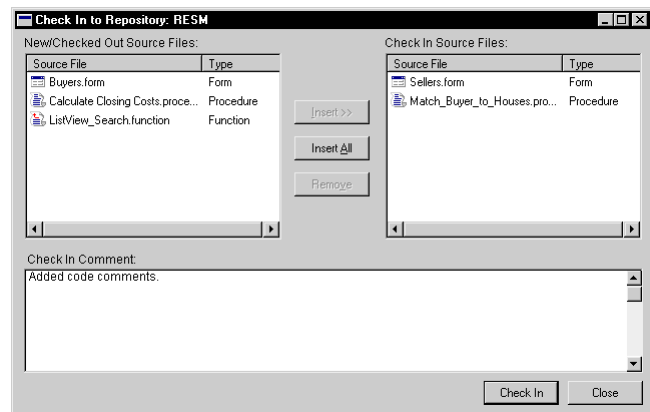
If any errors occurred during the check in process, refer to [Source code control errors](#) on page 460 for information about resolving them.

Checking in multiple source files

When you make changes to a dictionary, you often will make changes to multiple resources. Rather than checking in each resource individually, it is more efficient to check in multiple source files at one time. To do this, perform the following procedure:

1. Open the Check In to Repository window.

To open the Check In to Repository window, point to Source Control in the Explorer menu and choose Check In. The window is shown in the following illustration.



2. Select the source files to check in.

In the New/Checked Out Source Files list, select the source files you want to check in and click Insert.

3. Supply a check in comment.

This comment typically describes what source files were modified and for what reason.

4. Check in the source files.

Click Check In to begin checking source files into the repository. A dialog box will be displayed, indicating progress as source files are checked in.

When all source files are checked in, click Close to close the Check In to Repository window.

Locking and unlocking source files

Locking a source file in the repository prevents other users from making modifications to that file. Source files are automatically locked and unlocked when you check out and check in source files. However, in some cases it may be necessary to manually lock or unlock source files.



Use caution when you manually lock and unlock source files. If several developers are checking sources files into and out of the repository, it's easy to overwrite or lose the work a developer has done.

Locking source files

If you have made changes to a resource in your dictionary, but don't have the corresponding source file checked out, you can lock the source file in the repository. This allows you to check in your changes without having to perform a check out operation.

You can use the Resource Explorer to lock one source file at a time. To lock one source file, complete the following procedure.

1. Select the source file you want to lock.

In the Resource Explorer, select the category of source file you want to lock in the Repository tree. The right side of the Resource Explorer will be filled with a list of source files of the selected type. Select the source file you want to lock.



If you're locking one of the "base" resources, such as fields or data types, remember that you're locking all of the resources of that type.

2. Click the Lock button in the Resource Explorer.

To perform the lock operation, click the Lock button in the Resource Explorer.



Click Lock to lock the selected source file.

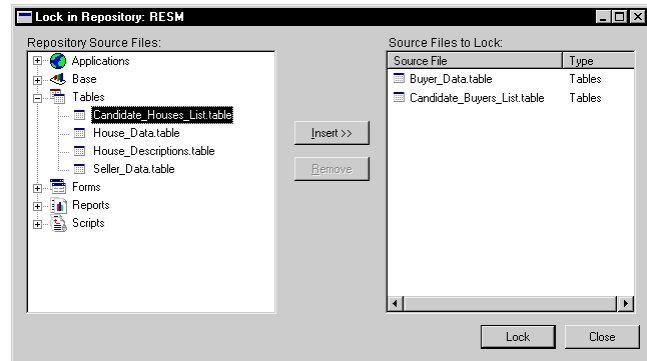


You can also lock a source file by selecting its corresponding resource in the Resource Explorer and clicking Lock. Dexterity will automatically lock the appropriate source file for the resource you have selected.

You can also lock multiple source files at one time. To do this, complete the following procedure.

1. Open the Lock in Repository window.

To open the Lock in Repository window, point to Source Control in the Explorer menu and choose Lock. The window is shown in the following illustration.



2. Select the source files to lock.

In the Repository Source Files list, select the source files you want to lock and click Insert.

3. Lock the source files.

Click Lock to begin locking source files in the repository. A dialog box will be displayed, indicating progress as source files are locked. When all source files are locked, click Close to close the Lock in Repository window.

Unlocking source files

If you have checked out a source file, but don't want to check it back into the repository, you can unlock the source file.

You can use the Resource Explorer to unlock one source file at a time. To unlock one source file, complete the following procedure.

1. Select the source file you want to lock.

In the Resource Explorer, select the category of source file you want to unlock in the Repository tree. The right side of the Resource Explorer will be filled with a list of source files of the selected type. Select the source file you want to unlock.



If you're unlocking one of the "base" resources, such as fields or data types, remember that you're unlocking all of the resources of that type.

2. Click the Unlock button in the Resource Explorer.

To perform the unlock operation, click the Unlock button in the Resource Explorer.



Click Unlock to unlock the selected source file.

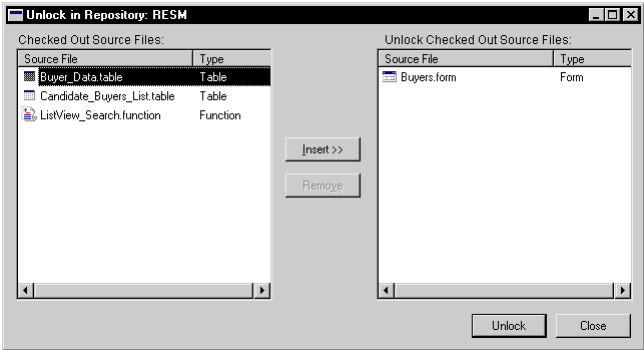


You can also unlock a source file by selecting its corresponding resource in the Resource Explorer and clicking Unlock. Dexterity will automatically unlock the appropriate source file for the resource you have selected.

You can also unlock multiple source files at one time. To do this, complete the following procedure.

1. Open the Unlock in Repository window.

To open the Unlock in Repository window, point to Source Control in the Explorer menu and choose Unlock. The window is shown in the following illustration.



2. Select the source files to unlock.

In the Checked Out Source Files list, select the source files you want to unlock and click Insert.

3. Unlock the source files.

Click Unlock to begin unlocking source files in the repository. A dialog box will be displayed, indicating progress as source files are unlocked. When all source files are unlocked, click Close to close the Unlock in Repository window.

Fetching source files

You can use the Versions window to fetch a specific version of a source file. Refer to [Viewing version information](#) on page 459 for more information.

In a fetch operation, the latest revision of a source file is retrieved from the repository and placed into your dictionary. Unlike a check out operation, the source file isn't locked.

Typically, you will fetch a source file when you know a newer version of the source file exists in the repository and you want to update your dictionary. Another common reason to fetch a source file occurs when you made changes to a resource, but want to discard them. Fetching the corresponding source file will overwrite the changes you made in your dictionary.

Fetching one source file

You can use the Resource Explorer to fetch one source file at a time. To fetch one source file, complete the following procedure.

1. Select the source file you want to fetch.

In the Resource Explorer, select the category of source file you want to fetch in the Repository tree. The right side of the Resource Explorer will be filled with a list of source files of the selected type. Select the source file you want to fetch.

If you're fetching one of the "base" resources, such as fields or data types, remember that you're fetching all of the resources of that type. If you created some new base resources that aren't in the source file in the repository, the Modified Resources window will appear, asking you which of the new resources you created you want to keep. By default, all new base resources you created will be kept. Mark the appropriate resources and click OK.

2. Click the Fetch button in the Resource Explorer.

To perform the fetch operation, click the Fetch button in the Resource Explorer.



Click Fetch to fetch the selected source file.

If the fetch operation was successful, the source file will be copied into your dictionary.



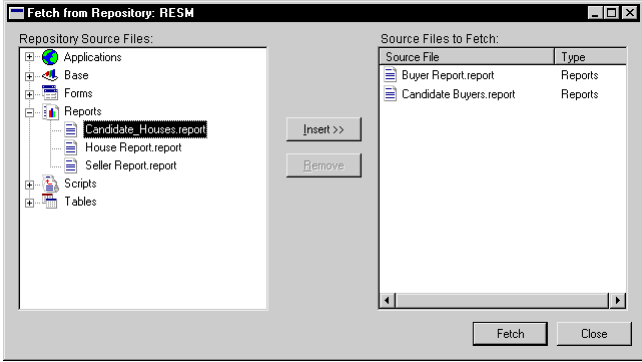
You can also fetch a source file by selecting its corresponding resource in the Resource Explorer and clicking Fetch. Dexterity will automatically fetch the appropriate source file for the resource you have selected.

Fetching multiple source files

You can fetch multiple source files at one time. To do this, perform the following procedure:

1. Open the Fetch from Repository window.

To open the Fetch from Repository window, point to Source Control in the Explorer menu and choose Fetch. The window is shown in the following illustration.



2. Select the source files to fetch.

In the Repository Source Files list, select the source files you want to fetch and click Insert.

3. Fetch the source files.

Click Fetch to begin fetching source files from the repository. A dialog box will be displayed, indicating progress as source files are retrieved from the repository. When all source files have been fetched, click Close to close the Fetch from Repository window.

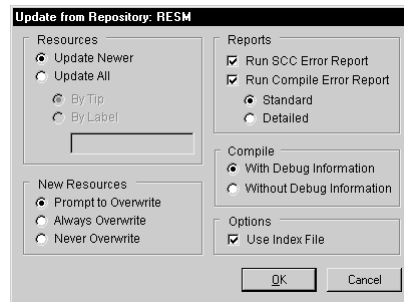
Updating a dictionary

In the update process, source files that meet specified criteria are read from the repository and imported into your dictionary. In a typical update, source files for resources that are newer than those in your dictionary are retrieved from the repository and imported into your dictionary. You can perform this type of update process simply by clicking the Update button in the Resource Explorer.



Click Update to update resources that have newer versions in the repository.

If you need to perform an update operation that retrieves source files that meet specific criteria from the repository, don't use the Update button. Instead, use the Update from Repository window, which is accessed by pointing to Source Control in the Explorer menu and choosing Update. The window is shown in the following illustration.



You can specify the following options that indicate which resources will be updated and what reports will be generated:

Resources

These options specify which source files will be retrieved from the repository.

Update Newer Selecting this option will update all resources that have a newer revision of their corresponding source file in the repository.

Update All Selecting this option will update all resources in the dictionary with a version from the repository. If you choose By Tip, the latest version of each source file in the repository will be used. If you choose By Label, the source files in the repository that have the specified label will be used.

New Resources

This option specifies how Dexterity will handle cases where a new resource you created in your dictionary has a corresponding resource that is being pulled from the repository. You can choose to be prompted to overwrite the resource you created, always overwrite the resource, or never overwrite it.

Reports

When you perform an update operation, two reports can be generated. The SCC Error Report lists all of the source code control errors that occurred during the update process. The Compile Error Report lists all of the compiler errors that occurred as scripts were compiled during the update process.

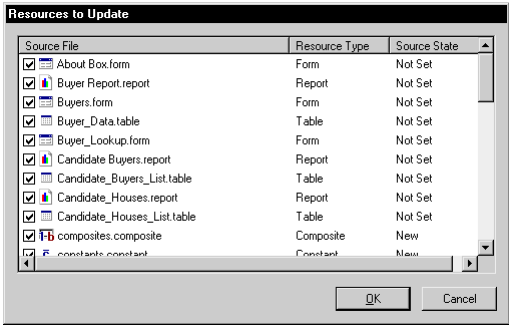
Compile

This option specifies whether scripts compiled as part of the update operation will be compiled with Debug information. If you choose to compile with Debug information, any **debug** and **assert** statements in your code will be compiled, and you will be able to use the Script Debugger to debug your code. We recommend compiling without Debug information in the final version of your application to ensure optimal script performance.

Options

When you perform the update operation, you can use the index file, which ensures that resource IDs are used consistently in your dictionary. An index file must have been created and checked into the repository before you can use this option. Index files are discussed in detail in [*Creating and updating the index file*](#) on page 455.

After you have set the update options, click Update. If you marked the Prompt to Overwrite option, the Resources to Update window will be displayed, as shown in the following illustration.



This window lists which sources files retrieved from the repository will overwrite existing resources in your dictionary. If you unmark any of the source files listed, they will not be imported into your dictionary. Click OK to complete the update process.

Creating and updating the index file

Each time you build a dictionary from the source files in the repository, it is desirable for each resource to have the same resource ID. The *index file* is a special file stored in the repository that is used to maintain consistent resource IDs in your dictionary. It stores a list of all resources and their corresponding resource IDs.

Typically, the index file is created and managed by the person responsible for producing production builds of your dictionary. Having one person manage the index file ensures that its content remains consistent.



The index file is stored in the Admin directory of the repository. We recommend that you limit write access to this directory to only those users you want to update the index file.

Creating the index file

To create the initial version of the index file, be sure that you have a complete dictionary that was built from the latest source files in the repository. This dictionary will serve as your baseline. To create the index file, point to Source Control in the Explorer menu and choose Update Index File. The index file will be created based on the current dictionary and checked into the repository.



The Update Index File menu item will be available only if you have marked the Enable Administrative Features option in the Source Control tab of the Options window.

Updating the index file

You will typically update the index file after you have created a production build of your dictionary. This is the best time to update the index file, because the dictionary will be in a complete and consistent state. To update the index file, point to Source Control in the Explorer menu and choose Update Index File. A new revision of the index file will be checked into the repository.

Building a dictionary

The build process involves creating a dictionary from scratch, based on the source files in the repository. Building a dictionary is typically performed by one person, and is done each time you create a production version of your dictionary.

The build process

A typical build process includes the following steps:

1. Create an empty dictionary.

You must create a “clean” dictionary that will contain the resources read from the repository. This can be done either with Dexterity or from the command line. The following command will start Dexterity and create a new “clean” dictionary named RESM.DIC.

```
dex.exe /nc RESM.DIC
```

2. Perform an update operation.

The update operation will retrieve source files from the repository and import them into your dictionary. Typically, you will be retrieving the latest version of each source file. However, if you label your source files, you can easily build versions of your dictionary based on the labels you've used in the repository.

When you perform the update operation, be sure that you use the index file. This ensures that resource IDs are used consistently in the dictionary.

3. Update the index file.

After the new dictionary has been successfully built, be sure to update the index file. The updated index file will contain entries for any new resources that were added to this build of the dictionary.

4. Label the source files in the repository (optional).

After the dictionary has been built, we recommend that you label all of the source files that were used to create the dictionary. This allows you to re-create this dictionary at any time simply by performing an update operation based on the label you supplied.

Automating the build process

Typically, you will want to perform the build process at a time when no developers are making changes to source files in the repository. Automating the build process allows it to be performed at any time, such as overnight, when no changes are being made to the repository.

The most basic automation of the build process involves creating a .BAT file that creates a new dictionary and runs a "build macro". The build macro is a Dexterity macro that performs the update process that builds the dictionary.

The following is a build macro that performs an update from the currently selected repository into an empty dictionary. The most recent (tip) version of each source file is used for the update.

Macro**Build.mac**

```
# DEXVERSION=DEX 6.00
Logging file 'macro.log'
CheckActiveWin dictionary 'default' form ResourceExplorer window
ResourceExplorer
MenuSelect title Explorer entry 'Update...'
```

```

NewActiveWin dictionary 'default' form SCCUpdate window Update
ClickHit field '(L) RG_ModifiedResource' item 1 # 'Always
Overwrite'
MoveTo field '(L) RG_Update' item 0 # 'Update Newer'
ClickHit field '(L) RG_Update' item 1 # 'Update All'
MoveTo field '(L) CB_IndexFile'
ClickHit field '(L) CB_IndexFile'
MoveTo field '(L) PB_Update'
ClickHit field '(L) PB_Update'
NewActiveWin dictionary 'default' form ResourceExplorer window
ResourceExplorer
NewActiveWin dictionary 'default' form ResourceExplorer window
ResourceExplorer
NewActiveWin dictionary 'default' form ResourceExplorer window
ResourceExplorer
NewActiveWin dictionary 'default' form 'Script_Debug_Res' window
CompileDebugInfo
NewActiveWin dictionary 'default' form ResourceExplorer window
ResourceExplorer

```

This macro was created by recording the user actions necessary to perform the update process. Notice the second line of the macro. It was added using a text editor such as Notepad. This line prevents any message displayed by the macro, such as the total running time, from being displayed on the screen. Instead, all messages generated by the macro will be written to the MACRO.LOG file.

To use this build macro to perform a build, start Dexterity from the command line, create a new “clean” dictionary, and then run the build macro. The following is a batch file that will start Dexterity, create a new dictionary named RESM.DIC, and execute the macro named Build.mac.

```

dex.exe /nc RESM.DIC Build.mac

```

To further automate the process you could have the build macro update the index file. If you were using Visual SourceSafe as your source code control provider, you could also add the following line to the batch file to label the source files used to for the build:

```

ss Label $/RESM -L6.0b1 "-CBuild 6.0b1"

```

This adds the label “6.0b1” and the comment “Build 6.0b1” to the latest version of each source file in the RESM project. Be sure that the SS.EXE application is in your path, that you have the “Add” access right that allows you to use the Label command.

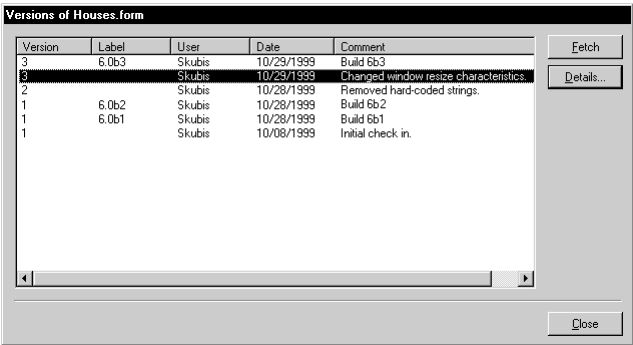
Viewing version information

You can view version information for a specific source file or resource from within Dexterity. To view version information, select a source file or resource in the Resource Explorer, then click the Version Information button.



*Click Version Information
to view version
information for the
selected source file.*

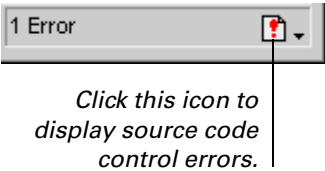
The Versions window will be displayed, as shown in the following illustration.



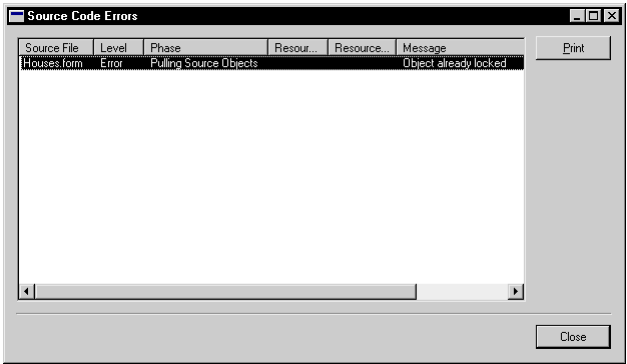
This window allows you to view the various versions of the selected source file, the comments associated with each version, and any labels that have been applied. You can also fetch a specific version of the resource from the source code control repository. Select the appropriate version in the list and click Fetch.

Source code control errors

As you perform source code control operations, you may encounter errors. If errors occur, an icon will be displayed in the lower-right corner of the Resource Explorer. Click the icon to display the Source Code Errors window.



This window lists any errors that occurred during the last source code control operation. Use the information provided to help you resolve any errors that occurred.



Chapter 54: Maintaining the Repository

When using source code control in Dexterity, you may need to perform maintenance activities in the repository. Information about maintaining the repository is divided into the following sections:

- [*Deleting resources from a dictionary*](#)
- [*Renaming resources*](#)
- [*Labels*](#)
- [*Updating source file states*](#)

Deleting resources from a dictionary

When using source code control, you must pay special attention when you delete resources from your dictionary. Depending on the type of resource you're deleting, you may need to perform some maintenance tasks in the repository.

Base resources

If you're deleting a base resource, such as a datatype or field, you won't need to perform any maintenance tasks in the repository. Because base resources are stored as a group, deleting a resource is just like any other change to the source file.

Standard resources

If you're deleting a standard resource, such as a form or table, you will need to perform an additional operation to have the repository be updated properly. When you delete a resource, you want future revisions of the dictionary to no longer include it. However, you want earlier revisions to still include the resource. To have this occur, do the following:

- 1. Check out the resource you want to delete.**

To delete a resource, you must have it check out from the source code control repository.

- 2. Delete the resource from the dictionary.**

Delete the resource from the dictionary, as you normally would.

3. Check the source file for the resource into the repository.

The source file for the resource will still be listed in the repository view in the Resource Explorer. Select the source file for the resource you just deleted and click Check In. An empty version of the source file will be check into the source code control repository.

All of the earlier revisions of the source file will still exist in the repository. You will still see the source file when you view the contents of the repository with Dexterity. However, the empty source file will prevent the source file from being converted into a resource in future revisions of your dictionary.

Renaming resources

If you are using source code control, you must pay special attention when you rename resources in your dictionary. Depending on the type of resource you're renaming, you may need to perform some maintenance tasks in the repository.

Base resources

If you're renaming a base resource, such as a datatype or field, you won't need to perform any maintenance tasks in the repository. Because base resources are stored as a group, renaming a resource is just like any other change to the source file.

Standard resources

If you're renaming a standard resource, such as a form or table, you will need to perform some maintenance tasks in the repository. The name of the source file for a standard resource is based on the resource's name. If you change the name of the resource, the source file name will no longer match. This can make it difficult to find the resource when you're looking for it in the repository.

If you must rename a standard resource, we recommend that you also change the name of its corresponding source file. To do this, perform the following procedure:

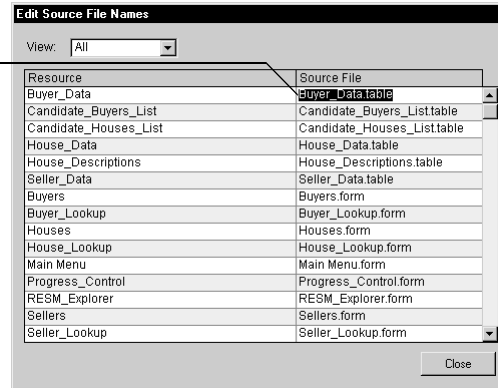
1. Rename the resource in the dictionary.

In Dexterity, rename the resource. Be sure to note the new name. You will need it for the next two steps.

2. Rename the source file in the dictionary.

Each dictionary keeps a list of the source files that correspond to the resources in the dictionary. You can edit this list using the Edit Source File Names window. To open this window, point to Source Control in the Explorer menu and choose Source Files.

*Edit the source file name
to match the resource
name you are changing.*



In the Edit Source Files window, locate the source file corresponding to the resource whose name you changed. In the Source File column, change the name of the source file to reflect the change you made to the resource name.

3. Rename the source file in the repository.

Use the utilities included with your source code control provider to change the name of the source file in the repository to match the name you specified in the previous step. For example, if you are using Microsoft Visual SourceSafe, you can use the Visual SourceSafe Explorer to change the name of a source file in the repository.

Labels

Labels are useful for indentifying specific revisions of source files in the repository. You can't apply labels to source files from within Dexterity. Instead, you must use the utilities included with your source code control provider to apply labels to source files.

Labeling is commonly performed during milestones in the development process, such as alpha, beta, release, and so on. It is also useful to label source files each time you build your application.

Updating source file states

Each source file has state information that is stored in the dictionary. This information allows Dexterity to know the current status of the source file, such as whether it is new or checked out. In some cases, changes may have been made to the state of source files in the repository that aren't reflected in your dictionary.

For example, assume you checked out a source from the repository. Later, someone else used the administrative tools for the source code control system to manually release the lock on the source file. Your dictionary will still indicate you have the source file checked out. If this occurs, you need to update the state information for the source files in your dictionary. You can do this by pointing to Source Control in the Explorer menu and choosing Update SCC State. This command will set the state information for the source files in your dictionary, based on the contents of the repository.

Updating source file states performs the following actions:

- Source files that exist in your dictionary, but not in the repository will be set to the New state.
- Source files you had locked in the repository, but that are no longer locked in the dictionary will be set to the Checked Out state.
- Source files that are in the Checked Out state in the dictionary, but are not locked in the repository will no longer be in the Checked Out state.
- Source files that are in the New state in your dictionary, but also exist in the repository will no longer be in the New state.

Chapter 55: Source Code Control for Integrating Applications

Using source code control for Dexterity-based applications that integrate with Microsoft Dynamics GP is very similar to using it with a stand-alone dictionaries. The following sections provide information about using source code control with integrating applications:

- [*Configuring Dexterity*](#)
- [*Developing an application*](#)
- [*Checking in a development dictionary*](#)
- [*Moving to new versions of Microsoft Dynamics GP*](#)

Configuring Dexterity

When configuring Dexterity to use source code control for an integrating application, be sure to set the Original Dictionary field in the SCC tab of the Options window in Dexterity. For resources to be properly checked into the repository, you must specify the location of an unmodified original dictionary for the version of Microsoft Dynamics GP in which you are developing.

Developing an application

As usual, you will develop your integrating application in a development dictionary. Remember that a development dictionary is the Dynamics.dic dictionary that contains the forms and reports you modified, as well as the additional resources you've added.

Preparing the development dictionary

Before you begin the development process, it's important that the resources in the development dictionary (Dynamics.dic) are in the correct state. All existing resources should be in the Main Product state, indicating they are not third-party resources. You can check this in the Resource Explorer.

If all of the resources in the development dictionary indicate they are in the New state, the source file state must be updated before you can continue developing your integration. To update the source file state, complete the following procedure.

1. Enable SCC administrative features.

In the Options window in Dexterity, display the Source Control tab. Mark the Enable Administrative Features option and click OK.

2. Open the development dictionary.

If you haven't done so already, open the Dynamics.dic dictionary you plan to use for development.

3. Select the appropriate repository.

You must choose an empty repository, or the repository that contains the resources from a previous version of your integration. You will be updating the state of the source files for the dictionary based on the content of this repository.

4. Update the source file state.

In the Explorer menu, point to Source Control and choose Update SCC State. A dialog will be displayed, indicating the source file states are being synchronized with the repository.

When the process is complete, the Resource Explorer will show that the resources from the dictionary are in the Main Product state. The development dictionary is now ready for use.

Adding new resources

You can add resources of any type to the development dictionary. When you add new resources, they will be in the New source code control state, allowing you to check them into the repository.

Main product resources

All of the original resources in the development dictionary will be in the Main Product source code control state. The only original resources you can modify are forms and reports. When you modify an original form or report, its source code control state will change to New, allowing you to check it into the repository.



Remember that an original form or report you modify is called an alternate form or report.

Checking in a development dictionary

To check in the development dictionary for an integrating application, open the development dictionary with Dexterity. When you check in the development dictionary, the repository will store only the new resources you added. It will also store the alternate forms and alternate reports you created.

Alternate forms

When you check an alternate form into the repository, the alternate form and the original are compared. Only the changes you made are checked into the repository.

Alternate reports

When you check an alternate report into the repository, the entire report is checked in. No comparison is made between the original report and the modified version.

Moving to new versions of Microsoft Dynamics GP

Using source code control when developing an integrating application makes it much easier to move to new versions of Microsoft Dynamics GP. To move your application to a new version of Microsoft Dynamics GP, complete the following procedure:

- 1. Make a copy of the new version of the Dynamics.dic dictionary.**

This copy will become your new development dictionary.

- 2. Specify the location of the original Dynamics.dic dictionary.**

In the Source Control tab of the Options window in Dexterity, specify the location of an unmodified copy of the dictionary for the new version of Microsoft Dynamics GP you are moving to.

- 3. Create a branch in the repository for the new version.**

We recommend that you create a new branch in the source code control repository for the new version of your application. This new branch will typically be based on the latest revision of the source files for the previous version of your integrating application. Refer to the documentation included with your source code control provider for information about creating a branch.

4. Configure Dexterity to access the files in the new project.

In the Source Control tab of the Options window in Dexterity, specify the new branch you created in the repository as the project you want to use.

5. Perform an update operation.

Point to Source Control in the Explorer menu and choose Update. Specify that you want to retrieve the latest version of each source file in the repository, and click Update. The resources for your application will be added to the new development dictionary.

Any new resources you created for your integrating application will be added to the development dictionary. For any alternate forms you created, the changes you made for your integrating application will be applied to the corresponding forms in the development dictionary. Any alternate reports you created will *replace* the corresponding reports in the development dictionary.

6. Verify the results of the update.

After the update is complete, verify that the appropriate resources were added to the development dictionary. Also verify that any alternate forms and alternate reports were updated properly.

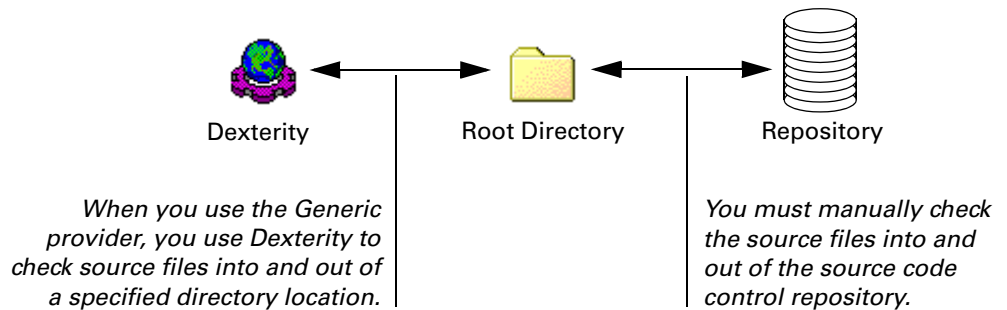
Chapter 56: Generic Provider

If the source code control provider you want to use isn't directly supported by Dexterity, you can use the Generic provider. Information about using the Generic provider is divided into the following sections:

- [Specifying the project location](#)
- [Creating a project](#)
- [Checking in a new dictionary](#)
- [Checking out source files](#)
- [Checking in source files](#)

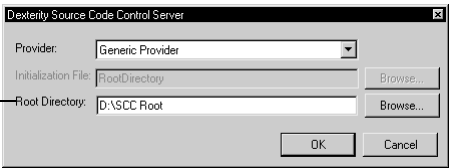
Specifying the project location

When you use the Generic provider, you aren't checking source files directly into and out of the source code control repository. Instead, you are checking them into and out of a directory location you specify. You will then manually check the source files into and out of the repository for the source code control provider you are using. The process is shown in the following illustration.



You will use the Dexterity Source Code Control Server control panel to specify the directory where Dexterity will check files into and out of. This “root” directory will contain project directories that store the source files for each of your dictionaries. Typically, this root directory will be located on the same machine that is running the Dexterity Source Code Control Server and the source code control provider you are using.

The Root Directory specifies the location of the project directories that contain the source files for your dictionaries.



Creating a project

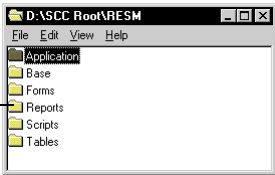
When using the Generic provider, you create a project by creating a directory in the location you specified as the Root Directory in the Dexterity Source Code Control Server control panel. For example, if you specified the location D:\SCC Root as the Root Directory, you could create the RESM project by creating the \RESM directory in that location.

Checking in a new dictionary

Check in a new dictionary just as you would with any of the source code control providers supported by Dexterity. Refer to [Checking in a new dictionary](#) on page 440 for a complete procedure for checking in a dictionary.

During the check in process, several directories will be created in the project directory you specified. These directories categorize all of the source files that you will need to manually check into the source code control repository. The following illustration shows the directories that were created when the RESM dictionary was checked in using the Generic provider.

The Generic provider created these directories. The directories contain the source files created from the dictionary.



Checking out source files

When you want to check out a source file and modify it in Dexterity, you must first manually check it out of the repository. If the source file already exists in the directory for the project, it will be overwritten. In Dexterity, you can perform a check out operation, which will read the source file from the project directory and import it into the current dictionary.

Keep in mind that even though Dexterity shows a source file as being Checked Out, no locking occurs in the project directory. Other developers could perform a check out operation and make modifications to the same resource.

Checking in source files

When you have finished making changes to a resource, you will check it in with Dexterity. The check in operation overwrites any existing source file in the project directory with the source file created from your dictionary. You will then manually check the source file into the source code control repository.

Part 10: Using the Runtime Engine

If your product integrates with Microsoft Dynamics GP, refer to the Integration Guide for information about packaging your application.

Use the information in this portion of the manual to learn about running your application using the runtime engine. Following is a list of the items that will be discussed:

- [Chapter 57, “Setting Product Information,”](#) describes how to add product information to an application dictionary.
- [Chapter 58, “Using Launch Files,”](#) describes how a launch file is used with the runtime engine to start an application, how to create a launch file, and what the launch file contains.

Chapter 57: Setting Product Information

Before you can use your application with the runtime engine, you must add product information to the dictionary. This portion of the documentation describes what product information is and how add it to an application dictionary. The information is divided into the following sections:

- [Product information](#)
- [Procedure: Setting product information](#)

Product information

Launch files are described in detail in the next chapter.

Product information is used when the launch file for an application dictionary is created. Without product information, a launch file can't be properly created. A launch file is used by the runtime engine to start the application. For example, the launch file RESM.SET is used to start the Real Estate Sales Manager application RESM.DIC.



RESM.SET



RESM.DIC

This launch file is used to start the RESM.DIC application dictionary using the runtime engine.

Product information, which is added to the application dictionary using the Product Information window in Dexterity Utilities, consists of the following items:

Launch File This is the default launch file name. The name is limited to eight characters plus a three-character extension. The name should have the .SET extension. You can assign a different name when the launch file is created using the runtime engine.

Launch ID The launch ID is an integer that uniquely identifies the launch file. If the application is a stand-alone application, this value should be the same as the product ID.

Product Name This is the complete name of the product.

Product ID Product ID is an integer that uniquely identifies the product. A product ID is obtained by registering your product with Microsoft Business Solutions. Products should be registered so they each have a unique product ID. This minimizes the possibility of conflicts between different Dexterity applications. If you haven't registered your product, use 0 as the Product ID.

Within applications, the product ID functions as the value for the Product ID parameter used for security and some of the functions in the function libraries. The product ID is also sometimes referred to as the Dictionary ID.

Forms Dictionary This is the name the forms dictionary will be given when it is created for the application. The name is limited to eight characters plus a three-character extension. The name should have the .DIC extension.

Reports Dictionary This is the name the reports dictionary will be given when it is created for the application. The name is limited to eight characters plus a three-character extension. The name should have the .DIC extension.



You must provide names for the forms and reports dictionaries, even if the Modifier and Report Writer won't be used in your application. Otherwise, the launch file won't work properly.

Compatibility ID The Compatibility ID is a string that uniquely identifies the version of your application. When you launch your application with the runtime engine, the compatibility ID in the application dictionary is compared with the compatibility IDs in the forms and reports dictionaries. If they match, items in the forms and reports dictionaries can be accessed.

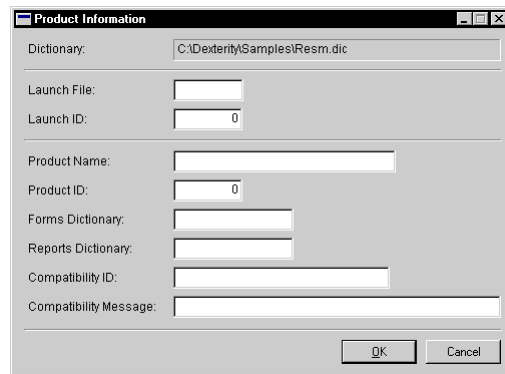
Compatibility Message The Compatibility Message is displayed when the compatibility IDs in the forms or reports dictionaries don't match the compatibility ID in the main dictionary. You should provide information in this message describing how to update your forms and reports dictionaries to make them compatible with the current version of your application. Refer to [Chapter 60, "Updating an Application,"](#) for more information about updating forms and reports dictionaries.

Procedure: Setting product information

For stand-alone applications, use the Product Information window in Dexterity to add product information a dictionary. If you are creating an application that integrates with Microsoft Dynamics GP, you must use Dexterity Utilities to add product information. To add product information to a stand-alone application, complete the following procedure:

1. Open the Product Information window.

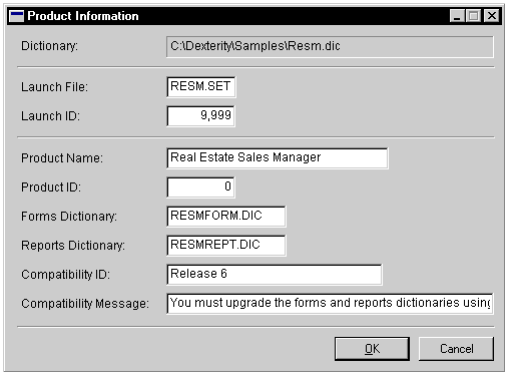
In the Resource Explorer, choose to display Application resources. Select the Product resource and click Open.



2. Add the product information and click OK.

Example

The following illustration shows the Product Information window for the Real Estate Sales Manager application.



When created, the launch file will have the name RESM.SET and the ID 9999. The product has the name Real Estate Sales Manager. It's a main (stand-alone) product with the ID 9999. This means the Product ID parameter used for security and several of the functions in the function libraries will also be 9999. When created, the forms dictionary will have the name RESMFORM.DIC and the reports dictionary will have the name RESMREPT.DIC. The Compatibility ID indicates this dictionary is Release 6, so any existing forms and reports dictionaries must be updated before they can be used with this version of the application.

Chapter 58: Using Launch Files

A launch file is required to run an application dictionary using the runtime engine. The launch file contains necessary information to start the application, such as the location of the dictionary, the product ID, and the locations of the forms and reports dictionaries. Information about using launch files is divided into the following sections:

- [*Creating a launch file*](#)
- [*Contents of a launch file*](#)
- [*Using the launch file to start your application*](#)

Creating a launch file

The runtime engine can be used to create a launch file for an application dictionary, but before that can happen, you must have added product information to the application dictionary. Refer to [Chapter 57, “Setting Product Information,”](#) for information about adding product information.

Refer to the COMPNT.PDF file included with Dexterity for a list of components for Dexterity.

The runtime engine along with necessary DLLs, DEX.DIC, and the application dictionary should all be in the same location in order for the launch file to be created properly. No other application dictionaries should be in this location. To create a launch file, perform the following steps:

1. Start the runtime engine.

Use the Windows Explorer and double-click the runtime engine (Dynamics.exe) file.

2. Choose Create Launch File from the File menu.

A dialog box will appear containing the default name of the launch file. You can change the name of the launch file if you want. Click Save. The launch file will be created.

3. Choose Open Launch File from the File menu.

A dialog box will appear. Locate the launch file you created and click Open. The application’s main menu will appear.

Contents of a launch file

The following illustration shows the contents of the launch file for the Real Estate Sales Manager application.

This is the number of dictionaries operating. For stand-alone applications, this will always be 1.

This is the launch file ID and the product ID of the main product.

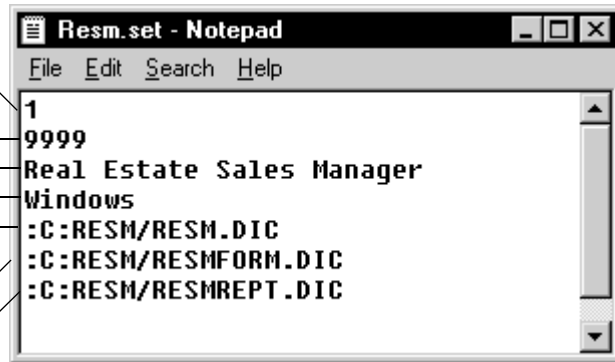
This is the main product name.

This is the dictionary location ID.

This is the location of the application dictionary.

This is the location of the forms dictionary.

This is the location of the reports dictionary.



The launch file contains the following items:

- The first line contains the number of dictionaries operating. For stand-alone applications, this will always be 1. Only launch files for multidictionary applications, such as Microsoft Dynamics GP, will have more than one dictionary operating simultaneously.
- The second line indicates both the ID of the launch file and the product ID of the main product.

Within the application, the product ID is the value passed to the Dictionary ID parameter used for security and online documentation.

- The third line contains the name of the main product.
- The next four lines contain a dictionary location ID and the locations of the application dictionary, the forms dictionary, and the reports dictionary for the main product. The locations are in generic format.

A launch file can have several sets of dictionary location IDs and dictionary locations. This may be useful if you have certain workstations that you want to access a different set of dictionaries. The set of dictionaries to use is specified by the **Workstation2** setting in the DEX.INI file. For example, if the application were started from a workstation that had “Alternate” as its **Workstation2** setting, the set of dictionaries identified by the dictionary location ID “Alternate” would be used.



*If the DEX.INI file doesn't contain a **Workstation2** setting when you attempt to create a launch file or use a launch file to start an application, a dialog box will appear. This dialog box allows you to select a dictionary location ID to use. Your selection will be added as the **Workstation2** setting.*

Using the launch file to start your application

Refer to the COMPNT.PDF file included with Dexterity for a list of components for Dexterity. Information about the DLLs required for Dexterity can be found there.

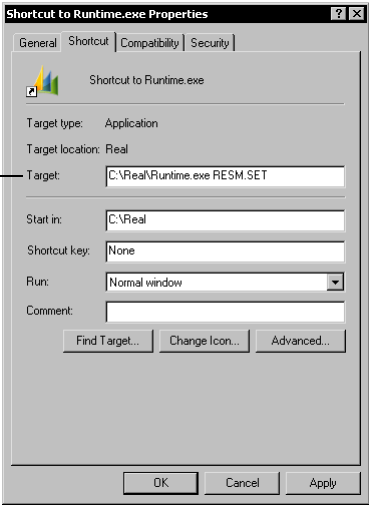
For Window, be sure the runtime engine (along with the necessary DLLs), launch file, application dictionary and the DEX.DIC file are all in the same location. Open the Explorer and locate the icon for the runtime engine (RUNTIME.EXE). Create a shortcut to the runtime engine by dragging its icon to the desktop. Edit the properties for the shortcut and add the name of the launch file to the Target field so the runtime engine will use the correct set of dictionaries.

For example, the RESM.SET launch file contains the information necessary to start the Real Estate Sales Manager. The Target field in the properties for the shortcut would look like the following:

```
C:\REAL\RUNTIME.EXE RESM.SET
```

The properties window for the shortcut to the Real Estate Sales Manager application is shown in the following illustration.

The launch file name is added to the end of the Target field.



Part 11: Packaging Applications

Use the information in this portion of the documentation to learn about packaging your application to deliver to customers. Following is a list of the topics that will be discussed:

- [Chapter 59, “Making Installation Files,”](#) describes how to use Dexterity Utilities to create “chunk” files that are used to install your application. Final packaging of your application is also described.
- [Chapter 60, “Updating an Application,”](#) describes how to create and send updates of your application to customers.

Chapter 59: Making Installation Files

After your application is complete, you will finish the development process by packaging your application for delivery to your customers. Typically, this involves writing some installation scripts, building chunk dictionaries, creating an “empty” launch file, and testing the installation process.

Information about making installation files is divided into the following sections:

- [Installation scripts](#)
- [Building chunk dictionaries](#)
- [Creating an empty launch file](#)
- [Testing the installation process](#)
- [Additional components](#)
- [Runtime command line parameters](#)

Installation scripts

Information about how to determine when install scripts run in your application is found in the next section.

Installation scripts are procedures in your application that run automatically when your application is installed. These scripts perform actions such as adding items to palettes or setting up default pathnames. Installation scripts are created just like ordinary procedures. For example, the following procedure is used as an installation script to add items to the Reports palette. The script runs when the application is installed.

Procedure name: Add_Reports_Palette_Items

```
{This script adds the items for the Reports palette.}
```

```
{Call the procedure to add the Buyers Report.}
```

```
call Add_Palette_Item, "Reports", "Buyers", 3, "Houses Report", "";
```

```
{Call the procedure to add the Sellers Report.}
```

```
call Add_Palette_Item, "Reports", "Sellers", 3, "Sellers Report", "";
```

```
{Call the procedure to add the Houses Report.}
```

```
call Add_Palette_Item, "Reports", "Houses", 3, "Houses Report", "";
```

Building chunk dictionaries

When your application dictionary is complete, you will use Dexterity Utilities to split your application dictionary into “chunks.” The dictionary chunks are part of what you deliver to your customer. When your application is installed, the dictionary chunks are recombined to create the application dictionary. Chunk dictionaries are used for two reasons:

- When an application dictionary is installed from dictionary chunks, the launch file for the application is updated automatically. In contrast, if you deliver your application as a complete dictionary, you’ll have to create a launch file using the runtime engine, as described in [Chapter 58](#).
- Chunks allow large dictionaries to be divided into more manageable pieces.

Dexterity Utilities has two utilities that create dictionary chunks. The Auto-Chunk utility is used to make a single dictionary chunk from your application dictionary, while the Create Chunk Dictionary utility can be used to make several chunks from your application dictionary.

You will typically use the Auto-Chunk utility to make a single chunk that contains all of the dictionary’s resources. If your application dictionary is large and your installation program doesn’t support segmenting files, you may want create several chunks using the Create Chunk Dictionary utility. A common way to divide an application dictionary into several chunks is to include core resources in one chunk, forms in another chunk, and reports in a third chunk.

Using the Auto-Chunk utility

To use the Auto-Chunk utility to create a single dictionary chunk for your application dictionary, perform the following steps.

1. Start Dexterity Utilities.

Open your application dictionary as an editable dictionary.

2. Choose Auto-Chunk from the Utilities menu on the toolbar.

The Auto-Chunk window will appear. Click the Chunk Dictionary lookup button; a dialog box will appear, prompting you to name the dictionary chunk you're creating. Name the chunk and click OK.



Due to a historical limitation imposed by early versions of Windows, the name of the chunk dictionary must not exceed 13 characters, including the .cnk extension. If the name exceeds 13 characters, the chunk cannot be accessed by the runtime engine and will not be unchunked.

3. Use the Auto-Chunk window to complete the process of creating a chunk.

Click the Chunk Dictionary lookup button to name the new dictionary chunk.

Dictionary Enter the name of the dictionary that the chunk will become part of. For instance, if you're creating a chunk for the RESM.DIC application dictionary, enter that name here.

Refer to the **Runtime GetModule Info()** function in the Function Library Reference manual for information about retrieving version and build numbers.

Module This selection indicates which module the Major Version, Minor Version and Build Number information is associated with. (These are the fields below the Module field.) The setting in the Module field is used when the version numbers and build number are retrieved from the dictionary.

Major Version/Minor Version Indicate the major and minor version numbers for your application. For instance, if this is release 1.3 of your application, enter a 1 for the major number and a 3 for the minor number. These version numbers can be retrieved from the dictionary and typically are displayed in the application's About Box.

The values you choose for the major version and minor version are significant. If the installation folder contains a dictionary for a previous version of your product dictionary, the dictionary chunk you are creating will unchunk only if the major version and minor version numbers in the chunk match those in the existing dictionary. If the version numbers don't match, an error indicating the issue will be written to the InstallErrors.txt file in the installation folder. The dictionary chunk won't be deleted.

Build Number Indicates the build of the current version of the dictionary. The build number can also be retrieved from the dictionary and is typically displayed in the application's About Box.

If you have multiple dictionary chunks with the same major version and minor version values, the build number is used to determine the order in which the dictionary chunks are unchunked. The chunks with lower build number values are unchunked first.

If the installation folder contains a dictionary for which the major and minor version numbers match those of the dictionary chunk, the build number will be examined. If the build number of the chunk is the same or greater than the build number of the dictionary, the chunk will be unchunked. If the build number of the chunk is lower, it will not be unchunked, and a message indicating the issue will be written to the InstallErrors.txt file in the installation folder. The dictionary chunk will be deleted.

Refer to [Installation scripts](#) on page 485 for information about writing installation scripts.

Starting Script/Ending Script These fields specify which installation scripts (procedures) will be run when the chunk is merged into the application dictionary. To specify a starting or ending script, mark the corresponding check box and then click the lookup button for the starting or ending script. A list of the procedures in the application dictionary is displayed, allowing you to select a script.

The installation scripts run only after the dictionary chunk has been merged into the application dictionary. The starting script is run first, then the ending script is run.

Compression Choose to remove either unused blocks from the chunk or to perform total compression, which removes unused blocks and all script source. Typically, total compression is performed when an application is being prepared to send to customers.

4. Click OK to begin the chunking process.

After the chunk dictionary has been created, the Auto-Chunk window will close.



At this point, we recommend that you make a backup of your chunk dictionary.

Using the Create Chunk Dictionary utility

To use the Create Chunk Dictionary utility to create one or more dictionary chunks for your application dictionary, perform the following steps.



If you're creating only one dictionary chunk for your application, we recommend that you use the Auto-Chunk utility. It is much easier to use.

1. Start Dexterity Utilities

Open your application dictionary as a source dictionary.

2. Choose Create Chunk Dictionary from the Utilities menu on the toolbar.

A window will appear asking whether you want to edit an existing dictionary chunk. Click Yes if you want to add resources to or make changes to existing dictionary chunk. Click No if you want to create a new dictionary chunk.

If you clicked Yes, a window will appear allowing you to select an existing chunk to open. If you clicked No, a window will appear allowing you to specify the name of a new chunk. Then the Create Chunk Dictionary window will appear.



Due to a historical limitation imposed by early versions of Windows, the name of the chunk dictionary must not exceed 13 characters, including the .cnk extension. If the name exceeds 13 characters, the chunk cannot be accessed by the runtime engine and will not be unchunked.

3. Use the fields in the Create Chunk Dictionary window to complete the process of creating a chunk.

Dictionary Enter the name of the dictionary that the chunk will become part of. For instance, if you’re creating a chunk for the RESM.DIC application dictionary, enter that name here.

Sequence Number Specify the order in which this chunk will be read when “unchunked.” If you’re creating a single chunk for the entire dictionary, specify 1. If you break the application dictionary into several smaller chunks, the chunks will be merged into the application dictionary in the order specified by the sequence number.

Refer to the [Runtime_GetModule Info\(\)](#) function in the Function Library Reference manual for information about retrieving version and build numbers.

Module This selection indicates which module the Major Version, Minor Version and Build Number information is associated with. (These are the fields below the Module field.) The setting in the Module field is used when the version numbers and build number are retrieved from the dictionary.

Major Version/Minor Version Indicate the major and minor version numbers for your application. For instance, if this is release 1.3 of your application, enter a 1 for the major number and a 3 for the minor number. These version numbers can be retrieved from the dictionary and typically are displayed in the application's About Box.

The values you choose for the major version and minor version are significant. If the installation folder contains a dictionary for a previous version of your product dictionary, the dictionary chunk you are creating will unchunk only if the major version and minor version numbers in the chunk match those in the existing dictionary. If the version numbers don't match, an error indicating the issue will be written to the InstallErrors.txt file in the installation folder. The dictionary chunk won't be deleted.

Build Number Indicates the build of the current version of the dictionary. The build number can also be retrieved from the dictionary and is typically displayed in the application's About Box.

If you have multiple dictionary chunks with the same major version and minor version values, the build number is used to determine the order in which the dictionary chunks are unchunked. The chunks with lower build number values are unchunked first.

If the installation folder contains a dictionary for which the major and minor version numbers match those of the dictionary chunk, the build number will be examined. If the build number of the chunk is the same or greater than the build number of the dictionary, the chunk will be unchunked. If the build number of the chunk is lower, it will not be unchunked, and a message indicating the issue will be written to the InstallErrors.txt file in the installation folder. The dictionary chunk will be deleted.

Refer to [Installation scripts](#) on page 485 for information about writing installation scripts.

Starting Script/Ending Script These fields specify which installation scripts (procedures) will be run when the chunk is merged into the application dictionary. To specify a starting or ending script, mark the corresponding check box and then click the lookup button for the starting or ending script. A list of the procedures in the application dictionary is displayed, allowing you to select a script.

The installation scripts run only after **all** dictionary chunks have been merged into the application dictionary. All starting scripts are run in the order the chunks were merged. Then all ending scripts are run in reverse order.

For example, if three dictionary chunks were merged, the starting and ending scripts would run in the following order:

1. Chunk 1 Starting Script
2. Chunk 2 Starting Script
3. Chunk 3 Starting Script
4. Chunk 3 Ending Script
5. Chunk 2 Ending Script
6. Chunk 1 Ending Script

Dictionary Module Type Select the type of resources (Forms, Reports or Core resources) you want to add to the chunk dictionary.

Series Select the series from which you want to add resources to the chunk dictionary.

Source Dictionary Modules Based upon your selection, this list fills with forms, reports or core resources for the selected series. To add a module to the chunk dictionary, select it from the list, then click Transfer. If you want to add all resources from the list, click All.

If you're making a single chunk dictionary, first add all forms and reports from your application dictionary to the Chunk Dictionary Modules list. Then add all core resources for the series listed from Financial through System in the Series list. Core resources for the Company through Report Writer series can be ignored because they contain duplicate resources from the System series. When you've finished, the size of the chunk dictionary should be nearly the same size as the source dictionary. (You can see this by viewing the Total Size fields beneath the source dictionary and the chunk dictionary lists.)

If you're making multiple chunk dictionaries, the entry in the Dictionary name field must be the same for each chunk. Also, each chunk *must contain* the **[Core] System Core Tables** and the **[Core] System Install** modules for the dictionary chunks to merge properly. Be sure that you haven't forgotten to include these resources in each dictionary chunk.

Chunk Dictionary Modules This list contains all of the modules from the application dictionary that will be included in the chunk dictionary. To delete a module from the chunk dictionary, select it from the list and click Delete.

4. Click OK to begin the chunking process.

When you've added the necessary resources to the Chunk Dictionary Modules list, click OK to begin the chunking process. After the modules have been copied to the chunk dictionary, the Create Chunk Dictionary window will close.



At this point, we recommend that you make backups of your chunk dictionaries.

Creating an empty launch file

To properly install your application using the chunk dictionaries you just created, you need to create an “empty” launch file for your application. The contents of an “empty” launch file are shown in the following illustration.

*The number of dictionaries
operating is zero.*

*This is the launch file ID
and the product ID for the
applications.*

0

9999

For the installation to work properly, the number of dictionaries operating must be set to 0, and the launch file ID must be the same as the product ID of the application.

The launch file's name must have the form *filename.SET* and be the same as the default launch file name you specified when you added product information to your application dictionary. Use a text editor such as Windows Notepad to create the empty launch file. Be sure to use the extension .SET when naming the launch file.

Testing the installation process

*Refer to the
COMPNT.PDF file
included with Dexterity
for a list of
components for
Dexterity. The list of
required components
can be found there.*

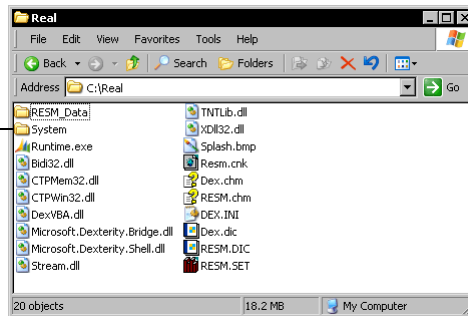


At this point, you're ready to test the installation procedure. Make copies of the runtime engine, the DEX.DIC file, necessary DLLs, your dictionary chunks, the empty launch file, and any other necessary files (such as online documentation files or sample data files) and place them in a folder or directory.

The components you're assembling here are the same components you'll be sending to your customers.

As an example, the files necessary to install the Real Estate Sales Manager application are shown in the following illustration.

System tables (such as Users and Security tables) and sample data for the Real Estate Sales Manager application are stored here.



Be sure you're using copies of the launch file and dictionary chunk files. The launch file will have information added to it and the dictionary chunk files will be deleted during the installation process.

When all of the files are in place, start the installation process by starting the application as you normally would. In a few moments, you should see a dialog box indicating that the dictionary chunks are being included and the application dictionary is being synchronized. Any installation scripts will be run. When these processes are complete, your application should be running.

If your application doesn't install properly, check the following:

- Be sure your application dictionary has product information before you create chunk dictionaries.
- Be sure the name of the chunk file is no more than 13 characters long, including the .cnk extension. If the name is longer than 13 characters, the chunk cannot be unchunked.
- Be sure the name of the launch file you are using is the same as the launch file name you specified in the product information for the dictionary.
- Verify that the version numbers and build number you applied to the dictionary chunk are correct. They must follow the rules described earlier.

- If you're using installation scripts, test them individually in your application.
- Before installing, be sure the empty launch file has 0 as the number of dictionaries operating and contains the correct launch (product) ID.
- If you're using multiple dictionary chunks, be sure the Dictionary name you've indicated for each dictionary chunk is exactly the same for each chunk. Also, be sure the sequence numbers are correct.
- Be sure your chunk dictionaries each contain the **[Core] System Core Tables** and the **[Core] System Install** resources.
- If you receive permissions errors, be sure the dictionary chunk file isn't locked. Also verify that no other process (such as an instance of the runtime engine) is accessing the dictionary to be updated.

After installation, a single application dictionary will have replaced the chunk dictionary or dictionaries. The launch file will have been updated to contain the correct location of the application dictionary.

Additional components

This section describes several additional components you may need to include some additional components with your installation.

DEX.INI file

If you would like your application to use its own DEX.INI file, be sure you include one with your application. You should install this file in the same location as the runtime engine. This ensures that your application will use this file instead of the file in the default location.



If you don't include your own DEX.INI file, one will be created in the default location. The default location is the WINDOWS directory.

Windows Data Access Components

Windows Data Access Components (Windows DAC) **must** be installed in order for Dexterity or the runtime engine to launch properly. If Windows DAC isn't installed, you will receive an error message indicating Dexterity or the runtime engine can't find certain ODBC components.

When packaging a stand-alone application, you need to ensure that Windows DAC is installed so the runtime engine will work properly. The components for Windows DAC are included on standard Windows installations, beginning with Windows XP.

SQL stored procedures

If your application will use Microsoft SQL Server, be sure to include any stored procedures you created that are required for your application.

Runtime command line parameters

The runtime engine (Dynamics.exe) can use several command line parameters to control its operation. The following table shows the sets of parameters that can be used.

Command and parameters	Description
Dynamics.exe	Launches the runtime, displaying the application shell.
Dynamics.exe <i>launch_file</i>	Launches the runtime, opening the set of dictionaries specified by the launch file. The complete path to the launch file can be specified.
Dynamics.exe <i>launch_file</i> /CNKONLY	Launches the runtime, extracting any dictionary chunk files found that can be merged into the set of dictionaries specified by the launch file. The runtime closes when the unchunking process is complete.
Dynamics.exe <i>launch_file Dex_ini</i>	Launches the runtime, opening the set of dictionaries specified by the launch file. The complete path to the launch file can be specified. Uses the Dex.ini file in the location specified.
Dynamics.exe <i>launch_file Dex_ini</i> /CNKONLY	Launches the runtime, extracting any dictionary chunk files found that can be merged into the set of dictionaries specified by the launch file. Uses the Dex.ini file in the location specified. Closes when the unchunking process is complete.
Dynamics.exe <i>launch_file macro_file</i>	Launches the runtime, opening the set of dictionaries specified by the launch file. The complete path to the launch file can be specified. Automatically runs the macro specified.

Specifying a macro to run on the command line can be useful to automatically log into an application when it is launched. To record a login macro, do the following:

1. Start the application.

Launch the application as you normally would, but don't log in.

2. Begin recording the macro.

Press ALT-F8 to begin recording the macro. Specify the name and location of the macro file.

3. Complete the login procedure.

Complete logging into the application as you normally would. Be sure to manually type any User ID or Password information so that it will be captured by the macro.

4. Stop recording the macro.

Once you have logged in, stop recording the macro by pressing ALT-F8.

5. Edit the macro.

The login macro is almost complete. However, you need to edit the macro using a text editor such as Notepad to add the following line to the second line of the macro:

```
Logging file 'macro.log'
```

Adding this line prevents any messages displayed by the macro, such as the total running time, from being displayed on the screen and preventing the login. Be sure to save your changes.

Chapter 60: Updating an Application

At some point after you've shipped your application to customers, you may need to send an update. This portion of the documentation contains information about how changing table definitions can affect your application, how to package an update, and how forms and reports dictionaries are affected by updating your application. Information is divided into the following sections:

- [Changing table definitions](#)
- [Converting c-tree and Pervasive.SQL tables](#)
- [Converting SQL tables](#)
- [How to package an update](#)
- [Creating an update dictionary chunk](#)
- [Updating forms and reports dictionaries](#)
- [Updating forms dictionaries](#)
- [Updating reports dictionaries](#)

Changing table definitions

If you change any table definitions in the process of updating your application, data from the previous version of your application may be incompatible with the updated version. Changes that make existing data tables incompatible include:

- Adding or deleting fields from a table definition
- Changing the storage size of a table field
- Adding or deleting table keys
- Changing the keys for a table

If you make these types of changes, existing data tables from the previous version of your application will have to be converted to be used with the new version. Table conversions are performed by procedures that you write.



We recommend that you duplicate a table definition before you make changes to it. Then you will have an original table definition that can be used in the table conversion process.

It's a good idea to track whether an update of your application requires tables to be converted. When the update is installed, code in your application can determine whether tables need to be converted, or have already been converted by a previous update. Typically, this information will be stored in a separate table for the application.

How you convert the data in a table depends on the type of table you want to convert. If you are converting c-tree or Pervasive.SQL tables, you can use the [Table_StartConversion\(\)](#) and [Table_EndConversion\(\)](#) functions to help perform the conversion. If you're converting tables stored in a SQL database, you can't use these two functions. Instead, you must use pass-through SQL to perform the conversion.



Be sure your customers make a backup of their data before any table conversions are performed.

Converting c-tree and Pervasive.SQL tables

The following procedure describes how to write a table conversion procedure for c-tree and Pervasive.SQL tables.

1. Duplicate the table definition that you will be changing.

In Dexterity, select the table you will be changing in the Resource Explorer, and choose Duplicate from the Utilities menu. Name the duplicate table definition. We suggest you give the duplicate table definition a name that indicates it was the original or old table definition. When you've finished, click OK.

2. Determine which tables have changed and need to be converted.

As an example, the table definition for the Seller_Data table for the Real Estate Sales Manager application will be changed as follows:

- The data type used for the Seller Name field will be changed from a 30-character string to a 45-character string.
- The Initial Contact Date field will be added to the table.
- A new key based on the Contact field will be added.

3. Make the necessary changes to the old and new table definitions.

Use the following chart to determine how to change the table definitions.

To do this	Perform these actions
Add new fields to the table	Add the new fields to the new table definition.
Remove fields from the table	Remove the fields from the new table definition.
Add, change or remove keys	Add, change or remove keys from the new table definition.
Change the storage size of fields	Create a “placeholder” field that has the same control type, keyable length, and storage size as original field you’re changing. In the old table definition, insert the placeholder field in place of the field you’re changing. Next, change the data type for the field you’re changing to reflect the new size. Finally, open the table definition for the new table so the correct buffer size is calculated.

Continuing the example, the duplicate table definition for the Seller_Data table was named Seller_Data_OLD. The new Initial Contact Date field was added to the Seller_Data table definition. The new key was also added to the Seller_Data table definition. The size of the Seller Name field was changed using the following steps:

- A new field named String30 was created as a placeholder for the Seller Name field.
- The String30 field was inserted in place of the Seller Name field in the Seller_Data_OLD table definition. The Seller Name field in the keys for the Seller_Data_OLD table was replaced with the String30 field as well.
- The Seller Name field was changed to use a new data type for a 45-character string.
- The Seller_Data table definition was opened to recalculate the table buffer size.

Refer to the *Function Library Reference manual* for information about [Table StartConversion\(\)](#) and [Table EndConversion\(\)](#)

4. Write the table conversion procedure.

Use the [Table StartConversion\(\)](#) and [Table EndConversion\(\)](#) functions in the conversion process. The table conversion procedure reads each record in the original table and copies it to the new table. During this process, fields you added should be given default values, fields that won't be in the converted table aren't copied, and changed fields are copied from the placeholders in the old table to the correct field in the new table. Also, a new table index is automatically created for any new or changed keys.



The [Table StartConversion\(\)](#) and [Table EndConversion\(\)](#) functions can't be used with SQL Server.

Use the example for the [Table StartConversion\(\)](#) function as a template when you're writing your own table conversion scripts.

5. Determine how the table conversion script will be called.

In some cases you may want to call this script from a maintenance window in your application. Table conversion scripts can also be run as installation scripts during the update process, which is described in the next section.



In instances when several tables need to be converted, you may want to write a separate Dexterity application to perform necessary table conversions.

Converting SQL tables

Table conversion actions for SQL tables are typically performed using pass-through SQL. The appropriate SQL commands are issued by your application, and the work is performed on the server. Refer to [Chapter 49, "Pass-through SQL,"](#) for more information about using pass-through SQL from within your application.

Performing conversions on SQL tables can be complex. This documentation describes the basic steps necessary for the common table conversions. For specific details about SQL table conversions, or for more complex table conversions, consult Dexterity technical support. The remainder of this section describes how the common table conversions are performed for SQL tables.

Adding fields

To add a field to a table, add the field to the table definition in Dexterity. Use the ALTER TABLE statement in pass-through SQL to add the new column to the table. Be sure the name matches the physical name for the field you added to the table definition in Dexterity.

Use the [Table DropProcedures\(\)](#) function to remove the existing stored procedures for the table, and then use the [Table CreateProcedures\(\)](#) function to re-create them. You will need to grant access to the new auto-generated stored procedures. Refer to [Granting access to tables](#) on page 393 for information about granting access to tables and their auto-generated stored procedures.

Removing fields

To remove a field from a table, remove the field from the table definition in Dexterity. Use the ALTER TABLE statement in pass-through SQL to remove the column from the table.

Use the [Table DropProcedures\(\)](#) function to remove the existing stored procedures for the table, and then use the [Table CreateProcedures\(\)](#) function to re-create them. You will need to grant access to the new auto-generated stored procedures. Refer to [Granting access to tables](#) on page 393 for information about granting access to tables and their auto-generated stored procedures.

Adding, changing, or removing keys

If you add, change, or remove keys for a Dexterity table, you must re-create the auto-generated stored procedures for that table. With the previous version of the table definition, use the [Table DropProcedures\(\)](#) function to remove the existing stored procedures. Then use the [Table CreateProcedures\(\)](#) function with the updated table definition to create new auto-generated stored procedures for the table. You will need to grant access to the new auto-generated stored procedures. Refer to [Granting access to tables](#) on page 393 for information about granting access to tables and their auto-generated stored procedures.

You should also use pass-through SQL to drop the indexes for the table whose keys you are changing. Then use the [Table CreateIndexes\(\)](#) function to re-create the indexes for the table.

Changing a field's storage size

To change a field's storage size, change the data type associated with the field whose size is changing. Using pass-through SQL, execute the `sp_rename` command to rename the existing table to some appropriate placeholder name. Create the new table that contains the field with the new storage size. Use pass-through SQL to bulk copy the data from the previous table to the new table. Once the copy is complete, you can delete the previous table. You will need to grant access to the new table. Refer to [Granting access to tables](#) on page 393 for information about granting access to tables.

How to package an update

The extent of changes in your application determines the method you will use to package your update.

- If you're including only problem fixes and minor application changes, you can send your update in the form of a dictionary chunk. This process is described in the next section.
- If you're adding major new functionality to your application, you should consider having your customers install a new version and migrate data to it from their current version.

Creating an update dictionary chunk

If you're distributing only problem fixes and minor application changes, you can use a dictionary chunk to package your update. Perform the following steps to create an update dictionary chunk.

1. Keep an unmodified copy of the version of the application dictionary your customers currently have.

You must have this so the update chunk can be created properly.

2. Make your changes in another copy of the application dictionary.

Be sure that you're making changes to an application dictionary that is the same version as the application dictionary your customers currently have.



If you will be changing table definitions, be sure you've read [Changing table definitions](#) on page 499. It contains important information about how to convert existing data tables to work with the updated application.

3. Prepare your updated application dictionary as if you were going to be sending it to your customers.

Perform the steps you ordinarily would to prepare a dictionary for distribution, including synchronizing and compressing the dictionary.

4. Use Dexterity Utilities to compare the original application dictionary to the updated application dictionary.

Start Dexterity Utilities and open the original application dictionary (the dictionary from step 1) as a source dictionary. Open the updated application dictionary as an editable dictionary. Choose Compare from the Utilities menu; the Compare Dictionaries window will appear. Click OK to compare the two dictionaries.

5. Examine the two reports generated by the Compare Dictionaries utility.

The first report contains a list of the resources that are new or have been changed in the updated version of the application. You will use this list to determine which modules to include when you create the update chunk. The second report, named CHANGES.TXT, contains a list of the forms and reports that have changed in the updated dictionary.

6. Decide which type of update chunk you want to create.

There are two possibilities:

- You can create a chunk dictionary that contains all resources for the application, including the changes for the new version. This method has the advantage that the same dictionary chunk can be used to update existing installations of your application as well as install new versions of your application. This reduces the work required, because you have only one chunk to create.
- You can create a chunk that contains only the changes to the application. This chunk can be used only to update existing installations of your application.



We recommend that you use the first approach and create an update chunk that contains all of the application dictionary's resources.

Auto-Chunk To create an update chunk that contains all of the application's resources, use the same procedure you use when packaging the application. We recommend that you use the Auto-Chunk utility in Dexterity Utilities to create a single dictionary chunk. Optionally, you could also use the Create Chunk Dictionary utility to create several dictionary chunks. Refer to [Chapter 59, "Making Installation Files,"](#) for information about packaging procedures. Then continue to step 7.

Create Chunk Dictionary If you're creating a update chunk containing only changes, use the Create Chunk Dictionary utility to create an update dictionary chunk. In Dexterity Utilities, open your updated application dictionary as a source dictionary. Choose Create Chunk Dictionary from the Utilities menu on the tool bar. A window will appear asking whether you want to edit an existing dictionary chunk. Click No. A window will appear allowing you to specify the name of the new chunk.

The Create Chunk Dictionary window will appear. Add the header information for the chunk. Be sure the name you enter in the Dictionary field is exactly the same as the name of the application dictionary you're updating. Choose the same Module you did for your original installation. Also, set the Major, Minor, and Build Number fields to reflect the version and build number of the updated application.

Major, Minor, and Build Numbers The values you choose for the major version and minor version are significant. If the installation folder contains a dictionary for a previous version of your product dictionary, the dictionary chunk you are creating will unchunk only if the major version and minor version numbers in the chunk match those in the existing dictionary. If the version numbers don't match, an error indicating the issue will be written to the InstallErrors.txt file in the installation folder. The dictionary chunk won't be deleted.

If you have multiple dictionary chunks with the same major version and minor version values, the build number is used to determine the order in which the dictionary chunks are unchunked. The chunks with lower build number values are unchunked first.

If the installation folder contains a dictionary for which the major and minor version numbers match those of the dictionary chunk, the build number will be examined. If the build number of the chunk is the same or greater than the build number of the dictionary, the chunk will be unchunked. If the build number of the chunk is lower, it will not be unchunked, and a message indicating the issue will be written to the InstallErrors.txt file in the installation folder. The dictionary chunk will be deleted.

Table Conversions If your application must perform any table conversions, you may want to run table conversion procedures from an install script. Next, add the appropriate modules to the update chunk. Use the report from step 5 that lists the modules that contain new or changed resources. These are the modules you must add to the update chunk. Click OK to create the update chunk.

Continuing the Real Estate Sales Manager example, after the application was updated, the Compare Dictionaries utility was used to compare it to the previous version. The comparison report is shown in the following illustration.

In this illustration, boxes have been drawn around the module names. These are the modules you must include in the update chunk.

```

Time printed: 10/20/1995 10:31:15

Dictionary Comparison

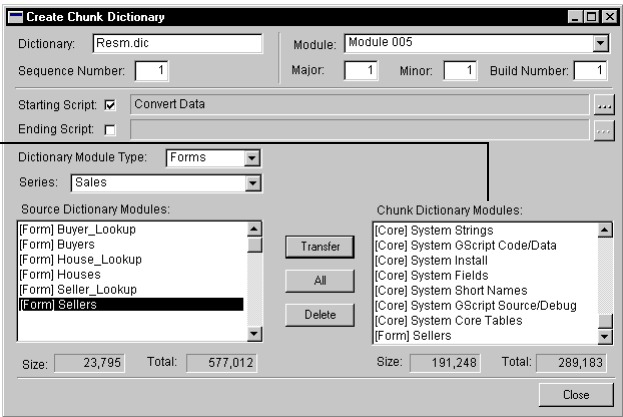
Source Dictionary      : HD1:RESM_V1.DIC
Compare Dictionary    : HD1:RESM_V2.DIC

* = New Resource or Name Change

[Core System Core 1]
Table
*      Seller_Data
      Seller_Data_OLD
*      Key
      Seller_Data_By_ID!
      Seller_Data_By_Name!
      Seller_Data_By_Initial Contact
*      Relation
      22005
[Core System Strings 2]
String
*      Convert Data
      Initial Contact:
[Core System GScript Code/Data 3]
Global Script Data
*      Convert Data
      Global Script Code
*      Convert Data
[Core System Data Types 6]
Data Type
*      Date
      STR45
[Core System Fields 7]
Field
*      Seller Name
      Initial Contact
*      String30
[Core System Short Names 8]
Field Short Name
*      Initial Contact
      String30
[Core System GScript Source/Debug 9]
Global Script Source
*      Convert Data
      Global Script Debug
*      Convert Data
[Core System Core Tables 1]
Core Table 3rd
      Local core table 1, 2
      Local core table 5, 2
      Local core table 6, 2
[Form Sellers 2003]
Form
      Sellers
Window
      Sellers
  
```

The information in the comparison report tells you which modules to include in the update chunk. Be sure the update chunk contains the **[Core] System Core Tables** and the **[Core] System Install** modules so the update chunk will merge properly. The following illustration shows the Create Chunk Dictionary window just before the update chunk for the Real Estate Sales Manager was created.

The dictionary comparison report indicates which modules to include in the update chunk.



7. Test the update chunk.

Move a copy of the update chunk so it is in the same location as the application dictionary for an installation of the previous version of your application. Start the application as you normally would. You should see a message asking you whether you want to include new code; click Yes. A message will be displayed indicating that new code is being merged into the application dictionary. When the application starts, check to determine whether the appropriate resources have been added and updated.

Updating forms and reports dictionaries

If your customers have been using the Modifier to modify forms or the Report Writer to modify reports, you need to ensure that the forms and reports dictionaries are updated properly. Otherwise, your application may not operate correctly.

The runtime engine has a built-in mechanism to ensure the forms and reports dictionaries being used are compatible with the application dictionary. When you add product information to an application dictionary, you supply a *compatibility ID* that uniquely identifies the version of your application.

When you launch your application, the runtime engine compares the compatibility ID in the application dictionary with the compatibility IDs in the forms dictionary and reports dictionary. If the compatibility IDs match exactly, items in the forms and reports dictionaries can be accessed. If the IDs don't match, the user won't be allowed to access items in the forms and reports dictionaries. If this occurs, the compatibility message you supplied when you added product information for the application dictionary will be displayed. You should provide information in this message describing how to update the forms and reports dictionaries for your application.

Updating forms dictionaries

During the updating process, resources in the forms dictionary are updated to make them compatible with the application dictionary. If the user has made any modifications to forms, as many of those changes as possible will be preserved. Forms dictionaries can be updated in one of the following ways:

- Using an update chunk to update your application. In the process of unchunking, the updated resources in your application are copied to the forms dictionary. Any errors that occur during the updating process will be logged in the `INSTALLERERRORS.TXT` file.
- Using the [`Dict_UpdateFormsDictionary\(\)`](#) function to update the forms dictionary. Typically, this function is run from another Dexterity-based application used to migrate an existing application to a new version. Any errors that occur during the updating process will be returned through the callback mechanism set up by the [`Dict_UpdateFormsDictionary\(\)`](#) function.

After the update operation is complete, the compatibility ID in the forms dictionary will be the same as the compatibility ID in the application dictionary. Then the runtime engine will be able to access resources from the forms dictionary.

Updating reports dictionaries

Updating the reports dictionary is a two-step process. Two steps are necessary because two types of changes can be made that affect reports. The reports themselves can be changed, and the database structure (tables, keys and so on) used by reports also can be changed.

Updating reports dictionary resources

The first step in updating the reports dictionary is similar to updating a forms dictionary. Resources in the reports dictionary are updated to make them compatible with the application dictionary. This process can be started in one of the following ways:

- Using an update chunk to update your application. In the process of unchunking, the updated resources in your application are copied to the reports dictionary. Any errors that occur during the updating process will be logged in the INSTALLERERRORS.TXT file.
- Using the [`Dict UpdateReportsDictionary\(\)`](#) function to update the reports dictionary. Typically, this function is run from another Dexterity-based application used to migrate an existing application to a new version. Any errors that occur during the updating process will be returned through the callback mechanism set up by the [`Dict UpdateReportsDictionary\(\)`](#) function.

After this portion of the update operation is complete, the compatibility ID in the reports dictionary will be the same as the compatibility ID in the application dictionary. Then the runtime engine will be able to access resources from the reports dictionary.

Updating reports for database changes

The second step in updating a reports dictionary is necessary if you have made database changes that can affect reports. For example, assume a field was removed from table A and added to table B. A report that looks in table A for the field won't be able to find it. The report needs to be updated to look in table B. The second part of the report update process performs these types of updates.

The information necessary to perform the second portion of the report update can't be derived from your application dictionary. You must supply a text file that contains the instructions describing how to handle each database change.

A portion of this input file is shown in the following illustration.

```
[ 'RESM 4.0' to 'RESM 5.0' ]

#The following tables were removed in version: RESM 5.0

#The following fields were removed in version: RESM 5.0
removed field 'SQL_Statements' = 22011.
removed field 'Picture' = 22023.
removed field 'Query_Results' = 22027.
removed field 'Progress_Bar_Status' = 22034.
removed field 'Houses Tree' = 22040.

#The following keys were modified or removed. Display an exception to
make the User aware of any potential sorting changes!
exception key_modified 'House_Data'.4
"House_Data.House_Data_By_Seller_First_Name has changed. Any reports
using this key may sort differently than before!".

# NOTE: If a key has been deleted from a table, move statements are
required for all keys greater in position than the one which was removed.
Reports reference table keys by position as opposed to resid. Hence, we
need to generate move key statements to adjust these changed key
positions!

# WARNING: The following fields and/or keys were left unmapped. All
fields and keys should either be moved or made obsolete.

#The following fields and/or keys will become obsolete:
# Deleted Field: 'House_Data'. 'Floors'.
# Deleted Key: 'House_Data'. 'House_Data_By_Seller_First_Name'.

#The following fields and/or keys will be moved:
```

Contact Dexterity Technical Support for information about how to create this input file for your integration.

The second part of the report update process can be started only by the [Dict_UpdateReportsDictionary\(\)](#) function. When you call this function, you can specify the input file that contains information about performing the second part of the report update. If you don't specify an input file, the second part of the report update performs a simple "cleanup" operation on the reports dictionary.



Because some "cleanup" tasks are performed, it's a good idea to run the second portion of the report update process, even if you haven't made any database changes that affect reports.

If you are using a chunk to package your application, you can call the [Dict_UpdateReportsDictionary\(\)](#) function from within an installation script that runs after the unchunking process. In this case, you won't supply the previous version of your dictionary, so this function will simply perform the second portion of the report update. The first portion will have already been performed as a part of the unchunking process.

Part 12: Utilities

Several utilities are available in Dexterity that are useful while you are developing your application. This portion of the documentation describes these utilities and explains when they should be used. The following utilities are described:

- [Chapter 61, “Synchronize Utility,”](#) describes the synchronize process for a dictionary.
- [Chapter 62, “Duplicate utility,”](#) explains how to duplicate resources in a dictionary.
- [Chapter 63, “Integration Reports,”](#) describes various reports available for applications that integrate with Microsoft Dynamics GP.

Chapter 61: Synchronize Utility

A synchronize operation should be performed on a dictionary under the following circumstances:

- A field's data type has changed
- A data type's control type has changed

A field's data type reference is stored with the window, table, key, report and global variable resources for faster processing. If the data type changes, synchronizing the dictionary will ensure all references to the data type are correct. The synchronize utility also recalculates the buffer sizes for tables and windows.

To synchronize a dictionary, simply choose Synchronize from the Utilities menu. A progress dialog will be displayed as the dictionary is synchronized.



For large dictionaries, the synchronize operation may take several minutes.

Chapter 62: Duplicate utility

The Duplicate utility allows you to duplicate various resources in the dictionary and save them under a new name. This utility is useful when creating new resources that are very similar to existing one. It is also used when table definitions are duplicated to prepare for table conversions, and when alternate versions of Microsoft Dynamics GP reports are created.

Information about the Duplicate utility is divided into the following sections:

- [*Procedure: Duplicating resources*](#)
- [*Additional resources duplicated*](#)

Procedure: Duplicating resources

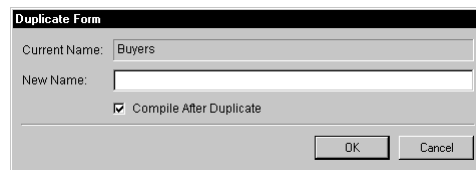
To duplicate a resource, complete the following procedure.

1. Select the resource to duplicate.

In the Resource Explorer, select the resource that you want to duplicate.

2. Open the Duplicate window.

Choose Duplicate from the Utilities menu, or click the Duplicate Resource Button on the Resource Explorer toolbar. The Duplicate window will be displayed.



3. Enter a name for the copied resource.

Enter a name for the new resource.

4. Choose whether to compile scripts.

If you are duplicating a resource that has scripts, such as a form or global procedure, mark the Compile After Duplication option to have the scripts for the new resource compiled.

5. Duplicate the resource.
- Click OK to duplicate the resource.

Additional resources duplicated

When duplicating resources, keep in mind that resources related to the resource you’re duplicating will also be duplicated. For example, when you duplicate a table resource, the keys and table relationships for the table are duplicated as well. The following table lists the additional resources that are duplicated when you duplicate a table, form or report.

Resource being duplicated	Additional resources duplicated
Fields	Field Physical Names (Short Names)
Tables	Table relationships Keys
Forms	Windows Local data types Local fields Forms, window and field scripts Form procedure scripts Form function scripts Form menus and scripts Form constants Commands Scrolling windows and scripts
Reports	Calculated fields Restrictions

When some of these related resources are duplicated, such as keys and field physical names, they must be given unique names. To automatically create unique names for these related resources, a number is appended to the name of the new resource.

Chapter 63: Integration Reports

The Integration Reports utility produces reports that describe applications that integrate with Microsoft Dynamics GP. These reports provide lists of the resources in your integrating application. They also list the main product resources used by your integrating application. Information about the Integration Reports utility is divided into the following sections:

- [*Procedure: Creating integration reports*](#)
- [*Creating a submission file*](#)

Procedure: Creating integration reports

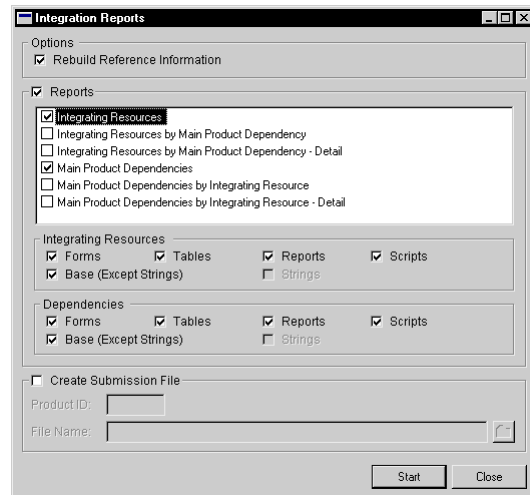
To create integration reports, complete the following procedure.

1. Open your development dictionary.

The development dictionary is the Dynamics.dic dictionary in which you have developed your integrating application.

2. Open the Integration Reports window.

Choose Integration Reports from the Utilities menu. The Integration Reports window will be displayed.



3. Indicate whether to rebuild the Reference Information table.

The information used for the integration reports comes from the Reference Information table. This table contains data that describes the dependencies among the resources in the dictionary.



This is the same table used for the Reference Information window.

To have the most up-to-date reports, mark the Rebuild Reference Information option. Be aware that rebuilding this table can take several minutes.

4. Indicate the reports to generate.

Mark the options that indicate the integration reports you want to generate. The following reports are available:

Integrating Resources This report lists all of the third-party resources in your integrating application.

Integrating Resources by Main Product Dependency This report lists the main product resources used by your application, along with the third-party resources that use them.

Integrating Resources by Main Product Dependency - Detail This report lists the main product resources used by your application, along with each occurrence of the third-party resources that use them. Items that are referenced by scripts are listed with the line number where the item is used.

Main Product Dependencies This report lists all of the resources in the main product dictionary that are used or referenced by your integrating application.

Main Product Dependencies by Integrating Resource This report lists all of the resources in your integrating application, along with the main product resources that they reference.

Main Product Dependencies by Integrating Resource - Detail This report lists all of the resources in your integrating application, along with every occurrence of the main product resources that they reference. Items that are referenced by scripts are listed with the line number where the item is used.

5. Indicate the resource types to include in the reports.

Mark the types of resources that you want to include in the reports that are generated. In most cases, you will want to mark all of the resource types.

6. Indicate whether to create a submission file.

If you want to create a submission file, mark the Create Submission File option. Submission files are described in the next section.



If you choose to create a Submission File, you will be required to mark the Rebuild Reference Information option.

7. Start the report generation process.

Click Start to start the report generation process. The Report Destination window will be displayed for each report being produced. Specify the desired destination for each report and click OK.

Creating a submission file

A submission File is a special text file that contains information about the main product resources used by an integrating application. When you mark the Create Submission File option, you need to specify the Product ID you use for your integrating application. You must also specify the location for the text file produced.

After you've created a submission file for your integrating product, you will send the file to Microsoft Business Solutions. We will add the contents of your submission file to a database that includes the submission files for other integrating applications. The information in the database is analyzed to learn what areas and specific resources integrating applications use. The database also allows us to notify developers affected when major changes are planned in a module.

Contact Dexterity Technical Support for information about sending a submission file for your integrating application.

Index

A

- About Box form, described 119
- About Dexterity, menu item 41
- access keys
 - for form-based menu items 168, 178
 - for form-based menus 167, 178
 - for push buttons 72
- active locking
 - allowing for tables 109, 385
 - for SQL tables 385
- Add button, how used in Dexterity 45
- adding fields
 - to a report layout 296
 - to a window layout 132
- adding records, *see* saving records
- addition operator, use in calculated fields 325
- additional footers
 - beginning a new report after 344
 - creating 343-345
 - described 341
 - illustration 342
 - position in report layout 342
 - printing after given number of records in body 343
 - relationship with additional headers 341
 - selecting field based upon 344
 - starting a new page after 344
 - suppressing 344, 345
 - use in reports 295
- additional headers
 - creating 343-345
 - described 341
 - illustration 342
 - position in report layout 342
 - relationship with additional footers 341
 - selecting field based upon 344
 - suppressing 344
 - use in reports 295
- Additional Headers and Footers, chapter 341-349
- Additional Resources, part 206-249
- alert messages
 - search the Dexterity online help for alert messages*
 - see also* messages
- align to bottom tool
 - using in report layout 298
 - using in window layout 130
- Align To Grid menu item 132
- align to left tool
 - using in report layout 298
 - using in window layout 130
- align to right tool
 - using in report layout 298
 - using in window layout 130
- align to top tool
 - using in report layout 298
 - using in window layout 130
- aligning
 - see also* formats
 - fields in reports 298
 - fields in windows 130
 - objects to the grid 132
- Alignment
 - drawn object property 142, 303
 - field property 140, 302
- alternate forms, checking into source
 - code control 467
- alternate image, for icons 234
- alternate reports, checking into source
 - code control 467
- analyzing script performance, *see* script profiler
- AND, *see* logical AND
- Appearance
 - drawn object property 142
 - field property 140
- application connectivity
 - COM (Component Object Model) 16
 - MAPI 16
- application development, chapter 21-30
- application dictionaries, *see* dictionaries
- application-level menus, described 165
- applications
 - basic components 11
 - developing 24
 - how to start creating 49
 - installing 485
 - integrating with Microsoft Dynamics GP 20, 22
 - multiuser 17
 - overview, illustration 13
 - packaging 484-511
 - stand-alone 21
 - starting 481
 - types of 21
 - updating 499-511
- arrange tools, described 130, 298
- arranging items, in report layout 298-299
- array fields
 - adding to a report layout 299
 - creating 90, 95
 - described 90, 93
 - elements of 90
 - example 93
 - illustration 90
 - size 90
- array indexes, setting for report fields 299
- ArrayIndex, field property 302
- arrow tool
 - using in report layout 296
 - using in window layout 129
- ask() function 221
- attached tables
 - described 120, 122
 - options 120, 122
- Auto Add Field option, for data types 75
- Auto-Chunk utility 487
- AutoComplete, field property 138
- AutoCopy, field property 138
- auto-generated stored procedures
 - described 391
 - disabling 109, 393
- auto-link tables
 - described 155
 - for windows 155
- AutoLinkTable, window property 137

automatic script naming, *see* names
 AutoOpen, window property 137
 AVERAGE, report restriction function 315
 Average, report display type 358

B

BackColor
 drawn object property 142, 303
 field property 140, 302
 window property 138
 Basics, part 32-45
 beta testing, *see* software testing
 big text tool 129
 bitmap images, as native pictures 213
 block size, for cursors 382
 boolean
 control type 55
 use in reports 304
 Border
 drawn object property 142
 field property 140
 buffers, *see also* table buffers
 build process
 automating 457
 described 456
 building an application, part 48-160
 building dictionaries with source code control 456
 built-in commands
 described 185
 table 185
 button drop lists
 control type 55
 static picture values 53
 static text values 53

C

calculated expression, *see* calculated fields
 Calculated Field Definition window 323
 calculated fields
 see also system-defined functions, user-defined functions
 adding fields to the expression 327
 calculated expressions 323
 chapter 323-339

calculated fields (*continued*)
 concatenating in 326
 conditional expressions 323
 creating, procedure 324-325
 dependencies 338
 described 323
 evaluation order 338
 hidden dependencies 338
 nesting system-defined functions 330
 operators used in 325-327
 result type restrictions 323
 using constants in 329
 using global variables in 329
 using legends in 328
 using other calculated fields in 327
 using report fields in 327
 using system-defined functions in 330
 using user-defined functions 330, 336
 Cancel, field property 138
 Cancel button, how used in Dexterity 45
 cascading menus, described 171
 CAT operator, use in calculated fields 326
 categories, for system colors 152
 centering text in a field, *see* formats
 change script, *see also* field change scripts
 characters per inch, for report text 306
 charts
 characteristics of control types 54
 common uses for static picture values 53
 common uses for static text values 53
 composite format options 86
 currency format options 85
 default form-based menu set 172
 Dexterity predefined constants 219
 drawn object properties 142, 303
 field properties 138-140, 302
 formats for currency fields 81
 formats for variable currency fields 81

charts (*continued*)
 inherited menu options 167
 numeric format options 85
 predefined menu actions 169
 report sections 294
 standard buttons in Dexterity 45
 string format options 86
 window properties 137-138
 check box, tool 129
 check boxes
 control type 56
 static text values 53
 use in reports 304
 Check In to Repository window 447
 check marks
 adding to a form-based menu item 173
 removing from a form-based menu item 173
 Check Out from Repository window 444
 CheckIn To Repository window 440
 checking in
 dictionaries 440
 source files 446
 source files, using generic provider 471
 checking out
 source files 442
 source files, using generic provider 471
 checklists, *see* procedures
 chunk dictionaries
 adding modules to 492
 attaching installation scripts 489, 491
 Create Chunk Dictionary window 490
 creating with Dexterity Utilities, procedure 486
 ending scripts 489, 491
 for updates 504
 modules to include 492
 not unchunking 494
 purpose 486
 starting scripts 489, 491
 troubleshooting 494

- chunk dictionaries (*continued*)
 - unchunking 494
 - unchunking with command line parameter 496
- chunks, *see* chunk dictionaries
- circle tool, using in report layout 297
- clean-up
 - for command-based menus 194
 - for toolbars 204
- Clear, menu item 40
- CloseBox, window property 137
- Clustered, key option 107
- clustered keys 107
- color properties
 - custom color support 153
 - system color support 149
- colors
 - custom colors for reports 307
 - for report items 307
 - system colors 149, 152
- COM, type libraries 227
- COM (Component Object Model), described 16
- combo box, tool 129
- combo boxes
 - control type 56
 - use in reports 304
- comma-delimited files, saving reports to 362
- command bars, *see* toolbars
- Command Definition window 181, 188
- command forms, described 184
- command line parameters, for runtime engine 496
- command lists, described 183, 184
- command-based menus
 - automatic clean-up 194
 - chapter 191-194
 - described 191
 - implementing 191
 - sample 191
 - script for adding 191
- commands
 - built-in commands 185
 - chapter 181-189
 - command form 184
 - controlling access 185
- commands (*continued*)
 - creating, procedure 188-189
 - described 181
 - disabling 185
 - display name 181
 - elements 181
 - enabling 185
 - hiding 185
 - images 183, 189
 - naming 181, 188
 - overview 181
 - ribbon options 183, 189
 - running 185
 - security 185
 - security form 182
 - shortcut 182, 188
 - showing 185
 - tags 184
 - toolbar options 183, 189
 - tooltips 182, 188
 - types 182, 189
 - use on menus 184
 - use on toolbars 185
 - using 184
- compare dictionaries report, illustration 507
- Compile, menu item 258
- Compile All, menu item 258
- Compile button 256
- Compile Dictionary button 257
- compiler errors, *search the Dexterity online help for Compiler messages*
- components of Dexterity, described 33
- composite formats
 - example 80, 86
 - format options 86
- composites
 - control type 57
 - format strings
 - described 80, 86
 - example 80, 86
 - formatting 80
- Comprehensive Index, described 5
- Compress Keys, key option 107
- concatenating, in reports 326
- conditional expression, *see* calculated fields
- connection pools
 - creating 378
 - described 378
- connections
 - described 377
 - for pass-through SQL 415
 - logging into a data source 375
 - maximum number 378
 - TCP/IP connection limits 378
- connectivity
 - for source code control 425
 - see* application connectivity
- Constant Definition window 218
- constants
 - chapter 217-221
 - character constants 220
 - creating, procedure 218, 219
 - described 217
 - Dexterity constants, *see* predefined constants
 - predefined constants
 - chart 219
 - example of use 221
 - using 220
 - script example 220
 - use 217
 - using in calculated fields 329
- Contents, menu item 41
- context menus
 - adding items 196
 - automatically-added items 196
 - built-in command list for 186, 195
 - chapter 195-199
 - closing 199
 - described 195
 - displaying 199
 - example 197
 - focus behavior 199
 - in Microsoft Dynamics GP 195
 - removing items 196
 - using 195
 - using in macros 199
 - when available 195
 - where used in applications 195
- Control Count
 - described 348
 - report display type 358

- ul style="list-style-type: none;">
- control types
 - assigning to data types 73
 - boolean 55
 - button drop list 55
 - check box 56
 - combo box 56
 - composite 57
 - currency 57
 - currency (variable) 58
 - date 59
 - descriptions 54-70
 - drop-down list 60
 - examples 54-70
 - horizontal list box 60
 - integer 61
 - list box 61
 - list view 62
 - long integer 62
 - multi-select list box 63
 - non-native list box 64
 - picture 65
 - progress indicator 65
 - push button 66
 - radio button 66
 - radio group 67
 - reference 67
 - string 68
 - text 68
 - time 69
 - tiny integer 69
 - tree view 70
 - visual switch 70
- ControlArea, window property 138
- controls, *see also* control types, window controls
- conventions, in documentation 7
- conversion procedures, *see* converting data tables
- converting data tables
 - conversion procedures 500
 - c-tree tables 500
 - overview 499
 - Pervasive.SQL tables 500
 - SQL tables 502
 - when required 499
- Copy, menu item 40
- copying resources 517
- Count, report display type 358
- counting
 - group items 348
 - groups 348
- Create Chunk Dictionary utility 489
- Create Chunk Dictionary window 490, 508
- Create Index, key option 107
- creating, *see* individual topics
- c-tree Plus database manager
 - described 28, 102
 - key options 106
- CUR_STR, calculated field function 331
- currency
 - control type 57
 - formats
 - format options 85
 - multiple format selector 81
 - tool 129
- currency (variable), control type 58
- cursors
 - block size 382
 - chapter 381-384
 - described 381
 - refreshing 383
 - stale data 383
- custom colors
 - described 153
 - for prompts 153
 - for report items 307
 - for scrolling windows 153
 - for zoom fields 153
- Cut, menu item 40
- D**
- Data, report display type 358
- data sources
 - chapter 369-373
 - configuring 370
 - connection pools 378
 - connections 377
 - described 369
 - logging into 375
 - login failures 376
 - login security 376
 - Login window 375
- Data Type Definition window 50, 73
- data types
 - see also* control types
 - assigning control types 73
 - assigning to fields 94
 - chapter 49-75
 - composites 51
 - control types 50
 - descriptions 54-70
 - examples 54-70
 - creating, procedure 73-75
 - described 27
 - elements 50
 - examples 27, 71
 - formats 51, 75
 - in database development 27
 - keyable length 50, 74
 - modifying, effects 51
 - naming 50, 73
 - overview 49
 - related items 75
 - selecting for fields 90
 - SQL equivalents 404
 - static values 51, 74
 - storage size 50, 54
 - using 71
- database development, overview 27-28
- database independence 18
- database managers
 - default 102
 - specifying 102
- database types
 - c-tree 102
 - described 28
 - Pervasive.SQL 102
 - selecting for tables 102, 113
 - SQL 102
 - supported by Dexterity 102, 113
- DataType, field property 138
- date tool
 - using in report layout 297
 - using in window layout 129
- dates
 - control type 59
 - use in reports 304
- DAY calculated field function 331
- DAY_NAME, calculated field function 331

- Debug menu, described 41
- debugger, *see* script debugger
- Default, field property 138
- default window position, *see* positioning windows
- DefaultDbClick, field property 138
- defaults file, *search the Dexterity online help for defaults file*
- Define Worksets window 259, 262
- Delete, menu item 257
- Delete button 256, how used in Dexterity 45
- delete row menu item 169
- deleting, resources when using source code control 461
- dependencies, among calculated fields 338
- Descending, key segment option 107
- designing, *see* individual topic
- development dictionary
 - checking into source code control 467
 - preparing for use 465
 - setting source file state 465
- development process
 - elements of development 24
 - for stand-alone applications 21
- development process for integrating applications
 - described 22
 - illustration 23
- DEX.DIC, described 11, 13
- DEX.INI file
 - packaging 495
 - specifying with command line parameter 496
- DEX_LOCK table
 - creating 388
 - described 387
 - role in SQL locking 387
 - table definition 387
- DEX_ROW_ID column, for SQL tables 385
- DEX_ROW_TS column 110
- DEX_SESSION table
 - creating 388
 - described 386
- DEX_SESSION table (*continued*)
 - role in SQL locking 386
 - table definition 386
- Dexterity
 - basics 32-45
 - creating dictionaries 34
 - described 12
 - editing dictionaries 35
 - features 17
 - getting started 33
 - launching 34
 - main menu, illustration 37
 - menu bar 38
 - new features 3
 - overview 10-30
 - starting 34
 - toolbar, described 38
 - tools 15
- Dexterity applications, *see* applications
- Dexterity constants, *see* constants
- Dexterity Programmer's Guide, described 5
- Dexterity Quick Start manual, described 5
- Dexterity Report Writer, *see* Report Writer
- Dexterity Stand-alone Application Guide, described 6
- Dexterity Utilities
 - Auto-Chunk utility 487
 - Create Chunk Dictionary utility 489
 - described 14
 - manual, described 5
 - using to create an update chunk 506
 - using to create chunk dictionaries 487, 489
- dialog boxes, *see also* communicating with the user, messages
- dictionaries
 - see also* applications, chunk dictionaries
 - building from source code control 456
 - checking in to source code control 440
- dictionaries (*continued*)
 - creating 34
 - described 11
 - editing 35
 - illustration 12
 - location in launch file 480
 - overview 12
 - synchronizing 515
 - updating from source code control 453
- dictionary chunks, *see* chunk
- dictionaries
- dictionary location IDs, in launch files 480
- Direction, field property 140
- DisableLookup, field property 139
- disabling, form-based menu items 173
- display names
 - for commands 181
 - for tables 101, 112
- display types
 - for reports 357
 - use in report sections 360
- DisplayName, report property 301
- DisplayType, field property 302
- distributed processing, described 18
- divider tool, using in report layout 297
- division operator, use in calculated fields 326
- documentation
 - Comprehensive Index 5
 - Dexterity Programmer's Guide 5
 - Dexterity Quick Start manual 5
 - Dexterity Stand-alone Application Guide 6
 - Dexterity Utilities manual 5
 - Function Library Reference 5
 - Integration Guide 5
 - Online help for Dexterity, described 5
 - organization 4
 - SanScript Reference manual 5
 - symbols and conventions 7
- DOW, calculated field function 331
- Drawing Options window 306
- drawn objects, properties, list 142, 303
- drop-down list, tool 129

- drop-down lists
 - control type 60
 - static text values 53
 - use in reports 304
- DropIndicator, field property 140
- DropPosX, field property 140
- DropPoxY, field property 140
- duplicate records, associated key
 - option 106
- Duplicate Resource button 256
- duplicate utility
 - additional resources duplicated 518
 - chapter 517-518
- Duplicate window 517
- Duplicates, key option 106
- Dynamic Link Libraries, *see* DLLs

E

- Edit menu, described 40
- Edit Source File Names window 438, 463
- Editable, field property 139
- EditMode, field property 139
- elements of a resource, *see* individual resource
- elements of arrays
 - described 90
 - illustration 90
 - specifying 299
- ellipses, in form-based menu items 168
- enabling, form-based menu items 173
- encrypted fields
 - for SQL Server 399
 - in tables 116
- ending scripts, for a chunk dictionary 489, 491
- EndTransaction, field property 139
- equal joins, described 243
- equality operator, use in calculated fields 326
- error handling, for SQL 395
- error messages
 - search the Dexterity online help for alert messages*
 - see* messages
- error statement 208

- evaluate after, setting for calculated fields 339
- evaluation order, for calculated fields 338
- Exchange Web Services, described 16
- EXISTS, report restriction function 315
- Exit, menu item 39
- Explorer menu, described 40, 257
- Export Dictionary to Text Files, menu item 258
- Export to Text File, menu item 258
- Export to Text File button 257
- exporting data with reports 362
- extended composites, *see also* composites

F

- Fetch from Repository window 452
- fetching, source files 451
- Field, field property 139, 302
- field, context menus 195
- Field Definition window 89, 94
- Field Descriptions, menu item 39
- field properties, list 138-140, 302
- FieldID, field property 139, 302
- fields
 - see also* global fields, local fields, report fields, table fields, window fields
 - adding to reports 296, 299
 - adding to tables 104, 113
 - adding to the calculated field expression 327
 - adding to windows 132
 - array fields 90, 95
 - described 93
 - example 93
 - as global variables 92
 - assigning data types 94
 - creating, procedure 94-95
 - described 27, 97
 - elements 89
 - encrypting 116
 - examples 91
 - formatting 77, 355
 - hiding on report layout 354
 - in database development 27

- fields (*continued*)
 - in graphics reports, applying drawing options 306
 - items related to 95
 - linking prompts
 - described 145, 156
 - illustration 145, 156
 - lookup form 91, 95
 - modifying data types 51
 - naming 90, 94
 - naming restrictions in SQL 402
 - object properties 138, 302
 - overview 89
 - physical names 90, 94
 - positioning in windows 155
 - properties, list 138, 302
 - required fields 149
 - resizing 146
 - selecting a data type 90
 - tab sequence 143, 156
 - table fields 91, 104
 - using 71, 91
 - visual properties 140
 - window fields 91
- FieldType, field property 139, 302
- File menu, described 38
- files
 - see also* tables
 - sending reports to 362
- Find, menu item 40
- Find button 257
- finding items in dictionaries 263
- firewall settings, for source code control 430
- FIRST, report restriction function 316
- First Item In Group, report display type 358
- First Occurrence Only, report display type 359
- Font
 - drawn object property 142, 303
 - field property 140, 302
- FontBold
 - drawn object property 142, 303
 - field property 140, 302
- FontColor
 - drawn object property 142, 303

- FontColor (*continued*)
 - field property 140, 302
- FontItalic
 - drawn object property 142, 303
 - field property 140, 302
- fonts
 - for graphics reports 306
 - for text reports 307
 - generic fonts for reports 306
 - specifying color for reports 306
 - specifying size for reports 306
- FontSize
 - drawn object property 303
 - field property 302
- FontUnderline
 - drawn object property 142, 303
 - field property 140, 302
- Footer Options window 343
- footers, for groups 347
- FORALL, report restriction function 317
- Form Constant Definition window 219
- form constants
 - creating 219
 - described 218
 - script example 220
- Form Definition window 119, 122
- form menus
 - creating 123
 - described 166
 - when displayed 176
 - where displayed 176
- form series, assigning to forms 120, 122
- Format, field property 302
- Format Definition window 78, 84
- format field, described 80
- format linking mode 80
- format strings
 - composite formats 80, 86
 - described 79
 - example 79, 80, 86
 - string formats 79
 - using to add static elements 79
- FormatField, field property 302
- FormatFieldSrc, field property 302
- formats
 - aligning field information 78
- formats (*continued*)
 - applying to a data type 51, 75
 - applying to report fields 355
 - chapter 77-86
 - composite formats 80, 86
 - creating, procedure 84-86
 - currency, format options 85
 - elements 78
 - format strings 79
 - items related to 86
 - linking to format fields 80
 - multiple format selector 80
 - naming 78, 84
 - numeric, format options 85
 - options 78, 85
 - overview 77
 - string formats 79, 86
- form-based menu items
 - access keys 168, 178
 - changing order 179
 - checking and unchecking 173
 - controlling content 174
 - disabling 173
 - ellipses in name 168, 178
 - enabling 173
 - name 168, 178
 - position 170
 - predefined actions 169, 179
 - pulling focus when selected 170, 179
 - scripts 170, 179
 - shortcut keys 169
- form-based menus
 - access keys 167, 178
 - accessing from scripts 174
 - application-level menus 165
 - cascading menus 171
 - changing order of menus 177
 - chapter 165-180
 - controlling content 174
 - creating, procedure 177-180
 - default menu set 172
 - elements 166
 - form-level
 - described 166
 - when displayed 176
 - where displayed 176
- form-based menus (*continued*)
 - inherited options, chart 167
 - naming
 - display name 166, 177
 - menu name 166, 177
 - overview 165
 - separator lines 168
 - to indicate options 173
 - to initiate actions 173
 - types of 165
 - using 173
- forms
 - attached tables
 - options 120, 123
 - purpose 120, 122
 - chapter 117-124
 - creating menus 123
 - creating windows 124
 - creating, procedure 122-124
 - described 24, 118
 - design basics 24
 - elements 119
 - illustration 25, 118
 - items related to 124
 - main window
 - described 124
 - specifying 124
 - naming 119
 - overview 24, 118
 - purpose 118
 - series 120, 122
 - updating modified forms 509
- forms dictionaries
 - location in launch file 480
 - updating 508, 509
- FREQUENCY, report restriction function 318
- Function Library Reference, described 5
- functions
 - user-defined, *see* user-defined functions

G

- Generate Help Files, menu item 39
- Generate Resource Reports, menu item 39

generic fonts, for reports 306
 generic provider
 chapter 469-471
 checking in new dictionary 470
 checking in source files 471
 checking out source files 471
 creating a project 470
 described 469
 project location 469
 getmsg() function 207
 Getting Started with Dexterity, chapter 33-35
 global constants
 creating 218
 described 217
 script example 220
 global fields
 see also fields
 adding to a window
 illustration 133
 procedure 133
 as table fields 104
 assigning data types 94
 described 89, 132
 items related to 95
 naming 94
 physical names 94
 global scripts, *see* global procedures, procedures
 global variables
 chapter 223-224
 creating, procedure 224
 described 92
 example 223
 items related to 224
 overview 223
 using in calculated fields 329
 using in scripts 223
 graphics
 adding to windows 155
 cross-platform issues 144
 in Windows 144
 using in layout window 144
 graphics reports
 see also reports
 aligning text items 308
 applying drawing options 306

graphics reports (*continued*)
 described 275
 tools specific to, described 296
 greater than operator, use in calculated fields 326
 greater than or equal to operator, use in calculated fields 326
 grid
 described 132
 for window layout 132
 grid spacing, setting 132
 Grid to Back menu item 132
 groups
 counting 348
 counting items in 348
 creating in reports using headers and footers 341
 described 341, 345
 footers 347
 headers 347
 sorting information for 346

H

Header Options window 343
 headers, for groups 347
 help files, included with Dexterity 33
 Help menu, described 41
 HelpContextID
 field property 139
 window property 137
 hidden dependencies, among
 calculated fields 338
 hierarchical menus, *see* cascading menus
 highest value of a report field, *see* Maximum report field type
 highest value of a report restriction
 expression, *see* MAXIMUM report restriction expression
 horizontal list box tool 129
 horizontal list boxes, control type 60
 How to Use Help, menu item 41
 HTML, saving reports to 362
 Hyperspace, field property 139

I

Icon Definition window 233, 234

icons
 alternate image 234
 chapter 233-235
 creating an icon resource,
 procedure 234-235
 described 233
 elements 233
 naming 233
 identity columns, for SQL tables 385
 Ignore Case, key segment option 108
 Import from Text File, menu item 258
 Import from Text File button 257
 Import utility, described 16
 index file
 creating 456
 described 455
 updating 456
 indexes, creating for tables 107
 IndicatorColor, field property 140
 inequality operator, use in calculated fields 326
 Inherited Menu Options window 167
 Insert button, how used in Dexterity 45
 insert row menu item 169
 installation
 see also chunk dictionaries,
 installation
 additional components 495
 assembling necessary components 493
 files, chapter 485-497
 overview 485
 problems 494
 scripts 485
 testing 493
 troubleshooting 494
 Windows Data Access
 Components (Windows DAC)
 495
 installation scripts
 see also starting and ending scripts
 example 485
 installing SQL Server 369
 INT_STR, calculated field function 331
 integer, tool 129
 integers, control type 61

- integrating applications
 - illustration 22
 - overview 22
 - using Microsoft Dynamics GP
 - features 22
 - using source code control 465
- integration
 - overview 20
 - with Microsoft Dynamics GP 20
- Integration Guide, described 5
- integration reports
 - chapter 519-521
 - described 519
 - submission file 521
 - types of 520
- Integration Reports window 519
- interface design
 - forms 24
 - overview 24-26
 - windows 25
- interface for Dexterity, described 37
- internal windows, described 127
- internationalization, of applications 225
- Introduction to Source Code Control, chapter 421-424
- invisible fields, showing in layout 131

J

- joins, for virtual tables 243

K

- Key Definition window 105, 114
- key options
 - for c-tree 106
 - for Pervasive.SQL 106
 - for SQL 107
- key segments
 - described 105
 - options 107
- keyable length
 - defining for data types 50, 74
 - described 50, 74
- keys
 - adding segments 114
 - creating for a table 105, 113
 - creating indexes for 107
 - described 99, 105, 113

- keys (*continued*)
 - example 105
 - key segments, described 105
 - multisegment keys, described 105
 - naming 114
 - options 106, 114
 - primary key, for SQL tables 107

L

- LAST, report restriction function 318
- Last Occurrence, report display type 359
- launch files
 - chapter 479-482
 - contents 480
 - creating 479
 - creating an empty launch file 493
 - dictionary location IDs 480
 - dictionary locations 480
 - illustration 480
 - launch file ID 480
 - location of application dictionary 480
 - overview 479
 - product ID 480
 - product information 475
 - product name 480
 - purpose 479
 - specifying with command line parameter 496
 - starting applications with 481
 - Workstation2 setting 481
- launching
 - see also* starting
 - Dexterity 34
- layout area
 - described 131
 - for windows 131
- layout grid, *see* grid
- layout tools, described 129
- layout windows
 - described 26, 43
 - for reports 293
 - for windows 128
- left outer joins, described 243
- left-justifying text in a field, *see* formats

legends

- chapter 351-352
- use in run report statement 361
- using in calculated fields 328
- less than operator, use in calculated fields 327
- less than or equal to operator, use in calculated fields 327
- LFT_STR, calculated field function 331
- libraries
 - adding a reference, procedure 228-229
 - chapter 227-232
 - described 227
 - icons in 233
 - types of 227
- Library Definition window 228
- light bulb symbol 7
- line tool
 - using in report layout 297
 - using in window layout 129
- LineColor, drawn object property 142, 303
- LineSize, drawn object property 142, 303
- LinkedFormat, field property 139
- LinkedLookup, field property 139
- LinkedPrompt, field property 139
- linking fields to format fields 80
- linking prompts to fields
 - described 145, 156
 - illustration 145, 156
- list box, tool 129
- list boxes
 - control type 61
 - static text values 53
 - use in reports 304
- list view, control type 62
- list view tool 129
- LNG_STR, calculated field function 332
- local area network
 - illustration 17
 - using with Dexterity applications 17
- Local Field Definition window 134

local fields
see also fields
 adding to a window, procedure 134
 created with prototyping tools 135
 creating 134
 described 89, 132
 locations, for tables 111
 Lock in Repository window 449
 locking
 active 385
 chapter 385-390
 source files 448
 locks, stranded
 clearing 389
 described 389
 logger, *see* script logger
 logging into a data source 375
 logical AND operator, use in calculated fields 326
 logical NOT operator, use in calculated fields 327
 logical OR operator, use in calculated fields 326
 login failures 376
 login macros, recording 496
 Login window
 displaying 376
 illustration 375
 Login, menu item 39
 logins, macro for 496
 Logins and Connections, chapter 375-379
 Logout, menu item 39
 long integers, control type 62
 Lookup, menu item 41
 lookup buttons, how used in Dexterity 45
 lookup forms, for global fields 91, 95
 lowest value of a report field, *see* Minimum report field type
 lowest value of a report restriction expression, *see* MINIMUM report restriction expression

M

Macro menu, described 40

macro system, described 19
 macros
 search the Dexterity online help for macro system
 automatic dictionary build process 457
 logging into an application 496
 starting from command line 496
 mail connectivity
 mailing reports 364
 overview 364
 mail enabling, described 16
 main menu
 chapter 163-164
 for Dexterity 37
 Main Menu form, described 119
 main window for a form
 described 124
 specifying 124
 MainTable, report property 301
 MainTableKey, report property 301
 MainTableName, report property 301
 Maintaining the Repository, chapter 461-464
 Making Installation Files, chapter 485-497
 manuals, *see* documentation
 MAPI
 described 16
 mailing reports 364
 using with Dexterity reports 364
 margin notes 7
 matching patterns in report restrictions 320
 MAXIMUM, report restriction function 319
 Maximum, report display type 359
 MaxRecords, report property 301
 member tables, for virtual tables 240
 memory-based tables, described 102
 Menu Definition window 166, 177
 menus
 command-based menus 191
 context menus 195
 creating, for forms 123

menus in Dexterity
 search the Dexterity online help for menus, in Dexterity
 Debug menu 41
 Edit menu 40
 Explorer menu 40, 257
 File menu 38
 Help menu 41
 Macro menu 40
 Utilities menu 40
 Windows menu 41
 messages
 search the Dexterity online help for alert messages
 see also communicating with the user
 chapter 207-210
 creating, procedure 210
 described 207
 items related to 210
 overview 207
 product names in 209
 substituting items into 208
 use 208
 viewing a list of 207
 Messages window 210
 metafiles
 as native pictures 213
 as pictures 211
 Microsoft Dynamics GP
 integrating applications 20, 22
 integration features 22
 Microsoft SQL Server
 see also SQL
 client/server architecture 18
 described 18
 MINIMUM, report restriction function 319
 Minimum, report display type 359
 modal dialogs, described 127
 modal windows, described 154
 Modifiable, key option 106
 modifiable keys, associated key option 106
 Modifier
 described 15
 updating modified forms 509

- modifying, reports 510
- Modifying Fields, chapter 353-360
- modules, adding to chunk dictionaries 492
- MONTH, calculated field function 332
- MONTH_NAME, calculated field function 332
- multidictionary
 - described 20
 - illustration 22
 - integrating process 22
 - using Microsoft Dynamics GP features 22
- multiplatform, issues for graphics 144
- multiple format selector
 - described 80, 355
 - for currency fields 81
 - for string fields 82
 - using with report fields 355
 - using with window fields 80
- multiplication operator, use in
 - calculated fields 326
- multisegment keys 105
- multi-select list box
 - control type 63
 - tool 129
 - use in reports 304
- multiuser processing, described 17

N

- Name
 - report property 301
 - window property 137
- names
 - see also* simple names and qualified names
 - commands 181, 188
 - data types 50, 73
 - fields 90, 94
 - formats 78, 84
 - form-based menus 166, 177
 - forms 119
 - global fields 90, 94
 - icons 233
 - local fields 134
 - tables 101, 112
 - windows 126

- Native Picture Definition window 214
- native pictures
 - as static picture values 53
 - blending with window background 150
 - control types used with 211
 - creating, procedure 214
 - described 53, 211
 - examples 213
 - for commands 183
 - synchronizing, procedure 215
 - using system colors 150
- Navigation, part 162-204
- network operation, described 17
- New, menu item 257
- New button 256, how used in Dexterity 45
- New Dictionary, menu item 38
- new features in Dexterity 3
- New Resource button 256
- new symbol 7
- New Workset window 260
- No Combining, key segment option 108
- No Length Byte, key segment option 108
- non-native list boxes, control type 64
- NOT, *see* logical NOT
- numeric formats, format options 85

O

- object properties, for fields 138, 302
- ODBC data sources, configuring 370
- OK button, how used in Dexterity 45
- one-to-many relationship, described 241, 279
- one-to-one relationship, described 241, 279
- online help
 - for Dexterity 5
 - for Dexterity-based applications 19
- Online Manuals, menu item 41
- Open, menu item 257
- Open button 256, how used in Dexterity 45

- Open Database Connectivity (ODBC), described 18
- Open Dictionary, menu item 38
- Open Script window 158
- opening position for windows, *see* positioning windows
- operators
 - parentheses, use in calculated fields 325
 - used in calculated fields 325-327
- Optimistic Concurrency Control, described 17
- Options, menu item 40
- OR, *see* logical OR
- Orientation, report property 301
- orientation of toolbars 131
- overriding, system colors 153
- Overview of Dexterity, part 10-30

P

- Packaging Applications, part 484-511
- packaging applications
 - see also* installing
 - described 485
 - updates 504
- page footers, use in reports 295
- page headers, use in reports 295
- page number tool, using in report layout 297
- palette windows, described 127, 155
- parentheses, use in calculated fields 325
- passing parameters, *see* parameters
- pass-through SQL
 - chapter 415-417
 - connections 415
 - executing statements 416
 - results sets 416
 - specifying a database 417
- Password, field property 139
- Paste, menu item 40
- pathname support, for Dexterity applications 19
- pathnames, for SQL tables 395
- Pattern
 - drawn object property 142, 303
 - field property 140, 302

- PatternColor
 - drawn object property 142, 303
 - field property 140, 302
- PatternSelect, field property 140
- PDF attachments, mailing reports as 364
- Pervasive.SQL database manager
 - described 28, 102
 - key options 106
- physical names
 - for global fields 90, 94
 - for tables 101, 112
- Picture Definition window 144, 212
- picture library
 - adding pictures to 212, 297
 - using pictures from 144, 212-213
- picture tool
 - using in report layout 297
 - using in window layout 129
- pictures, control type 65
- pictures (resource)
 - adding to picture library 212, 297
 - adding to reports 297
 - adding to windows 212
 - as static picture values 53
 - blending with window
 - background 150
 - described 53, 211
 - example 211
 - native, *see* native pictures
 - using system colors 150
- Pictures and Native Pictures, chapter 211-215
- port number, specifying for source code control 431
- port used, Dexterity Source Code Control Server service 430
- positioning
 - fields in reports 296
 - fields in windows 155
 - windows 145, 156
- Position-Left
 - drawn object property 142, 303
 - field property 140, 302
 - window property 138
- Position-Top
 - drawn object property 142, 303
- Position-Top (*continued*)
 - field property 140, 303
 - window property 138
- POWER_10, calculated field function 332
- predefined actions, for form-based menu items, chart 169, 179
- predefined constants
 - chart 219
 - script example 221
 - using 219
- preprinted forms, for reports 290
- prerequisites, for learning Dexterity 6
- previewing, windows 142
- Previous Occurrence, report display type 359
- Primary, key option 107
- primary key, for SQL tables 107
- primary windows, described 127
- Print Report Definition window 291
- Print Setup, menu item 39
- printing reports 362
- privileges, for SQL tables 393
- procedures
 - Adding a library reference 228-229
 - Adding a picture to the picture library 212
 - Adding global fields to a window 133
 - Adding local fields to a window 134
 - Checking in a new dictionary 440-441
 - Checking in multiple source files 447
 - Checking in one source file 446-447
 - Checking out multiple source files 444
 - Checking out one source file 443-444
 - Configuring ODBC data sources 370-373
 - Creating a calculated field 324-325
 - Creating a command 188-189
 - Creating a prototype procedure 409
- procedures (*continued*)
 - Creating a resource library 229-232
 - Creating a virtual table 246-249
 - Creating additional footers 343-345
 - Creating additional headers 343-345
 - Creating an icon resource 234-235
 - Creating data types 73-75
 - Creating fields 94-95
 - Creating formats 84-86
 - Creating form-based menus 177-180
 - Creating form-level constants 219
 - Creating forms 122-124
 - Creating global constants 218
 - Creating global variables 224
 - Creating integration reports 519-521
 - Creating messages 210
 - Creating native pictures 214
 - Creating tables 112-116
 - Creating windows 154-156
 - Creating worksets 259
 - Defining a new report 285-291
 - Defining a table group 237-238
 - Defining a table relationship 280-283
 - Defining report restrictions 313-314
 - Duplicating resources 517
 - Fetching multiple source files 452
 - Fetching one source file 451-452
 - Locking source files 448-449, 449
 - Modifying a string 226
 - Preparing the development dictionary 465-466
 - Setting tab sequence 143
 - Synchronizing native pictures 215
 - Unlocking source files 449-450, 450-451
 - Using a picture from the picture library 212-213
 - Using Find 264-265
 - Using legends 351
 - Using Replace 265-266
 - Using the picture library 144

- procedures (resource)
 - anonymous parameters, *see* anonymous tables and fields
 - for table conversion 500
 - form level, *see* form procedures
 - prototype procedures 408
 - SQLException 396
- Process Monitor, menu item 39
- Process Server
 - described 18
 - illustration 18
- product IDs, in a launch file 480
- product information
 - adding with Dexterity 477
 - chapter 475-478
 - elements of 475
 - example 478
 - overview 475
 - to create a launch file 475
- Product Information window 477
- product names, using in messages 209
- product support, for Dexterity 8
- profiler, *see* script profiler
- programs, *see* dictionaries
- progress indicator
 - control type 65
 - tool 129
- prompts
 - linking to fields 145, 156
 - unlinking 145
- properties
 - for drawn objects 142, 303
 - for fields 138, 302
 - for windows 137
 - setting 136, 301
- Properties window
 - described 44, 126
 - illustration 126
 - using 136, 300
- prototype procedure, for stored
 - procedures 408, 409
- prototyping tools
 - creating local fields 135
 - described 129
 - using 135
- providers
 - described 422
- providers (*continued*)
 - for source code control 423
- push buttons
 - access keys 72
 - control type 66
 - data type 71
 - example 71
 - native pictures for 213
 - static picture values 53
 - static text values 53
 - tool 129
 - using, illustration 71
- R**
- radio buttons
 - control type 66
 - data types 72
 - static text values 53
 - tool 129
 - using 72
- radio groups
 - control type 67
 - data types 72
 - tool 129
 - use in reports 305
 - using 72
- records
 - allowing active locking 109
 - described 97
 - fixed-length portion 104
 - size
 - calculated 105
 - of tables 104
 - variable-length portion 105
- rectangle tool, using in report layout 297
- Reference By, menu item 258
- Reference Information button 257
- references, control type 67
- Refers To, menu item 258
- Refresh, menu item 258
- relations, *see* table relationships
- Relationship Definition window 248
- relationships, for virtual tables 241
- Remove button, how used in Dexterity 45
- renaming, resources when using
 - source code control 462
- replacement markers
 - described 208
 - script example 208
- replacing items in dictionaries 263
- Report Definition, chapter 285-291
- report definition, described 285
- Report Definition window 285
- Report Destination button 257
- report display types, use in report sections 360
- Report Field Options window 353
- report fields
 - use in calculated fields 327
 - use in report sections 360
- report footers, use in reports 295
- report headers, use in reports 295
- Report Layout, chapter 293-308
- Report Layout window 293
- Report Restriction Definition window 313
- Report Table Relationships window 287
- Report Writer
 - described 15, 275
 - Dexterity and Runtime versions, compared 276
 - updating modified reports 510
- Report Writer Overview, chapter 275-278
- ReportID, report property 301
- reports
 - see also* graphics reports, text reports
 - adding fields 296, 299
 - adding pictures to 297
 - adding tables to 286
 - additional footers 295
 - additional headers 295
 - aligning fields 298
 - arranging items in layout area, described 298-299
 - body, described 295
 - changing field display types 357
 - database changes 510
 - defining, procedure 285-291

- reports (*continued*)
 - described 29
 - destinations 362
 - empty fields, hiding 354
 - exporting to a file 362
 - field characteristics 304
 - footer printing options 290
 - format options 288
 - format required for mailing 364
 - formatting fields 355
 - generating, overview 29
 - generic fonts 306
 - header printing options 290
 - hiding fields in layout 354
 - illustration 29
 - layout window 29
 - described 293
 - illustration 293
 - sections 294
 - legends 351-352
 - limiting the records included 287
 - mailing 364
 - main table for report 286
 - modifying 510
 - page footers, described 295
 - page headers, described 295
 - part 274-365
 - planning 277
 - positioning items in 299
 - printing 362
 - on preprinted forms 290
 - options 290
 - printing report definition 291
 - report footers, described 295
 - report headers, described 295
 - report sections, chart 294
 - restrictions
 - adding at runtime 361
 - defining 313-314
 - described 313
 - restriction functions 315-322
 - secondary tables 281
 - sending to a printer 362
 - sending to the screen 362
 - series 286
 - sizing items in 299
- reports (*continued*)
 - sorting
 - defining sort order at runtime 361
 - using a sorting definition 310
 - using main table key 309
 - table relationships
 - defining 280-283
 - described 279
 - illustration 280
 - temporary tables 364
 - text report options 289
 - tile tools used in layout 298
 - totaling and subtotaling 349
 - types, described 275
 - updating modified reports 510
 - using report footers 291
 - viewing
 - at runtime 361
 - in tools mode 308
 - virtual tables 245
- reports dictionaries
 - location in launch file 480
 - updating 508, 510
- repository
 - building a dictionary 456
 - checking in a new dictionary 440
 - checking in source files 446
 - checking out source files 442
 - configuring 428
 - deleting resources 461
 - described 422
 - examining 441
 - fetching source files 451
 - index file 455
 - labels for source files 464
 - locking source files 448
 - maintaining 461
 - renaming resources 462
 - unlocking source files 449
 - updating a dictionary 453
 - updating source file states 464
 - validating connection to 432
 - viewing version information 459
- Required, field property 139
- required fields, described 149
- Resizable, window property 137
- Resize-Horizontal
 - drawn object property 142
 - field property 140
- Resize-Vertical
 - drawn object property 142
 - field property 141
- resizing windows
 - automatic resizing 146
 - described 146
 - horizontal resize properties 147
 - per field resizing 146
 - vertical resize properties 148
- resource definition windows,
 - described 43
- Resource Descriptions, described 16
- Resource Explorer
 - buttons 256
 - chapter 253-258
 - described 42, 253
 - examining source code control repository 441
 - menu item 39
 - refreshing view 258
 - worksets 259
- Resource Explorer window 42
- Resource Find window 263
- resource IDs, maintaining in source
 - code control 455
- resource libraries
 - creating with Visual Studio 229
 - creating, procedure 229-232
 - described 227
 - icons in 233
- resource names, *see* names
- resources
 - adding to worksets 260
 - changing when using source code control 445
 - copying 517
 - creating when using source code control 445
 - deleting when using source code control 461
 - duplicating 517
 - removing from worksets 261
 - renaming when using source code control 462

- resources (*continued*)
 - worksets 260
 - Resources to Update window 455
 - restriction functions
 - see also* specific function
 - described 315
 - restrictions
 - adding to reports 313
 - adding to reports at runtime 361
 - chapter 313-322
 - results sets, for pass-through SQL 416
 - revision management, described 422
 - RGT_STR, calculated field function 332
 - ribbons, command display options 183, 189
 - right-click, *see* context menus
 - right-justifying text in a field, *see* formats
 - rounded rectangle tool, using in report layout 297
 - run report statement
 - destination clause 362
 - legends clause 361
 - running applications, *see* test mode or starting an application
 - Running Sum, report display type 359
 - running sums for reports 349
 - runtime engine
 - command line parameters 496
 - described 11, 13
 - purpose 13
 - renaming 13
 - used with Microsoft Dynamics GP 13
 - Runtime Report Writer, *see* Report Writer
 - Runtime Script Editor, described 158
 - runtime tools, *see* tools
- S**
- sanScript
 - search the Dexterity online help for individual commands*
 - compiler, *see* compiling
 - described 30
 - SanScript Reference manual, described 5
 - SavedOnRestart, field property 139
 - Scaling, field property 141
 - screens, *see* windows
 - script debugger, described 159
 - script logger, described 159
 - script lookup mode, described 159
 - script performance, *see* script profiler
 - script profiler, described 159
 - scripts
 - attaching, to form-based menu items 170
 - debugging 159
 - editing at runtime 158, 159
 - logging 159
 - overview 30
 - profiling 159
 - searching 263
 - that affect form-based menus 174
 - ScrollBars, field property 139
 - scrolling window tool, using in window layout 129
 - scrolling windows
 - context menus 195
 - virtual tables 245
 - Search and Replace, chapter 263-266
 - Search for Help On, menu item 41
 - searching
 - for items in scripts, Resource Find 263
 - searching and replacing, Resource Find 263
 - Section
 - drawn object property 303
 - field property 302
 - security
 - described 19
 - for Dexterity applications 19
 - for SQL logins 376
 - security form, for commands 182
 - segment options, for key segments 107
 - Select All, menu item 40
 - separator lines
 - built-in command for 186
 - in form-based menus 168, 178
 - sequence numbers, for toolbars 202
 - Series, report property 301
 - series
 - for forms 120, 122
 - for reports 286
 - for tables 101, 112
 - server software, client/server applications 18
 - session management, for SQL logins 386
 - SET files, *see* launch files
 - SetChangeFlag, field property 139
 - Setting Product Information, chapter 475-478
 - Setting up Source Code Control, chapter 425-433
 - settings box
 - illustration 136, 301
 - using 136, 301
 - Shape, drawn object property 142, 303
 - shape tool, using in window layout 129
 - shortcut keys
 - for commands 182, 188
 - for form-based menu items 169
 - Show Field Names menu item 131
 - Show Grid menu item 132
 - Show Invisible Fields menu item 131
 - ShowPartialItems, field property 141
 - size to default tool
 - using in report layout 298
 - using in window layout 130
 - size to narrowest tool
 - using in report layout 298
 - using in window layout 130
 - size to shortest tool
 - using in report layout 298
 - using in window layout 130
 - size to tallest tool
 - using in report layout 298
 - using in window layout 130
 - size to widest tool
 - using in report layout 298
 - using in window layout 130
 - Size-Height
 - drawn object property 142, 303
 - field property 141, 303
 - window property 138
 - Size-Width
 - drawn object property 142, 303

Size-Width (*continued*)

- field property 141, 303
- window property 138
- sizing windows 155
- slashes, in form-based menu items 168
- software testing, *see also* test procedures
- sorting
 - chapter 309-311
 - for groups 346
 - key segment options 107
 - reports 309, 361
 - SQL Server setting 369
- sorting definition
 - creating 310
 - sorting options 310
- Sorting Definition window 310
- source code control
 - benefits 421
 - building a dictionary 456
 - changing resources 445
 - checking in a dictionary 440
 - checking in source files 446
 - checking out source files 442
 - configurations 422
 - configuring Dexterity 431
 - configuring the repository 428
 - connectivity required 425
 - creating new resources 445
 - described 421
 - enabling in Dexterity 431
 - errors 460
 - examining the repository 441
 - fetching source files 451
 - firewall settings 430
 - for integrating applications 465
 - generic provider 469
 - index file 455
 - installing 425
 - labels for source files 464
 - locking source files 448
 - maintaining the repository 461
 - part 420-471
 - providers 423
 - repository maintenance 461
 - resource IDs 455
 - setting source file state 465

source code control (*continued*)

- setting up 425
- source code control server 429
- source files 435
- TCP/IP 425
- terminology 422
- troubleshooting 432, 460
- unlocking source files 449
- updating a dictionary 453
- updating source file states 464
- using 439
- validating connection to repository 432
- version information 459
- source code control server
 - configuring 429
 - described 422
 - installing 426
 - port used 430
 - restarting 429
- Source Code Errors window 460
- Source Control, menu item 258
- source files
 - changing names 438
 - chapter 435-438
 - checking in 446
 - checking out 442
 - contents 436
 - described 422, 435
 - fetching 451
 - labels for 464
 - locking 448
 - names 437
 - renaming 462
 - setting source file state 465
 - structure 436
 - unlocking 449
 - updating state information 464
 - viewing version information 459
- SQA, *see* software quality
- SQL
 - accessing existing SQL data 402-404
 - active locking 385
 - auto-generated stored procedures
 - described 391
 - disabling 109

SQL (*continued*)

- connection pools 378
- connections 377
- creating tables 391
- cursors 381
- data sources 369
- data types, Dexterity equivalents 404
- database manager, described 28, 102
- encrypted fields 399
- error handling 395
- installing SQL Server 369
- key options 107
- login failures 376
- login security 376
- Login window
 - displaying 376
 - illustration 375
- pass-through SQL 415
- pathnames 395
- privileges for tables 393
- record locking 385
- session management 386
- sorting order 369
- stale data 383
- stored procedures 496
- SQL Database Manager, part 368-417
- SQL Server, installing 369
- SQL symbol 7
- SQL tables
 - chapter 391-404
 - privileges 393
- SQLError procedure
 - described 396
 - parameters 396
 - reading error information 397
 - sample 397
- SQLLoginTimeout defaults file setting 376
- SQLMaxConnections defaults file setting 378
- SQLPath procedure, described 395
- SQLProcsTimeout defaults file setting 414
- SQLScriptPath procedure
 - described 413

- SQLScriptPath procedure (*continued*)
 - parameters 413
 - use for stored procedures 413
- stale data, described 383
- stand-alone applications, overview 21
- standard composites, *see also*
 - composites
- standard windows, described 154
- starting
 - applications, using a launch file 481
 - Dexterity 34
- starting scripts, for a chunk dictionary 489, 491
- static picture values
 - see also* static values
 - for controls, chart 53
 - native pictures 211
 - selecting for a data type 53
 - uses 53
- static text values
 - see also* static values
 - for controls, chart 53
 - selecting for a data type 52
 - uses 53
- Static Text Values window 52
- static values
 - see also* static text values, static
 - picture values
 - described 51, 74
 - for control types 54
 - for data types 51, 74
- storage, in tables 97
- storage sizes
 - for control types 54
 - for data types 50
- storage types, *see also* data types
- stored procedures
 - auto-generated
 - described 391
 - disabling 109
 - benefits 407
 - calling from sanScript 408
 - chapter 407-414
 - described 407
 - examples 410, 411
 - pathnames for 413
- stored procedures (*continued*)
 - prototype procedure
 - creating 409
 - described 408
 - return values 410
 - SQLScriptPath procedure 413
 - timing out 414
 - using 408
- STR_CUR, calculated field function 332
- STR_DAT, calculated field function 332
- STR_LEN, calculated field function 333
- STR_LNG, calculated field function 333
- STR_VCUR, calculated field function 333
- stranded locks
 - clearing 389
 - described 389
- String Definition window 226
- string formats
 - described 79
 - example 79
 - format options 86
 - multiple format selector
 - described 82
 - example 82
- strings
 - chapter 225-226
 - control type 68
 - data type 71
 - described 225
 - format strings
 - described 79, 86
 - example 79, 86
 - formatting 79
 - items related to 226
 - modifying, procedure 226
 - tool 129
- STRIP, calculated field function 333
- structured exception handler, *see also*
 - exception handling
- Style, field property 141
- submenus, *see* cascading menus
- submission file, described 521
- substituting fonts for reports 306
- substituting items into messages 208
- SUBSTRING
 - calculated field function 334
 - report restriction function 320
- subtotaling data for reports 349
- subtraction operator, use in calculated fields 325
- Sum, report display type 359
- SUMMATION, report restriction function 321
- summing data for reports 349
- support, available for Dexterity 8
- suppressing
 - additional footers 344, 345
 - additional headers 344
- symbols in documentation 7
- synchronize utility, chapter 515
- synchronizing, native pictures 215
- system colors
 - automatically assigned 149
 - color categories 152
 - color properties 149
 - customizable 152
 - overriding default colors 153
 - pictures and native pictures 150
 - recommendations 150
 - window background color 150
- system variables, *see* global variables
- system-defined functions
 - CUR_STR, described 331
 - DAY, described 331
 - DAY_NAME, described 331
 - DOW, described 331
 - INT_STR, described 331
 - LFT_STR, described 331
 - LNG_STR, described 332
 - MONTH, described 332
 - MONTH_NAME, described 332
 - nesting in calculated field 330
 - POWER_10, described 332
 - RGT_STR, described 332
 - STR_CUR, described 332
 - STR_DAT, described 332
 - STR_LEN, described 333
 - STR_LNG, described 333
 - STR_VCUR, described 333
 - STRIP, described 333
 - SUBSTRING, described 334

system-defined functions (*continued*)
 UCASE, described 335
 using in calculated fields 330
 WILDCARD, described 335
 YEAR, described 335

T

tab sequence
 described 143
 removing fields from 143
 setting 143, 156
 tab-delimited files, saving reports to 362
 Table, field property 302
 table access modes
 exclusive use 120, 123
 read only 120, 123
 read/write 120, 123
 table buffers, recalculating size 515
 table conversions, *see* converting data tables
 Table Definition window 100, 112
 Table Descriptions, menu item 39
 table fields
 see also tables, fields
 creating 91
 described 91, 104
 example 92
 table groups
 chapter 237-238
 creating 111
 defining, procedure 237-238
 described 111, 237
 purpose 111, 237
 table keys, *see* keys
 Table Options window 120, 123
 Table Relationship Definition window 281
 table relationships
 chapter 279-283
 creating, procedure 280-283
 described 110, 115, 279
 illustration 280
 one-to-many relationship,
 described 279
 one-to-one relationship, described 279

table relationships (*continued*)
 secondary tables 281
 types 279
 table series, assigning to tables 101, 112
 TableName, field property 302
 tables
 4 byte header 110
 see also attached tables, keys
 access options 120, 123
 active locking 109, 385
 adding fields 104
 as a data storage method 97
 attached to a form 120, 122
 auto-generated stored procedures
 described 391
 disabling 109
 changing table definitions 499
 chapter 97-116
 concepts 97
 converting data 499
 creating
 procedure 112-116
 SQL tables 391
 c-tree Plus 28
 database types 28
 assigning 113
 described 102
 described 28, 98
 display names 101, 112
 elements 100
 encrypting table fields 105, 116
 example 99
 in database development 28
 inserting fields 113
 keys 99
 adding segments 114
 defining 105, 113
 described 105, 113
 key options 106, 114
 naming 114
 locations 111
 naming 101, 112
 options 109, 115
 overview 97
 Pervasive.SQL 28
 physical names 101, 112
 privileges for SQL tables 393

tables (*continued*)
 record size 104
 records 97
 relationships between 279
 restrictions 110
 sorting data in SQL tables 369
 SQL 28
 structure, illustration 99
 table fields 104
 table groups
 creating 111
 defining 237
 described 111, 237
 purpose 111, 237
 table relationships 110, 115, 279
 table series 101, 112
 temporary
 creating 110
 database types used 111
 location 111
 using in reports 364
 timestamp for each row 110
 using 97
 virtual tables 239
 TabStop, field property 139
 tags, for commands 184
 TCP/IP
 connection limits 378
 use for source code control 425
 TCP/IP port, used for Dexterity Source Code Control service 430
 technical support, for Dexterity 8
 temporary tables
 creating 110
 database types used 111
 location 111
 using in reports 364
 test mode
 chapter 157-160
 debugging scripts 159
 determining if running in 160
 editing scripts 158, 159
 leaving 157
 limitations 160
 logging scripts 159
 overview 157
 profiling scripts 159

- test mode (*continued*)
 - script lookup 159
 - switching to 157
 - tools available 158
- text
 - see also* static text
 - aligning 78
 - control type 68
- text attachments, mailing reports as 364
- text fields, use in reports 305
- text files, saving reports to 362
- text reports
 - see also* reports
 - described 275
 - line spacing 289
 - options 289
 - text size 289, 307
- text tool
 - using in report layout 297
 - using in window layout 129
- The Dexterity Interface, chapter 37-45
- The Dexterity System, chapter 11-14
- themes
 - using with controls 26
 - using with field properties 138
- tile horizontally tool
 - using in report layout 298
 - using in window layout 130
- tile vertically tool
 - using in report layout 298
 - using in window layout 130
- time, control type 69
- time tool
 - using in report layout 297
 - using in window layout 129
- time values, use in reports 305
- timeouts, for SQL data sources 376
- timestamp, for rows in tables 110
- tiny integers, control type 69
- Title, window property 137
- titles of windows, described 126, 154
- tokens, in messages 209
- Toolbar window 38
- toolbars
 - automatic clean-up 204
 - chapter 201-204
- toolbars (*continued*)
 - command display options 183, 189
 - creating 201
 - described 127, 155, 201
 - Dexterity's toolbar 38
 - drop-down lists in 201
 - implementing 201
 - orientation 131
 - re-creating arrangement 203
 - removing 202
 - retrieving position 203
 - rows 202
 - sample script 203
 - sequence values 202
 - toolbar commands 201
- Toolbox
 - arrange tools 130
 - described 44, 129
 - for reports 296
 - illustration 296
 - layout tools 129
 - tools 129-130
- tools
 - built into Dexterity 15
 - Import Utility 16
 - macro system 19
 - Modifier 15
 - Report Writer 15
 - Resource Descriptions 16
 - VBA 15
- Tools and Features, chapter 15-20
- tools mode, viewing reports 308
- Tooltip, field property 139
- tooltips, for commands 182, 188
- totaling data for reports 349
- translating applications, using strings resource 225
- trapping for errors, *see* error trapping
- tree view
 - control type 70
 - static picture values 53
- TreeView tool 129
- Type, report property 301
- type libraries, described 227
- U**
- UCASE, calculated field function 335
- unchunking, runtime command line parameter 496
- Undo, menu item 40
- unfocus, option for form-based menu items 170
- Unique, key option 107
- unlinking, prompts from fields 145
- Unlock in Repository window 450
- unlocking, source files 449
- update chunks
 - creating 506
 - dictionary modules to include 505
 - example 507
- Update from Repository window 453
- updates
 - packaging 504
 - procedure 504
 - using dictionary chunks 504
- updating
 - applications 499-511
 - dictionaries 453
 - forms and reports dictionaries 508
- updating an application, chapter 499-511
- user-defined constants, *see* constants
- user-defined functions
 - form level, *see* form functions
 - using for reports 336
 - using in calculated fields 330, 336
- UseUniqueHelp, field property 140
- Using Launch Files, chapter 479-482
- Using Reports in Applications, chapter 361-365
- Using Source Code Control, chapter 439-460
- Using the Runtime Engine, part 474-482
- Utilities, part 514-521
- Utilities menu, described 40
- V**
- variable currency 58
- VBA, described 15
- version numbers, for chunk dictionaries 488, 491, 506
- Versions window 459
- View, menu item 257

View button 257
 views, created for virtual tables 243
 Virtual Table Definition window 246
 virtual tables
 chapter 239-249
 concepts 239
 creating, procedure 246-249
 described 239
 elements 240
 joins 243
 member tables 240
 relationships 241, 242
 scripting 244
 using 244
 views created for SQL 243
 Visibility, field property 303
 Visible, field property 140
 VisibleItems, field property 140
 Visual Basic for Applications, *see* VBA
 visual properties
 for drawn objects 142
 for fields 140
 Visual Studio, using to create a
 resource library 229
 visual switch, tool 129
 visual switches
 control type 70
 static picture values 53
 use in reports 305

W

walkthroughs, *see* software
 walkthroughs
 warning messages, *see* communicating
 with the user, messages
 warning statement 208
 warning symbol 7
 WILDCARD
 calculated field function 335
 report restriction function 321
 window controls, standard buttons in
 Dexterity 45
 Window Descriptions, menu item 39
 window fields
 creating 91
 described 91, 132
 example 91

Window Help, menu item 41
 Window Layout window 128
 window layouts
 designing 128, 155
 saving 128
 window properties, list of 137-138
 WindowID, window property 137
 windows
 adding
 global fields 133
 graphics 155
 local fields 134
 auto-link table 155
 background color 150
 chapter 125-156
 context menus 195
 creating a layout 128
 creating for forms 124
 creating, procedure 154-156
 described 25
 design basics 25
 elements 126
 internal 127
 invisible fields 131
 modal dialogs 127
 modal windows 154
 naming 126, 154
 orientation 131
 overview 25, 125
 palettes 127, 155
 positioning 145, 156
 positioning fields 155
 previewing 142
 primary windows 127
 properties, list of 137
 resizing 146
 saving 128
 scrolling, *see* scrolling windows
 setting
 properties 126
 tab sequence 143
 sizing 131, 155
 standard windows 154
 title 126, 154
 toolbars 127, 131, 155
 types 127, 154

Windows Data Access Components
 (Windows DAC), required by
 runtime 495
 Windows graphics 144
 Windows Help, support in Dexterity
 applications 19
 windows in Dexterity
 *search the Dexterity online help for
 windows in Dexterity*
 Calculated Field Definition 323
 Check In to Repository 447
 Check in To Repository 440
 Check Out from Repository 444
 Command Definition 181, 188
 Constant Definition 218
 Data Type Definition 50, 73
 Define Worksets 259, 262
 Drawing Options 306
 Duplicate 517
 Edit Source File Names 438, 463
 Fetch from Repository 452
 Field Definition 89, 94
 Footer Options 343
 Form Constant Definition 219
 Form Definition 119, 122
 Format Definition 78, 84
 Header Options 343
 Icon Definition 233, 234
 Inherited Menu Options 167
 Integration Reports 519
 Key Definition 105, 114
 Library Definition 228
 Local Field Definition 134
 Lock in Repository 449
 Menu Definition 166, 177
 Messages 210
 Native Picture Definition 214
 New Workset 260
 Open Script 158
 Picture Definition 144, 212
 Print Report Definition 291
 Product Information 477
 Properties 126, 136
 Relationship Definition 248
 Report Definition 285
 Report Field Options 353
 Report Restriction Definition 313

- windows in Dexterity (*continued*)
 - Report Table Relationships 287
 - Resource Explorer 42
 - Resource Find 263
 - Resources to Update 455
 - Sorting Definition 310
 - Source Code Errors 460
 - Static Text Values 52
 - String Definition 226
 - Table Definition 100, 112
 - Table Relationship Definition 281
 - Toolbox 129
 - Unlock in Repository 450
 - Update from Repository 453
 - Versions 459
 - Virtual Table Definition 246
 - Window Layout 128
- windows in Dexterity Utilities, Create
 - Chunk Dictionary 490, 508
- windows in Report Writer, Properties
 - 300
- Windows menu, described 41
- Windows metafiles, *see* metafiles
- WindowType, window property 137
- WordWrap, field property 140
- Worksets, button 256
- worksets
 - adding resources 260
 - chapter 259-262
 - creating 259
 - deleting 262
 - described 259
 - removing resources 261
 - viewing 261
- Workstation2 setting 481

X

- XOR, *see* logical XOR

Y

- YEAR calculated field function 335

Z

- Zoom
 - drawn object property 142
 - field property 141

