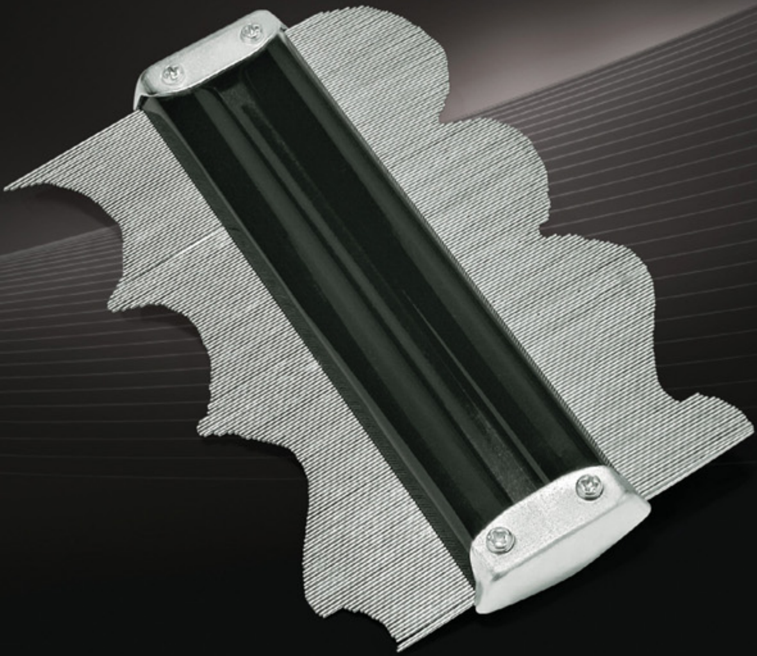
Microsoft

Foreword by Hans J. Skovgaard
Product Unit Manager, Microsoft Corporation

Inside Microsoft Dynamics® AX 2009



The Microsoft Dynamics AX Team

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2009927101

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWT 4 3 2 1 0 9

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Acquisitions Editor: Ben Ryan
Developmental Editor: Maria Gargiulo
Project Editor: Valerie Woolley
Editorial Production: Macmillan Publishing Solutions
Technical Reviewer: Arijit Basu; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Cover: Tom Draper Design

Body Part No. X15-77861

Contents at a Glance

Table of Contents

Part IV Appendices

Foreword

In the course of our engagement with numerous partners and customers, we have learned how much developers enjoy working with Microsoft Dynamics AX. We love building ever-more-powerful versions of Dynamics AX that help our customers increase their competitiveness; our partners love our product's powerful set of development tools, which allow them to build affordable and flexible enterprise resource planning (ERP) systems whose quality and adaptability are second to none. Some of the examples in this book, such as creating and exposing custom data sets to external applications through a Web service, can be performed in 10 minutes or less. The very same examples would take at least a week to complete in other ERP systems.

This is the second edition of this book. We received very positive feedback on the first edition, *Inside Microsoft Dynamics AX 4.0*, from people who used the book for a variety of purposes: from ramping up new teams and using the information in it to build presentations, to giving copies to customers, prospects, and colleagues. Our aspiration for this new edition is to incorporate the feedback we received on the original book and to cover the technologies added to Dynamics AX 2009, such as ASP.NET for Enterprise Portal, new Workflow functionality, Role Centers with advanced business intelligence, and the new Batch framework.

We believe that this book, which aims to provide developers with solid information on advanced concepts, can make your entry into the powerful toolset for building business applications a much smoother and more digestible learning experience. We hope that this book is received as an insightful resource for many people working with Dynamics AX 2009.

I want to thank the coauthors for using their evenings and weekends to contribute to this book.

I wish you success with your implementation and customization of Microsoft Dynamics AX 2009.

Hans J. Skovgaard
Product Unit Manager
Microsoft Corporation

Acknowledgments

We want to thank all the people who assisted us in making this book become a reality. The list of people is long—if we inadvertently missed anyone, we apologize. A special thanks goes to the following people on the Microsoft Dynamics AX product team:

- Heidi Boeh, who pitched this book to Microsoft Press and then worked to pull together great content from 24 contributors (from 10 countries of origin), across three development centers.
- Hal Howard and Hans Jørgen Skovgaard, who sponsored the project.
- Michael Fruergaard Pontoppidan, one of the principle authors of this book. Michael's dedication and broad and deep product knowledge brought excellence to this book that it would not have otherwise achieved.
- The people on the product team who provided reviews, comments, and editorial assistance, and otherwise moved small mountains to make this book happen:

  Wade Baird
  Arijit Basu
  Sue Brandt
  Ben Buresh
  Hua Chu
  Christian Heide Damm
  Hitesh Dani
  Julie Deutz
  Krishnan Duraisamy
  Ahmad El Husseini
  Arthur Greef
  Jan Jakobsen
  Peter Jerkewitz
  Andrew Jewsbury
  Brian King
  Steve Kubis
  Arif Kureshy
  Stuart Macrae
  Donna Marshall
  Scott McDonald
  Gene Milener
  Becky Newell
  Adrian Orth
  Marko Perisic
  Jen Pfau
  Gustavo Plancarte
  Pepijn Richter
  David Robinson
  Finnur Saemundsson
  Karen Scipi
  Dianne Siebold
  Stella Sorensen
  Gandhi Swaminathan
  Piotr Szymanski
  Tracy Taylor
  Bill Thompson
  Shyla Thompson
  Van Vanslette
  Peter Villadsen

- The early reviewers from the Dynamics AX Partner Advisory Board, who provided us with the crucial partner perspective.

We also want to thank the people at Microsoft Press who helped support us throughout the book writing and publishing process:

- Ben Ryan, who championed the book project at Microsoft Press.
- Maria Gargiulo, who provided valuable feedback during the developmental editing cycle.
- Valerie Woolley, who was our outstanding, efficient, and always calming project editor. Val, we can't thank you enough.
- Arijit Basu, the technical reviewer, who asked the countless questions that made this book better than it would have been otherwise. Arijit, you rock.
- Sally Stickney, who provided the final editing passes on this book. Her questions were insightful and precise, and her feedback was excellent. She went above and beyond the call of duty to bring excellence to this project.

The Microsoft Dynamics AX author team

Introduction

We understand if you're a bit skeptical when we tell you that you are about to fall for a piece of software. We want you to know up front that our intention is to show you all the wonderful and amazing benefits that Microsoft Dynamics AX 2009 has to offer your business.

Here are some reactions from our partners and customers who have been involved in the early adoption.

> The release of Microsoft Dynamics AX 2009 shows a tremendous development effort on Microsoft's side not only for those already familiar with the product but for newcomers too. This is the best release yet.
>
> *Jesper R. Hansen, Partner, thy:innovation*

> The completeness of the . . . release of Microsoft Dynamics AX 2009 will help existing and new customers reduce the cost of additional customization and therefore reduce the total implementation cost.
>
> *Sven Sieverink, Business Consultant, Dynamics Software*

> The out-of-the-box Role Centers in Microsoft Dynamics AX 2009 add value from day one for all employees in our organization. Our users see enormous value with the visuals, the ability to easily design cues, and the year-to-year comparison.
>
> *John Elmer, Vice President of Information Systems, Rodgers and Hammerstein*

> The Business Intelligence possibilities and integrated workflow offer my team powerful tools to do the job themselves.
>
> *Greg Brock, Director Information Systems, Techmer PM*

Who Is This Book For?

This book delves into the technology and tools in Dynamics AX 2009. New and experienced developers are the intended target audience, and consultants will also benefit from reading this book. The intention is not to give guidance on application functionality but rather to offer as much technical information between the two covers as possible. It is also beyond the scope of this book to include details about installation, upgrade, deployment, and sizing of production environments. Refer to the extensive installation and implementation documentation supplied with the product for more information on these topics.

To get full value from this book, you should have knowledge of common object-oriented concepts from languages such as C++, C#, and Java. Knowledge of Structured Query Language (SQL) is also an advantage. SQL statements are used to perform relational database tasks such as data updates and data retrieval.

Note If you don't have the Dynamics AX license that provides developer rights, you won't be able to perform most of the actions in this book. A virtual PC version of Dynamics AX 2009, with developer rights, is available for partners to download at *https://mbs.microsoft.com/partnersource/deployment/methodology/vpc/vpcimageax2009.htm*.

The History of Microsoft Dynamics AX

Historically, Dynamics AX envelops more than 25 years of experience in business application innovation and developer productivity. Microsoft acquired Dynamics AX in 2002; the success of the product has spurred an increasing commitment of research and development resources, which allow Dynamics AX to continuously grow and strengthen its offering.

The development team that created Dynamics AX 2009 consists of three large teams, two of which are based in the United States (Fargo, North Dakota, and Redmond, Washington) and one based in Denmark (Copenhagen). The Fargo team focuses on finance and human resources (HR), the Redmond team concentrates on Microsoft Project and CRM, and Copenhagen team delivers Supply Chain Management (SCM). In addition, a framework team, distributed over the three sites, develops infrastructure components. Finally, a worldwide distributed team localizes the Dynamics AX features to meet national regulations or local differences in business practices, allowing the product to ship in 24 main languages in 38 countries.

To clarify a few aspects of the origins of Dynamics AX, the authors contacted people who participated in the early stages of the Dynamics AX development cycle. The first question we asked was, How was the idea of using X++ as the programming language for Dynamics AX conceived?

> We had been working with an upgraded version of XAL for a while called OO XAL back in 1996/1997. At some point in time, we stopped and reviewed our approach and looked at other new languages like Java. After working one long night, I decided that our approach had to change to align with the latest trends in programming languages, and we started with X++.
>
> *Erik Damgaard, cofounder of Damgaard Data*

Of course, there were several perspectives among the developers on this breakthrough event.

One morning when we came to work, nothing was working. Later in the morning, we realized that we had changed programming languages! But we did not have any tools, so for months we were programming in Notepad without compiler or editor support.

Anonymous developer (but among the authors of this book!)

Many hypotheses exist regarding the origin of the original product name, Axapta. Axapta was a constructed name, and the only requirement was that the letter *X* be included, to mark the association with the predecessor XAL. The *X* association carries over in the name Dynamics AX.

Organization of This Book

Part I , "A Tour of the Development Environment," is mainly for people new to Dynamics AX. It describes the application architecture from the perspective of development, deployment, and administration. The chapters in Part I also provide a tour of the internal Dynamics AX development environment to help new developers familiarize themselves with designers, tools, the X++ programming language, and the object-oriented application framework that they will use to implement their customizations, extensions, and integrations.

Parts II ("Core Development Concepts") and III ("Under the Hood") are largely devoted to illustrating how developers use the Dynamics AX application framework. Through code samples written for a fictitious bicycle sales and distribution company, Part II describes how to customize and extend Dynamics AX. The examples show how the fictitious company customizes, extends, and integrates the application to support its online make-to-order sales, distribution, and service operations.

Reading Guide

If you are an experienced Dynamics AX developer, you might want to skip the tour of the development environment after reading Chapter 1, "Architectural Overview," and move straight to Part II or Part III, which consist of chapters that can be read in random order. Or use the book as a reference for subjects that you are especially interested in.

Differences from Inside Microsoft Dynamics AX 4.0

This book is an update to the book *Inside Microsoft Dynamics AX 4.0*. Along with changes made to existing chapters, we added several new chapters, on workflow, Role Centers, the Batch framework, reporting, and code upgrade. We have significantly expanded the performance chapter, and the Enterprise Portal chapter now describes the new ASP.NET tooling.

We greatly extended the chapter "XML Document Integration" in the first edition, renaming it as "The Application Integration Framework" (Chapter 17).

We removed the chapters on advanced MorphX forms and system classes because these were least referenced by readers. You can find extensive documentation on MorphX forms and system classes in the Dynamics AX 2009 software development kit (SDK), which is on MSDN. And the previous version of this book is still a good source of information because some of the technologies haven't changed much. We also removed the chapter on upgrade and data migration and replaced that content with a significantly enhanced series of chapters on the version upgrade process, found in the bonus eBook, which can be found on this book's companion Web site: *http://www.microsoft.com/learning/en/us/books/13345*. Finally, the chapter on unit testing has been merged into the chapter on the MorphX tools (Chapter 3).

Product Documentation

In addition to this book, you can read thousands of topic pages of product documentation on application and system issues in the online Help. Extensive documentation on installation and implementation is available in the Microsoft Dynamics AX 2009 SDK and the Microsoft Dynamics AX Implementation Guide, both supplied with the product. You can also find the product documentation on MSDN. And if you have an installation of Dynamics AX 2009, you have access to the following topic areas on the Help menu: Administrator Help, Developer Help, and User Help.

Product Web Site

The user portal for Dynamics AX encompasses product and purchase information as well as guidelines for using the product and links to online newsgroups and user communities.

For more information, visit the site *http://www.microsoft.com/dynamics/ax*.

Naming

With the latest version of the application, the name of the product changed to Microsoft Dynamics AX 2009. The previous product versions were named Microsoft Axapta and Microsoft Dynamics AX 4.0. For easier reading, this book refers to the 2009 version of the product as Dynamics AX 2009 or just Dynamics AX and refers specifically to earlier versions where appropriate.

Code

All relevant code examples are available for download. For details on the companion Web site, see the "Code Samples" section later in this introduction. You might need to modify some of the code samples to execute them. The necessary changes are described either in the .xpo files themselves or in the readme file associated with the code samples on the companion Web site.

Glossary

Like all software, Dynamics AX involves the use of many abbreviations, acronyms, and technical expressions. Much of this information is available in a glossary that you will find at the back of the book. For a larger list of terms and abbreviations, refer to the glossary provided with the product documentation.

Special Legend

To distinguish between SQL and X++ statements, this book uses the common practice for SQL keywords, which is to display them in all capital letters. The following code shows an example of this in connection with nested transactions, where a transaction is started in X++ and later sent to a SQL server.

```
boolean b = true;  ;  ttsbegin; // Transaction is not initiated here
update_recordset custTable       setting creditMax = 0; // set implicit transactions on
if ( b == true )          ttscommit; // COMMIT TRANSACTION  else
    ttsabort; // ROLLBACK TRANSACTION
```

System Requirements

You need the following hardware and software to build and run all the code samples for this book:

- Microsoft Dynamics AX 2009: .NET Business Connector, Microsoft Dynamics AX 2009 Rich Client, Application Object Server (AOS; up and running)
- Windows Vista Business Edition, Ultimate Edition, or Enterprise Edition, Service Pack 1 or Windows XP Professional Edition, Service Pack 2/3 (for Microsoft Dynamics AX 2009 Rich Client)
- Windows Server 2003 with Service Pack 2 or Windows Server 2008 (AOS Server)
- Microsoft SQL Server 2008 or Microsoft SQL Server 2005, Service Pack 2, Service Pack 3, or Oracle Database 10*g* R2
- Windows SharePoint Services 3.0 with Service Pack 1 or Microsoft Office SharePoint Server 2007, Enterprise Edition Service Pack 1 (to run Enterprise Portal or Role Centers)
- Microsoft SQL Server 2008 Reporting Services or Microsoft SQL Server 2005 Reporting Services with SQL Server Service Pack 2/3 (to run SQL Reporting Services)

- Microsoft Visual Studio 2008
- Microsoft .NET Framework 3.5
- Intel Pentium/Celeron family or compatible Pentium III Xeon or higher processor minimum; 1.1 gigahertz (GHz) or higher recommended
- 1 gigabyte (GB) RAM or more recommended
- Video: at least 1024 × 768 high color 16-bit
- DVD-ROM drive
- Microsoft mouse or compatible pointing device

Because the requirements typically evolve with service packs that support new versions of underlying technologies, we recommend that you check for the latest updated system requirements at *http://www.microsoft.com/dynamics/ax/using/2009systemrequirements.mspx*.

Release Software

This book was reviewed and tested against the RTM version of Dynamics AX 2009. Any changes or corrections to this book will be added to a Microsoft Knowledge Base article. For details, see the "Support for This Book" section in this introduction.

Technology Updates

As technologies related to this book are updated, links to additional information will be added to the Microsoft Press Technology Updates Web site. Visit this site periodically for updates on Microsoft Visual Studio 2005 and other technologies: *http://www.microsoft.com/mspress/updates*.

Code Samples

All code samples discussed in this book can be downloaded from the book's companion content page at the following address: *http://www.microsoft.com/learning/en/us/books/13345.aspx*.

Bonus Content

On the companion Web site you'll find an eBook that contains several bonus chapters. Chapter 1, "Introduction to Upgrade," Chapter 2, "Code Upgrade" (also Chapter 18 of this book), Chapter 3, "Data Upgrade," and Chapter 4 "Upgrade Additional Topics."

The upgrade information you find in this eBook gives you a solid overview of the Dynamics AX 2009 upgrade process, tells you about the tools that are available to walk you through the upgrade process, and gives you some tips and best practice guidelines; it does not give you detailed procedures—simply because version upgrade is such a large topic. A wealth of procedural and other information on the upgrade process is available with the Dynamics AX product; we have included a list of those resources at the end of this introduction. You can download the eBook from *http://www.microsoft.com/learning/en/us/books/13345.aspx*.

Find Additional Content Online

As new or updated material becomes available that complements your book, it will be posted online on the Microsoft Press Online Developer Tools Web site. The type of material you might find includes updates to book content, articles, links to companion content, errata, sample chapters, and more. This Web is available at *www.microsoft.com/learning/books/online/developer*, and is updated periodically.

Support for This Book

Every effort has been made to ensure the accuracy of this book and the companion content. As corrections or changes are collected, they will be added to a Microsoft Knowledge Base article. To view the list of known corrections for this book, visit the following article:

http://support.microsoft.com/kb

Microsoft Press provides support for books and companion content at the following Web site:

http://www.microsoft.com/learning/support/books

Questions and Comments

If you have comments, questions, or ideas regarding the book or the companion content, or questions that are not answered by visiting the sites mentioned earlier, please send them to Microsoft Press via e-mail:

mspinput@microsoft.com

You may also send your questions via postal mail to

Microsoft Press
Attn: *Inside Microsoft Dynamics AX 2009* Project Editor
One Microsoft Way
Redmond, WA 98052-6399

Please note that Microsoft software product support is not offered through these addresses.

Chapter 3

The MorphX Tools

In this chapter:

The objectives of this chapter are to:

- Provide an overview of the tools used when developing a Microsoft Dynamics AX 2009 enterprise resource planning (ERP) application with MorphX.
- Share tips and tricks on how to use the MorphX tools efficiently.
- Demonstrate how to personalize and extend the MorphX tools.

Introduction

Dynamics AX includes a set of tools, the MorphX development tools, that allow developers to build and modify Dynamics AX business applications. Each feature of a business application uses the application model elements described in Chapter 2, "The MorphX Development Environment." The MorphX tools enable developers to create, view, modify, and delete the application model elements, which contain metadata, structure (ordering and hierarchies of elements), properties (key and value pairs), and X++ code. For example, a table element includes the name of the table, the properties set for the table, the fields, the indices, the relations, the methods, and so on.

This chapter describes the most commonly used tools and offers some tips and tricks for working with them. You can find additional information and an overview of other MorphX tools in the MorphX Development Tools section of the Microsoft Dynamics AX software development kit (SDK) 2009 on MSDN.

Tip To enable the development mode of Dynamics AX 2009, press Ctrl+Shift+D. Ctrl+Shift+D is a toggle key that also returns you to content mode.

Table 3-1 lists the MorphX tools that are discussed in this chapter.

TABLE 3-1 MorphX Tools

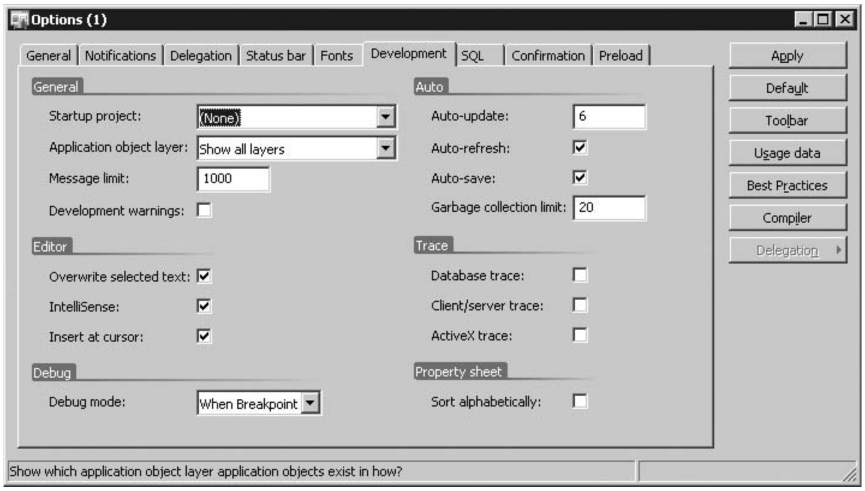
Tool	Use This Tool To:
Application Object Tree (AOT)	Start development activities. The AOT is the main entry point for all development activities. It is the repository for all elements that together comprise the business application. You can use the AOT to invoke the other tools and to browse and create elements.
Project Designer	Group related elements into projects.
Property sheet	Inspect and modify properties of elements. The property sheet shows key and value pairs.
X++ Code Editor	Inspect and write X++ source code.
Label Editor	Create and inspect localizable strings.
Visual Form Designer and the Visual Report Designer	Design forms and reports in a What You See Is What You Get (WYSIWYG) fashion.
Compiler	Compile X++ code into an executable format.
Best Practices tool	Automatically detect defects in both your code and your elements.
Debugger	Find bugs in your X++ code.
Reverse Engineering tool	Generate Microsoft Office Visio Unified Modeling Language (UML) and Entity Relationship Diagrams (ERDs) from elements.
Table Browser tool	View the contents of a table directly from the table element.
Find tool	Search for code or metadata patterns in the AOT.
Compare tool	See a line-by-line comparison of two versions of the same element.
Cross-reference tool	Determine where an element is used.
Version Control tool	Track all changes to elements and see a full revision log.
Unit Test tool	Build automated tests that can exercise your code and detect regressions.

You can access these development tools from the following places:

- The Development Tools submenu on the Tools menu. From the Microsoft Dynamics AX drop-down menu, point to Tools, and then point to Development Tools.
- The context menu on elements in the AOT.

Note The Microsoft Dynamics AX SDK contains valuable developer documentation and is updated frequently. Find it in the Microsoft Dynamics AX Developer Center on *msdn.microsoft.com*.

You can personalize the behavior of many MorphX tools by clicking Options on the Tools menu. Figure 3-1 shows the Options dialog box.



General
Startup project: (None)
Application object layer: Show all layers
Message limit: 1000
Development warnings: ☐

Editor
Overwrite selected text: ☑
IntelliSense: ☑
Insert at cursor: ☑

Debug
Debug mode: When Breakpoint

Auto
Auto-update: 6
Auto-refresh: ☑
Auto-save: ☑
Garbage collection limit: 20

Trace
Database trace: ☐
Client/server trace: ☐
ActiveX trace: ☐

Property sheet
Sort alphabetically: ☐

Show which application object layer application objects exist in how?

FIGURE 3-1 Options dialog box, in which development options are specified

Application Object Tree

The AOT is the main entry point to MorphX and is the repository explorer for all metadata. You can open the AOT by clicking the AOT icon on the toolbar or by pressing Ctrl+D. The AOT icon looks like this:

Navigating the AOT

As the name implies, the AOT is a tree view. The root of the AOT contains the element categories, such as Classes, Tables, and Forms. Some elements are grouped into subcategories to provide a better structure. For example, Tables, Maps, Views, and Extended Data Types reside under Data Dictionary, and all Web-related elements are found under Web. Figure 3-2 shows the AOT.

You can navigate the AOT by using the arrow keys on the keyboard. Pressing the Right arrow key expands a node if it has any children.

Elements are ordered alphabetically. Because thousands of elements exist, understanding the naming conventions and adhering to them is important to effectively using the AOT.

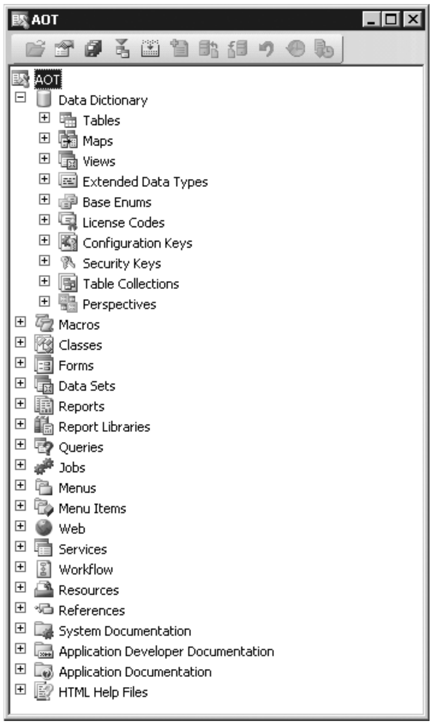


FIGURE 3-2 Application Object Tree

All element names in the AOT follow this structure:

<Business area name> + <Business area description> + <Action performed or type of content>

In this naming convention, similar elements are placed next to each other. The business area name is also often referred to as the *prefix*. Prefixes are commonly used to indicate the team responsible for an element.

Table 3-2 contains a list of the most common prefixes and their descriptions.

TABLE 3-2 Common Prefixes

Prefix	Description
Ax	Dynamics AX typed data source
Axd	Dynamics AX business document
BOM	Bill of material
COS	Cost accounting
Cust	Customer
HRM	Human resources management
Invent	Inventory management
JMG	Shop floor control
KM	Knowledge management
Ledger	General ledger
PBA	Product builder
Prod	Production
Proj	Project
Purch	Purchase
Req	Requirements
Sales	Sales
SMA	Service management
SMM	Sales and marketing management
Sys	Application frameworks and development tools
Tax	Tax engine
Vend	Vendor
Web	Web framework
WMS	Warehouse management

Tip When creating new elements, make sure to follow the recommended naming conventions. Any future development and maintenance will be much easier.

The Project Designer, described in detail later in this chapter, provides an alternative view of the information organized by the AOT.

Creating New Elements in the AOT

You can create new elements in the AOT by right-clicking the element category node and selecting New <*Element Name*>, as shown in Figure 3-3.

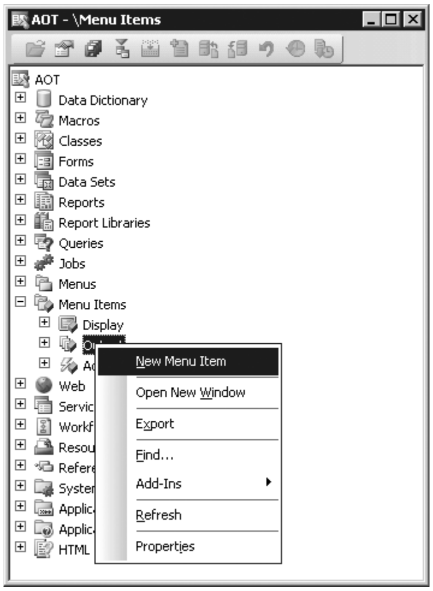


FIGURE 3-3 Creating a new element in the AOT

Objects are given automatically generated names when they are created. However, you should replace the default names with new names in accordance with the naming conventions.

Modifying Elements in the AOT

Each node in the AOT has a set of properties and either subnodes or X++ code. You can use the property sheet (shown in Figure 3-9) to inspect or modify properties, and you can use the X++ code editor (shown in Figure 3-11) to inspect or modify X++ code.

The order of the subnodes can play a role in the semantics of the element. For example, the tabs on a form display in the order in which they are listed in the AOT. You can change the order of nodes by selecting a node and pressing the Alt key while pressing the Up or Down arrow key.

A red vertical line next to an element name marks it as modified and unsaved, or *dirty*, as shown in Figure 3-4.

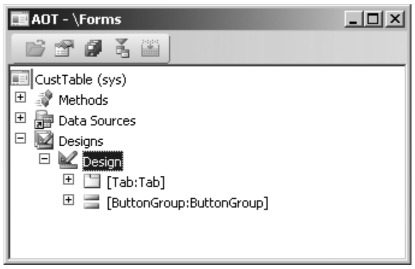


FIGURE 3-4 A dirty element in the AOT, indicated by a vertical line next to *CustTable (sys)*

A dirty element is saved in the following situations:

- The element is executed.
- The developer explicitly invokes the Save or Save All action.
- Autosave takes place. You specify the frequency of autosave in the Options dialog box accessible from the Tools menu.

Refreshing Elements in the AOT

If several developers modify elements simultaneously in the same installation of Dynamics AX, each developer's local elements could become out of sync with the latest version. To ensure that the local versions of remotely changed elements are updated, an autorefresh thread runs in the background. This autorefresh functionality eventually updates all changes, but you might want to explicitly force a refresh. You do this by right-clicking the element you want to restore and then selecting Restore. This action refreshes both the on-disk and the in-memory versions of the element. The following is a less elegant way of ensuring that the latest elements are used:

1. Close the Dynamics AX client to clear in-memory elements.
2. Close the Dynamics Server service on the Application Object Server (AOS) to clear in-memory elements.
3. Delete the application object cache files (*.auc) from the Local Application Data folder (located in Documents and Settings\<*User*>\Local Settings\Application Data) to remove the on-disk elements.

Note Before Dynamics AX 4.0, the application object cache was stored in .aoc files. To support Unicode, the file extension was changed to .auc in Dynamics AX 4.0.

Element Actions in the AOT

Each node in the AOT contains a set of available actions. You can access these actions from the context menu, which you can open by right-clicking the node in question.

Here are two facts to remember about actions:

- The actions available depend on the type of node you select.
- You can select multiple nodes and perform actions simultaneously on all the nodes selected.

A frequently used action is Open New Window, which is available for all nodes. It opens a new AOT window with the current nodes as the root. We used this action to create the screen capture of the *CustTable* element shown in Figure 3-4. Once you open a new AOT window, you can drag elements into the nodes, saving time and effort when you're developing an application.

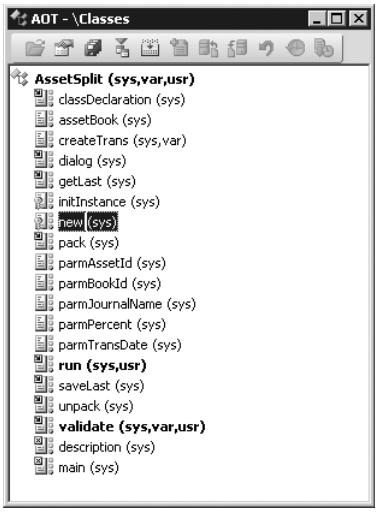
You can extend the list of available actions on the context menu. You can create custom actions for any element in the AOT by using the features provided by MorphX. In fact, all actions listed on the Add-Ins submenu are implemented in MorphX by using X++ and the MorphX tools.

You can enlist a class as a new add-in by following this procedure:

1. Create a new menu item and give it a meaningful name, a label, and Help text.
2. Set the menu item's *Object Type* property to Class.
3. Set the menu item's *Object* property to the name of the class to be invoked by the add-in.
4. Drag the menu item to the *SysContextMenu* menu.
5. If you want the action available only for certain nodes, you need to modify the *verifyItem* method on the *SysContextMenu* class.

Element Layers in the AOT

When you modify an element from a lower layer, a copy of the element is placed in the current layer. All elements in the current layer appear in bold type (as shown in Figure 3-5), which makes it easy to recognize changes. For a description of the layer technology, see the section "Application Model Layering System" in Chapter 1, "Architectural Overview."


AOT - \Classes

AssetSplit (sys,var,usr)
classDeclaration (sys)
assetBook (sys)
createTrans (sys,var)
dialog (sys)
getLast (sys)
initInstance (sys)
new (sys)
pack (sys)
parmAssetId (sys)
parmBookId (sys)
parmJournalName (sys)
parmPercent (sys)
parmTransDate (sys)
run (sys,usr)
saveLast (sys)
unpack (sys)
validate (sys,var,usr)
description (sys)
main (sys)

FIGURE 3-5 An element in the AOT that exists in several layers

You can use the Application Object Layer setting in the Options dialog box to personalize the layer information shown in the AOT. Figure 3-5 shows a class with the option set to All Layers. As you can see, each method is suffixed with information about the layers in which it is defined, such as *sys*, *var*, and *usr*. If an element exists in several layers, you can right-click it and select Layers to access its versions from lower layers. We highly recommend the All Layers setting during code upgrade because it provides a visual representation of the layer dimension directly in the AOT.

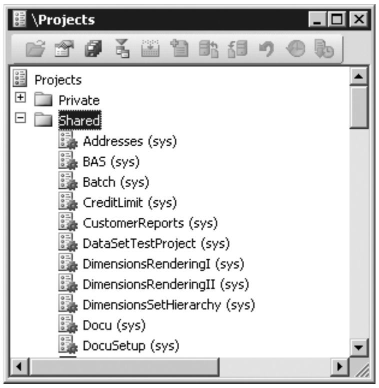
Note If you modify an element that exists in a higher layer than your current layer, all modifications are redirected to the upper layer where the element is defined.

Project Designer

For a fully customizable overview of the elements, you can use projects. In a *project*, elements can be grouped and structured according to the developer's preference. The Project Designer is a powerful alternative to the AOT because you can collect all the elements needed for a feature in one project.

Creating a New Project

You open the Project Designer by clicking the Project button on the toolbar. Figure 3-6 shows the Project Designer and its Private and Shared projects.



Addresses (sys)
BAS (sys)
Batch (sys)
CreditLimit (sys)
CustomerReports (sys)
DataSetTestProject (sys)
DimensionsRenderingI (sys)
DimensionsRenderingII (sys)
DimensionsSetHierarchy (sys)
Docu (sys)
DocuSetup (sys)

FIGURE 3-6 Project Designer, showing available private and shared projects

Except for its structure, the Project Designer behaves exactly like the AOT. Every element in a project is also present in the AOT.

When you create a new project, you must decide whether it should be private or shared among all developers. You can't set access requirements on shared projects. You can make a shared project private (and a private project shared) by dragging it from the shared category into the private category.

Note Central features of Dynamics AX 2009 are captured in shared projects to provide an overview of all the elements in a feature. No private projects are included with the application.

You can specify a startup project in the Options dialog box. If specified, the chosen project automatically opens when Dynamics AX is started.

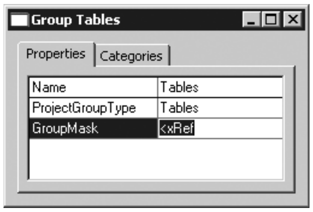
Automatically Generated Projects

Projects can be automatically generated in several ways—from using group masks to customizing special project types—to make working with them easier. We discuss the various ways to automatically generate projects in the sections that follow.

Group Masks

Groups are folders in a project. When you create a group, you can have its contents be automatically generated by setting the *ProjectGroupType* property (All is an option) and a regular expression as the *GroupMask* property. The contents of the group are created automatically and kept up to date as elements are created, deleted, and renamed. Using group masks ensures that your project is always current, even when elements are created directly in the AOT.

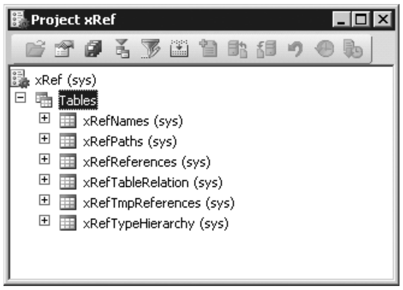
Figure 3-7 shows the *ProjectGroupType* property set to Tables and the *GroupMask* property set to <xRef on a project group. All table names starting with *xRef* (the prefix for the Cross-reference tool) will be included in the project group.



Name	Tables
ProjectGroupType	Tables
GroupMask	<xRef

FIGURE 3-7 Property sheet specifying settings for *ProjectGroupType* and *GroupMask*

Figure 3-8 shows the resulting project when the settings from Figure 3-7 are used.



```
xRef (sys)
  Tables
    xRefNames (sys)
    xRefPaths (sys)
    xRefReferences (sys)
    xRefTableRelation (sys)
    xRefTmpReferences (sys)
    xRefTypeHierarchy (sys)
```

FIGURE 3-8 Project created by using group masks

Filters

You can also generate a project based on a filter. Because all elements in the AOT persist in a database format, you can use a query to filter elements and have the results presented in a project. You create a project filter by clicking the Filter button on the project's toolbar. Depending on the complexity of the query, a project can be generated instantly or might take several minutes.

Filters allow you to create projects containing the following kinds of elements:

- Elements created or modified within the last month
- Elements created or modified by a named user
- Elements from a particular layer

Development Tools

Several development tools, such as the Wizard wizard, produce projects containing elements the wizard creates. The result of running the Wizard wizard is a new project that includes a form, a class, and a menu item—all the elements comprising the newly created wizard.

You can also use several other wizards, such as the Report Wizard and the Class Wizard, to create projects. You can access these wizards from the Microsoft Dynamics AX drop-down menu by clicking Tools\Development Tools\Wizards.

Layer Comparison

You can compare all the elements in one layer with the elements in another layer, called the reference layer. If an element exists in both layers, and the definitions of the element are different or the element doesn't exist in the reference layer, the element will be added to the resulting project. You can compare layers by clicking Tools\Development Tools\Code Upgrade from the Microsoft Dynamics AX drop-down menu.

Upgrade Projects

When you upgrade from one version of Dynamics AX to another or install a new service pack, you need to deal with any new elements that are introduced and existing elements that have been modified. These changes might conflict with customizations you've implemented in a higher layer.

The Create Upgrade Project feature makes a three-way comparison to establish whether an element has any upgrade conflicts. It compares the original version with both the customized version and the updated version. If a conflict is detected, the element is added to the project.

The resulting project provides a list of elements to update based on upgrade conflicts between versions. You can use the Compare tool, described later in this chapter, to see the conflicts in each element. Together, these features provide a cost-effective toolbox to use when upgrading. For more information about upgrading code, see Chapter 18, "Code Upgrade."

You can create upgrade projects by clicking Tools\Development Tools\Code Upgrade\Detect Code Upgrade conflicts from the Microsoft Dynamics AX drop-down menu.

Project Types

When you create a new project, you can specify a project type. So far in this chapter, we've limited our discussion to standard projects. Two specialized project types are also provided in Dynamics AX:

- **Test project** Project used to group a set of classes for unit testing
- **Help Book project** Project used for the table of contents in the online Help system

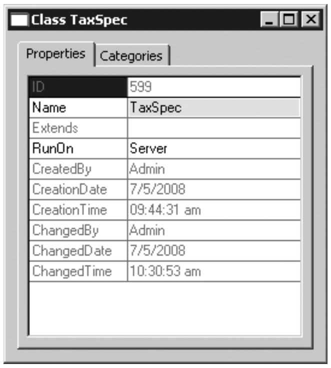
You can create a custom specialized project by creating a new class that extends the *ProjectNode* class. Specialized projects allow you to control the structure, icons, and actions available to the project.

Property Sheet

Properties are an important part of the metadata system. Each property is a key and value pair. The property sheet allows you to inspect and modify properties of elements.

Open the property sheet by pressing Alt+Enter or by clicking the Properties button. The property sheet automatically updates itself to show properties for any element selected in the AOT. You don't have to manually open the property sheet for each element; you can simply leave it open and browse the elements. Figure 3-9 shows the property sheet for a *TaxSpec* class. The two columns are the key and value pairs for each property.

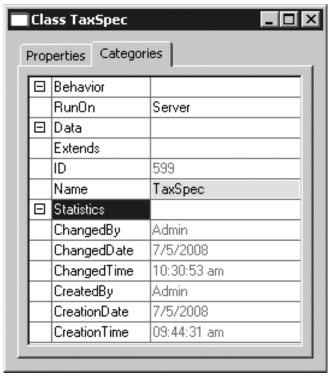
Tip Pressing Esc in the property sheet sets the focus back to your origin.



ID	599
Name	TaxSpec
Extends	
RunOn	Server
CreatedBy	Admin
CreationDate	7/5/2008
CreationTime	09:44:31 am
ChangedBy	Admin
ChangedDate	7/5/2008
ChangedTime	10:30:53 am

FIGURE 3-9 Property sheet for an element in the AOT

Figure 3-10 shows the Categories tab for the class shown in Figure 3-9. Here, related properties are categorized. For elements with many properties, this view can make it easier to find the right property.



⊟ Behavior	
RunOn	Server
⊟ Data	
Extends	
ID	599
Name	TaxSpec
⊟ Statistics	
ChangedBy	Admin
ChangedDate	7/5/2008
ChangedTime	10:30:53 am
CreatedBy	Admin
CreationDate	7/5/2008
CreationTime	09:44:31 am

FIGURE 3-10 Categories tab on the property sheet for an element in the AOT

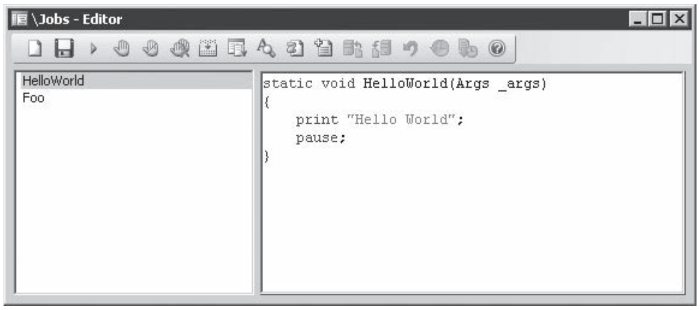
Read-only properties appear in gray. Just like files in the file system, elements contain information about who created them and when they were modified. The Microsoft build process ensures that all elements that ship from Microsoft have the same time and user stamp.

The default sort order places related properties near each other. Categories were introduced in an earlier version of Dynamics AX to make finding properties easier, but you can also sort properties alphabetically by setting a parameter in the Options dialog box. (Thanks to Erik Damgaard, founder of Damgaard Data, the default sorting order is retained in the current version for developers familiar with the original layout of properties.)

You can dock the property sheet on either side of the screen by right-clicking the title bar. Docking ensures that the property sheet is never hidden behind another tool.

X++ Code Editor

You write all X++ code with the X++ code editor. You open the editor by selecting a node in the AOT and pressing Enter. The editor contains two panes. The left pane shows the methods available, and the right pane shows the X++ code for the selected method, as shown in Figure 3-11.



```
static void HelloWorld(Args _args)
{
    print "Hello World";
    pause;
}
```

FIGURE 3-11 X++ code editor

The X++ code editor is a basic text editor that supports color coding and IntelliSense.

Shortcut Keys

Navigation and editing in the X++ code editor use standard shortcuts, as described in Table 3-3.

TABLE 3-3 X++ Code Editor Shortcut Keys

Action	Shortcut	Description
Show Help window	F1	Opens context-sensitive Help for the type or method currently selected in the editor.
Go to next error message	F4	Opens the editor and positions the cursor at the next compilation error, based on the contents of the compiler output window.
Execute current element	F5	Starts the current form, report, or class.
Compile	F7	Compiles the current method.
Toggle a breakpoint	F9	Sets or removes a breakpoint.
List enumerations	F11	Provides a drop-down list of all enumerations available in the system.
List reserved words	Shift+F2	Provides a drop-down list of all reserved words in X++.
List built-in functions	Shift+F4	Provides a drop-down list of all built-in functions available in X++.
Run an editor script	Alt+R	Lists all available editor scripts and lets you select one to execute (such as Send to mail recipient).
Open the Label Editor	Ctrl+Alt+Spacebar	Opens the Label Editor and searches for the selected text.

Action	Shortcut	Description
Show parameter information or IntelliSense list members	Ctrl+Spacebar	Shows parameter information as a ScreenTip or shows members in a drop-down list.
Go to implementation (drill down in code)	Ctrl+Shift+Spacebar	Goes to the implementation of the selected method. Highly useful for fast navigation.
Go to the next method	Ctrl+Tab	Sets focus on the next method in the editor.
Go to the previous method	Ctrl+Shift+Tab	Sets focus on the previous method in the editor.
Enable block selection	Alt+O	Enables block selection, instead of the default line selection.

Editor Scripts

The X++ code editor contains a set of editor scripts that you can invoke by clicking the Script button on the X++ Code Editor toolbar or by pressing Alt+R. Editor scripts provide functionality such as the following:

- Send to mail recipient.
- Send to file.
- Comment or uncomment code.
- Check out element, if version control is enabled.
- Generate code for standard code patterns.
- Open the AOT for the element that owns the method.

Note Code generation allows you to create, in a matter of minutes, a new class with the right constructor method and the right encapsulation of member variables by using *parm* methods. *Parm* methods (*parm* is short for "parameter") are used as simple property getters/setters on classes. Naturally, code generation is carried out in accordance with X++ best practices.

The list of editor scripts is extendable. You can create your own scripts by adding new methods to the *EditorScripts* class.

Label Editor

The term *label* in Dynamics AX simply refers to a localizable text resource. Text resources are used throughout the product as messages to the user, form control labels, column headers, Help text in the status bar, captions on forms, and text on Web forms, to name just a few places. Labels are localizable, meaning that they can be translated into most languages. Because the space requirement for displaying text resources typically depends on the language, you might fear that the actual user interface must be manually localized as well. However, with IntelliMorph technology, the user interface is dynamically rendered and honors any space requirements imposed by localization.

The technology behind the label system is simple. All text resources are kept in a Unicode-based label file that must have a three-letter identifier. The label file is located in the application folder (Program Files\Microsoft Dynamics AX\50\Application\Appl\Standard) and follows this naming convention:

Ax<Label file identifier><Locale>.ALD

The following are two examples, the first showing U.S. English and the second a Danish label file:

Axsysen-us.ALD

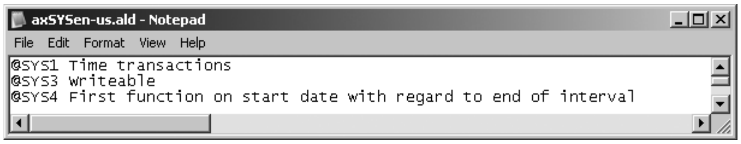
Axtstda.ALD

Each text resource in the label file has a 32-bit integer label ID, label text, and an optional label description. The structure of the label file is very simple:

@*<Label file identifier><Label ID> <Label text>*

[*Label description*]

Figure 3-12 shows an example of a label file.



```
@SYS1 Time transactions
@SYS3 Writeable
@SYS4 First function on start date with regard to end of interval
```

FIGURE 3-12 Label file opened in Microsoft Notepad showing a few labels from the en-us label file

This simple structure allows for localization outside Dynamics AX using third-party tools.

After the localized label files are in place, the user can choose a language in the Options dialog box. When the language is changed, the user must close and restart the Dynamics AX client.

You can create new label files by using the Label File Wizard, which you access from the Microsoft Dynamics AX drop-down menu by clicking Tools\Development Tools\Wizards\Label File Wizard. The wizard guides you through the steps for adding a new label file or a new language to an existing label file. After you run the wizard, the label file is ready to use.

Note You can use any combination of three letters when naming a label file, and you can use any label file from any layer. A common misunderstanding is that the label file identifier must be the same as the layer in which it is used. This misunderstanding is caused by the Microsoft label file identifiers. Dynamics AX ships with a SYS layer and a label file named SYS; service packs contain a SYP layer and a label file named SYP. This naming standard was chosen because it is simple, easy to remember, and easy to understand. Dynamics AX doesn't impose any limitations on the label file name.

The following are tips for working with label files:

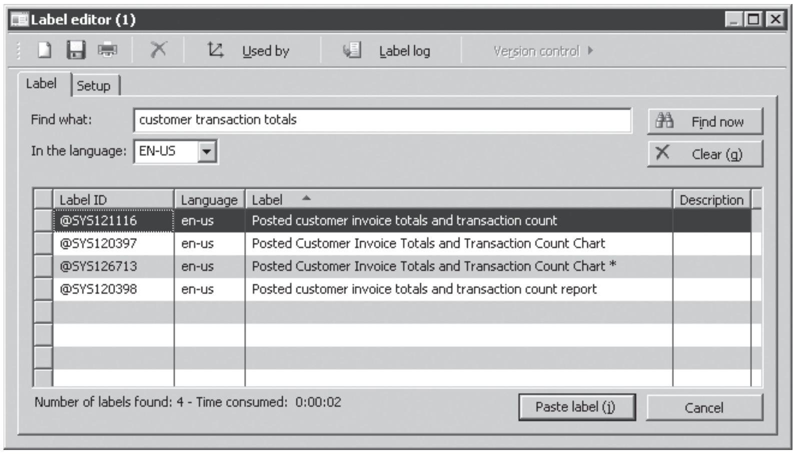
- When naming a label file, choose a three-letter ID that has a high chance of being unique, such as your company's initials. Don't choose the name of the layer, such as VAR or USR. Eventually, you'll likely merge two separately developed features into the same installation, a task that will be more difficult if the label files collide.
- Feel free to reference labels in the Microsoft-provided label files, but avoid making changes to labels in these label files, because they are updated with each new version of Dynamics AX.

Creating a New Label

You use the Label Editor to create new labels. You can start it using any of the following procedures:

- Clicking Tools\Development Tools\Label\Label Editor from the Microsoft Dynamics AX drop-down menu
- Clicking the Lookup Label/Text button on the X++ code editor toolbar
- Clicking the Lookup button on text properties in the property sheet

The Label Editor (shown in Figure 3-13) allows you to find existing labels. Reusing a label is often preferable to creating a new one. You can create a new label by pressing Ctrl+N or by clicking the New button.



Find what: customer transaction totals

In the language: EN-US

Label ID	Language	Label	Description
@SYS121116	en-us	Posted customer invoice totals and transaction count	
@SYS120397	en-us	Posted Customer Invoice Totals and Transaction Count Chart	
@SYS126713	en-us	Posted Customer Invoice Totals and Transaction Count Chart *	
@SYS120398	en-us	Posted customer invoice totals and transaction count report	

Number of labels found: 4 - Time consumed: 0:00:02

FIGURE 3-13 Label Editor

In addition to allowing you to find and create new labels, the Label Editor can also show where a label is used. It also logs any changes to each label.

The following are tips to consider when creating and reusing labels:

- When reusing a label, make sure that the label meaning is what you intend it to be in all languages. Some words are homonyms, meaning words that have many meanings, and they naturally translate into many different words in other languages. For example, the English word *can* is both a verb and a noun. The description column describes the intended meaning of the label.
- When creating new labels, make sure to use complete sentences or other stand-alone words or phrases in each label. Don't construct complete sentences by concatenating labels with one or few words, because the order of words in a sentence differs from one language to another.

Referencing Labels from X++

In the MorphX design environment, labels are referenced in the format @<*LabelFileIdentifier*><*LabelID*>. If you don't want a label reference to automatically convert to the label text, you can use the *literalStr* function. When a placeholder is needed to display the value of a variable, you can use the *strFmt* function and a string containing %*n*, where *n*> = 1. Placeholders can also be used within labels. The following code shows a few examples.

```
// prints: Time transactions
print "@SYS1";

// prints: @SYS1
print literalStr("@SYS1");

// prints: Microsoft Dynamics is a Microsoft brand
print strFmt("%1 is a %2 brand", "Microsoft Dynamics", "Microsoft");
```

The following are some best practices to consider when referencing labels from X++:

- You should always create user interface text by using a label. When referencing labels from X++ code, use double quotation marks.
- You should never create system text such as file names by using a label. When referencing system text from X++ code, use single quotation marks. You can place system text in macros to make it reusable.

Using single and double quotation marks to differentiate between system text and user interface text allows the Best Practices tool to find and report any hard-coded user interface text. The Best Practices tool is described in depth later in this chapter.

Visual Form Designer and Visual Report Designer

MorphX has two visual designers, one for forms and one for reports, that allow you to drag controls onto the design surface in WYSIWYG fashion. IntelliMorph determines the actual position of the controls, so you can’t place them precisely.

You can override these layout restrictions by changing property values, such as Top, Left, Height, and Width, from Auto to a fixed value, allowing the visual designers to lay out the controls. However, doing so interferes with the automated layout attempted by IntelliMorph, which means that there is no guarantee that your forms and reports will display well when translated, configured, secured, and personalized.

It is a best practice to let IntelliMorph control all the layout. (More detailed information about IntelliMorph is in Chapter 13, “Configuration and Security.”) Most forms and reports that ship with Dynamics AX are designed by using the AOT. When the visual designer is opened, a tree structure of the design is displayed, making it fairly simple to add new controls to the design. You can either drag fields or field groups from the data source to the design or right-click the design and choose New Control.

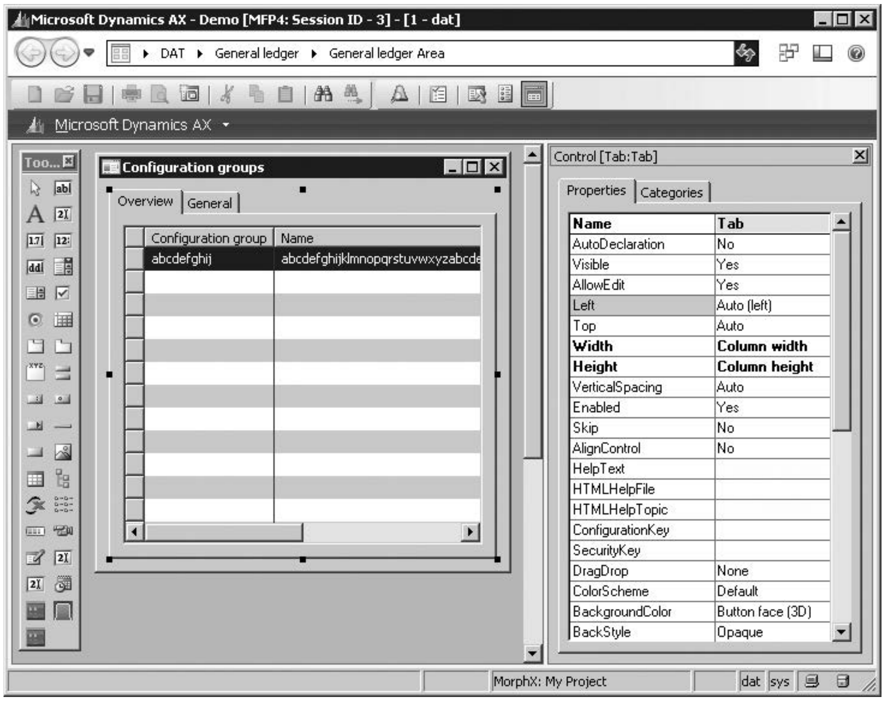
Note IntelliMorph and MorphX treat form and report designs as hierarchical structures. A control can be next to another control or inside a group control. This arrangement makes a lot of sense for business applications. If you require controls to be on top of one another, you must use absolute pixel positions. The order of the controls in the AOT mandates the z-order—that is, the order in which controls are virtually stacked in the display.

You can use a Report Wizard, accessed from the Microsoft Dynamics AX drop-down menu at Tools\Development Tools\Wizards, to help you create reports. The wizard guides you through the process step by step, allowing you to specify data sources, sorting, grouping, layout, and other settings before producing a report in the AOT. You can read more about developing reports in Chapter 11, “Reporting in Dynamics AX.”

Visual Form Designer

The designers can be helpful tools for learning how the IntelliMorph layout scheme works. If you have the Visual Form Designer open when you start designing a form, you immediately see what the form will look like, even when it is modified in the AOT. In fact, after creating a few forms, you'll probably feel so confident of the power of IntelliMorph and the effectiveness of designing forms in the AOT that you'll only rarely use the Visual Form Designer.

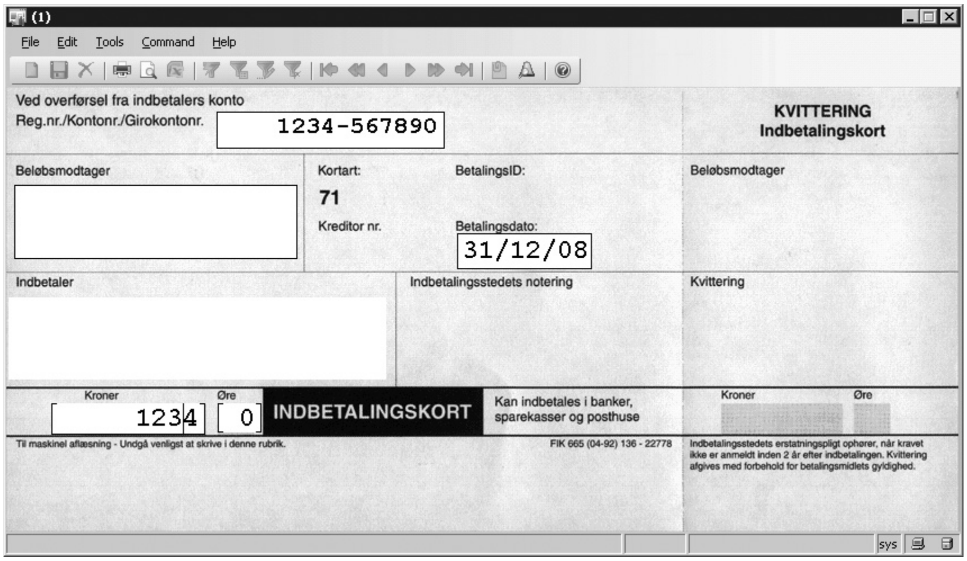
You open the Visual Form Designer by right-clicking a form's design in the AOT and selecting Edit. The designer is shown in design mode in Figure 3-14. Next to the form is a toolbar with all the available controls, which can be dragged onto the form's surface. You can also see the property sheet showing the selected control's properties.



Name	Tab
AutoDeclaration	No
Visible	Yes
AllowEdit	Yes
Left	Auto (left)
Top	Auto
Width	**Column width**
Height	**Column height**
VerticalSpacing	Auto
Enabled	Yes
Skip	No
AlignControl	No
HelpText	
HTMLHelpFile	
HTMLHelpTopic	
ConfigurationKey	
SecurityKey	
DragDrop	None
ColorScheme	Default
BackgroundColor	Button face (3D)
BackStyle	Opaque

FIGURE 3-14 Visual Form Designer

One interesting form that overrides IntelliMorph is the form *tutorial_Form_freeform*. Figure 3-15 shows how a scanned bitmap of a payment form is used as a background image for the form, and the controls positioned where data entry is needed.



Ved overførsel fra indbetalers konto
Reg.nr./Kontonr./Girokontonr. 1234-567890

KVITTERING
Indbetalingskort

Beløbsmodtager

Kortart: 71

BetalingsID:

Kreditor nr.

Betalingsdato: 31/12/08

Beløbsmodtager

Indbetaler

Indbetalingsstedets notering

Kvittering

Kroner 1234 Øre 0

INDBETALINGSKORT

Kan indbetales i banker, sparekasser og posthuse

Kroner Øre

Til maskinel aflæsning - Undgå venligst at skrive i denne rubrik.

FIK 665 (04-92) 136 - 22778

Indbetalingsstedets erstatningspligt ophører, når kravet ikke er anmeldt inden 2 år efter indbetalingen. Kvittering afgives med forbehold for betalingsmidlets gyldighed.

FIGURE 3-15 Nonstandard form that uses a bitmap background

Visual Report Designer

The majority of MorphX reports fall into two categories—internal and external. Requirements for reports used internally in a company are often more relaxed than requirements for external reports. External reports are often part of the company's face to the outside world. An invoice report is a classic example of an external report.

Leveraging the features of IntelliMorph, internal reports typically follow an autodesign that allows the consumer of the report to add and remove columns from the report and control its orientation, font, and font size.

External reports typically use a generated design, which effectively overrides IntelliMorph. So for external reports, the Visual Report Designer is clearly preferable. Often, external reports are printed on preprinted paper containing, for example, the company's letterhead, so the ability to easily control the exact position of each control is essential.

You create a generated design from an autodesign by right-clicking a design node of a report in the AOT and selecting Generate Design. You can open the Visual Report Designer by right-clicking a generated design and selecting Edit. As shown in Figure 3-16, each control can be moved freely, and new controls can be added.

Notice the zoom setting in the lower-right corner of Figure 3-16. This setting allows you to get a close-up view of the report and, with a steady hand, position each control exactly where you want it.

The rendering subsystem of the report engine can print only generated designs because it requires all controls to have fixed positions. If a report has only an autodesign, the report engine generates a design in memory before printing.

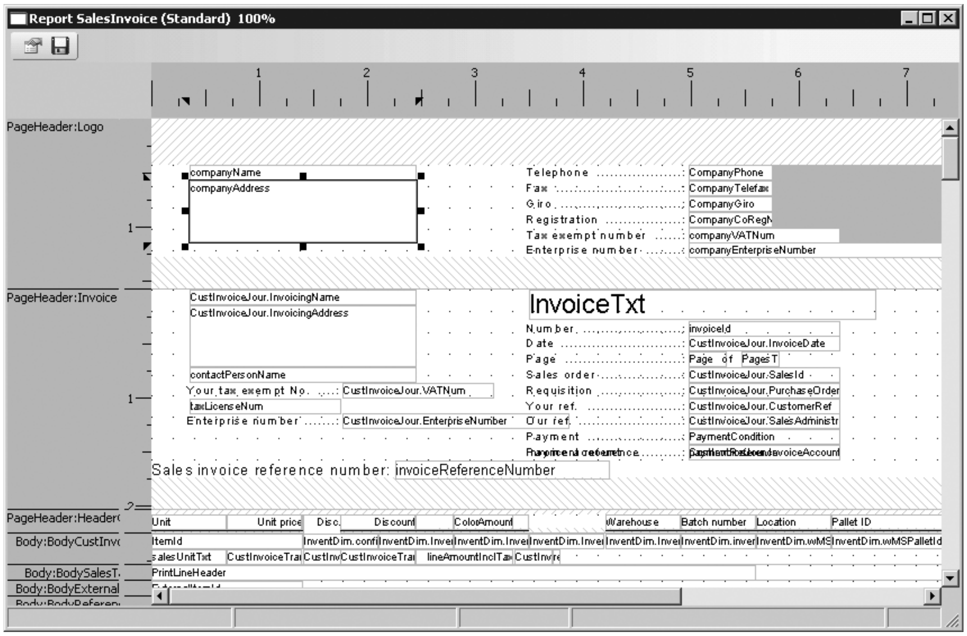


FIGURE 3-16 Visual Report Designer

Code Compiler

Whenever you make a change to X++ code, you must recompile, just as you would in any other development language. You start the recompile by pressing F7 in the X++ code editor. Your code also recompiles whenever you close the editor or save a dirty element.

The compiler also produces a list of the following information:

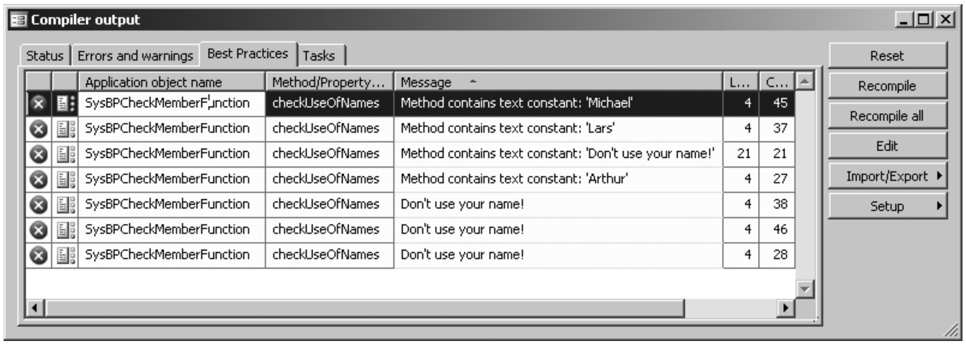
- **Compiler errors** These prevent code from compiling and should be fixed as soon as possible.
- **Compiler warnings** These typically indicate that something is wrong in the implementation. See Table 3-4, later in this section, for a list of compiler warnings. Compiler warnings can and should be addressed. Check-in attempts with compiler warnings are rejected.
- **Tasks (also known as to-dos)** The compiler picks up single-line comments that start with TODO. These comments can be useful during development for adding reminders, but you should use them only in cases in which implementation can't be completed.

For example, you might use a to-do comment when you're waiting for a check-in from another developer. You should avoid using to-do comments just to postpone work. For a developer, there is nothing worse than debugging an issue at a customer site and finding a to-do comment indicating that the issue was already known but overlooked.

Note Unlike other languages, X++ requires that you compile only code you've modified. This is because the intermediate language the compiler produces is persisted along with the X++ code and metadata. Of course, your changes can require other methods consuming your code to be changed and recompiled if, for example, you rename a method or modify its parameters. If the consumers are not recompiled, a run-time error is thrown when they are invoked. This means that you can execute your business application even when compile errors exist, as long as you don't use the code that can't compile. You should always compile the entire AOT when you consider your changes complete, and you should fix any compilation errors found.

- **Best practice deviations** The Best Practices tool carries out more complex validations. See the section "Best Practices Tool" later in this chapter for more information.

The Compiler Output dialog box provides access to everything reported during compilation, as shown in Figure 3-17. Each category of findings has a dedicated tab: Status, Errors And Warnings, Best Practices, and Tasks. Each tab contains the same information for each issue that the compiler detects—a description of the issue and its location. For example, the Status tab shows a count of the detected issues.



Application object name	Method/Property...	Message	L...	C...
SysBPCheckMemberFunction	checkUseOfNames	Method contains text constant: 'Michael'	4	45
SysBPCheckMemberFunction	checkUseOfNames	Method contains text constant: 'Lars'	4	37
SysBPCheckMemberFunction	checkUseOfNames	Method contains text constant: 'Don't use your name!'	21	21
SysBPCheckMemberFunction	checkUseOfNames	Method contains text constant: 'Arthur'	4	27
SysBPCheckMemberFunction	checkUseOfNames	Don't use your name!	4	38
SysBPCheckMemberFunction	checkUseOfNames	Don't use your name!	4	46
SysBPCheckMemberFunction	checkUseOfNames	Don't use your name!	4	28

FIGURE 3-17 Compiler Output dialog box

You can export compile results. This capability is useful if you want to share the list of issues with team members. The exported file is an HTML file that can be viewed in Microsoft Internet Explorer or re-imported into the Compiler Output dialog box in another Dynamics AX session.

In the Compiler Output dialog box, click Setup and then click Compiler to define the types of issues that the compiler should report. Compiler warnings are grouped into four levels, as shown in Table 3-4.

TABLE 3-4 Compiler Warnings

Warning Message	Level
Break statement found outside legal context	1
The new method of a derived class does not call super()	1
The new method of a derived class may not call super()	1
Function never returns a value	1
Not all paths return a value	1
Assignment/comparison loses precision	1
Unreachable code	2
Empty compound statement	3
Class names should start with an uppercase letter	4
Member names should start with a lowercase letter	4

Dynamics AX SDK

Constructing quality software has become a daunting task in the 21st century. Many new competencies are expected of the developer, and mastering them fully and at all times is nearly impossible. Today you must write code conforming to many technical requirements, including security, localization, internationalization, customization, performance, accessibility, reliability, scalability, compatibility, supportability, interoperability, and so on. The list seems to grow with each software revision, and keeping up with all of these competencies is increasingly difficult.

Microsoft Dynamics AX 2009 includes a software development kit (SDK) that explains how to satisfy these requirements when you use MorphX. You can access the SDK from the application Help menu under Developer Help. We highly recommend that you read the Developer Help section—it's not just for novices but also for experienced developers, who will find that the content has been extensively revised for Dynamics AX 2009. The SDK is frequently refreshed with new content, so you might want to check it often on MSDN.

Among other critical information, the Developer Help section of the SDK includes an important discussion on conforming to best practices in Dynamics AX. The motivation for conforming to best practices should be obvious to anyone. Constructing code that follows proven standards and patterns can't guarantee a project's success, but it certainly minimizes the risk of failure. To ensure your project's success, you should learn, conform to, and advocate best practices within your group.

The following are a few benefits of following best practices:

- You avoid less-than-obvious pitfalls. Following best practices helps you avoid many obstacles, even those that surface only in border scenarios that would otherwise be difficult and time consuming to detect and test. Using best practices allows you to leverage the combined experiences of Dynamics AX expert developers.

- The learning curve is flattened. When you perform similar tasks in a standard way, you are more comfortable in an unknown area of the application. Consequently, adding new resources to a project is more cost efficient, and downstream consumers of the code are able to make changes more readily.
- You are making a long-term investment. Code that conforms to standards is less likely to require rework during an upgrade process, whether you're upgrading to Dynamics AX 2009, installing service packs, or upgrading to future releases.
- You are more likely to ship on time. Most of the problems you face when implementing a solution in Dynamics AX have been solved at least once before. Choosing a proven solution results in faster implementation and less regression. You can find solutions to known problems in both the Developer Help section of the SDK and the code base.

Best Practices Tool

A powerful supplement to the best practices discussion in the SDK is the Best Practices tool. This tool is the MorphX version of a static code analysis tool, similar to FxCop for the Microsoft .NET Framework and PREfix and PREfast for C and C++. The Best Practices tool is embedded in the compiler, and the results are located on the Best Practices tab of the Compiler Output dialog box.

The purpose of static code analysis is to automatically detect defects in the code. The longer a defect exists, the more costly it becomes to fix—a bug found in the design phase is much cheaper to correct than a bug in shipped code running at several customer sites. The Best Practices tool allows any developer to run an analysis of his or her code and application model to ensure that it conforms to a set of predefined rules. Developers can run analysis during development, and they should always do so before implementations are tested.

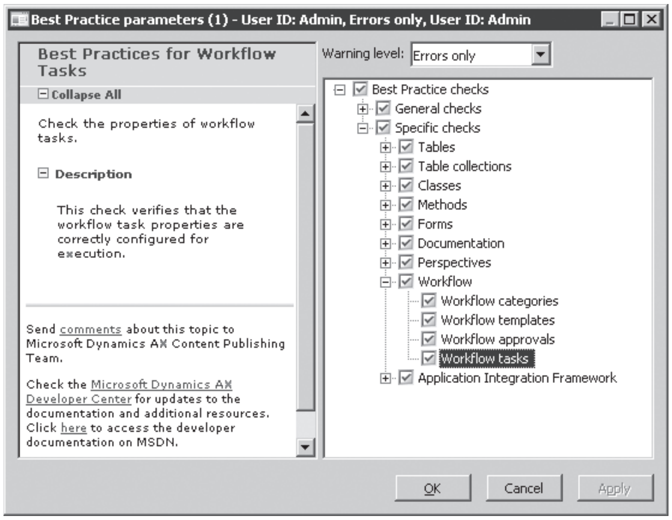
The Best Practices tool displays deviations from the best practice rules, as shown in Figure 3-17. Double-clicking a line on the Best Practices tab opens the X++ code editor on the violating line of code.

Understanding Rules

The Best Practices tool includes about 350 rules, a small subset of the best practices mentioned in the SDK. You can define the best practice rules that you want to run in the Best Practice Parameters dialog box: from the Microsoft Dynamics AX drop-down menu, click Tools\Options and then the Best Practices button.

Note You must set the compiler error level to 4 if you want best practice rule violations to be reported. To turn off the Best Practices tool, click Tools\Options\Compiler, and then set the diagnostic level to less than 4.

The best practice rules are divided into categories. By default, all categories are turned on, as shown in Figure 3-18.


Best Practice parameters (1) - User ID: Admin, Errors only, User ID: Admin

Best Practices for Workflow Tasks

Collapse All

Check the properties of workflow tasks.

Description

This check verifies that the workflow task properties are correctly configured for execution.

Send comments about this topic to Microsoft Dynamics AX Content Publishing Team.

Check the Microsoft Dynamics AX Developer Center for updates to the documentation and additional resources. Click here to access the developer documentation on MSDN.

Warning level: Errors only

- ☑ Best Practice checks
  - ☑ General checks
  - ☑ Specific checks
    - ☑ Tables
    - ☑ Table collections
    - ☑ Classes
    - ☑ Methods
    - ☑ Forms
    - ☑ Documentation
    - ☑ Perspectives
    - ☑ Workflow
      - ☑ Workflow categories
      - ☑ Workflow templates
      - ☑ Workflow approvals
      - ☑ Workflow tasks
    - ☑ Application Integration Framework

FIGURE 3-18 Best Practice Parameters dialog box

The best practice rules are divided into three levels of severity:

- **Errors** The majority of the rules focus on errors. Any check-in attempt with a best practice error is rejected. You must take all errors seriously and fix them as soon as possible.
- **Warnings** Follow a 95/5 rule for warnings. This means that you should treat 95 percent of all warnings as errors; the remaining 5 percent constitute exceptions to the rule. You should provide valid explanations in the design document for all warnings you choose to ignore.
- **Information** In some situations, your implementation might have a side effect that isn't obvious to you or the user (e.g., if you're assigning a value to a variable but you never use the variable again). These are typically reported as information messages.

Suppressing Errors and Warnings

The Best Practices tool allows you to suppress errors and warnings. A suppressed best practice deviation is reported as information. This gives you a way to identify the deviation as reviewed and accepted. To identify a suppressed error or warning, place a line containing the following text just before the deviation.

```
//BP Deviation Documented
```

Only a small subset of the best practice rules can be suppressed. Use the following guidelines for selecting which rules to suppress:

- Where exceptions exist that are impossible to detect automatically, you should examine each error to ensure the correct implementation. Dangerous APIs are often responsible for such exceptions. A dangerous API is an API that can compromise a system's security when used incorrectly. If a dangerous API is used, a suppressible error is reported. You are allowed to use some so-called dangerous APIs when you take certain precautions, such as using code access security. You can suppress the error after you apply the appropriate mitigations.
- About 5 percent of all warnings are false positives and can be suppressed. Note that only warnings caused by actual code can be suppressed, not warnings caused by metadata.

After you set up the best practices, the compiler automatically runs the best practices check whenever an element is compiled. The results are displayed on the Best Practices tab in the Compiler Output dialog box.

Adding Custom Rules

The X++ Best Practices tool allows you to create your own set of rules. The classes used to check for rules are named *SysBPCheck<ElementKind>*. You call the *init*, *check*, and *dispose* methods once for each node in the AOT for the element being compiled.

One of the most interesting classes is *SysBPCheckMemberFunction*, which is called for each piece of X++ code whether it is a class method, form method, macro, or other method. For example, if developers don't want to include their names in the source code, you can implement a best practice check by creating the following method on the *SysBPCheckMemberFunction* class.

```
protected void checkUseOfNames()
{
    #Define.MyErrorCode(50000)
    container devNames = ["Arthur", "Lars", "Michael"];
    int i;
    int j;
    int pos;
    str line;
    int lineLen;

    for (i=scanner.lines(); i; i--)
    {
        line = scanner.sourceLine(i);
        lineLen = strlen(line);
        for (j=conlen(devNames); j; j--)
        {
                pos = strscan(line, conpeek(devNames, j), 1, lineLen);
                if (pos)
                {
                    sysBPCheck.addError(#MyErrorCode, i, pos,
                        "Don't use your name!");
                }
            }
        }
    }
```

To enlist the rule, make sure to call the preceding method from the *check* method. Compiling this sample code results in the best practice errors shown in Table 3-5.

TABLE 3-5 Best Practice Errors in *checkUseOfNames*

Message	Line	Column
Method contains text constant: 'Arthur'	4	27
Don't use your name!	4	28
Method contains text constant: 'Lars'	4	37
Don't use your name!	4	38
Method contains text constant: 'Michael'	4	45
Don't use your name!	4	46
Method contains text constant: 'Don't use your name!'	20	59

In a real-world implementation, names of developers would probably be read from a file. Make sure to cache the names to prevent the compiler from going to the disk to read the names for each method being compiled.

Debugger

Like most development environments, MorphX features a debugger. The debugger is a stand-alone application, not part of the Dynamics AX shell like the rest of the tools mentioned in this chapter. As a stand-alone application, the debugger allows you to debug X++ in any of the Dynamics AX components in the following list:

- Microsoft Dynamics AX client
- Application Object Server (AOS)
- Enterprise Portal
- Business Connector

Using the Debugger

For the debugger to start, a breakpoint must be hit during execution of X++ code. You set breakpoints by using the X++ code editor in the Microsoft Dynamics AX client. The debugger starts automatically when any component hits a breakpoint.

You must enable debugging for each component as follows:

- In the Microsoft Dynamics AX client, click the Microsoft Dynamics AX drop-down menu, point to Tools and then Options. On the Development tab, select When Breakpoint in the Debug Mode list.
- For the AOS, open the Microsoft Dynamics AX Server Configuration utility under Start\Administrative Tools. Create a new configuration (if necessary), and select the check box labeled Enable Breakpoints To Debug X++ Code Running On This Server.
- For Batch jobs, open the Microsoft Dynamics AX Server Configuration utility under Start\Administrative Tools. Create a new configuration (if necessary), and select the check box labeled Enable Global Breakpoints To Debug X++ Code Running In Batch Jobs.
- For Enterprise Portal and Business Connector, open the Microsoft Dynamics AX Configuration utility under Start\Administrative Tools. Select one of two check boxes on the Developer tab: Enable User Breakpoints For Debugging Code Running In The Business Connector or Enable Global Breakpoints For Debugging Code Running In The Business Connector Or Client. The latter is useful for debugging incoming Web requests.

Caution We recommend that you do not enable any of the debugging capabilities in a live environment. If you do, execution will stop when it hits a breakpoint, and users will experience a hanging client.

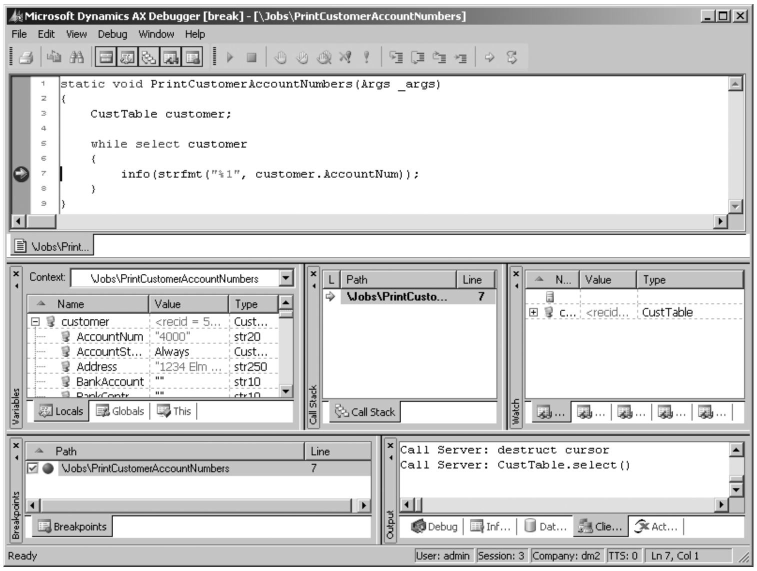
The debugger allows you to set and remove breakpoints by pressing F9. You can set a breakpoint on any line you want. If you set a breakpoint on a line without an X++ statement, however, the breakpoint will be triggered on the next X++ statement in the method. A breakpoint on the last brace will never be hit.

You can enable or disable a breakpoint by pressing Ctrl+F9. For a list of all breakpoints, press Shift+F9.

Breakpoints are persistent in the *SysBreakpoints* database table. Each developer has his or her own set of breakpoints. This means that your breakpoints are not cleared when you close Dynamics AX and that other Dynamics AX components can access them and break where you want them to.

Debugger Interface

The main window in the debugger initially shows the point in the code where a breakpoint was hit. You can control execution one step at a time while variables and other aspects are inspected. Figure 3-19 shows the debugger opened to a breakpoint with all the windows enabled.



Microsoft Dynamics AX Debugger [break] - [\Jobs\PrintCustomerAccountNumbers]

```
static void PrintCustomerAccountNumbers(Args _args)
{
    CustTable customer;

    while select customer
    {
        info(strfmt("%1", customer.AccountNum));
    }
}
```

Context: \Jobs\PrintCustomerAccountNumbers

Name	Value	Type
customer	<recid = 5...	Cust...
AccountNum	"4000"	str20
AccountSt...	Always	Cust...
Address	"1234 Elm ...	str250
BankAccount	""	str10

L	Path	Line
⇨	\Jobs\PrintCusto...	7

N...	Value	Type
c...	<recid...	CustTable

Path	Line
✓ ● \Jobs\PrintCustomerAccountNumbers	7

```
Call Server: destruct cursor
Call Server: CustTable.select()
```

Ready | User: admin | Session: 3 | Company: dm2 | TTS: 0 | Ln 7, Col 1

FIGURE 3-19 Debugger with all windows enabled

In the following subsections, we briefly describe the debugger's various windows and some of its other features.

Main Window

The main debugger window shows the current X++ code. Each variable has a ScreenTip that reveals its value. You can drag the next-statement pointer in the left margin. This pointer is particularly useful if the execution path isn't what you expected or if you want to repeat a step.

Variables Window

In this window, local, global, and member variables are shown. Local variables are variables in scope at the current execution point. Global variables are the global classes that are always instantiated: *Appl*, *Infolog*, *ClassFactory*, and *VersionControl*. Member variables make sense only on classes, and they show the class member variables.

The Variables window shows the name, value, and type of each variable. If a variable is changed during execution stepping, it is marked in red. Each variable is shown associated with a client or server icon. You can modify the value of a variable by double-clicking the value.

Tip As a developer, you might want to provide more information in the value field than what is provided by default. For a class, the defaults are *New* and *Null*. You can change the defaults by overriding the *toString* method. If your class doesn't explicitly extend *object* (the base class of all classes), you must add a new method named *toString*, returning *str* and taking no parameters, to implement this functionality.

Call Stack Window

The Call Stack window shows the code path followed to arrive at a particular execution point. Clicking a line in the Call Stack window opens the code in the Code window and updates the local Variables window. A client or server icon indicates the tier on which the code is executed.

Watch Window

In the Watch window, you can inspect variables without the scope limitations of the Variables window. You can drag a variable here from the Code window or the Variables window.

The Watch window shows the name, value, and type of the variables. Five different Watch windows are available. You can use these to group the variables you're watching in the way that you prefer.

Breakpoints Window

The Breakpoints window lists all your breakpoints. You can delete, enable, and disable the breakpoints via this window.

Output Window

The Output window shows the traces that are enabled and the output sent to the Infolog application framework, which we introduced in Chapter 1. The Output window includes the following pages:

- **Debug** You can instrument your X++ code to trace to this page by using the *printDebug* static method on the *Debug* class.
- **Infolog** This page contains messages in the queue for the Infolog.
- **Database, Client/Server, and ActiveX Trace** Any traces enabled on the Development tab in the Options dialog box appear on these pages.

Status Bar

The status bar at the bottom of the debugger offers the following important context information:

- **Current user** The ID of the user who is logged on to the system. This information is especially useful when you are debugging incoming Web requests.
- **Current session** The ID of the session on the AOS.
- **Current company accounts** The ID of the current company accounts.
- **Transaction level** The current transaction level. When reaching zero, the transaction is committed.

Debugger Shortcut Keys

Table 3-6 lists the most important shortcut keys available in the debugger.

TABLE 3-6 Debugger Shortcut Keys

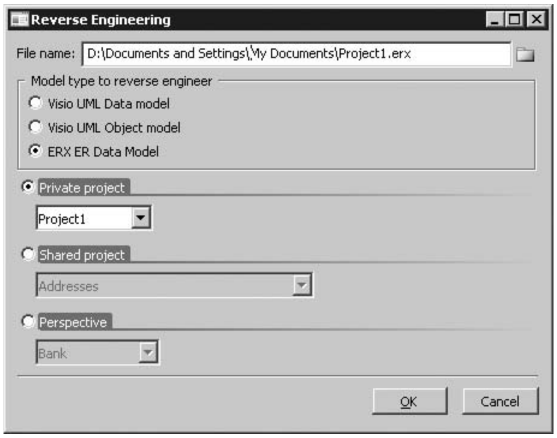
Action	Shortcut	Description
Run	F5	Continue execution
Stop debugging	Shift+F5	Break execution
Step over	F10	Step over next statement
Run to cursor	Ctrl+F10	Continue execution but break at the cursor's position
Step into	F11	Step into next statement
Step out	Shift+F11	Step out of method
Toggle breakpoint	Shift+F9	Insert or remove breakpoint
Variables window	Ctrl+Alt+V	Open or close Variables window
Call Stack window	Ctrl+Alt+C	Open or close Call Stack window
Watch window	Ctrl+Alt+W	Open or close Watch window
Breakpoints window	Ctrl+Alt+B	Open or close Breakpoints window
Output window	Ctrl+Alt+O	Open or close Output window

Visio Reverse Engineering Tool

Dynamics AX allows you to generate Visio models from existing metadata. Considering the amount of metadata available in Dynamics AX 2009 (more than 30,000 elements and more than 7 million lines of text when exported), it's practically impossible to get a clear view of how the elements relate to each other just by using the AOT. The Visio Reverse Engineering tool is a great aid when you need to visualize metadata.

Note You must have Office Visio 2003 or later installed to use the Visio Reverse Engineering tool.

The Reverse Engineering tool can generate a Unified Modeling Language (UML) data model, a UML object model, or an entity relationship data model, including all elements from a private or shared project. To open the tool, right-click a project or a perspective, point to Add-Ins, and then click Reverse Engineer. You can also open the tool by selecting Reverse Engineer from the Development Tools menu. In the dialog box shown in Figure 3-20, you must specify a file name and model type.


Reverse Engineering

File name: D:\Documents and Settings\My Documents\Project1.erx

Model type to reverse engineer

( ) Visio UML Data model
( ) Visio UML Object model
(●) ERX ER Data Model

(●) Private project: Project1
( ) Shared project: Addresses
( ) Perspective: Bank

FIGURE 3-20 Visio Reverse Engineering dialog box

When you click OK, the tool uses the metadata for all elements in the project to generate a Visio document that opens automatically in Visio. You can drag elements from the Visio Model Explorer onto the drawing surface, which is initially blank. Any relationship between two elements is automatically shown.

UML Data Model

When generating a UML data model, the Reverse Engineering tool looks for tables in the project. The UML model contains a class for each table and view in the project and its attributes and associations. Figure 3-21 shows a class diagram with the *CustTable* (Customers), *InventTable* (Inventory Items), *SalesTable* (Sales Order Header), and *SalesLine* (Sales Order Line) tables. To simplify the diagram, some attributes have been removed.

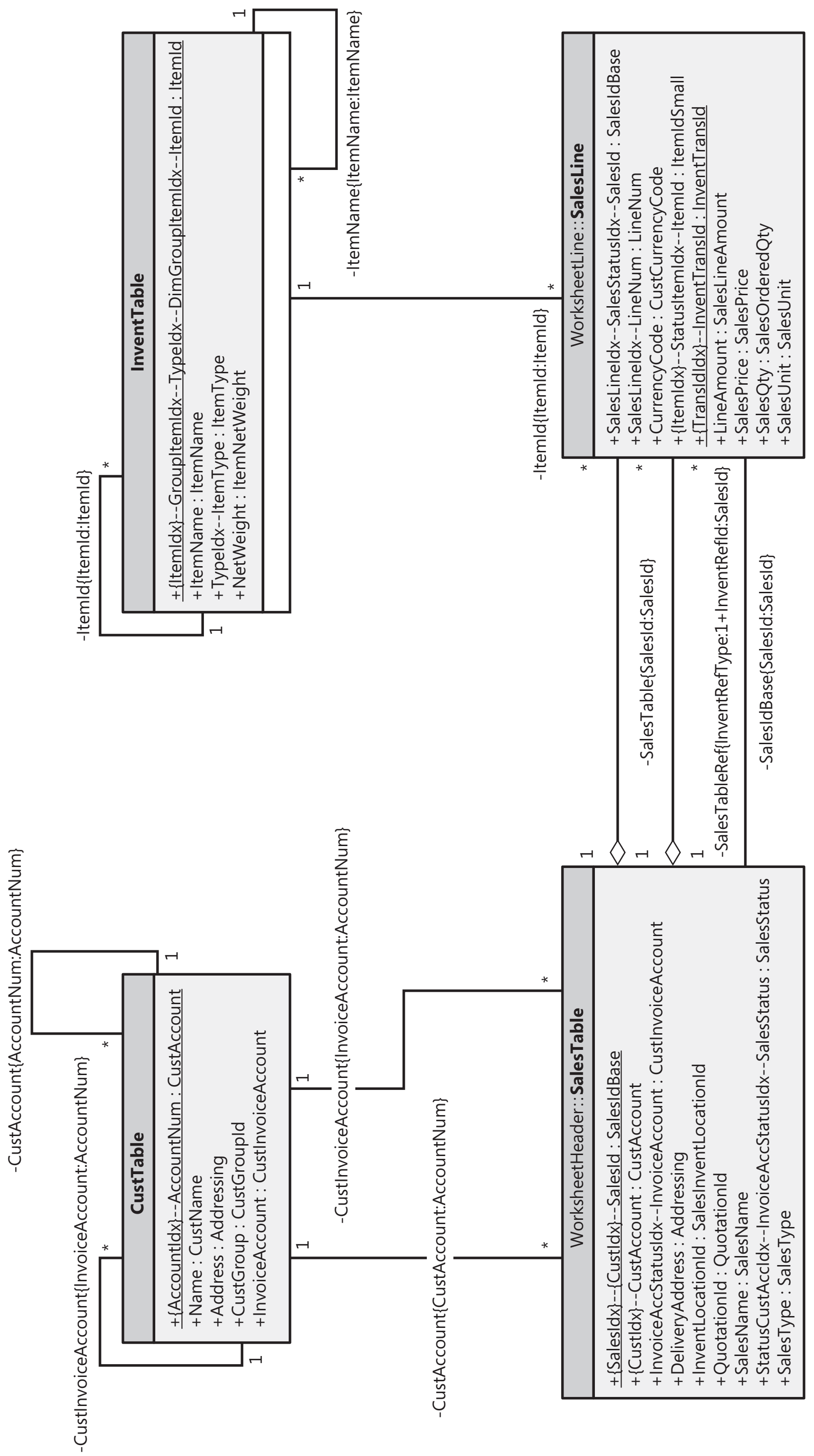

InventTable
+{ItemIdx}--GroupItemIdx--TypeIdx--DimGroupItemIdx--ItemId : ItemId
+ItemName : ItemName
+TypeIdx--ItemType : ItemType
+NetWeight : ItemNetWeight
-ItemId{ItemId:ItemId}
-ItemName{ItemName:ItemName}
WorksheetLine::SalesLine
+SalesLineIdx--SalesStatusIdx--SalesId : SalesIdBase
+SalesLineIdx--LineNum : LineNum
+CurrencyCode : CustCurrencyCode
+{ItemIdx}--StatusItemIdx--ItemId : ItemIdSmall
+{TransIdIdx}--InventTransId : InventTransId
+LineAmount : SalesLineAmount
+SalesPrice : SalesPrice
+SalesQty : SalesOrderedQty
+SalesUnit : SalesUnit
-ItemId{ItemId:ItemId}
-SalesTable{SalesId:SalesId}
-SalesTableRef{InventRefType:1+InventRefId:SalesId}
-SalesIdBase{SalesId:SalesId}
CustTable
+{AccountIdx}--AccountNum : CustAccount
+Name : CustName
+Address : Addressing
+CustGroup : CustGroupId
+InvoiceAccount : CustInvoiceAccount
-CustAccount{AccountNum:AccountNum}
-CustInvoiceAccount{InvoiceAccount:AccountNum}
-CustInvoiceAccount{InvoiceAccount:AccountNum}
-CustAccount{CustAccount:AccountNum}
WorksheetHeader::SalesTable
+{SalesIdx}--{CustIdx}--SalesId : SalesIdBase
+{CustIdx}--CustAccount : CustAccount
+InvoiceAccStatusIdx--InvoiceAccount : CustInvoiceAccount
+DeliveryAddress : Addressing
+InventLocationId : SalesInventLocationId
+QuotationId : QuotationId
+SalesName : SalesName
+StatusCustAccIdx--InvoiceAccStatusIdx--SalesStatus : SalesStatus
+SalesType : SalesType
1
*

FIGURE 3-21 UML data model diagram

The UML model also contains referenced tables and all extended data types, base enumerations, and X++ data types. You can include these items in your diagrams without having to run the Reverse Engineering tool again.

Fields in Dynamics AX are generated as UML attributes. All attributes are marked as public because of the nature of fields in Dynamics AX. Each attribute also shows the type. The primary key field is underlined. If a field is a part of one or more indexes, the names of the indexes are prefixed to the field name; if the index is unique, the index name is noted in brackets.

Relationships in Dynamics AX are generated as UML associations. The aggregation property of the association is set based on two conditions in metadata:

- If the relationship is validating (the validate property is set to Yes), the aggregation property is set to shared. This is also known as UML aggregation, visualized by a white diamond.
- If a cascading delete action exists between the two tables, a composite association is added to the model. A cascading delete action ties the life span of two or more tables and is visualized by a black diamond.

The end name on associations is the name of the Dynamics AX relationship, and the names and types of all fields in the relationship appear in brackets.

UML Object Model

When generating an object model, the Reverse Engineering tool looks for Dynamics AX classes, tables, and interfaces in the project. The UML model contains a class for each Dynamics AX table and class in the project and an interface for each Dynamics AX interface. The UML model also contains attributes and operations, including return types, parameters, and the types of the parameters. Figure 3-22 shows an object model of the most important *RunBase* and *Batch* classes and interfaces in Dynamics AX. To simplify the view, some attributes and operations have been removed, and operation parameters are suppressed.

The UML model also contains referenced tables, classes and tables, and all extended data types, base enumerations, and X++ data types. You can include these elements in your diagrams without having to run the Reverse Engineering tool again.

Fields and member variables in Dynamics AX are generated as UML attributes. All fields are generated as public attributes, whereas member variables are generated as protected attributes. Each attribute also shows the type. Methods are generated as UML operations, including return type, parameters, and the types of the parameters.

The Reverse Engineering tool also picks up any generalizations (classes extending other classes), realizations (classes implementing interfaces), and associations (classes using each other). The associations are limited to references in member variables.

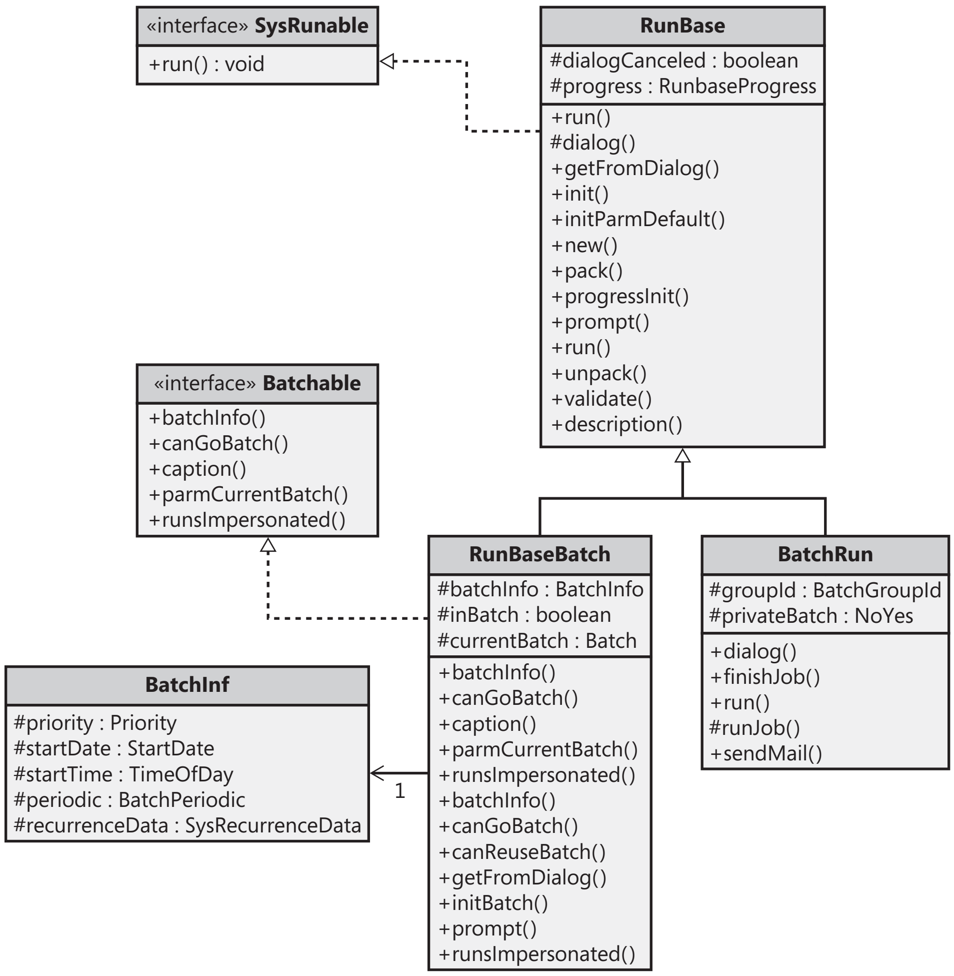


FIGURE 3-22 UML object model diagram

Note To get the names of operation parameters, you must reverse engineer in debug mode. The names are read from metadata only and placed into the stack when in debug mode. You can enable debug mode on the Development tab in the Options dialog box by selecting When Breakpoint in the Debug Mode list.

Entity Relationship Data Model

When generating an entity relationship data model, the Reverse Engineering tool looks for tables and views in the project. The entity relationship model contains an entity type for each AOT table in the project and attributes for each table's fields. Figure 3-23 shows an Entity Relationship Diagram (ERD) with the *CustTable* (Customers), *InventTable* (Inventory Items), *SalesTable* (Sales Order Header), and *SalesLine* (Sales Order Line) tables. To simplify the diagram, some attributes have been removed.

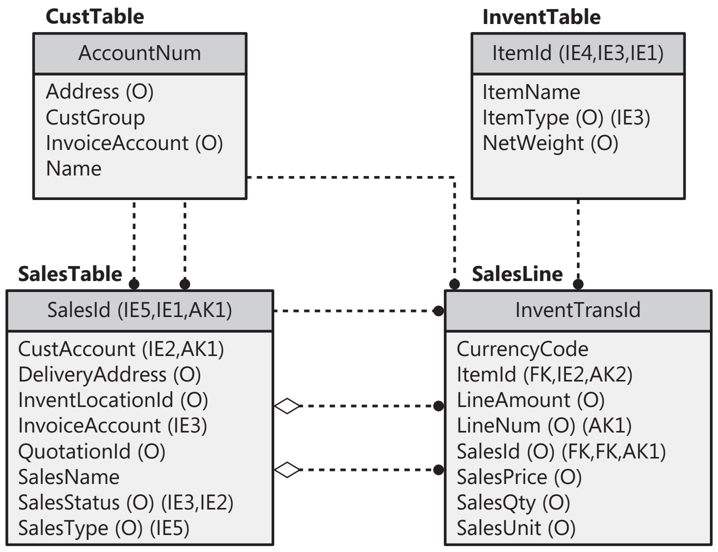


FIGURE 3-23 ERD using IDEF1X notation

Fields in Dynamics AX are generated as entity relationship columns. Columns can be foreign key (FK), alternate key (AK), inversion entry (IE), and optional (O). A foreign key column is used to identify a record in another table, an alternate key uniquely identifies a record in the current table, an inversion entry identifies zero or more records in the current table (these are typical of the fields in nonunique indexes), and optional columns don't require a value.

Relationships in Dynamics AX are generated as entity relationships. The *EntityRelationshipRole* property of the relationship in Dynamics AX is used as the foreign key role name of the relation in the entity relationship data model.

Note The Reverse Engineering tool produces an ERX file. To work with the generated file in Visio, you must start Visio, create a new Database Model Diagram and select Database, and point to Import and then Import Erwin ERX file. Afterward you can drag relevant tables from the Tables And Views pane (available from the Database menu) to the diagram canvas.

Table Browser Tool

This small, helpful tool can be used in numerous scenarios. The Table Browser tool lets you see the records in a table without requiring you to build any user interface. This tool is useful when you're debugging, validating data models, and modifying or cleaning up data, to name just a few uses.

You can access the Table Browser tool from the Add-Ins submenu in the AOT on:

- Tables
- Tables listed as data sources in forms, reports, Web forms, and Web reports
- System tables listed in the AOT under System Documentation\Tables

Note The Table Browser tool is implemented in X++. You can find it in the AOT under the name *SysTableBrowser*. It is a good example of how to bind the data source to a table at run time.

Figure 3-24 shows the Table Browser tool started from the *CustTable* table. In addition to the querying, sorting, and filtering capabilities provided by the grid control, the Table Browser tool allows you to type an SQL SELECT statement directly into the form using X++ *SELECT* statement syntax and see a visual display of the result set. This tool is a great way to try out complex SELECT statements. It fully supports grouping, sorting, aggregation, and field lists.

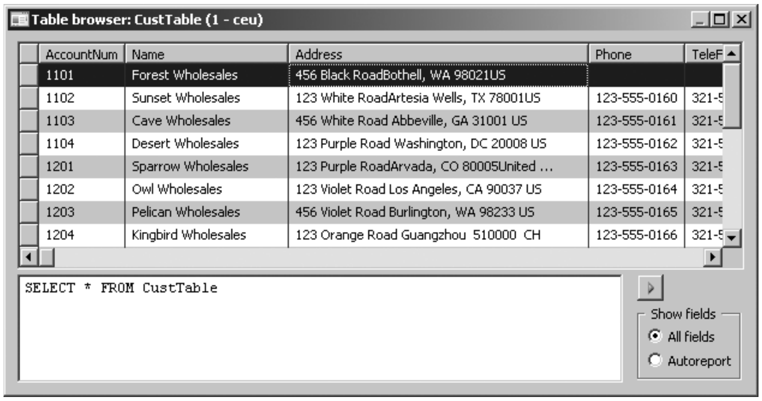


Table browser: CustTable (1 - ceu)

AccountNum	Name	Address	Phone	TeleF
1101	Forest Wholesales	456 Black RoadBothell, WA 98021US		
1102	Sunset Wholesales	123 White RoadArtesia Wells, TX 78001US	123-555-0160	321-5
1103	Cave Wholesales	456 White Road Abbeville, GA 31001 US	123-555-0161	321-5
1104	Desert Wholesales	123 Purple Road Washington, DC 20008 US	123-555-0162	321-5
1201	Sparrow Wholesales	123 Purple RoadArvada, CO 80005United ...	123-555-0163	321-5
1202	Owl Wholesales	123 Violet Road Los Angeles, CA 90037 US	123-555-0164	321-5
1203	Pelican Wholesales	456 Violet Road Burlington, WA 98233 US	123-555-0165	321-5
1204	Kingbird Wholesales	123 Orange Road Guangzhou 510000 CH	123-555-0166	321-5

```
SELECT * FROM CustTable
```

Show fields: [✓ All fields] Autoreport

FIGURE 3-24 Table Browser tool, showing *CustTable* from demo data

The Table Browser tool also allows you to choose to see only the fields from the auto-report field group. These fields are printed in a report when the user clicks Print in a form with this table as a data source. Typically, these fields hold the most interesting information. This option can make it easier to find the values you're looking for in tables with many fields.

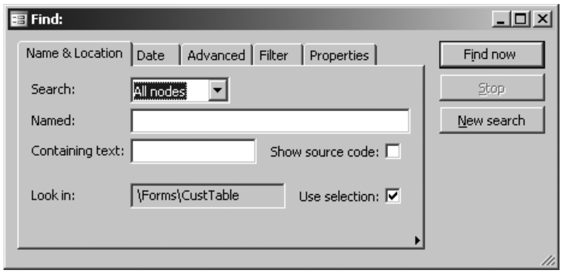
Note The Table Browser tool is just a normal form that uses IntelliMorph. It can't display fields for which the *visible* property is set to No or fields that the current user doesn't have access to.

Find Tool

Search is everything! The size of Dynamics AX applications calls for a powerful and effective search tool.

Tip You can use the Find tool to search for an example of how to use an API. Real examples can complement the examples found in the documentation.

You can start the Find tool, shown in Figure 3-25, from any node in the AOT by pressing Ctrl+F or by clicking Find on the context menu. The Find tool supports multiple selections in the AOT.



Search: All nodes
Named:
Containing text:
Show source code: ☐
Look in: \Forms\CustTable
Use selection: ✓

FIGURE 3-25 Find tool

The Name & Location tab defines what you're searching for and where to look:

- In Search, the menu options are Methods and All Nodes. When you choose All Nodes, the Properties tab appears.
- The Named text box limits the search to nodes with the name you specify.
- The Containing Text box specifies the text to look for in the method expressed as a regular expression.
- When you select the Show Source Code check box, results include a snippet of source code containing the match, making it easier to browse the results.
- By default, the Find tool searches the node (and its subnodes) selected in the AOT. If you change focus in the AOT while the Find tool is open, the Look In value is updated. This is quite useful if you want to search several nodes using the same criterion. You can disable this behavior by clearing the Use Selection check box.

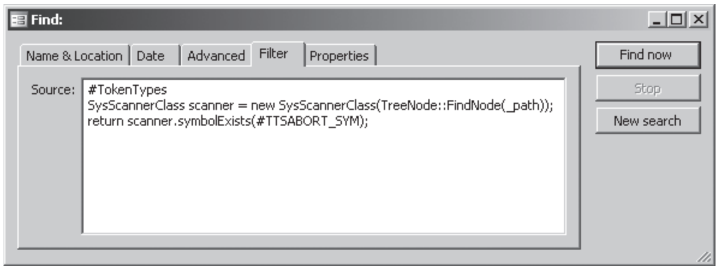
In the Date tab, you specify additional ranges for your search, such as Modified Date and Modified By.

On the Advanced tab, you can specify more-advanced settings for your search, such as the layer to search, the size range of elements, the type of element, and the tier on which the element is set to run.

The Filter tab, shown in Figure 3-26, allows you to write a more complex query by using X++ and type libraries. The code written in the Source text box is the body of a method with the following profile.

```
boolean FilterMethod(str _treeNodeName,
                     str _treeNodeSource,
                     XRefPath _path,
                     ClassRunMode _runMode)
```

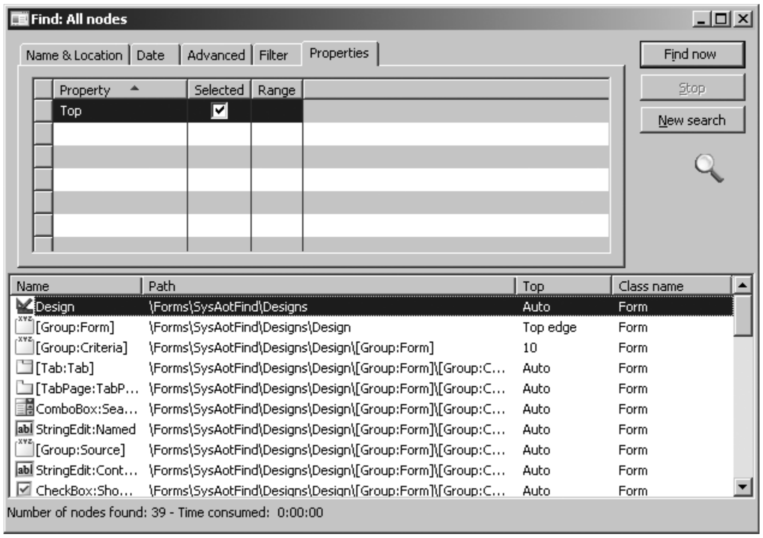
The example in Figure 3-26 uses the class *SysScannerClass* to find any occurrence of the *TTSAbort* X++ keyword. The scanner is primarily used to pass tokens into the parser during compilation. Here, however, it detects the use of a particular keyword. This tool is more accurate (though slower) than using a regular expression, because X++ comments don't produce tokens.



```
#TokenTypes
SysScannerClass scanner = new SysScannerClass(TreeNode::FindNode(_path));
return scanner.symbolExists(#TTSABORT_SYM);
```

FIGURE 3-26 Filtering in the Find tool

The Properties tab appears when All Nodes is selected in the Search menu. You can specify a search range for any property. Leaving the range blank for a property is a powerful setting when you want to inspect properties: it matches all nodes, and the property value is added as a column in the results, as shown in Figure 3-27. The search begins when you click Find Now. The results appear at the bottom of the dialog box as they are found.



Property	Selected	Range
Top	✓	

Name	Path	Top	Class name
Design	\Forms\SysAotFind\Designs	Auto	Form
[Group:Form]	\Forms\SysAotFind\Designs\Design	Top edge	Form
[Group:Criteria]	\Forms\SysAotFind\Designs\Design\[Group:Form]	10	Form
[Tab:Tab]	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form
[TabPage:TabP...	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form
ComboBox:Sea...	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form
StringEdit:Named	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form
[Group:Source]	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form
StringEdit:Cont...	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form
CheckBox:Sho...	\Forms\SysAotFind\Designs\Design\[Group:Form]\[Group:C...	Auto	Form

Number of nodes found: 39 - Time consumed: 0:00:00

FIGURE 3-27 Search results in the Find tool

Double-clicking any line in the result set opens the X++ code editor with focus on the matched code example. When you right-click the lines in the result set, a context menu containing the Add-Ins menu opens.

Compare Tool

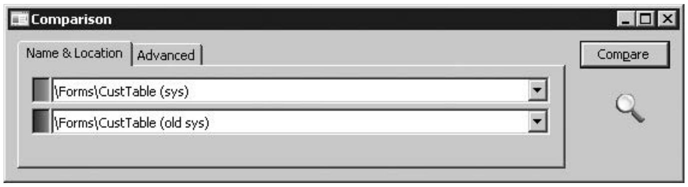
Several versions of the same element typically exist. These versions might emanate from various layers or revisions in version control, or they could be modified versions that exist in memory. Dynamics AX has a built-in Compare tool that highlights any differences between two versions of an element.

The comparison shows changes to elements, which can be modified in three ways:

- A metadata property can be changed.
- X++ code can be changed.
- The order of subnodes can be changed, such as the order of tabs on a form.

Starting the Compare Tool

You open the Compare tool by right-clicking an element and then clicking Compare on the Add-Ins submenu. A dialog box allows you to select the versions of the element you want to compare, as shown in Figure 3-28.



\Forms\CustTable (sys)
\Forms\CustTable (old sys)

FIGURE 3-28 Comparison dialog box

The versions to choose from come from many sources. The following is a list of all possible types of versions:

- **Standard layered version types** These include sys, syp, gls, glp, hfx, sl1, sl2, sl3, bus, bup, var, vap, cus, cup, usr, usp.
- **Old layered version types (old sys, old syp, and so on)** If .aod files are present in the Old Application folder (located in Program Files\Microsoft Dynamics AX\50\Application\Appl\Standard\Old), elements from the files are available here. This allows you to compare an older version of an element with its latest version. See Chapter 1 for more information on layers. In Chapter 1, Figure 1-3 illustrates the components in the application model layering system.
- **Version control revisions (Version 1, Version 2, and so on)** You can retrieve any revision of an element from the version control system individually and use it for comparison. The version control system is explained later in this chapter.
- **Best practice washed version (Washed)** A few simple best practice issues can be resolved automatically by a best practice “wash.” Selecting the washed version shows you how your implementation differs from best practices. To get the full benefit of this, select the Case Sensitive check box on the Advanced tab.
- **Export/import file (XPO)** Before you import elements, you can compare them with existing elements (which they overwrite during import). You can use the Compare tool during the import process (Command\Import) by selecting the Show Details check box in the Import dialog box and right-clicking any elements that appear in bold. Objects in bold already exist in the application; objects not in bold do not.
- **Upgraded version (Upgraded)** MorphX can automatically create a proposal for how a class should be upgraded. The requirement for upgrading a class arises during a version upgrade. The Create Upgrade Project step in the Upgrade Checklist automatically detects customized classes that conflict with new versions of the class. A class is conflicting when you’ve changed the original version of the class, and the publisher of the class has also changed the original version. MorphX constructs the proposal by merging your changes and the publisher’s changes to the class. MorphX requires access to all three versions of the class—the original version in the Old Application folder, a version with your changes in the current layer in the Old Application folder, and a version with the publisher’s changes in the same layer as the original. The installation program

ensures that the right versions are available in the right places during an upgrade. The conflict resolution is shown in Figure 3-29.

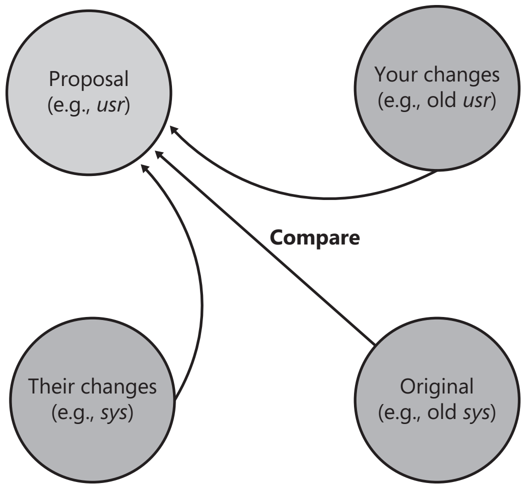
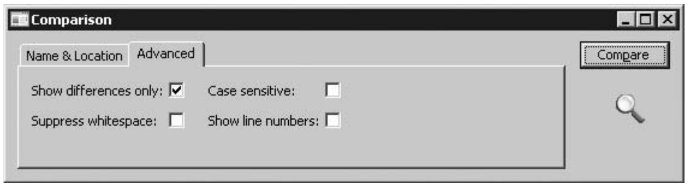

Proposal (e.g., *usr*)
Your changes (e.g., old *usr*)
Compare
Their changes (e.g., *sys*)
Original (e.g., old *sys*)

FIGURE 3-29 How the upgraded version proposal is created

Note You can also compare two different elements. To do this, select two elements in the AOT, right-click, point to Add-Ins, and then click Compare.

Figure 3-30 shows the Advanced tab, on which you can specify comparison options.



Show differences only: [✓]
Case sensitive: [ ]
Suppress whitespace: [ ]
Show line numbers: [ ]

FIGURE 3-30 Comparison options on the Advanced tab

The comparison options shown in Figure 3-30 are described in the following list:

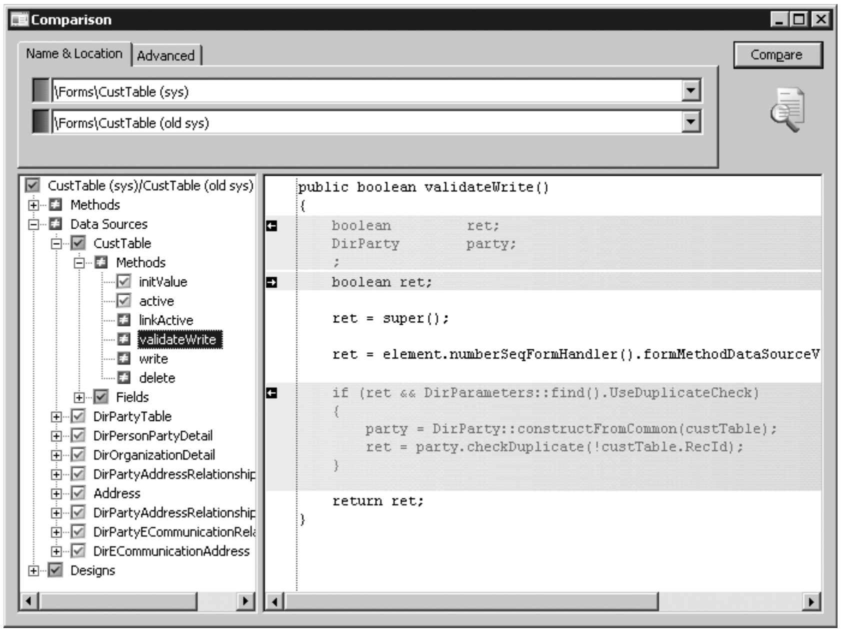
- **Show Differences Only** All equal nodes are suppressed from the view, making it easier to find the changed nodes. This option is selected by default.
- **Suppress Whitespace** White space, such as spaces and tabs, is suppressed into a single space when comparing. The Compare tool can ignore the amount of white space, just as the compiler does. This option is selected by default.
- **Case Sensitive** Because X++ is not case-sensitive, the Compare tool is also not case-sensitive by default. In certain scenarios, case sensitivity is required and must be

enabled, such as when you're using the best practice wash feature mentioned earlier in this section. The Case Sensitive option is not selected by default.

- **Show Line Numbers** The Compare tool can add line numbers to all displayed X++ code. This option is not selected by default but can be useful during an upgrade of large chunks of code.

Using the Compare Tool

After you choose elements and set parameters, you can start the comparison by clicking Compare. Results are displayed in a three-pane dialog box, as shown in Figure 3-31. The top pane is the element selection, the left pane is a tree structure resembling the AOT, and the right pane shows details of the tree selection.



\Forms\CustTable (sys)

\Forms\CustTable (old sys)

```
public boolean validateWrite()
{
    boolean         ret;
    DirParty        party;
    ;
    boolean ret;

    ret = super();

    ret = element.numberSeqFormHandler().formMethodDataSourceV

    if (ret && DirParameters::find().UseDuplicateCheck)
    {
        party = DirParty::constructFromCommon(custTable);
        ret = party.checkDuplicate(!custTable.RecId);
    }

    return ret;
}
```

FIGURE 3-31 Comparison results

The icons in the tree structure indicate how each node has changed. A red or blue check mark indicates that the node exists only in a red or blue element. Red corresponds to the *sys* layer, and blue corresponds to the old sys layer. A gray check mark indicates that the nodes are identical but one or more subnodes are different. A not-equal-to symbol (≠) on a red and blue background indicates that the nodes are different in the two versions.

Note Each node in the tree view has a context menu that provides access to the Add-Ins submenu and the Open New Window option. The Open New Window option provides an AOT view on any element, including old layer elements.

Details of the differences are shown in the right pane. Color coding is also used in this pane to highlight differences. If an element is editable, small action icons appear. These icons allow you to make changes to source, metadata, and nodes, which can save you time when performing an upgrade. A right or left arrow removes or adds the difference, and a bent arrow moves the difference to another position. These arrows always come in pairs, so you can see where the difference is moved to and from. An element is editable if it is from the current layer and checked out if a version control system is used.

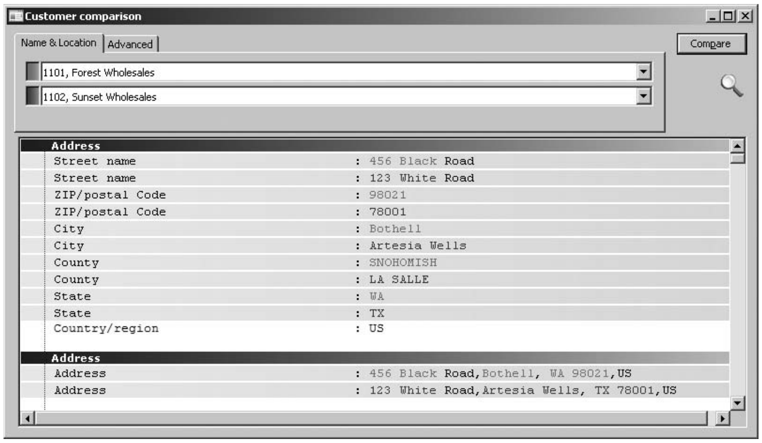
Compare APIs

Although Dynamics AX uses the comparison functionality for development purposes only, the general comparison functionality can be used more widely. The available APIs allow you to compare and present differences in the tree structure or text representation of any type of entity.

The *Tutorial_CompareContextProvider* class shows how simple it is to compare business data by using these APIs and presents it by using the Compare tool. The tutorial consists of two parts:

- ***Tutorial_Comparable*** This class implements the *SysComparable* interface. Basically, it creates a text representation of a customer.
- ***Tutorial_CompareContextProvider*** This class implements the *SysCompareContext-Provider* interface. It provides the context for comparison. For example, it lists a *tutorial_Comparable* class for each customer, sets the default comparison options, and handles context menus.

Figure 3-32 shows a comparison of two customers, the result of running the tutorial.



1101, Forest Wholesales
1102, Sunset Wholesales

```
Address
Street name                          : 456 Black Road
Street name                          : 123 White Road
ZIP/postal Code                      : 98021
ZIP/postal Code                      : 78001
City                                 : Bothell
City                                 : Artesia Wells
County                               : SNOHOMISH
County                               : LA SALLE
State                                : WA
State                                : TX
Country/region                       : US

Address
Address                              : 456 Black Road,Bothell, WA 98021,US
Address                              : 123 White Road,Artesia Wells, TX 78001,US
```

FIGURE 3-32 Result of comparing two customers using the Compare API

You can also use the line-by-line comparison functionality directly in X++. The static *run* method on the *SysCompareText* class, shown in the following code, takes two strings as parameters and returns a container that highlights differences in the two strings. You can also use a set of optional parameters to control the comparison.

```
public static container run(str _t1,
                            str _t2,
                            boolean _caseSensitive      = false,
                            boolean _suppressWhiteSpace = true,
                            boolean _lineNumbers        = false,
                            boolean _singleLine         = false,
                            boolean _alternateLines     = false)
```

Refer to the Microsoft Dynamics AX 2009 SDK for documentation of the classes.

Cross-Reference Tool

The concept of cross-references in Dynamics AX is simple. If an element uses another element, the reference is recorded. Cross-references allow you to determine which elements a particular element uses as well as which elements other elements are using. Dynamics AX provides the Cross-reference tool to access and manage cross-reference information.

You must update the Cross-reference tool regularly to ensure accuracy. The update typically takes several hours. The footprint in your database is about 1 gigabyte for the standard application.

You can update the Cross-reference tool by going to the Microsoft Dynamics AX drop-down menu and then pointing to Tools\Development Tools\Cross-reference\Periodic\Update. Updating the Cross-reference tool also compiles the entire AOT because the compiler emits cross-reference information.

Tip Keeping the Cross-reference tool up to date is important if you want to rely on its information. If you work in a shared development environment, you share cross-reference information with your team members. Updating the Cross-reference tool nightly is a good approach for a shared environment. If you work in a local development environment, you can keep the Cross-reference tool up to date by enabling cross-referencing when compiling. This option does slow down the compilation, however. Another option is to manually update cross-references for the elements in a project. You can do so by right-clicking the project, pointing to Add-Ins, pointing to Cross-reference, and then clicking Update.

In addition to the main cross-reference information, two smaller cross-reference subsystems exist:

- **Data model** This cross-reference subsystem stores information about relationships between tables. It is primarily used by the query form and the Reverse Engineering tool.
- **Type hierarchy** This cross-reference subsystem stores information about class and data type inheritance. It is used only in the Application Hierarchy Tree. The Application Hierarchy Tree is available from the Microsoft Dynamics AX drop-down menu, at Tools\Development Tools\Application Hierarchy Tree.

Further discussion of these tools is beyond the scope of this book. Refer to the Microsoft Dynamics AX 2009 SDK for more information on these subsystems and the tools that rely on them.

The cross-reference information the Cross-reference tool collects is quite complete. The following list shows the kinds of elements it cross-references. (Cross-reference information for elements followed by an asterisk is new in Dynamics AX 2009.) You can find the following list of cross-referenced elements and their values by opening the AOT, expanding the System Documentation node, and clicking Enums and then xRefKind.

BasicType
Class
ClassInstanceMethod
ClassStaticMethod
ClrType
ClrTypeMethod
ConfigurationKey
Dataset*
Enum
Enumerator
ExtendedType
Form*
Job*
Label
LicenseCode
Map
MapField
MapInstanceMethod
MapStaticMethod
Menu*
MenuItemAction
MenuItemDisplay
MenuItemOutput
Predefined (system functions)
Query*
Report*
SecurityKey
Table
TableField
TableIndex
TableInstanceMethod
TableStaticMethod
WebActionItem
WebDisplayContentItem
WebForm*
WebManagedContentItem*
WebMenu*
WebModule*
WebOutputContentItem
WebReport*
WebUrlItem

When the Cross-reference tool is updated, it scans all metadata and X++ code for references to elements of the kinds listed here.

Tip It's a good idea to use intrinsic functions when referring to elements in X++ code. An intrinsic function can evaluate to either an element name or an ID. The intrinsic functions are named <*ElementKind*>Str or <*ElementKind*>Num, respectively. Using intrinsic functions provides two benefits: you have compile-time verification that the element you reference actually exists, and the reference is picked up by the Cross-reference tool. Also, there is no run-time overhead. An example follows.

```
// Prints ID of MyClass, such as 50001
print classNum(myClass);

// Prints "MyClass"
print classStr(myClass);

// No compile check or cross-reference
print "MyClass";
```

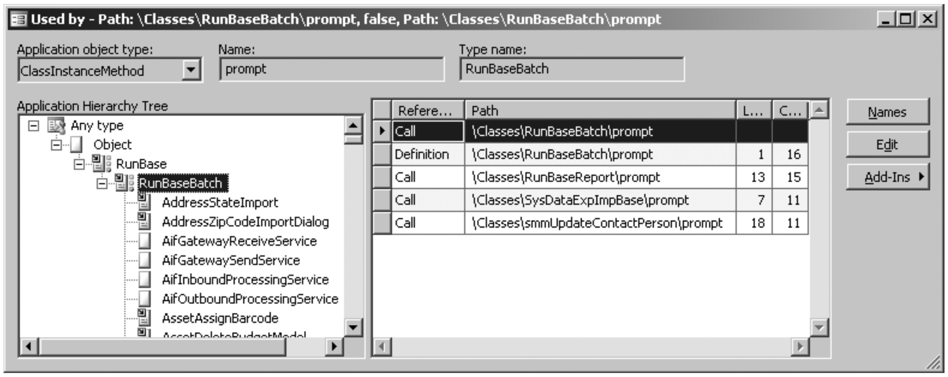
See Chapter 15, "Reflection," for more information about intrinsic functions.

The primary function of the Cross-reference tool is to determine where a particular element is being used. Here are a couple of scenarios:

- You want to find usage examples. If the product documentation doesn't help, you can use the Cross-reference tool to find real implementation examples.
- You need to perform an impact analysis. If you're changing an element, you need to know which other elements are affected by your change.

To access usage information, right-click any element in the AOT, point to Add-Ins, point to Cross-reference, and then click Used By. If the option isn't available, either the element isn't used or that cross-reference hasn't been updated.

Figure 3-33 shows where the *prompt* method is used on the *RunBaseBatch* class.


Used by - Path: \Classes\RunBaseBatch\prompt, false, Path: \Classes\RunBaseBatch\prompt

Application object type: ClassInstanceMethod
Name: prompt
Type name: RunBaseBatch

Application Hierarchy Tree
- Any type
  - Object
    - RunBase
      - RunBaseBatch
        - AddressStateImport
        - AddressZipCodeImportDialog
        - AifGatewayReceiveService
        - AifGatewaySendService
        - AifInboundProcessingService
        - AifOutboundProcessingService
        - AssetAssignBarcode

Refere...	Path	L...	C...
Call	\Classes\RunBaseBatch\prompt		
Definition	\Classes\RunBaseBatch\prompt	1	16
Call	\Classes\RunBaseReport\prompt	13	15
Call	\Classes\SysDataExpImpBase\prompt	7	11
Call	\Classes\smmUpdateContactPerson\prompt	18	11

FIGURE 3-33 Cross-reference tool, showing where *RunBaseBatch.prompt* is used

When you view cross-references for a class method, the Application Hierarchy Tree is visible, allowing you to see whether the same method is used on a parent or subclass. For types that don't support inheritance, such as tables, table methods, and table fields, the Application Hierarchy Tree is hidden.

Version Control

The Version Control tool is a feature in MorphX that makes it possible to use a version control system, such as Microsoft Visual SourceSafe or Microsoft Visual Studio Team Foundation Server, to keep track of changes to elements in the AOT. The tool is accessible from several places: from the Microsoft Dynamics AX drop-down menu at Tools\Development Tools\Version Control, from toolbars in the AOT and X++ code editor, and from the context menu on elements in the AOT.

Using a version control system offers several benefits:

- **Revision history of all elements** All changes are captured along with a description of the change, making it possible to consult change history and retrieve old versions of an element.
- **Code quality enforcement** The implementation of version control in Dynamics AX enables a fully configurable quality bar for all check-ins. With the quality bar, all changes are verified according to coding practices. If the change doesn't meet the criteria, it is rejected. Microsoft uses the quality bar for all check-ins, which has helped raise the quality of X++ code to an unprecedented level. Microsoft developers cannot check in code with compiler errors, compile warnings, or best practice errors. In the final stages of development, tasks in code (to-dos) are also prohibited.

- **Isolated development** Each developer can have a local installation and make all modifications locally. When modifications are ready, they can be checked in and made available to consumers of the build. So a developer can rewrite fundamental areas of the system without causing any instability issues for others. Developers are also unaffected by any downtime of a centralized development server.

Even though using a version control system when developing is optional, we strongly recommend you consider one for any development project. Dynamics AX 2009 supports three version control systems: Visual SourceSafe 6.0 and Team Foundation Server, which are designed for large development projects, and MorphX VCS. MorphX VCS is designed for smaller development projects that previously couldn't justify the additional overhead that using a version control system server adds to the entire process. Table 3-7 shows a side-by-side comparison of the version control system options.

TABLE 3-7 Overview of Version Control Systems

	Classic MorphX (No Version Control)	MorphX VCS	Visual SourceSafe	Team Foundation Server
Application Object Servers required	1	1	1 per developer	1 per developer
Database servers required	1	1	1 per developer	1 per developer
Team server required	No	Optional	Yes	Yes
Build process required	No	No	Yes	Yes
Master file	AOD	AOD	XPOs	XPOs
Isolated development	No	No	Yes	Yes
Multiple check-out	N/A	No	Configurable	Configurable
Change description	No	Yes	Yes	Yes
Change history	No	Yes	Yes	Yes
Change list support (atomic check-in of a set of files)	N/A	No	No	Yes
Code quality enforcement	No	Configurable	Configurable	Configurable

The elements persisted in the version control server are file representations of the elements in the AOT. The file format used is the standard Dynamics AX export format (.xpo). Each .xpo file contains only one element.

There are no additional infrastructure requirements when you use MorphX VCS, which makes it a perfect fit for partners running many parallel projects. In such setups, each developer often works simultaneously on several projects, toggling between projects and returning to past projects. In these situations, the benefits of having change history are enormous. With just a few clicks, you can enable MorphX VCS to persist the changes in the business database. Although MorphX VCS provides many of the same capabilities as a version control server, it does have some limitations. For example, MorphX VCS does not provide any tools for maintenance, such as backup, archiving, or labeling.

In contrast, Visual SourceSafe and Team Foundation Server are designed for large projects in which many developers work together on the same project for an extended period of time (e.g., an independent software vendor building a vertical solution).

Figure 3-34 shows a typical deployment using Visual SourceSafe or Team Foundation Server, in which each developer locally hosts the AOS and the database. Each developer also needs a copy of all .xpo files. When a developer communicates with the version control server, the .xpo files are transmitted. A unique ID is required when the developer creates a new element or label. A Team Server, available as a Microsoft .NET Web service, is required to ensure uniqueness of IDs across all the local developers' environments. The Team Server is a component available with Dynamics AX.

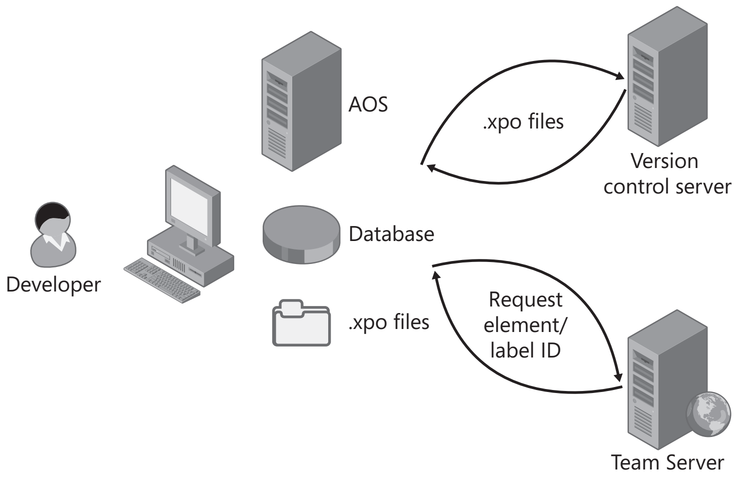


FIGURE 3-34 Typical deployment using version control

Element Life Cycle

Figure 3-35 shows the element life cycle in a version control system. When the element is in a state marked with green, it can be edited; otherwise it is read-only.

You can create a new element in two ways:

- Create a completely new element.

- Customize an existing element, resulting in an overlayered version of the element. Because elements are stored per layer in the version control system, customizing an element effectively creates a new element.

After you create an element, you must add it to the version control system. First give it a proper name in accordance with naming conventions, and then click Add To Version Control on the context menu. After you create the element, you must check it in.

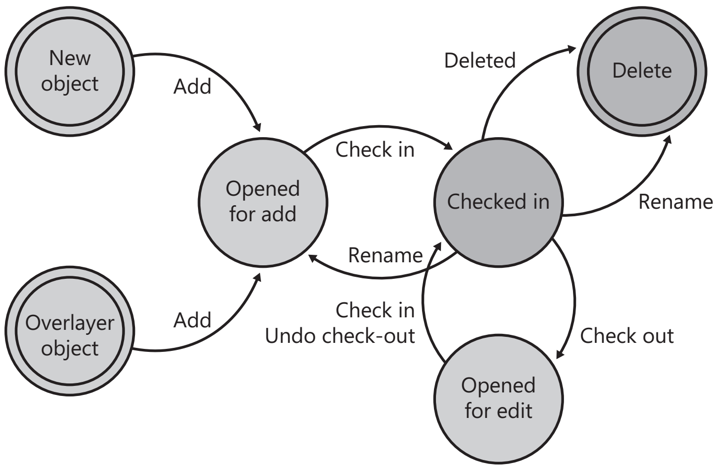


FIGURE 3-35 Element life cycle

An element that is checked in can be renamed. Renaming an element deletes the element with the old name and adds an element with the new name.

Check-Out

To modify an element, you must check it out. Checking out an element locks it so that others can't modify it while you're working.

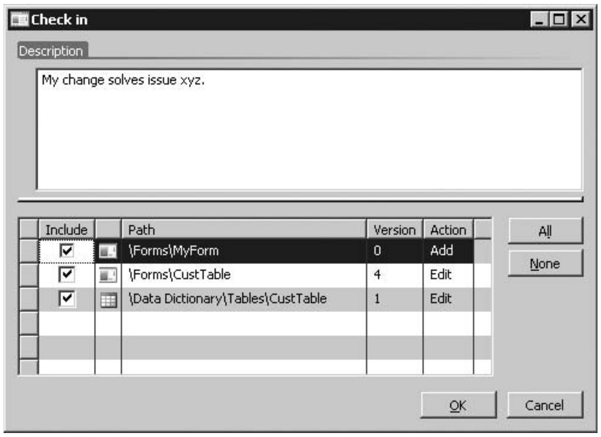
By clicking Tools\Development Tools\Version Control\Pending Objects from the Microsoft Dynamics AX drop-down menu, you can see which elements you currently have checked out. The elements you've checked out (or that you've created and not yet checked in), appear in blue, rather than black, in the AOT.

Undo Check-Out

If you decide that you don't want to modify an element that you checked out, you can undo the check-out. This releases your lock on the element and imports the server version of the element to undo your changes.

Check-In

When you have finalized your modifications, you must check in the elements for them to be part of the next build. When you click Check-In on the context menu, the dialog box shown in Figure 3-36 appears, displaying all the elements that you currently have checked out. The Check In dialog box shows all open elements by default; you can remove any elements not required in the check-in from the list by pressing Alt+F9.



My change solves issue xyz.

Include		Path	Version	Action
✓		\Forms\MyForm	0	Add
✓		\Forms\CustTable	4	Edit
✓		\Data Dictionary\Tables\CustTable	1	Edit

FIGURE 3-36 Check In dialog box

We recommend the following procedure for checking in your work:

1. Perform synchronization to update all elements in your environment to the latest version.
2. Verify that everything is still working as intended. Compilation is not enough!
3. Check in the elements.

Quality Checks

Before the version control system accepts a check-in, it might subject the elements to quality checks. You define what is accepted in a check-in when you set up the version control system. The following checks are supported:

- Compiler errors
- Compiler warnings
- Compiler tasks
- Best practice errors

When a check is enabled, it is carried out when you do a check-in. If the check fails, the check-in stops. You must address the issue and restart the check-in.

Updating Source Code Casing

You can set the Source Code Titlecase Update tool, available on the Add-Ins submenu, to automatically execute before elements are checked in to ensure uniform casing in variable and parameter declarations and references. You can specify this parameter when setting up the version control system by selecting the Run Title Case Update check box.

Creating New Elements

When using version control, you create new elements just as you normally would in the MorphX environment without a version control system. These elements are not part of your check-in until you click Add To Version Control on the context menu.

You can also create all element types except those listed in System Settings (from the Microsoft Dynamics AX drop-down menu: Tools\Development Tools\Version Control\Setup\System Settings). By default, jobs and private projects are not accepted.

New elements should follow Dynamics AX naming conventions. The best practice naming conventions are enforced by default, so you can't check in elements with names such as *aaaElement*, *Del_Element*, *element1*, or *element2*. (The only *Del* elements allowed are those required for version upgrade purposes.) You can change naming requirements in System Settings.

Renaming Elements

An element must be in the checked-in state to be renamed. Because all references in .xpo files are strictly name based (not ID based), all references to renamed elements must be updated. For example, when you rename a table field, you must also update any form or report that uses that field. Most references in metadata in the AOT are ID based, thus not affected when an element is renamed; in most cases, it is enough to simply check out the form or report and include it in the check-in to update the .xpo file. You can leverage the cross-reference functionality to identify references. References in X++ code are name based. You can use the compiler to find affected references.

An element's revision history is kept intact when elements are renamed. No tracking information in the version control system is lost because of a rename.

Deleting Elements

You delete an element as you normally would in Dynamics AX. The delete operation must be checked in before the deletion is visible to other users of the version control system. You can see pending deletions in the Pending Objects dialog box.

Labels

Working with labels is very similar to working with elements. To change, delete, or add a label, you must check out the label file containing the label. You can check out the label file from the Label Editor dialog box.

The main difference between checking out elements and checking out label files is that simultaneous check-outs are allowed for label files. This means that others can change labels while you have a label file checked out.

When you check in a label file, your changes are automatically merged into the latest version of the file. If you modify or delete a label that another person has also modified or deleted, your changes are lost. Lost changes are shown in the Infolog.

The ID server guarantees that label IDs are unique; adding labels won't generate conflicts.

Get Latest

If someone else has checked in a new version of an element, the Get Latest option on the context menu allows you to get the version of the element that was most recently checked in. This option isn't available when you have the element checked out yourself.

Get Latest is not applicable to MorphX VCS.

Synchronization

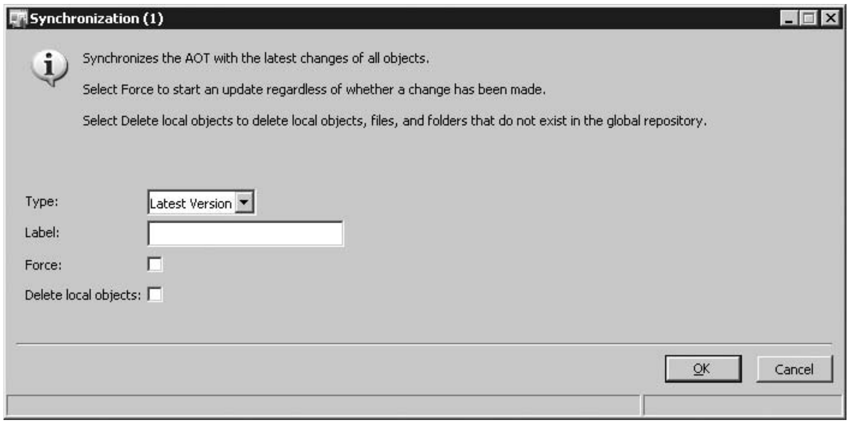
Synchronization allows you to get the latest version of all elements. This step is required before you can check in any elements. You can initiate synchronization from the Microsoft Dynamics AX drop-down menu: Tools\Development Tools\Version Control\Periodic\Synchronize.

Synchronization is divided into three operations that happen automatically in the following sequence:

1. Copy the latest files from the version control server to the local disk.
2. Import the files into the AOT.
3. Compile the imported files.

You should use synchronization to make sure your system is up to date. Synchronization won't affect any new elements that you have created or any elements that you have checked out.

Figure 3-37 shows the Synchronization dialog box.


Synchronization (1)

Synchronizes the AOT with the latest changes of all objects.

Select Force to start an update regardless of whether a change has been made.

Select Delete local objects to delete local objects, files, and folders that do not exist in the global repository.

Type: Latest Version

Label:

Force: ☐

Delete local objects: ☐

FIGURE 3-37 Synchronization dialog box

Selecting the Force check box gets the latest version of all files, whether or not they have changed, and then imports every file.

When using Visual SourceSafe, you can also synchronize to a label defined in Visual SourceSafe. This way you can easily synchronize to a specific build or version number.

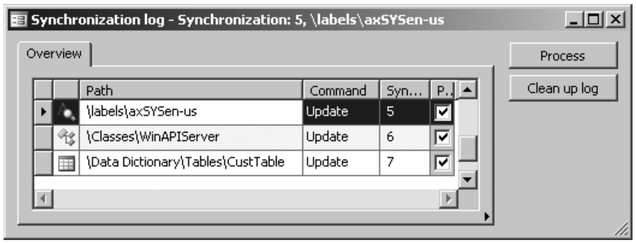
Synchronization is not applicable to MorphX VCS.

Synchronization Log

How you keep track of versions on the client depends on the version control system being used. Visual SourceSafe requires that Dynamics AX keep track of itself. When you synchronize the latest version, it is copied to the local repository folder from the version control system. Each file must be imported into Dynamics AX to be reflected in the AOT. To minimize the risk of partial synchronization, a log entry is created for each file. When all files are copied locally, the log is processed, and the files are automatically imported into Dynamics AX.

When synchronization fails, the import operation is usually the cause of any problems. Synchronization failure leaves your system in a partially synchronized state. To complete the synchronization, you must restart Dynamics AX and restart the import. You use the synchronization log to restart the import, and you access it from the Microsoft Dynamics AX drop-down menu at Tools\Development Tools\Version Control\Inquiries\Synchronization Log.

The Synchronization Log dialog box, shown in Figure 3-38, displays each batch of files, and you can restart the import by clicking Process. If the Processed check box is not selected, the import has failed and should be restarted.


Synchronization log - Synchronization: 5, \labels\axSYSen-us

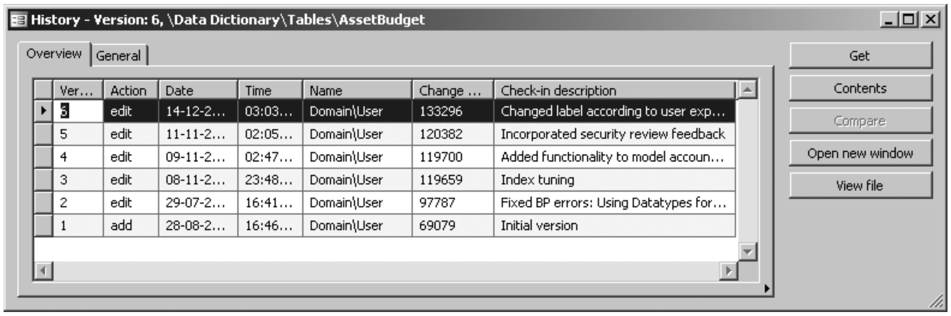
Path	Command	Syn...	P...
\labels\axSYSen-us	Update	5	✓
\Classes\WinAPIServer	Update	6	✓
\Data Dictionary\Tables\CustTable	Update	7	✓

FIGURE 3-38 Synchronization Log dialog box

The Synchronization Log is not available in MorphX VCS.

Show History

One of the biggest advantages of version control is the ability to track changes to elements. Selecting History on an element's context menu displays a list of all changes to an element, as shown in Figure 3-39.


History - Version: 6, \Data Dictionary\Tables\AssetBudget

Ver...	Action	Date	Time	Name	Change ...	Check-in description
6	edit	14-12-2...	03:03...	Domain\User	133296	Changed label according to user exp...
5	edit	11-11-2...	02:05...	Domain\User	120382	Incorporated security review feedback
4	edit	09-11-2...	02:47...	Domain\User	119700	Added functionality to model accoun...
3	edit	08-11-2...	23:48...	Domain\User	119659	Index tuning
2	edit	29-07-2...	16:41...	Domain\User	97787	Fixed BP errors: Using Datatypes for...
1	add	28-08-2...	16:46...	Domain\User	69079	Initial version

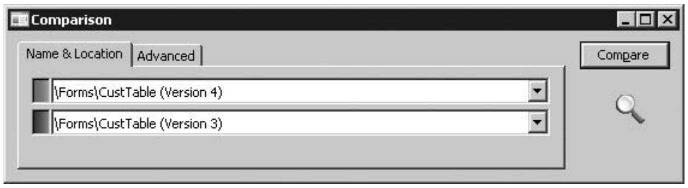
FIGURE 3-39 Revision history of an element

This dialog box shows the version number, the action performed, the time the action was performed, and who performed the action. You can also see the change number and the change description.

A set of buttons in the revision history dialog box allows further investigation of each version. Clicking Contents opens a form that shows other elements included in the same change. Clicking Compare opens the Compare dialog box, which allows you to do a line-by-line comparison of two versions of the element. The Open New Window button opens an AOT window that shows the selected version of the element, which is useful for investigating properties because it allows you to use the standard MorphX toolbox. Clicking View File opens the .xpo file for the selected version in Notepad.

Revision Comparison

Comparison is the key to harvesting the benefits of a version control system. You can start a comparison from several places, including the Compare option on the Add-Ins submenu. Figure 3-40 shows the Comparison dialog box where two revisions of the form *CustTable* are selected.



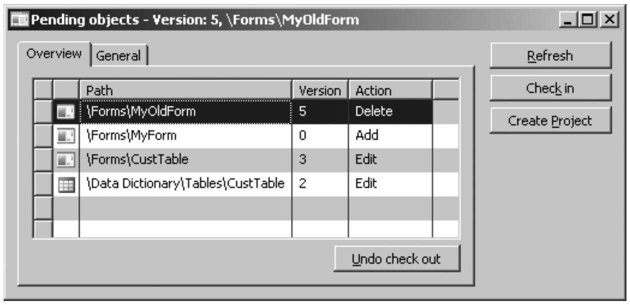
\Forms\CustTable (Version 4)
\Forms\CustTable (Version 3)

FIGURE 3-40 Comparing element revisions from version control

The Compare dialog box contains a list of all checked-in versions, in addition to the layer element versions, when a version control system is used.

Pending Elements

When you're working on a project, it's easy to lose track of which elements you've opened for editing. The Pending Objects dialog box, shown in Figure 3-41, lists the elements that are currently checked out in the version control system. Notice the column containing the action performed on the element. Deleted elements are available only in this dialog box; they are no longer shown in the AOT.



Path	Version	Action
\Forms\MyOldForm	5	Delete
\Forms\MyForm	0	Add
\Forms\CustTable	3	Edit
\Data Dictionary\Tables\CustTable	2	Edit

FIGURE 3-41 Pending elements

You can access the Pending Objects dialog box from the Microsoft Dynamics AX drop-down menu: Tools\Development Tools\Version Control\Pending Objects.

Build

Because the version control system contains .xpo files, and not an .aod file, a build process is required to generate an .aod file from the .xpo files. The following procedure is a high-level overview of the build process.

1. Use the CombineXPOs command-line utility to create one .xpo file by combining all .xpo files. The purpose of this step is to make the .xpo file consumable by Dynamics AX. Dynamics AX requires all referenced elements to be present in the .xpo file or to already exist in the AOT to maintain the references during import.
2. Import the new .xpo file by using the command-line parameter -AOTIMPORTFILE=<*FileNam*e.xpo> to Ax32.exe. This step imports the .xpo file and compiles everything. After it is complete, the new .aod file is ready.

You must follow these steps for each layer you build. The steps are described in more detail in the Microsoft Dynamics AX 2009 SDK.

The build process doesn't apply to MorphX VCS.

Integration with Other Version Control Systems

The implementation of the version control system in Dynamics AX is fully pluggable. This means that any version control system can be integrated with Dynamics AX.

Integrating with another version control system requires a new class implementing the *SysVersionControlFileBasedBackEnd* interface. It is the implementation's responsibility to provide the communication with the version control system server being used.

Unit Test Tool

A *unit test* is a piece of code that exercises another piece of code and ascertains that it behaves correctly. The developer who implements the unit to be tested typically writes the unit test. Thought leaders in this area recommend writing unit tests as early as possible, even before writing a single line of the unit's code. This principle is called *test-driven development*. (You can read more about test-driven development on MSDN and more about unit testing in the Unit Test Framework section of the Microsoft Dynamics AX 2009 SDK.)

Writing unit tests early forces you to consider how your code will be consumed; this in turn makes your APIs easier to use and understand, and it results in constructs that are more likely to be robust and long lasting. With this technique, you must have at least one unit test for each requirement; a failing unit test indicates an unfulfilled requirement. Development efforts should be targeted at making the failing unit test succeed—no more, no less.

To reap the full benefits of unit testing, you should execute test cases regularly, preferably each time code is changed. The Unit Test framework in Dynamics AX supports you regardless of your approach to writing unit tests. For example, the unit test capabilities are fully embedded in MorphX, and you can easily toggle between writing test cases and writing business logic.

If you're managing an implementation project for Dynamics AX, you should advocate testing and support your team members in any way required. At first glance, unit testing might seem like more work, but the investment is well worth the effort. If you're a team member on a project that doesn't do unit testing, you should convince your manager of its benefits. Plenty of recent literature describes the benefits in great detail.

When implementing unit tests, you write a test class, also referred to as a *test case*. Each test case has several test methods that exercise the object being tested in a particular way. As you build your library of test cases, you'll find that you need to organize them into groups. You can group test cases into test suites. The simplest way to do this is to use test projects, which are simply special kinds of AOT projects.

Test Cases

To implement a unit test case, you must create a new class that extends the *SysTestCase* class, which is part of the Unit Test framework. You should give the class the same name as the class it is testing, suffixed with *Test*. This is illustrated in the following example, where a unit test for the *Stack* class is declared.

```
class StackTest extends SysTestCase
{
}
```

If you were to run the unit test at this point, you would find that zero tests were run and zero tests failed.

This default naming convention tells the Unit Test framework which test class to collect code coverage data for. If the default test class name doesn't suit your needs, you can override the *testsElementName* method. You can also override the *testsElementType* method to set the kind of element for which the framework collects code coverage data.

To create a useful test, you must add one or more test methods to the class. All test method names must start with *test*. The test methods must return void and take no parameters. In the following code, a test method is added to the *StackTest* class.

```
void testPushPop()
{
    //Create an instance of the class to test.
    Stack stack = new Stack();
    ;
    //Push 123 to the top of the stack.
    stack.push([123]);
    //Pop the value from the stack and assert that it is 123.
    this.assertEquals([123], stack.pop());
}
```

Within each test method, you should exercise the object you test and confirm that it behaves correctly. Running the unit test at this point tells you that one test was run and zero tests failed.

Your testing needs should be met by the assertion methods available on *SysTestCase* (which extends *SysTestAssert*), as shown in Table 3-8.

TABLE 3-8 Assertion Methods on the *SysTestCase* Class

Method	Parameters	Action
assertEquals	(*anyType*, *anyType*)	Asserts that two values are equal. When the argument is of type *object*, the *equal* method is called to compare them.
assertFalse	*(boolean)*	Asserts that the value is false.
assertNotEqual	*(anyType, anyType)*	Asserts that two values are different.
assertNotNull	*(object)*	Asserts that the value is not null.
assertNotSame	*(object, object)*	Asserts that the objects referenced are not the same.
assertNull	*(object)*	Asserts that the value is null.
assertRealEquals	*(real, real [, real delta])*	Asserts that real values differ no more than the delta.
assertSame	*(object, object)*	Asserts that the objects referenced are the same.
assertTrue	*(boolean)*	Asserts that the value is true.

If an assertion fails, the test method fails. You can configure the framework to stop at first failure or continue with the next test method in the Unit Test Parameters dialog box: from the Microsoft Dynamics AX drop-down menu, point to Tools\ Development Tools\Unit Test\ Parameters. The following code adds a new failing test method.

```
//Test the qty method, which returns the quantity of values on the stack.
void testQty()
{
    //Create an instance of the class to test.
    Stack stack = new Stack();
    ;
    //Push 123 to the top of the stack.
    stack.push([123]);
    //Pop the value from the stack and assert that it is 0.
    this.assertEquals(0, stack.qty());
}
```

Running the unit test at this point shows that two tests were executed and one failed. The failing test appears in the Infolog. Clicking Edit opens the X++ code editor on the assert call that failed.

You might have noticed code redundancy in the test methods shown so far. In many cases, initialization code is required before the test method can run. Instead of duplicating this code in all test methods, you can refactor it into the *setUp* method. If teardown logic is required, you can place it in the *tearDown* method. When the framework runs a test method, it instantiates a new test case class, which is followed by calls to *setUp* and test methods, and finally a call to the *tearDown* method. This prevents in-memory data from one test method from affecting another test method. Test suites, which are covered in the next section, provide ways to isolate data persisted in the database between test cases and methods. The following code uses the *setUp* method to refactor the sample code.

```
class StackTest extends SysTestCase
{
    Stack stack;

    public void setUp()
    {;
        super();
        //Create an instance of the class to test.
        stack = new Stack();
    }
    void testPushPop()
    {;
        stack.push([123]);
        this.assertEquals([123], stack.pop());
    }
    ...
}
```

The Unit Test framework also supports testing of exceptions. If a method is expected to throw an exception, you can instruct the framework to expect an exception to be thrown. If you expect an exception and none is thrown, the framework reports the test case as failed. You inform the framework that an exception is expected by calling *parmExceptionExpected([boolean, str])*. You can specify an exception text that must exactly match the text thrown with the exception, or the test case will fail. You shouldn't write more asserts after the method call expected to throw an exception because execution should never get that far. The following code adds a test method that expects an exception message to be thrown.

```
void testFailingPop()
{;
    //Assert that an exception is expected.
    this.parmExceptionExpected(true, "Stack is empty!");

    //Call the method expected to throw an exception.
    stack.pop();
}
```

The sample test case now has three test methods. By following these steps, you can run the test case from MorphX:

1. Right-click the method, point to Add-Ins, and then click Run Tests.
2. Type the name in the Test toolbar, and then click Run.
3. Start the Dynamics AX client with the following command line:

```
StartupCmd=RunTestProject_<Name of test case class>
```

If you wanted to run the test case programmatically, you could use a test runner class. To do this, you would typically place the following logic in your test class's *main* method, which is invoked when you press F5 in the X++ code editor.

```
static void main(args _args)
{
    SysTestRunner runner = new SysTestRunner(classStr(StackTest));
    SysTestListenerXML listener =
        new SysTestListenerXML(@"c:\tmp\StackTest.xml");
    ;
    runner.getResult().addListener(listener);
    runner.run();
}
```

Notice that you also register a listener. If you didn't register a listener, you wouldn't know the result of the test. Listeners are described in the section "Test Listeners" later in this chapter.

Test Suites

Test suites serve two purposes:

- **Collection of test cases and test suites** A test suite can contain any number of test cases and other test suites. This arrangement means that you can group test cases in a hierarchy.
- **Test case isolation** Each test case could have different needs for isolation, depending on what data it changes. In fact, each method within the test case could have a need for isolation.

Dynamics AX includes the following five test suites that provide different levels of isolation:

- ***SysTestSuite*** This test suite is the default. It provides no isolation. You can override the *setUp* and *tearDown* methods, if necessary. Note that these methods are not the same as the *setUp* and *tearDown* methods on the test case.
- ***SysTestSuiteCompanyIsolateClass*** This test suite constructs an empty company account for the entire test class and runs each test method in the company account. After all test methods have been executed, the company account is deleted.
- ***SysTestSuiteCompanyIsolateMethod*** This test suite constructs an empty company account for each test method and runs the test method in the company account. After the test methods have been executed, the company account is deleted. This test suite provides the highest isolation level. It does, however, have a noticeable effect on performance.
- ***SysTestSuiteTTS*** This test suite wraps each test method in a transaction. After the test method has been completed, the transaction is aborted. This provides a fast alternative to the company isolation suites, but it has a couple of limitations:
    - Exceptions can’t be handled. Exceptions thrown inside a transaction abort the transaction automatically and can’t be caught inside the transaction.
    - Test cases that require data to be committed can’t use this test suite.
- ***SysTestSuiteCompIsolateClassWithTts*** This test suite provides a combination of *SysTestSuiteCompanyIsolateClass* and *SysTestSuiteTTS*.

For each test case, you can override the *createSuite* method to select the appropriate suite for your test case. The following code shows how to use the company isolation test suite in the *StackTest* class.

```
public SysTestSuite createSuite()
{;
    return new SysTestSuiteCompanyIsolateClass(this);
}
```

We recommend that you use test projects to group your test cases into suites. You can, however, create your own class extending from *SysTestSuite* and programmatically add test cases and other test suites to it. You can run each test suite in one of the following ways:

- Type the name in the Test toolbar, and then click Run.
- Start the Dynamics AX client with the following command line:

  ```
  StartupCmd=RunTestProject_<Name of test suite class>
  ```

- Implement a static *main* method similar to the one shown in the test case example.

The following code shows the entire *StackTest* test case. Notice the refactoring and the changes in *testQty* to make the test case succeed.

```
class StackTest extends SysTestCase
{
    Stack stack;

    public SysTestSuite createSuite()
    {;
        return new SysTestSuiteCompanyIsolateClass(this);
    }
    public void setUp()
    {;
        super();
        stack = new Stack();
    }
    void testPushPop()
    {;
        stack.push([123]);
        this.assertEquals([123], stack.pop());
    }
    void testQty()
    {;
        stack.push([100]);
        this.assertEquals(1, stack.qty());
        stack.push([200]);
        this.assertEquals(2, stack.qty());
        stack.clear();
        this.assertEquals(0, stack.qty());
    }
    void testFailingPop()
    {;
        //Assert that an exception is expected.
        this.parmExceptionExpected(true, "Stack is empty!");

        //Call the method expected to throw an exception.
        stack.pop();
    }
    static void main(args _args)
    {
        //This method illustrates how to run a test case programmatically.
```

```
        SysTestRunner runner = new SysTestRunner(classStr(StackTest));
        SysTestListenerXML listener =
           new SysTestListenerXML(@"c:\tmp\StackTest.xml");
        ;
        runner.getResult().addListener(listener);
        runner.run();
    }
}
```

Test Projects

The easiest way to group test cases is to use a test project. You can create a test project with the Project Designer in MorphX. The test project can contain groups of test case classes and references to other test projects. You create a new test project by selecting the project type Test Project when creating either a shared or private project. A test project can also contain references to other test projects, which allows the project to scale across many development teams. You create a reference by right-clicking the project root node and selecting New Reference To Test Project.

Figure 3-42 shows a test project that includes a group of common tests containing the test case example and references to two other test projects.

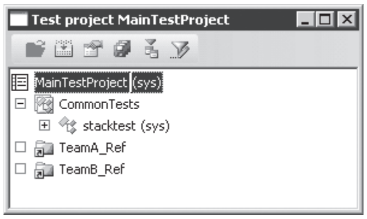


FIGURE 3-42 Test project containing references and a test case

Each test project has its own settings that are persisted with the project definition. This allows you to specify test project settings that follow the project, even through import and export, redeployment, and so on.

You can run a test project in several ways:

- Right-click it, and then click Run.
- Type the name in the Test toolbar, and then click Run.
- Start the Dynamics AX client with the following command line:

  `StartupCmd=RunTestProject_<Name of test project>`

- Use the version control functionality during check-in. Check-in stops if the test fails. You specify the project to run during check-in: from the Microsoft Dynamics AX drop-down menu, point to Tools\Development Tools\Version Control\Setup\System Settings.

The Test Toolbar

When you're working with unit testing, you should open the Test toolbar. You access the Test toolbar, shown in Figure 3-43, from the Microsoft Dynamics AX drop-down menu: Tools\Development Tools\Unit Test\Show Toolbar.

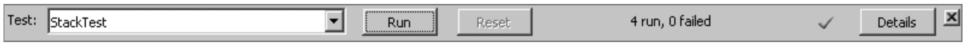


FIGURE 3-43 Test toolbar

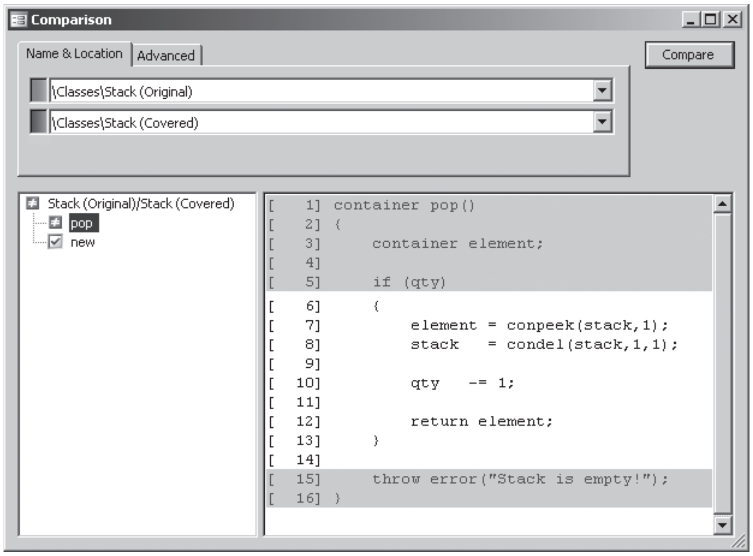
You can type the name of the test case, test suite, or test project you want to run, click Run to execute it, and then, to get information about the result, click Details to open the Test Jobs window. The Test Jobs window shows you the following information collected during the test execution:

- The status of each test case
- Environmental information
- Timing (when the test started and stopped, the duration of the test, and so on)
- Code coverage, when enabled
- Information sent to the Infolog during the test case execution

Note The database listener collects the information displayed in the Test Jobs window. This listener is automatically registered when you run a test via the toolbar.

Code Coverage

The Unit Test framework can collect code coverage information during execution, including a percentage value that indicates how thoroughly you have tested the unit. It also allows you to focus your implementation of the test cases on the parts not covered by other test cases. In addition, the Test Jobs window offers a line-by-line view of the code lines visited. You can enable code coverage in the Unit Test Parameters dialog box: from the Microsoft Dynamics AX drop-down menu, point to Tools\Development Tools\Unit Test\Parameters. However, because much more data is collected, enabling code coverage when executing unit tests dramatically affects performance during execution. Figure 3-44 shows an example of the code coverage recorded by the *testFailingPop* method from the preceding test case example.



\Classes\Stack (Original)

\Classes\Stack (Covered)

Stack (Original)/Stack (Covered)
- pop
- new

```
[   1] container pop()
[   2] {
[   3]     container element;
[   4]
[   5]     if (qty)
[   6]     {
[   7]         element = conpeek(stack,1);
[   8]         stack   = condel(stack,1,1);
[   9]
[  10]         qty   -= 1;
[  11]
[  12]         return element;
[  13]     }
[  14]
[  15]     throw error("Stack is empty!");
[  16] }
```

FIGURE 3-44 Visualization of code coverage

The lines highlighted in gray (lines 1 through 5 and 15 through 16) are the lines visited during execution. The lines not shaded (lines 6 through 14) haven't been visited.

Test Listeners

The value of running a test case is dramatically increased if good reporting options exist. When running a test case or a suite of tests, you can enable one or more listeners. Each listener produces its unique output. Dynamics AX includes many listeners, allowing output to text files, XML files, the database, the Infolog, the Message window, the Print window, and the Progress bar. You can enable test listeners in the Unit Test Parameters dialog box.

Here is the XML generated by the XML listener when you run the *StackTest* unit test.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!-- Created by SysTestListenerXML -->
<test-results date="08-07-2008" time="10:51:34" success="false">
  <test-suite name="stacktest" time="52" success="true" coverage="61.54">
    <results>
      <test-case name="stacktest.testFailingPop" time="31" success="true"
coverage="23.08" />
      <test-case name="stacktest.
testPushPop" time="0" success="true" coverage="50.00" />
      <test-case name="stacktest.testQty" time="21" success="true" coverage="30.77" />
    </results>
  </test-suite>
</test-results>
```

Note Listeners that generate a file write the file to the application log directory. The only way to change the file name and location is to manually register a listener, which we demonstrated in the *StackTest* code on pages 102 to 103.

If you must create a new listener to output to a type of media not supported by default, you can do so by following these steps:

1. Create your own listener class implementing the *SysTestListener* interface. Alternatively, you can inherit from one of the existing test listeners. The methods on your class are invoked when events such as the start and end of test suites and test cases occur and when test cases fail. A *SysTestListenerData* object is passed to each method. The object contains information about the test case or suite, coverage data, and much more. By extracting the information, you can generate output to suit your needs.
2. Modify the base enumeration *SysTestListeners*. You must add an entry that has the same name as your listener class and a label of your choice. This causes the listener to appear in the test parameters form.

Object Model

So far in this chapter we've described the classes in the Unit Test framework and explained how they interact. Figure 3-45 shows this information as a UML object model.

The *SysTestCase* class implements quite a few interfaces. In fact, the Unit Test framework can use any class that implements the *SysTestable* interface as a test case. You can implement the other interfaces if you want more control. It's far easier, however, to create test case classes that extend the *SysTestCase* base class. (For simplicity, Figure 3-45 doesn't show the *SysTestSuite* derived classes or the *SysTestListener* derived classes.)

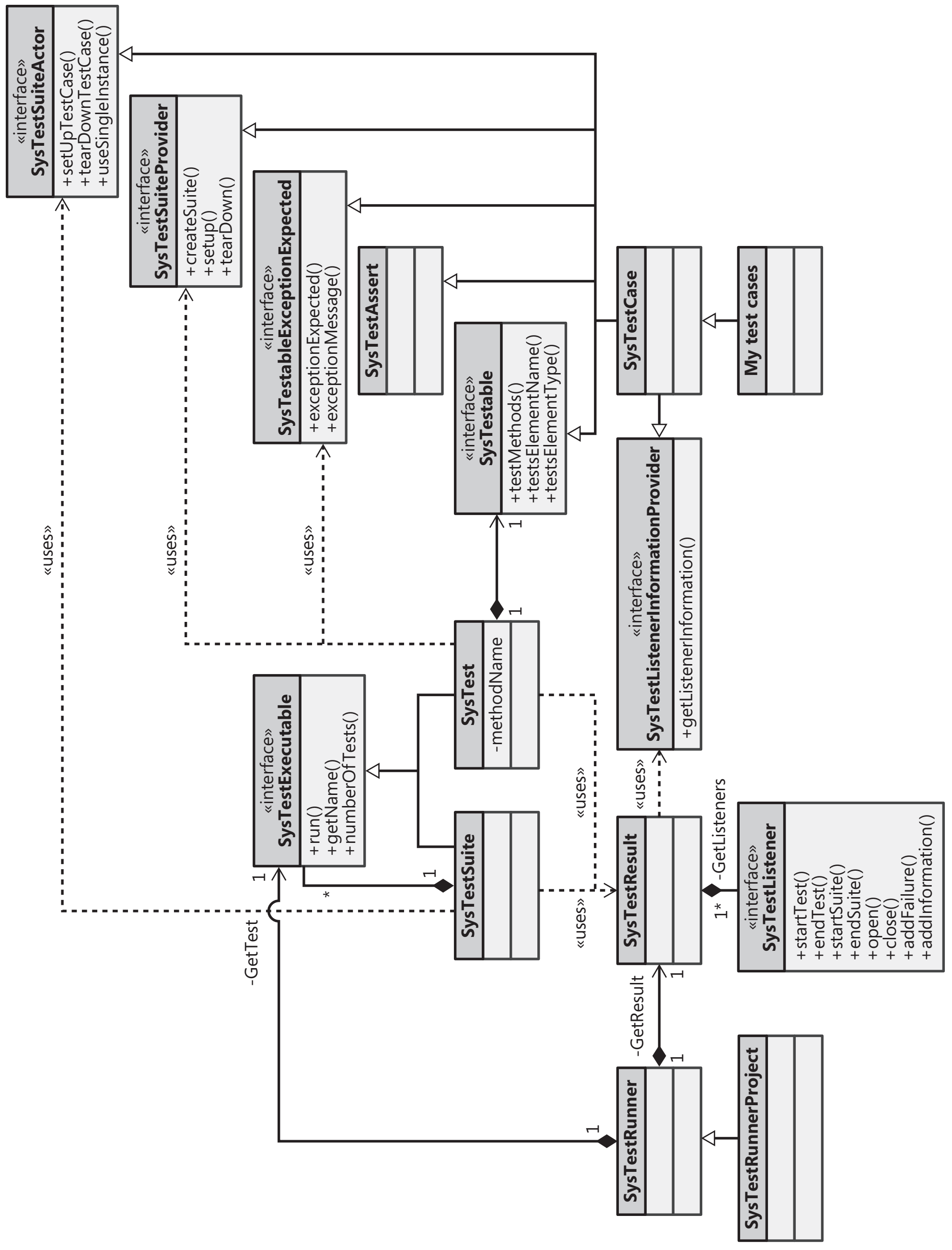


FIGURE 3-45 UML diagram of the Unit Test framework

Chapter 5

Customizing Dynamics AX

In this chapter:

The objectives of this chapter are to:

- Describe how to customize Microsoft Dynamics AX 2009 inventory tables and classes to implement new inventory dimensions.
- Explain how to customize forms in Dynamics AX.
- Describe how to customize reports in Dynamics AX.
- Explain how to customize the number sequence classes in Dynamics AX to implement a new number sequence.

Introduction

Dynamics AX allows you to customize an application by changing or adding metadata or modifying the application's source code. The unique layering feature ensures that you can always return to the point at which you began to make modifications or restore the original metadata and X++ code.

The next section of this chapter describes how to customize Dynamics AX to include a set of new inventory dimensions by customizing a set of tables and classes. The new dimensions automatically appear in forms and reports without requiring you to change the original code or metadata of any of these elements.

The chapter also describes form and report customizations. The sales order form is modified to include a product image, and the sales invoice report is modified to include promotional text.

The last section of the chapter explains how to customize the number sequence classes to enable the use of a new number sequence, which is useful for creating invoice numbers, voucher numbers, and so on.

Table and Class Customization

By default, Dynamics AX 2009 comes with nine default inventory dimensions. (The user can define additional inventory dimensions.) Dimensions describe the characteristics of items or item lots. Item dimensions might include characteristics such as configuration, model, and size. Item lots might have storage dimensions, such as site, warehouse, location, or pallet, or they might be identified by a serial number and batch number. The site dimension is new in Dynamics AX 2009.

The following customization scenario describes how to customize tables and classes used by the inventory dimension feature to implement two new item dimensions that describe a specific bicycle configuration: frame size and wheel size. This description isn't an exhaustive list of elements that you must change; instead, it offers guidelines for finding the elements necessary to customize the full implementation of a new inventory dimension.

Creating New Dimension Types

When implementing new inventory dimensions, your first task is to create extended data types for each of the dimensions. Doing so provides the following benefits:

- To apply the inventory dimensions to multiple tables, you define the type just once and then apply it to each table.
- The *Label* property, the *HelpText* property, and a few constraints can be defined on the data type, ensuring consistent behavior and appearance of fields of the same type.
- If the type is declared as a parameter or a return type for a method, you can declare variables of the type in X++ code to optimize IntelliSense responsiveness and to improve the readability of the code.

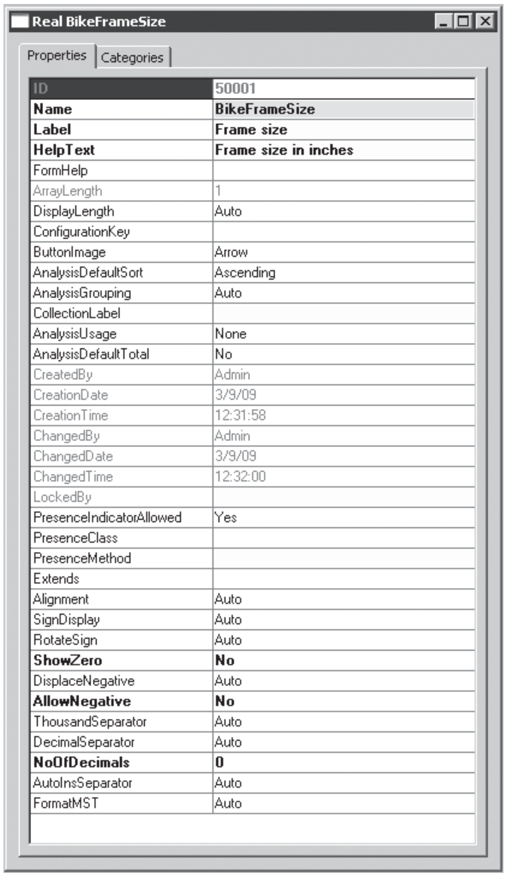
This scenario defines a table in which a field of the specific type is part of the primary key. You can define the relationship to this table on the extended data type and subsequently instruct the application runtime to provide lookups and Go To The Main Table Form support.

In this example, you enter the Data Dictionary in the Application Object Tree (AOT) and create a *BikeFrameSize* extended data type and a *BikeWheelSize* extended data type. Table 5-1 lists the property settings that deviate from the default settings.

TABLE 5-1 ***BikeFrameSize* and *BikeWheelSize* Property Settings**

Property	*BikeFrameSize*	*BikeWheelSize*
Type	Real	Real
Label	Frame size	Wheel size
HelpText	Frame size in inches	Wheel size in inches
AllowNegative	No	No
ShowZero	No	No
NoOfDecimals	0	0

Figure 5-1 shows the property sheet for the *BikeFrameSize* extended data type, accessible by clicking Properties on the context menu for the type.



ID	50001
Name	BikeFrameSize
Label	Frame size
HelpText	Frame size in inches
FormHelp	
ArrayLength	1
DisplayLength	Auto
ConfigurationKey	
ButtonImage	Arrow
AnalysisDefaultSort	Ascending
AnalysisGrouping	Auto
CollectionLabel	
AnalysisUsage	None
AnalysisDefaultTotal	No
CreatedBy	Admin
CreationDate	3/9/09
CreationTime	12:31:58
ChangedBy	Admin
ChangedDate	3/9/09
ChangedTime	12:32:00
LockedBy	
PresenceIndicatorAllowed	Yes
PresenceClass	
PresenceMethod	
Extends	
Alignment	Auto
SignDisplay	Auto
RotateSign	Auto
ShowZero	No
DisplaceNegative	Auto
AllowNegative	No
ThousandSeparator	Auto
DecimalSeparator	Auto
NoOfDecimals	0
AutoInsSeparator	Auto
FormatMST	Auto

FIGURE 5-1 Property sheet for the *BikeFrameSize* extended data type

Best Practices Creating labels for text in the *Label* and *HelpText* properties is, of course, a best practice, but the text in this example is written as a literal (as opposed to referencing a label) to improve readability.

Next, create two tables, named *BikeFrameSizeTable* and *BikeWheelSizeTable*, in which the frame and wheel sizes for each item can be stored. In addition to the specific inventory dimension types, the tables also contain an *ItemId* field and a *Name* field. The *ItemId* and dimension in each table constitute the table's primary index.

Table 5-2 lists the *BikeFrameSizeTable* property settings that deviate from the default settings. (The property settings for *BikeWheelSizeTable* are identical except for the *BikeWheelSize* field and its extended property type.)

TABLE 5-2 Field Property Settings

Property	*ItemId*	*BikeFrameSize*	*Name*
Type	*String*	Real	String
ExtendedDataType	*ItemId*	*BikeFrameSize*	Name
Mandatory	Yes	Yes	No (default)
AllowEdit	No	No	Yes (default)

Create a unique index on both tables. For *BikeFrameSizeTable*, name the index *FrameIdx* and make it contain the *ItemId* field and the *BikeFrameSize* field. For *BikeWheelSizeTable*, name the index *WheelIdx* and make it contain the *ItemId* field and the *BikeWheelSize* field. Declare the indexes as the *PrimaryIndex* on the respective tables. In the AOT, the fields and the indexes appear as shown in Figure 5-2.

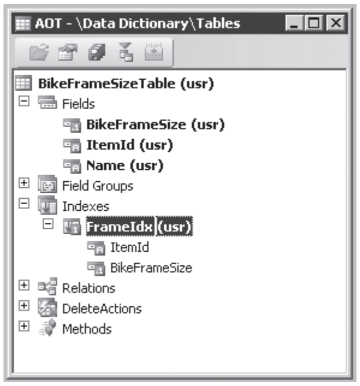
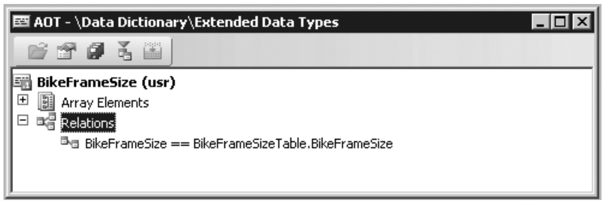


FIGURE 5-2 *BikeFrameSizeTable* definition

In addition to the fields and index shown in Figure 5-2, you should also set properties in the tables for caching, form references, and so on, and the table should contain field groups and methods for checking the validity of the fields. However, it is beyond the scope of this chapter to describe these enhancements. The Microsoft Dynamics AX 2009 software development kit (SDK) contains guidelines and best practices for creating tables.

After you define the tables, you should update the extended data types to reflect their relationship to the individual tables, as shown in Figure 5-3.



BikeFrameSize == BikeFrameSizeTable.BikeFrameSize

FIGURE 5-3 Extended data type relations of *BikeFrameSize*

This relationship instructs the Dynamics AX runtime to provide lookup and Go To The Main Table Form functionality when fields of these types appear on forms. The application runtime uses the related table as the data source for the lookup form and also to find the main table form from the *FormRef* property on the table. You must therefore create forms for the *BikeFrameSizeTable* and *BikeWheelSizeTable* tables and menu items to open the forms. These menu items are added to the *FormRef* properties on the corresponding tables. You could design the forms to mirror the form shown in Figure 5-4. See the Microsoft Dynamics AX 2009 SDK for general information on designing forms.

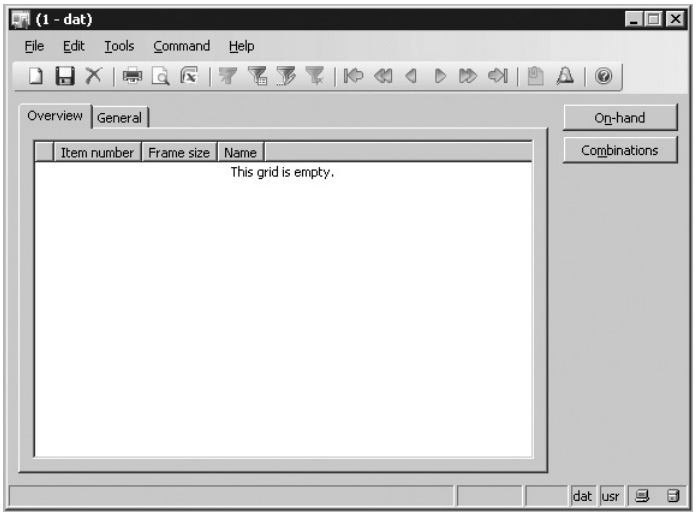


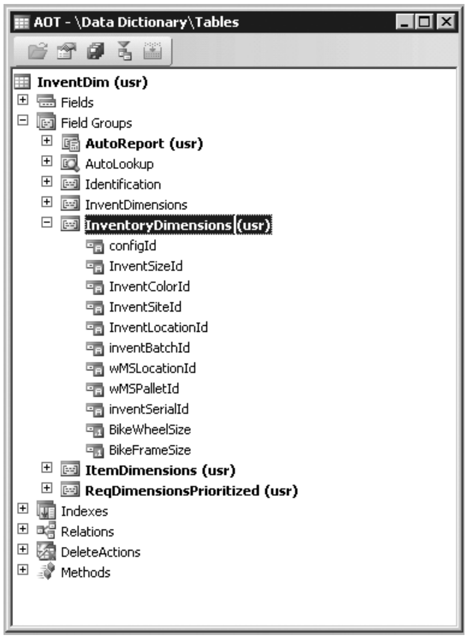
FIGURE 5-4 Frame Sizes form

Adding New Dimensions to a Table

To store transactions with the new inventory dimensions, the dimensions must be added to the *InventDim* table. You do this by creating two new fields, *BikeFrameSize* and *BikeWheelSize*, of the corresponding type on the *InventDim* table. You should also add these fields to the unique *DimIdx* index because any combination of inventory dimensions can exist only once in the *InventDim* table.

The display of inventory dimensions in almost any form in the Dynamics AX application is based on field groups and where the content of the field group in the form is built at run time. The forms runtime in Dynamics AX builds the group from the list of fields in the associated field group defined on the *InventDim* table. Therefore, by adding the new fields to the *InventoryDimensions* field group on the *InventDim* table, you make the two new fields available in almost any form that displays inventory dimensions. Position the fields in the field group based on where you want them to appear relative to the other dimensions, as shown in Figure 5-5.

Figure 5-5 shows *usr* flags on the *AutoReport* and *ItemDimensions* field groups, indicating that the custom fields have been added to these groups as well. The *AutoReport* group is modified so that it prints the new dimensions if you create an auto report by clicking Print on a form; the *ItemDimensions* group is modified because the new dimensions are considered to be item dimensions.


AOT - \Data Dictionary\Tables

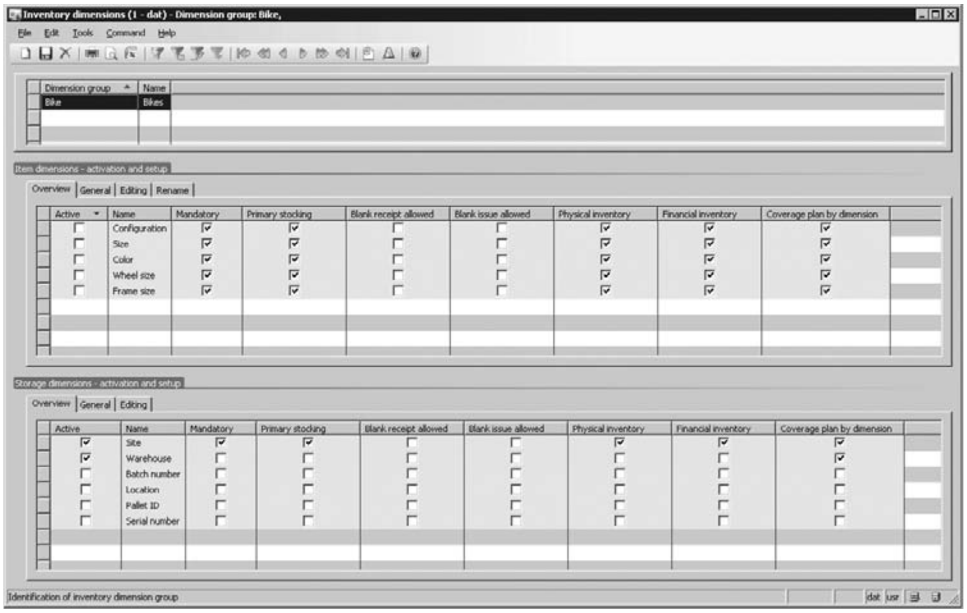
```
InventDim (usr)
+ Fields
- Field Groups
  + AutoReport (usr)
  + AutoLookup
  + Identification
  + InventDimensions
  - InventoryDimensions (usr)
      configId
      InventSizeId
      InventColorId
      InventSiteId
      InventLocationId
      inventBatchId
      wMSLocationId
      wMSPalletId
      inventSerialId
      BikeWheelSize
      BikeFrameSize
  + ItemDimensions (usr)
  + ReqDimensionsPrioritized (usr)
+ Indexes
+ Relations
+ DeleteActions
+ Methods
```

FIGURE 5-5 *InventDim* table with customized *InventoryDimensions* field group

Although the inventory dimensions are now available in any form because of the interpretation of the field groups by the Dynamics AX forms runtime, the fields still aren't visible or editable because they aren't enabled in any inventory dimension group. Moreover, the two new inventory dimensions automatically appear in the Dimension Groups form because the inventory dimension feature also interprets the *InventoryDimensions* field group on the *InventDim* table to find all the currently available inventory dimensions. To make the form work with the new dimensions, you merely state whether the new dimensions are item dimensions. You do this by adding the new dimensions to the *isFieldItemDim* method on the *InventDim* table, as shown in the following X++ code. The added lines are shown in bold.

```
static public boolean isFieldIdItemDim(fieldId dimFieldId)
{
    ;
    #InventDimDevelop

    switch (dimFieldId)
    {
        case (fieldnum(InventDim,ConfigId))         :
        case (fieldnum(InventDim,InventSizeId))     :
        case (fieldnum(InventDim,InventColorId))    :
        case (fieldnum(InventDim,BikeFrameSize))    : // Frame size added
        case (fieldnum(InventDim,BikeWheelSize))    : // Wheel size added
            return true;

        case (fieldnum(InventDim,InventSiteId))     :
        case (fieldnum(InventDim,InventLocationId)) :
        case (fieldnum(InventDim,InventBatchId))    :
        case (fieldnum(InventDim,wMSLocationId))    :
        case (fieldnum(InventDim,wMSPalletId))      :
        case (fieldnum(InventDim,InventSerialId))   :
            return false;
    }

    throw error("@SYS70108");
}
```

The new dimensions will be available for setup in the Dimension Groups form, which is reached through the navigation pane under Inventory Management\Setup\Dimensions\ Dimension Groups. The dimensions are located in the Item Dimensions grid, as shown in Figure 5-6.



Inventory dimensions (1 - dat) - Dimension group: Bike,

Dimension group	Name
Bike	Bikes

Item dimensions - activation and setup

Overview | General | Editing | Rename

Active	Name	Mandatory	Primary stocking	Blank receipt allowed	Blank issue allowed	Physical inventory	Financial inventory	Coverage plan by dimension
☐	Configuration	✓	✓	☐	☐	✓	✓	✓
☐	Size	✓	✓	☐	☐	✓	✓	✓
☐	Color	✓	✓	☐	☐	✓	✓	✓
☐	Wheel size	✓	✓	☐	☐	✓	✓	✓
☐	Frame size	✓	✓	☐	☐	✓	✓	✓

Storage dimensions - activation and setup

Overview | General | Editing

Active	Name	Mandatory	Primary stocking	Blank receipt allowed	Blank issue allowed	Physical inventory	Financial inventory	Coverage plan by dimension
✓	Site	✓	✓	☐	☐	✓	✓	✓
✓	Warehouse	☐	☐	☐	☐	☐	☐	✓
☐	Batch number	☐	☐	☐	☐	☐	☐	☐
☐	Location	☐	☐	☐	☐	☐	☐	☐
☐	Pallet ID	☐	☐	☐	☐	☐	☐	☐
☐	Serial number	☐	☐	☐	☐	☐	☐	☐

Identification of inventory dimension group dat usr

FIGURE 5-6 Dimension Groups form with new item dimensions

Important You need to restart the Application Object Server (AOS) after adding fields to the *InventoryDimensions* field group because the list of fields in the group is cached in memory on both the client and the server tiers.

Enabling New Dimensions in Forms

You can enable new dimensions by setting up dimension groups, but you won't see them yet in the forms. The inventory dimension feature uses a temporary table named *InventDimParm* to carry certain information, such as whether a dimension has the following attributes:

- Is enabled
- Is an item dimension
- Is a primary stocking dimension
- Is visible
- Serves as a filter-by term
- Serves as a group-by term
- Serves as an order-by term

The dimension groups are enabled and controlled by reflecting each inventory dimension as a boolean flag field on the *InventDimParm* table and then matching the corresponding fields in the X++ code. For example, when a dimension group is queried to determine which dimensions are active, an *InventDimParm* record is returned where the corresponding flag field is set to true for the active dimensions. The remaining flags are set to false. You must therefore add a frame-size flag and a wheel-size flag to the *InventDimParm* table, as shown in Table 5-3.

TABLE 5-3 ***BikeFrameSizeFlag* and *BikeWheelSizeFlag* Property Settings**

Property	*BikeFrameSizeFlag*	*BikeWheelSizeFlag*
Type	enum	enum
Label	Frame size	Wheel size
HelpText	View by frame size	View by wheel size
ExtendedDataType	*NoYesId*	*NoYesId*
Enum	*NoYes*	*NoYes*

You should also add the new fields to the *FixedView* and *View* field groups defined on the *InventDimParm* table, because they are used in forms from which it is possible to specify whether a dimension should be visible.

When you add fields to the table and field groups, you must map the new fields on the *InventDim* table to the corresponding fields on the *InventDimParm* table in the X++ code. To do this, you modify the *dim2DimParm* method on the *InventDim* table, as shown in the following X++ code. The added mappings of *BikeFrameSize* and *BikeWheelSize* appear in bold.

```
static public fieldId dim2dimParm(fieldId dimField)
{
    ;
    #InventDimDevelop

    switch (dimField)
    {
        case (fieldnum(InventDim,ConfigId))         :
            return fieldnum(InventDimParm,ConfigIdFlag);
        case (fieldnum(InventDim,InventSizeId))     :
            return fieldnum(InventDimParm,InventSizeIdFlag);
        case (fieldnum(InventDim,InventColorId))    :
            return fieldnum(InventDimParm,InventColorIdFlag);
        case (fieldnum(InventDim,InventSiteId))     :
            return fieldnum(InventDimParm,InventSiteIdFlag);
        case (fieldnum(InventDim,InventLocationId)) :
            return fieldnum(InventDimParm,InventLocationIdFlag);
        case (fieldnum(InventDim,InventBatchId))    :
            return fieldnum(InventDimParm,InventBatchIdFlag);
        case (fieldnum(InventDim,wMSLocationId))    :
            return fieldnum(InventDimParm,WMSLocationIdFlag);
        case (fieldnum(InventDim,wMSPalletId))      :
            return fieldnum(InventDimParm,WMSPalletIdFlag);
        case (fieldnum(InventDim,InventSerialId))   :
            return fieldnum(InventDimParm,InventSerialIdFlag);
        case (fieldnum(InventDim,BikeFrameSize))    : // Add mapping
            return fieldnum(InventDimParm,BikeFrameSizeFlag);
        case (fieldnum(InventDim,BikeWheelSize))    : // Add mapping
            return fieldnum(InventDimParm,BikeWheelSizeFlag);
    }

    throw error(strfmt("@SYS54431",funcname()));
}
```

You must make the same modification to the *dimParm2Dim* method on the same table to map *InventDimParm* fields to *InventDim* fields.

Customizing Other Tables

The customizations made so far allow the new dimensions to be enabled on dimension groups and presented in forms. However, you should also consider customizing the following tables by adding inventory dimensions to them:

- *BOMTmpUsedItem2ProducedItem*
- *InventCostTmpTransBreakdown*
- *InventDimCombination*
- *InventSumDateTrans*
- *InventSumDeltaDim*
- *PBADefault*
- *PBATreeInventDim*
- *PriceDiscTmpPrintout*
- *InterCompanyInventDim*

Whether and how you should customize these tables depends on the functionality you're implementing. Be sure to examine how the inventory dimensions are implemented and used for each of the tables before you begin customizing.

Adding Dimensions to Queries

Because of the generic implementation of the inventory dimension concept using the *InventDim* and *InventDim Parm* tables, a substantial number of queries written in X++ use just a few patterns to select, join, and filter the inventory dimensions. So that you don't have to repeatedly copy and paste the same X++ code, these patterns exist as macros that you can apply in your code. To modify these queries, you simply customize the macros and then recompile the entire application to update the X++ code with the new dimensions.

You should customize the following macros:

- *InventDimExistsJoin*
- *InventDimGroupAllFields*
- *InventDimJoin*
- *InventDimSelect*

The bold text in the following X++ code shows the changes that you must make to the *InventDimExistsJoin* macro to enable the two new dimensions for all exists joins written as statements involving the *InventDim* table.

```
/* %1 InventDimId           */
/* %2 InventDim             */
/* %3 InventDimCriteria     */
/* %4 InventDimParm         */
/* %5 Index hint            */

exists join tableId from %2
    where
  (%2.InventDimId      == %1) &&
  (%2.ConfigId         == %3.ConfigId         || ! %4.ConfigIdFlag)         &&
  (%2.InventSizeId     == %3.InventSizeId     || ! %4.InventSizeIdFlag)     &&
  (%2.InventColorId    == %3.InventColorId    || ! %4.InventColorIdFlag)    &&
  (%2.BikeFrameSize    == %3.BikeFrameSize    || ! %4.BikeFrameSizeFlag)    &&
  (%2.BikeWheelSize    == %3.BikeWheelSize    || ! %4.BikeWheelSizeFlag)    &&
  (%2.InventSiteId     == %3.InventSiteId     || ! %4.InventSiteIdFlag)     &&
  (%2.InventLocationId == %3.InventLocationId || ! %4.InventLocationIdFlag) &&
  (%2.InventBatchId    == %3.InventBatchId    || ! %4.InventBatchIdFlag)    &&
  (%2.WMSLocationId    == %3.WMSLocationId    || ! %4.WMSLocationIdFlag)    &&
  (%2.WMSPalletId      == %3.WMSPalletId      || ! %4.WMSPalletIdFlag)      &&
  (%2.InventSerialId   == %3.InventSerialId   || ! %4.InventSerialIdFlag)

#InventDimDevelop
```

The three remaining macros are just as easy to modify. Just remember to recompile the entire application after you make your changes.

Adding Lookup, Validation, and Defaulting X++ Code

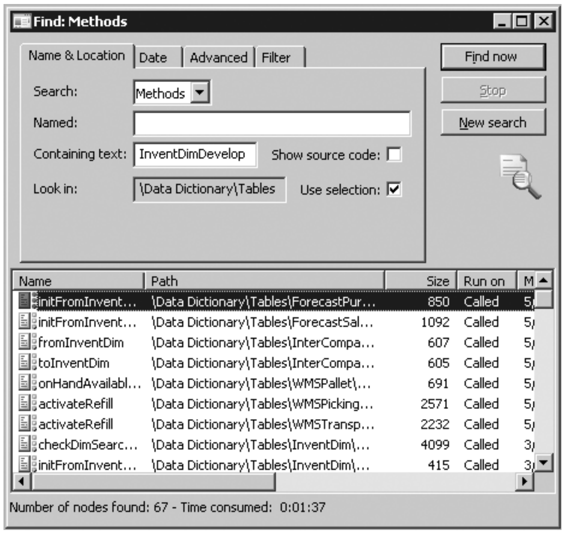
In addition to macro customizations and the customizations to the previously mentioned methods on the *InventDim* table, you must implement and customize lookup, validation, and defaulting methods. These include methods such as the *InventDim::findDim* lookup method, the *InventDim.validateWriteItemDim* validation method, and the *InventDim.initFromInvent-DimCombination* defaulting method. The necessary changes in the *InventDim::findDim* lookup method for the new inventory dimensions are shown in bold in the following X++ code.

```
server static public InventDim findDim(InventDim _inventDim,
                                       boolean   _forupdate = false)
{
    InventDim   inventDim;
    ;
    if (_forupdate)
        inventDim.selectForUpdate(_forupdate);

    select firstonly inventDim
        where inventDim.ConfigId         == _inventDim.ConfigId
           && inventDim.InventSizeId     == _inventDim.InventSizeId
           && inventDim.InventColorId    == _inventDim.InventColorId
           && inventDim.BikeFrameSize    == _inventDim.BikeFrameSize
           && inventDim.BikeWheelSize    == _inventDim.BikeWheelSize
           && inventDim.InventSiteId     == _inventDim.InventSiteId
           && inventDim.InventLocationId == _inventDim.InventLocationId
           && inventDim.InventBatchId    == _inventDim.InventBatchId
           && inventDim.wmsLocationId    == _inventDim.wmsLocationId
           && inventDim.wmsPalletId      == _inventDim.wmsPalletId
           && inventDim.InventSerialId   == _inventDim.InventSerialId;

    #inventDimDevelop

    return inventDim;
}
```

Notice the use of the *inventDimDevelop* macro in the preceding method. This macro contains only the following comment:

```
/* used to locate code with direct dimension references */
```

Performing a global search for use of the *inventDimDevelop* macro should be sufficient to find all the X++ code that you must consider when implementing a new dimension. This search returns all the methods that require further investigation. Figure 5-7 shows results of a search for the use of the macro on all tables.


Find: Methods

Name & Location | Date | Advanced | Filter

Search: Methods

Named:

Containing text: InventDimDevelop   Show source code: ☐

Look in: \Data Dictionary\Tables   Use selection: ☑

Find now | Stop | New search

Name	Path	Size	Run on	M
initFromInvent…	\Data Dictionary\Tables\ForecastPur…	850	Called	5/
initFromInvent…	\Data Dictionary\Tables\ForecastSal…	1092	Called	5/
fromInventDim	\Data Dictionary\Tables\InterCompa…	607	Called	5/
toInventDim	\Data Dictionary\Tables\InterCompa…	605	Called	5/
onHandAvailabl…	\Data Dictionary\Tables\WMSPallet\…	691	Called	5/
activateRefill	\Data Dictionary\Tables\WMSPicking…	2571	Called	5/
activateRefill	\Data Dictionary\Tables\WMSTransp…	2232	Called	5/
checkDimSearc…	\Data Dictionary\Tables\InventDim\…	4099	Called	3/
initFromInvent…	\Data Dictionary\Tables\InventDim\…	415	Called	3/

Number of nodes found: 67 - Time consumed: 0:01:37

FIGURE 5-7 Search results for the *inventDimDevelop* macro

Best Practices Inserting the *inventDimDevelop* macro in X++ code when it makes a direct reference to an inventory dimension is considered a best practice. Doing so makes implementing new dimensions easier.

Most of the methods you find when searching for the macro are lookup, validation, and defaulting methods, but you also see methods that aren't in these categories. Such methods include those that modify the *Query* object, such as the *InventDim::queryAddHintFromCaller* method, and methods that describe dimensions, such as *InventDimParm.isFlagSelective*. You should also review these methods when investigating the X++ code.

Tip Although the inventory dimension feature is implemented with the *inventDimDevelop* macro to direct developers to the methods they need to change, you might encounter methods with no macro included or tables, forms, or reports for which the inventory dimensions are not used generically. We therefore advise you to use the cross-reference system on an existing dimension that has the same behavior as the new dimension to determine its use and review it appropriately. You should also investigate whether the new dimension is or should be available in the same element.

Form Customization

Like most of the elements in the AOT, forms can be customized to include additional information and actions, such as fields and buttons, and to fulfill user requirements. The design and behavior of a form are provided by a combination of the form and the tables that are bound to the form.

Best Practices Even though you can implement all necessary customizations by modifying just the form, we don't recommend this approach. As a best practice, you should implement application customizations at the lowest level possible, preferably through changes to a table or a class rather than through changing specific forms.

The best way to implement forms is to keep most of the business logic and design decisions in tables and classes, focusing only on the positioning of fields and menu items when designing the form. This approach has several advantages:

- X++ code in forms is executed on the client tier only; X++ code in table methods can be executed on the server tier to optimize performance.
- Customizations made to a form are restricted to that form; customizations made to a table or a class apply to all forms that use that table or class as a source of data. This results in a consistent user experience wherever the table is used.
- When a form is customized, the entire form is copied to the current layer; customizations to tables and classes are more granular. When fields, field groups, and methods are customized, a copy of the specific element is in the current layer only. This makes upgrading to service packs and new versions easier.
- X++ customizations to the validate, default, and database trigger methods on forms, such as create, validateWrite, and write, affect only the records that are modified through the user interface. If records are modified somewhere other than that form, then that customized form's X++ code doesn't execute.

The following actions be customized only on the form, not by customizing a table:

- Enable and disable fields and other user interface elements (*Enabled* = Yes/No)
- Show and hide fields and other user interface elements (*Visible* = Yes/No)

However, you should consider having a table or a class method determine the business logic on the form. An example of this is shown in the following lines of X++ code from the InventTable form, in which a method on the table determines whether a field can be edited.

```
void setItemDimEnabled()
{
    boolean     configActive    = inventTable.configActive();
    ...
    inventTable_ds.object(
              fieldnum(InventTable,StandardConfigId)).allowEdit(configActive);
    inventTable_ds.object(
              fieldnum(InventTable, StandardConfigId)).skip(!configActive);
    ...
}
```

By moving these decision-making methods to a table or class, you make them available to other forms.

Learning Form Fundamentals

The rich client user interface in Dynamics AX is made up of forms that are declared in metadata and often contain associated code. Ideally, you should customize these forms as changes to metadata and make any changes at the lowest level (i.e., table level rather than form level) possible to ensure the greatest amount of metadata and code reuse.

The most visible change from Dynamics AX 4.0 to Dynamics AX 2009 is the change from a predominantly *multiple-document interface* (MDI) to a predominantly *single-document interface* (SDI). Forms with a *WindowType* property value of *Standard* (the default) are now SDI forms, and the *WindowType* values of *ListPage* and *ContentPage* have been added to fill the Workspace content area to provide a navigation experience similar to that in Microsoft Office Outlook. The different *WindowType* values share the same object model, metadata, and method overrides, so form customization skills are applicable across all forms.

Customizing with Metadata

Metadata customization is preferred over code customization because metadata changes (also called deltas) are easier to merge than code changes.

When customizing forms, you should be aware of the important properties, the metadata associations, and the metadata inheritance that is being used to fully define the form and its contents.

Metadata associations You edit the metadata in Dynamics AX by using the AOT. The base definitions for forms contained within the *AOT\Forms* node is composed of a hierarchy of metadata that is located in other nodes in the AOT. To fully understand a form, you should investigate the metadata associations it makes. For example, a form uses tables that are declared in the *AOT\Data Dictionary\Tables* node, security keys that are declared in the *AOT\Data Dictionary\Security Keys* node, menu items that are declared in the *AOT\Menu Items* node, queries that are declared in the *AOT\Queries* node, and classes that are declared in the *AOT\Classes* node.

Metadata inheritance You need to be aware of the inheritance within the metadata used by forms. For example, tables use Base Enums, Extended Data Types, and Configuration Key. A simple example of inheritance is that the *Image* properties on a *MenuItemButton* are inherited from the associated *MenuItem* if they aren't explicitly specified on that *MenuItemButton*. Table 5-4 shows important examples of pieces of metadata that are inherited from associated metadata.

Inheritance also occurs within forms. Controls that are contained within other controls receive certain metadata property behaviors from their parents unless different property values are specified, including HTMLHelpFile, HTMLHelpTopic, Security Key, Configuration Key, Enabled, and the various Font properties.

TABLE 5-4 Examples of Metadata Inheritance

Type of Metadata	Sources
Labels and HelpText	MenuItem→MenuItemButton Control Base Enum→Extended Data Type→Table Field→Form DataSource Field→Form Control (The *Base Enum Help* property is the equivalent of the *HelpText* property found in the other types.)
Relations	Extended Data Type→Table
Security keys	Table Field→Table→Form Control MenuItem→MenuItemButton Control Form→Form Control
Configuration keys	Base Enum→Extended Data Type→Table Field→Form DataSource Field→Form Control
Image properties (e.g., *NormalImage*)	MenuItem→MenuItemButton Control

Menu definitions Dynamics AX 2009 has a number of new navigation capabilities in the form of area pages and the address bar to complement the existing navigation pane (sometimes referred to as the "WunderBar"). In terms of metadata, the area pages and address bar are mostly just additional methods of exposing the existing menu metadata defined in the *AOT\Menus* and *AOT\Menu Items* nodes. The modules are defined in AOT\Menus\MainMenu, and you can follow the menu structure from that starting point. For example, the *Accounts Receivable* module is represented by the AOT\Menus\MainMenu\Cust MenuReference and is defined as AOT\Menus\Cust.

The menu metadata for list pages and content pages has some small changes. A primary list page is implemented as a submenu with *IsDisplayedInContentArea=Yes*, *MenuItemType=Display*, and *MenuItemName* populated. A secondary list page, a list page that adds ranges to a primary list page, is implemented as a menu item under the sub-menu of its primary list page. The list pages and content pages are navigation places, so all their menu item and submenu references are set to *IsDisplayedInContentArea=Yes* so that they appear in the Places group in the area pages and the Places section in the navigation pane. The other menu items in the root of each module's menu definition are displayed in the Common Forms group in the area pages and in the root of the Forms section in the navigation pane.

Important metadata properties Many properties are available to developers, but some are more important than others. Table 5-5 describes the most important form design properties, and Table 5-6 describes the most important form data source properties.

TABLE 5-5 Important Form *Design* Metadata Properties

Property	Explanation
Caption	The caption text shown in the title bar of a standard form or in the Filter Pane of a list page.
TitleDataSource	The data source information displayed in a standard form's caption text and used to provide filter information in the caption text of a list page.
WindowType	*Standard* - (Default) A standard SDI form that opens as a separate window with a separate entry in the Windows taskbar. *ContentPage* - A form that fills the Workspace content area. *ListPage* - A special style of ContentPageused to display records in a simple way that provides quick access to filtering capabilities and actions. It requires at least an Action Pane and a Grid. *Workspace* - A form that opens as an MDI window within the workspace. Workspace forms should be developer-specific forms. *Popup* - A form that opens as a subform to its parent. Popup forms don't have a separate entry in the Windows taskbar and can't be layered with other windows.
AllowFormCompanyChange	Specifies whether the form allows company changes when used as a child form with a cross-company dynalink. No - (Default) Form closes if parent form changes its company scope. Yes - Form dynamically changes company scope as needed.
HTMLHelpFile	Specifies the path to the Help topic file.
HTMLHelpTopic	Specifies the topic to use from the referenced Help file.

TABLE 5-6 Important Form *DataSource* Metadata Properties

Property	Explanation
Name	Named reference for the data source. A best practice is to use the same name as the table name.
Table	Specifies the table used as the data source.
CrossCompanyAutoQuery	*No* - (Default) Data source gets data from the current company. *Yes* - Data source gets data from all companies (e.g., retrieves customers from all companies).

Property	Explanation
JoinSource	Specifies the data source to link or join to as part of the query. For example, in the SalesTable form, SalesLine is linked to SalesTable. Data sources joined together are represented in a single query whereas links are represented as a separate query.
LinkType	Specifies the link or join type used between this data source and the data source specified in the *JoinSource* property. Joins are required when two data sources are displayed in the same grid. Joined data sources are represented in a single query whereas a linked data source is represented in a separate query. **Links** *Delayed* - (Default) A pause is inserted before linked child data sources are updated, enabling faster navigation in the parent data source because the records from the child data sources are not updated immediately. For example, the user could be scrolling past several orders without immediately seeing each order line. *Active* - The child data source is updated immediately when a new record in the parent data source is selected. Continuous updates consume lots of resources. *Passive* - Linked child data sources are not updated automatically. The link is established by the kernel, but the application developer must trigger the query to occur when desired by calling "ExecuteQuery" on the linked data source. **Joins** *InnerJoin* - Selects records from the main table that have matching records in the joined table, and vice versa. There is one record for each match. Records without related records in the other data source are eliminated from the result. *OuterJoin* - Selects records from the main table whether or not they have matching records in the joined table. An outer join doesn't require each record in the two joined tables to have a matching record. *ExistJoin* - Selects a record from the main table for each matching record in the joined table. *NotExistJoin* - Selects records from the main table that don't have a match in the joined table.
InsertIfEmpty	*Yes* - (Default) A record is automatically created for the user if none is present. *No* - The user needs to manually create the first record. This setting is often used when a special record creation process or interface is used.

Image metadata Dynamics AX 2009 makes greater use of images and icons throughout the application to provide the user with additional visual cues. Icons are used extensively in list pages to help users identify specific actions. The metadata properties used to associate images and icons with buttons, menus (menu items), and other controls depends on their location. Table 5-7 describes the metadata properties used for the three common image locations.

TABLE 5-7 Image Metadata

Image Location	Explanation
Embedded	Embedded image resources are associated with buttons, menus, and other controls using the *NormalResource* and *DisabledResource* properties. These resources are compiled into the kernel and therefore you can't add to the embedded resources list. The full list of embedded resources can be seen in the Embedded Resources form (Tools\Development Tools\Embedded Resources).
File	File image resources are associated with buttons, menus, and other controls using the *NormalImage* and *DisabledImage* properties. File image resources should be on the local computer wherever possible for performance reasons, but can be on a file share if needed.
AOT	AOT image resources cannot be utilized simply through metadata. AOT resources can be utilized only by adding code that copies the AOT resource into a temp folder and sets the *NormalImage* property at run time. The following code should be added into a dependable form method override (such as the *Init* method after the super call) for execution at run time: `SomeButton.normalImage(SysResource::getImagePath("<AOT Resource Name>"));`

Customizing with Code

You should customize forms with code only as a last resort. Customizing with metadata is much more upgrade friendly since metadata change conflicts are straight forward to resolve whereas code change conflicts need deeper investigation that sometimes involves creating a new merged method that attempts to replicate the behavior from the two original methods.

When you start to customize Dynamics AX, the following ideas may provide good starting points for investigation:

- Leverage examples in the base Dynamics AX 2009 codebase by using the *Find* command on the *Forms* node in the AOT (Ctrl+F).
- Refer to the system documentation entries (AOT\System Documentation) for information about system classes, tables, functions, enumerations, and other system elements that have been implemented in the AX kernel.
- When investigating the form method call hierarchy for a suitable location to place customization code, add a debug breakpoint in the *Form Init* method and step through the execution of method overrides. Note that control events (e.g., clicked) do not trigger debugging breakpoints. An explicit breakpoint (i.e., "*breakpoint;*") keyword is needed in the X++ code.

To enable simpler code maintenance, the following rules should be followed:

- Utilize the table and field functions of *FieldNum* (e.g., *fieldnum(SalesTable, SalesId)*) and *TableNum* (e.g., *tablenum(SalesTable)*) when working with form data sources.
- Avoid hard coding strings by using *sys* labels (e.g., *throw error("@SYS88659");*) and functions like *FieldStr* (e.g., *fieldstr(SalesTable, SalesId*)) and *TableStr* (e.g., *tablestr(SalesTable)*).
- Use as few method overrides as possible. Each additional method override has a chance of causing merge issues during future upgrades, patch applications, or code integrations.

When X++ code is executed in the scope of a form, there are some form-specific global variables created in X++ to help developers access important objects related to the form. These global variables are described in Table 5-8.

TABLE 5-8 Form-Specific Global X++ Variables

Variable	Use and Example

Element	Variable that provides easy access to the *FormRun* object in scope. Commonly used to call methods or change the design.
	`element.args().record().TableId == tablenum(SalesTable)`
	
	`name = element.design().addControl(FormControlType::String, "X");`

DataSourceName (e.g., *SalesTable*)	Variable that provides easy access to the *current/active record/cursor* in each data source. Commonly used to call methods or *get/set* properties on the current record. `if (SalesTable.type().canHaveCreditCard())`
DataSourceName_DS (e.g., *SalesTable_DS*)	Variable that provides easy access to each data source. Commonly used to call methods or *get/set* properties on the data source. `SalesTable_DS.research();`

DataSourceName_Q (e.g., *SalesTable_Q*)	Variable that provides easy access to each data source's *Query* object. Commonly used to access the data source query to add ranges prior to query execution/run. Equivalent to *SalesTable_DS.query*.
	`rangeSalesLineProjId    = salesLine1_q.dataSourceTable-(tablenum(SalesLine)).addRange(fieldnum(SalesLine, ProjId));`
	
	`rangeSalesLineProjId.value(ProjTable.ProjId);`

Variable	Use and Example
DataSourceName_QR (e.g., *SalesTable_QR*)	Variable that provides easy access to each data source *QueryRun* object that contains a copy of the query that was most recently executed. The query inside the *QueryRun* object is copied during the *FormDataSource ExecuteQuery* method. Commonly used to access the query that was executed so that query ranges can be inspected. Equivalent to *SalesTable_DS.queryRun*. `SalesTableQueryBuildDataSource = SalesTable_QR.query().dataSourceTable(tablenum(SalesTable));`
ControlName (e.g., *SalesTable_ SalesId*)	Variable created for each control set as *AutoDeclaration=Yes*. Commonly used to access controls not bound to a data source field, such as the fields used to implement custom filters. `backorderDate.dateValue(systemdateget());`

Form method overrides allow developers to influence the form life cycle and how the form responds to some user-initiated events. The most important form method overrides are described in Table 5-9. The two most overridden form methods are *Init* and *Run*.

TABLE 5-9 Form Method Override Explanations

Method	Explanation
Init	Called when the form is initialized. Prior to the call to *super*, much of the form (*FormRun*) is not initialized, including the controls and the query. Commonly overridden to access the form at the earliest stage possible.
Run	Called when the form is initialized. Prior to the call to *super*, the form is initialized but isn't visible to the user. Commonly overridden to make changes to form controls, layout, and cursor focus.
Close	Called when the form is being closed. Commonly overridden to release resources and save user settings and selections.
CloseOk	Called when the form is being closed via the *Ok* command/task, such as when the user clicks a *CommandButton* with a *Command* property of *Ok*. Commonly overridden on dialog forms to perform the action the user has initiated.
CloseCancel	Called when the form is being closed via the *Cancel* command/task, such as when the user clicks a *CommandButton* with a *Command* property of *Cancel*. Commonly overridden on dialog forms to clean up after the user indicates that an action should be cancelled.
CanClose	Called when the form is being closed. Commonly overridden to ensure that data is in a good state before the form is closed. Returning *false* aborts the close action and keeps the form open.

Form data source and form data source field method overrides allow developers to influence how the form reads and writes its data and allows developers to respond to user-initiated data-related events. The most important form data source method overrides are described in

Table 5-10. The five most overridden form data source methods are *Init*, *Active*, *ExecuteQuery*, *Write*, and *LinkActive*.

TABLE 5-10 Form Data Source Method Override Explanations

Method	Explanation
Active	Called when the active/current record changes, such as when the user clicks a different record. Commonly overridden to enable and disable buttons based on whether or not they are applicable to the current record.
Create	Called when a record is being created, such as when the user presses Ctrl+N. Commonly overridden to change the user interface in response to a record creation.
Delete	Called when a record is being deleted, such as when the user presses Alt+F9. Commonly overridden to change the user interface in response to a record creation.
ExecuteQuery	Called when the data source's query is executed, such as when the form is run (from the *super* of the form's *Run* method) or when the user refreshes the form by pressing F5. Commonly overridden to implement the behavior of a custom filter added to the form.
Init	Called when the data source is initialized during the *super* of the form's *Init* method. Commonly overridden to add or remove query ranges or change dynalinks.
InitValue	Called when a record is being created. Record values set in this method count as original values rather than changes. Commonly overridden to set the default values of a new record.
LeaveRecord	Called when the user is moving focus from one data source join hierarchy to another, which can happen when the user moves between controls. Commonly overridden to coordinate between data sources, but developers are encouraged to use the *ValidateWrite* and *Write* methods where possible. *ValidateWrite* and *Write* are called immediately after *LeaveRecord*.
LinkActive	Called when the active method in a dynalinked parent form is called. Commonly overridden to change the user interface to correspond to a different parent record (*element.args().record()*).
MarkChanged	Called when the marked set of records changes, such as when the user multi-selects a set of records. Commonly overridden to enable/disable buttons that work on a multi-selected (marked) set of records.
ValidateDelete	Called when the user attempts to delete a record. Commonly overridden to provide form-specific deletion event validation. Return *false* to abort the delete. Use the *ValidateDelete* table method to provide record deletion validation across all forms.
ValidateWrite	Called when the record is being saved, such as when the user presses the Close or Save buttons or clicks a field from another data source. Commonly overridden to provide form-specific write/save event validation. Return *false* to abort the write. Use the *ValidateWrite* table method to provide record write/save validation across all forms.
Write	Called when the record is being saved after validation has succeeded. Commonly overridden to perform additional form-specific write/save event logic such as updating the user interface. Use the *Write* table method to respond to the record write/save event across all forms.

Three commonly used form data source field method overrides are described in Table 5-11. The most overridden form data source field method is the *Modified* method.

TABLE 5-11 Form Data Source Field Method Override Explanations

Method	Explanation
Modified	Called when the value of a field changes. Commonly overridden to make a corresponding change to the user interface or to change other field values.
Lookup	Called when the Lookup button of the field is clicked. Commonly overridden to build a custom lookup form. Use the *EDT.FormHelp* property to provide lookup capabilities to all forms.
Validate	Called when the value of a field changes. Commonly overridden to perform form-specific validation needed prior to saving or to validate. Return *false* to abort the change. Use the *ValidateField* table method to provide field validation across all forms.

Displaying an Image

The following example illustrates how to customize the sales order form to allow a user to upload and display an image of a custom order. In this example, a customer must be able to place an order for a bike through Enterprise Portal and upload a sketch of the bike at the same time. An example of a customer-supplied bike image is shown in Figure 5-8.

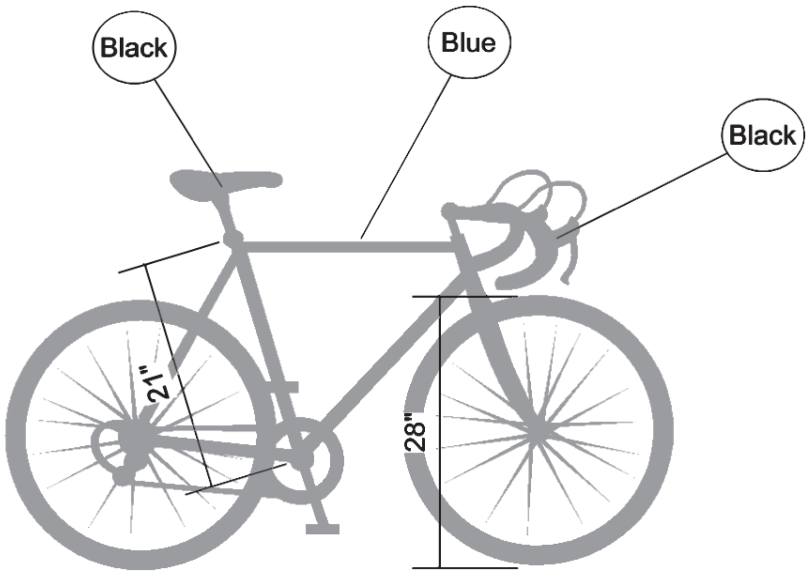


FIGURE 5-8 Uploaded bike image

This image must be stored in the database and attached to the sales order line. Sales order lines are stored in the *SalesLine* table. You could add a new field to the *SalesLine* table of the type container and store the image in this field, but this example uses the document management functionality in Dynamics AX. The image is therefore stored in the *DocuValue* table with a reference to a record in the *DocuRef* table from the image record in *DocuValue* to the *SalesLine* record. The relationship and multiplicity among the three tables is shown in Figure 5-9.

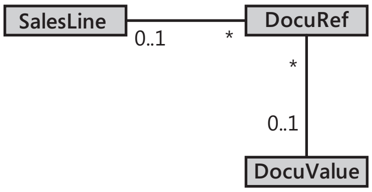
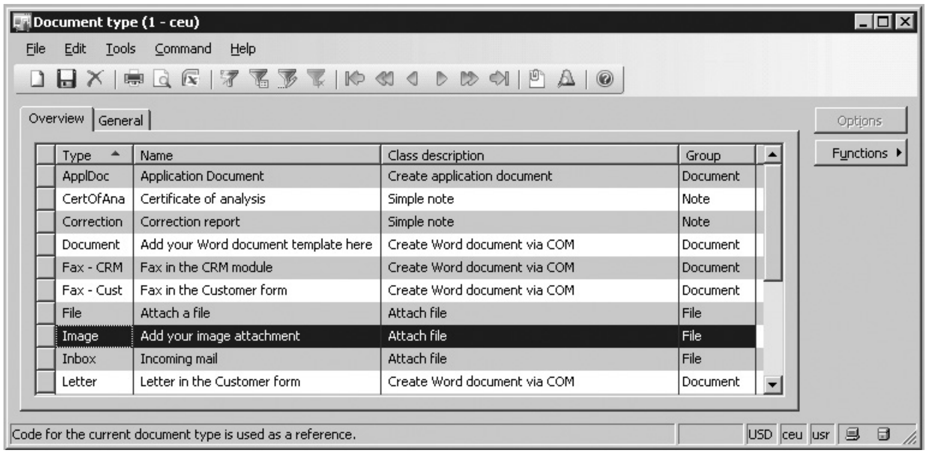


FIGURE 5-9 Relationship among the *SalesLine*, *DocuRef*, and *DocuValue* tables

In this example, a document type named Image stores the attached file in the disk folder. The Image document type is shown in Figure 5-10. The Document Type form is located in the navigation pane, Basic\Setup\Document Management\Document Types.

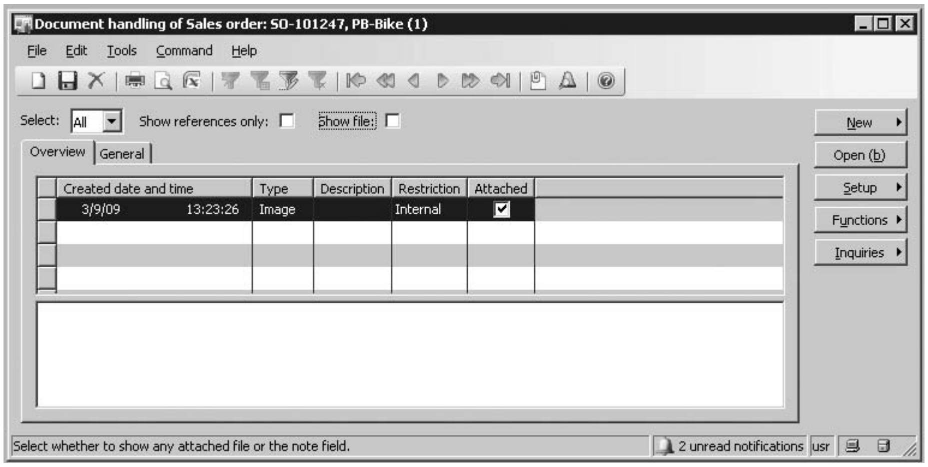


Type	Name	Class description	Group
ApplDoc	Application Document	Create application document	Document
CertOfAna	Certificate of analysis	Simple note	Note
Correction	Correction report	Simple note	Note
Document	Add your Word document template here	Create Word document via COM	Document
Fax - CRM	Fax in the CRM module	Create Word document via COM	Document
Fax - Cust	Fax in the Customer form	Create Word document via COM	Document
File	Attach a file	Attach file	File
Image	Add your image attachment	Attach file	File
Inbox	Incoming mail	Attach file	File
Letter	Letter in the Customer form	Create Word document via COM	Document

Code for the current document type is used as a reference. USD ceu usr

FIGURE 5-10 Image document type

Any uploaded image is therefore stored in the document management system; a user can view the image by either clicking the Document Handling icon on the status bar or choosing Document Handling on the Command menu. The user sees the dialog box shown in Figure 5-11, in which the image can be viewed, modified, or deleted, and additional notes or documents can be attached.


Document handling of Sales order: SO-101247, PB-Bike (1)
File Edit Tools Command Help
New
Open (b)
Setup
Functions
Inquiries
Overview
General

Select: All Show references only: ☐ Show file: ☐

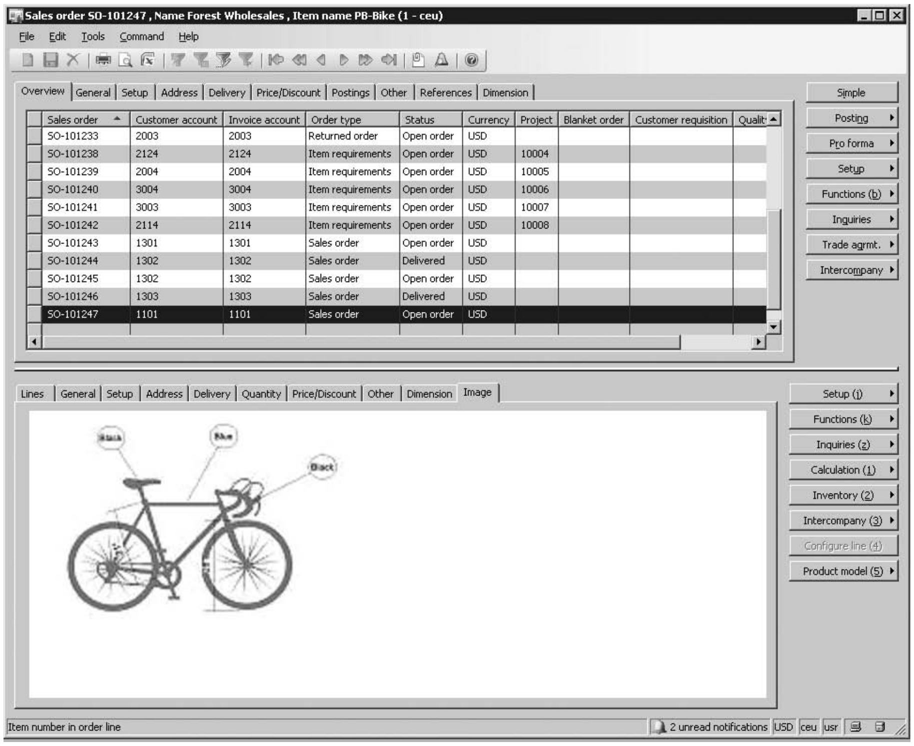
Created date and time	Type	Description	Restriction	Attached
3/9/09 13:23:26	Image		Internal	✓

Select whether to show any attached file or the note field. 2 unread notifications usr

FIGURE 5-11 Storage of the uploaded bike image in the document management system

Displaying an Image on a Form

You can display the image directly by placing it on a separate Image tab on the sales order form. Figure 5-12 shows an order for a bike with a frame size of 21 inches and a wheel size of 28 inches. The user can click the Image tab to view the uploaded bike image and confirm that it matches the ordered item before confirming the sales order. The Sales Order form (AOT\Forms\SalesTable) is located in the navigation pane, Accounts Receivable\Sales Order.


Sales order SO-101247 , Name Forest Wholesales , Item name PB-Bike (1 - ceu)

Sales order	Customer account	Invoice account	Order type	Status	Currency	Project	Blanket order	Customer requisition	Qualit
SO-101233	2003	2003	Returned order	Open order	USD				
SO-101238	2124	2124	Item requirements	Open order	USD	10004			
SO-101239	2004	2004	Item requirements	Open order	USD	10005			
SO-101240	3004	3004	Item requirements	Open order	USD	10006			
SO-101241	3003	3003	Item requirements	Open order	USD	10007			
SO-101242	2114	2114	Item requirements	Open order	USD	10008			
SO-101243	1301	1301	Sales order	Open order	USD				
SO-101244	1302	1302	Sales order	Delivered	USD				
SO-101245	1302	1302	Sales order	Open order	USD				
SO-101246	1303	1303	Sales order	Delivered	USD				
SO-101247	1101	1101	Sales order	Open order	USD				

Item number in order line

2 unread notifications USD ceu usr

FIGURE 5-12 Uploaded bike image displayed on the Sales Order form Image tab

The following two example implementations describe how to use the document management tables as data sources in the form and how to create a separate method on the *SalesLine* table. These examples demonstrate customization of the *SalesTable* sales order form and the *SalesLine* table.

Displaying an Image by Using Joined Data Sources

One way to display the image is to apply the *DocuRef* and *DocuValue* tables as data sources for the *SalesTable* form. The following example creates a *DocuRef* data source based on the relationship among the *SalesLine*, *DocuRef*, and *DocuValue* tables shown in Figure 5-9. The *DocuRef* data source relates to the *DocuRef* table and is joined to the *SalesLine* data source. Additionally, a *DocuValue* data source is created to connect to the *DocuRef* data source. Table 5-12 shows additional properties of the data sources.

TABLE 5-12 ***DocuRef* and *DocuValue* Property Settings**

Property	*DocuRef*	*DocuValue*
Table	*DocuRef*	*DocuValue*
AllowEdit	No	No
AllowCreate	No	No
AllowDelete	No	No
JoinSource	*SalesLine*	*DocuRef*
LinkType	Active	Active

The properties *JoinSource* and *LinkType* allow the *DocuRef* and *DocuValue* records to be fetched when the user moves from one line to another. The remaining properties disable editing of the records.

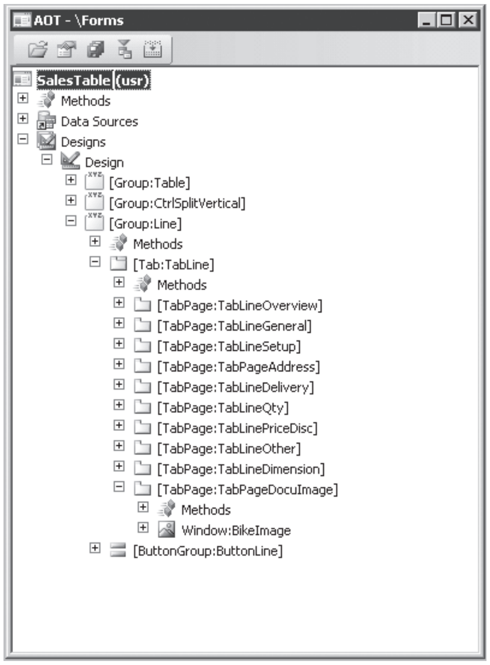
You can attach multiple files, documents, and notes to a *SalesLine* record by using the document management feature, but the goal of this example is to display an image from a linked document named Image. You can limit the retrieved records from the *DocuRef* table by adding a range to the query used by the *DocuRef* data source. You do this by customizing the *Init* method on the *DocuRef* data source, as shown here.

```
public void init()
{
    super();

    docuRef_ds.query().dataSourceTable(
                          tableNum(DocuRef)).addRange(
                          fieldNum(DocuRef,TypeId)).value(queryValue('Image'));
}
```

This X++ code limits the query so that it retrieves only records from the *DocuRef* table in which the *TypeId* field is equal to the value *'Image'*.

Note The use of a constant such as the word *Image* is not a best practice. The value must be retrieved from a configuration table so that the user can decide the naming. *'Image'* is hard coded in the preceding example only to improve the readability and limit the scope of the example.

The image is displayed by using a window control, which is placed in a tab control, as shown in Figure 5-13.



```
AOT - \Forms
SalesTable (usr)
+ Methods
+ Data Sources
- Designs
  - Design
    + [Group:Table]
    + [Group:CtrlSplitVertical]
    - [Group:Line]
      + Methods
      - [Tab:TabLine]
        + Methods
        + [TabPage:TabLineOverview]
        + [TabPage:TabLineGeneral]
        + [TabPage:TabLineSetup]
        + [TabPage:TabPageAddress]
        + [TabPage:TabLineDelivery]
        + [TabPage:TabLineQty]
        + [TabPage:TabLinePriceDisc]
        + [TabPage:TabLineOther]
        + [TabPage:TabLineDimension]
        - [TabPage:TabPageDocuImage]
          + Methods
          + Window:BikeImage
      + [ButtonGroup:ButtonLine]
```

FIGURE 5-13 Tab and window controls in the SalesTable form

Although the image is stored in the *File* field on the *DocuValue* table, to display the image you can't simply link the field as a *DataField* value on the window control property sheet. The image must be parsed to the control by using a method on the control in X++ that uses the *FormWindowControl* object. The *AutoDeclaration* property on the *FormWindowControl* object is therefore set to Yes so that the forms designer automatically declares an object handle with the same name. This handle can be used in X++ and manipulated at run time because the form application runtime automatically ensures that it is a handle to the *FormWindowControl* object. The *Width* and *Height* properties are set to *Column width* and *Column height* so that the image takes up all the space on the tab.

The last step is to parse the retrieved image from the *DocuValue* table to the *BikeImage FormWindowControl* object. You can do this when a *DocuValue* record buffer is present. This record must contain an image that is stored in the database, and the X++ code should be placed in the active method on the *DocuValue* data source and look like the following.

```
public int active()
{
    Image   image;
    int    ret;
    ret = super();
    if (docuValue.File)
    {
            image = new Image();
            image.setData(docuValue.File);
            bikeImage.image(image);
        }
        else
        {
            bikeImage.imageResource(0);
        }
        return ret;
    }
```

This code determines whether a value exists in the *File* field and, if so, instantiates an image object and parses the *File* field value to the image object. This object is then parsed by using the *Image* method to the *FormWindowControl* object that displays the image. If the *File* field doesn't contain a value, the *imageResource* method on the *FormWindowControl* object is called with a value of 0 to clear the control of any previous content. The active method is executed only if a *DocuValue* record has been retrieved. However, if a user moves from an order line with an image to an order line without an image, the image isn't cleared because the active method isn't executed. If you add the following line to the active method on the *SalesLine* data source, the image is cleared when a new order line becomes active and before the *DocuRef* and *DocuValue* records are retrieved.

```
    docuBikeImage.imageResource(0);
```

The customizations described in this section make it possible to display the image on the Image tab. This solution has one downside, however. Whenever a user moves from one order line to another or a line is created or saved, calls are made from the client to the server, and lookups are made in the database for the *DocuRef* and *DocuValue* data sources. You can see this by turning on the client/server or SQL trace option in the Options dialog box, which you access from the Tools menu. The next section addresses this issue and offers a solution—decreasing the number of client/server calls and lookups in the database.

Displaying an Image When Activating the Image Tab

The following example implements a solution similar to the previous example, but it results in calls to the server and the database only when the image is actually displayed.

The *TabPage* control must be added to the *SalesTable* form and contain a *FormWindowControl* with property settings similar to those in the preceding example. The *DocuRef* and *DocuValue* tables are not, however, added as data sources for the form. Instead, this example retrieves the image—the only element shown on the Image tab—from the database only when the user chooses to display the content of the Image tab. You configure this by adding the following X++ code to the *pageActivated* method on the *TabPage* control.

```
public void pageActivated()
{
    Image           image;
    DocuValueFile   docuValueFile;
    ;
    docuValueFile = salesLine.bikeImage();
    if (docuValueFile)
    {
        image = new Image();
        image.setData(docuValueFile);
        bikeImage.image(image);
    }
    else
    {
        bikeImage.imageResource(0);
    }

    super();
}
```

This code is very similar to the code added to the *DocuValue* active method, but in this case the value is retrieved from a *bikeImage* method on the *SalesLine* table. The *bikeImage* method is a new method created on the *SalesLine* table with the following content.

```
server public DocuValueFile bikeImage()
{
    DocuRef     docuref;
    DocuValue   docuValue;
    ;
    select  firstonly tableid from docuRef
        where docuRef.RefCompanyId  == this.DataAreaId  &&
              docuRef.RefTableId    == this.TableId     &&
              docuRef.RefRecId      == this.RecId       &&
              docuRef.TypeId        == 'Image'
    join file from docuValue
        where docuValue.RecId   == docuRef.ValueRecId;

    return docuValue.File;
}
```

The select statement in the *bikeImage* method is a combination of the two lookups in the database produced by the runtime shown in the first sample implementation, which used data sources. However, the statements in this method are joined. The *bikeImage* method could simply be implemented in the *SalesTable* form, but implementing it on the *SalesLine* table allows it to be reused in other forms or reports and executed on the server tier, if required.

The advantage of this implementation method is that both database lookups and calls from the client to the server are reduced by half. And because calls are made only when the Image tab is activated, they aren't made when a user simply moves through the order lines without viewing the content of the Image tab. The disadvantage, however, is that the user can't personalize the form or move the display of the image to another tab because retrieval of the image is dependent on activation of the Image tab.

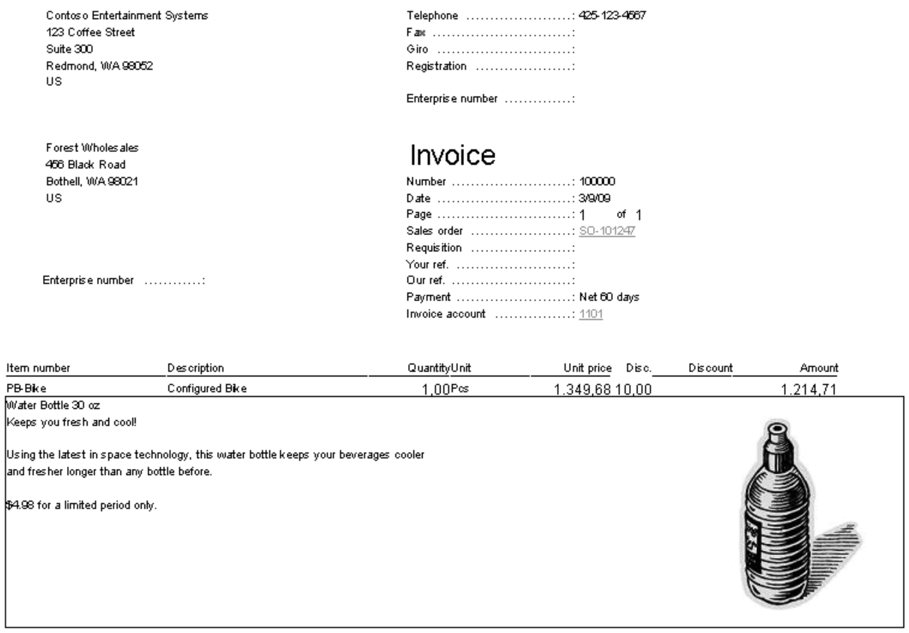
Report Customization

Reports, like forms, can be customized to include and exclude information, and you can modify their design and layout. As with forms, the design and layout of a report depend on settings on the table and on the report itself. The best practice is, once again, to keep as much of the business logic as possible with the table methods or metadata. The X++ code in reports must deal with the functionality for the specific report. All other X++ code must generally be implemented on the table to be reused by other areas in the application. Here are some of the advantages to such an approach:

- Customizations made to a report are isolated; customizations made to a table affect all reports using that table, resulting in a consistent user experience wherever the table is used.
- Customization of a report copies the entire report to the current layer; customizations made to tables are more granular because customization of fields, field groups, and methods results in a copy of the specific element to the current layer only. This makes upgrading to service packs and new versions easier.
- Methods in reports always execute on the tier where the report is generated; methods on tables can be targeted to execute on the server tier. Where a report is generated is controlled by the *RunOn* property on the menu item that starts the report. The property can be set to Client, Server, or Called From.

Creating Promotional Materials

The example in this section demonstrates how to customize the sales order invoice report named SalesInvoice (AOT\Reports\SalesInvoice). The invoice is customized to include promotions based on items listed on the invoice. The promotion appears below each item on the invoice associated with a promotion. Figure 5-14 shows an example of an invoice that displays a promotion for a water bottle.



Contoso Entertainment Systems
123 Coffee Street
Suite 300
Redmond, WA 98052
US

Telephone	425-123-4567
Fax	
Giro	
Registration	
Enterprise number	

Forest Wholesales
456 Black Road
Bothell, WA 98021
US

Enterprise number ....:

Invoice

Number	100000
Date	3/9/09
Page	1 of 1
Sales order	SO-101247
Requisition	
Your ref.	
Our ref.	
Payment	Net 60 days
Invoice account	1101

Item number	Description	Quantity	Unit	Unit price	Disc.	Discount	Amount
PB-Bike	Configured Bike	1,00	Pcs	1.349,68	10,00		1.214,71

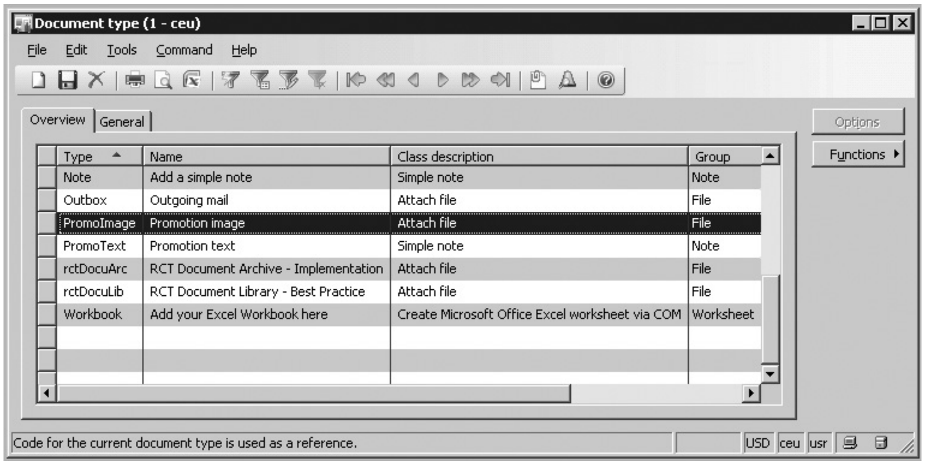
Water Bottle 30 oz
Keeps you fresh and cool!

Using the latest in space technology, this water bottle keeps your beverages cooler
and fresher longer than any bottle before.

$4.98 for a limited period only.

FIGURE 5-14 Promotion on an invoice

Like the forms example, this example uses the document management feature in Dynamics AX. You use document handling to store the text and image in the database. The information is attached to the item table as two different types of document information, named *PromoText* and *PromoImage*, for storing the text and image. Figure 5-15 shows the *PromoText* and *PromoImage* document types.

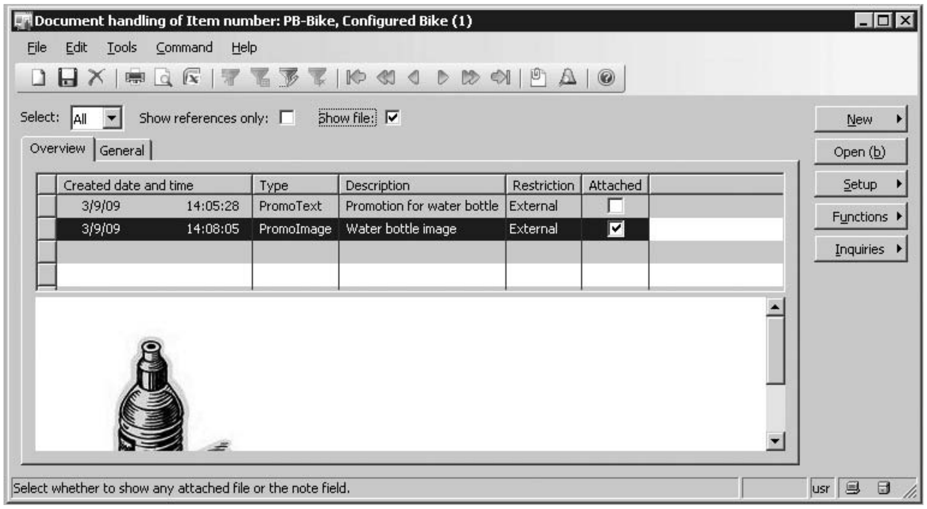


Type	Name	Class description	Group
Note	Add a simple note	Simple note	Note
Outbox	Outgoing mail	Attach file	File
PromoImage	Promotion image	Attach file	File
PromoText	Promotion text	Simple note	Note
rctDocuArc	RCT Document Archive - Implementation	Attach file	File
rctDocuLib	RCT Document Library - Best Practice	Attach file	File
Workbook	Add your Excel Workbook here	Create Microsoft Office Excel worksheet via COM	Worksheet

Code for the current document type is used as a reference. USD ceu usr

FIGURE 5-15 *PromoText* and *PromoImage* document types

Figure 5-16 shows the text and image attached to an item named PB-Bike.



Select: All   Show references only: ☐   Show file: ☑

Created date and time		Type	Description	Restriction	Attached
3/9/09	14:05:28	PromoText	Promotion for water bottle	External	☐
3/9/09	14:08:05	PromoImage	Water bottle image	External	☑

Select whether to show any attached file or the note field.

FIGURE 5-16 Text and image attached to an item

The X++ code used to display the promotion on the invoice looks up the item in the *InventTable* table and searches the document handling for documents of type *PromoText* and *PromoImage* to print on the invoice. If neither type is attached to the item, no promotion information prints.

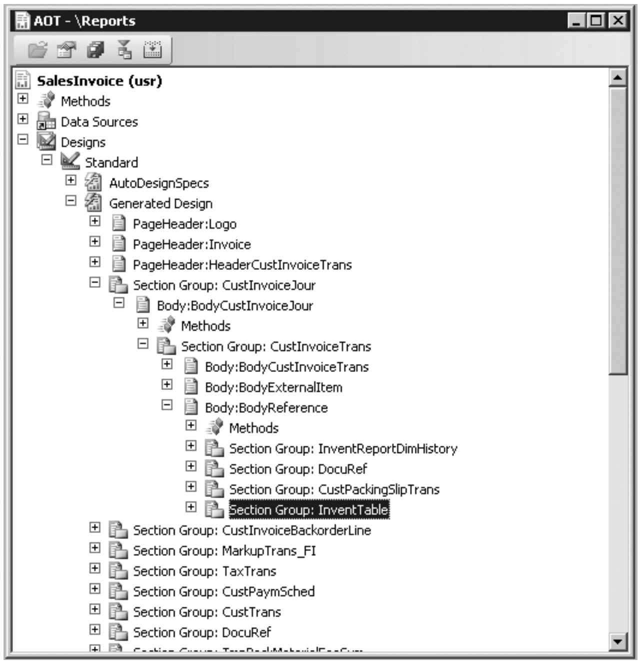
Adding Promotional Materials to an Invoice Report

Before you customize the *SalesInvoice* report for this example, you must decide where in the design of the report to place the printing of the promotion. The printed information should be printed for each invoiced item, so you must place it under the *CustInvoiceTrans* section group because the *CustInvoiceTrans* table contains the invoiced items. The *CustInvoiceTrans* section group contains a reference body section that can print other pieces of reference information, such as from inventory dimensions or the packing slip lines posted when the invoiced item is shipped. The promotion resembles this kind of information in terms of when and how it is printed.

This example, therefore, creates a new section group within the reference body section below the existing three groups. The new section group must reference a table type so that it can be invoked when a record buffer of the same type is sent to the report by using the *element.send* method. The *DocuRef* table stores the promotion text, and the *DocuValue* table stores the promotion image with an association created in the *DocuRef* table.

Although the storage of the text and image results in the creation of *DocuRef* records, the choice of *DocuRef* as the reference table type for the new section group isn't an optimal solution. First, the information is stored as two records in the *DocuRef* table, but the text and image should be printed side by side for this example. The *element.send* method should be called only once, parsing in only a single record buffer. Also, two other section groups already use *DocuRef* as the table type, so using this type might result in the other section groups getting invoked as well when the promotion prints. You could prevent this by introducing a variable to control which section group to invoke, but then you would have to customize even more of the report, making it harder to upgrade the report when a new version or service pack is installed.

Both of the *DocuRef* records are, however, related to the same *InventTable* record, so you can use this table as the type for the section group, and an *InventTable* record buffer is sent to the report to print the promotion text and image. Figure 5-17 shows the new section group, named *InventTable*, and its positioning within the report.



```
SalesInvoice (usr)
⊞ Methods
⊞ Data Sources
⊟ Designs
    ⊟ Standard
        ⊞ AutoDesignSpecs
        ⊟ Generated Design
            ⊞ PageHeader:Logo
            ⊞ PageHeader:Invoice
            ⊞ PageHeader:HeaderCustInvoiceTrans
            ⊟ Section Group: CustInvoiceJour
                ⊟ Body:BodyCustInvoiceJour
                    ⊞ Methods
                    ⊟ Section Group: CustInvoiceTrans
                        ⊞ Body:BodyCustInvoiceTrans
                        ⊞ Body:BodyExternalItem
                        ⊟ Body:BodyReference
                            ⊞ Methods
                            ⊞ Section Group: InventReportDimHistory
                            ⊞ Section Group: DocuRef
                            ⊞ Section Group: CustPackingSlipTrans
                            ⊞ Section Group: InventTable
            ⊞ Section Group: CustInvoiceBackorderLine
            ⊞ Section Group: MarkupTrans_FI
            ⊞ Section Group: TaxTrans
            ⊞ Section Group: CustPaymSched
            ⊞ Section Group: CustTrans
            ⊞ Section Group: DocuRef
```

FIGURE 5-17 *InventTable* section group in the *SalesInvoice* report

Implementing Promotional Methods

When the promotion text and image print, an *InventTable* record buffer is sent to the report. For this reason, this example implements two methods to return the text and image by using an *InventTable* record buffer. The methods can be implemented directly in the report, but because the methods are not report specific—and therefore can be reused in other reports, or even forms—they are implemented as instance methods on *InventTable*. The following code shows the new methods. The *PromotionImage* method is implemented like the *BikeImage* method in the forms example discussed earlier. However, the *PromotionImage* method must look in only the *DocuRef* table to find the text.

```
display server public DocuValueFile PromotionImage()
{
    DocuRef     docuref;
    DocuValue   docuValue;
    ;
    select  firstonly tableid from docuRef
        where docuRef.RefCompanyId  == this.DataAreaId  &&
              docuRef.RefTableId    == this.TableId     &&
              docuRef.RefRecId      == this.RecId       &&
              docuRef.TypeId        == 'PromoImage'
    join file from docuValue
        where docuValue.RecId   == docuRef.ValueRecId;

    return docuValue.File;
}



display server public Notes PromotionText()
{
    DocuRef     docuref;
    ;
    select firstonly notes from docuRef
        where docuRef.RefCompanyId  == this.DataAreaId  &&
              docuRef.RefTableId    == this.TableId     &&
              docuRef.RefRecId      == this.RecId       &&
              docuRef.TypeId        == 'PromoText';

    return docuRef.Notes;
}
```

Both methods are implemented as display methods to allow them to bind directly to report controls and to print the information.

Binding Display Methods to Report Controls

The next step is to bind the methods to report controls. A new body section named *BodyInventTable* is created in the *InventTable* section group, and several of its properties are altered, as shown in Table 5-13.

TABLE 5-13 ***BodyInventTable* Property Settings**

Property	Settings
NoOfHeadingLines	0
LineAbove	Solid
LineBelow	Solid
LineLeft	Solid
LineRight	Solid

The *NoOfHeadingLines* property must be set to 0 because the text and image must not include any headings when printed. The *Line* property settings create a border around the promotion.

In the body section, a string control, named *PromotionText*, and a bitmap control, named *PromotionImage*, are added and bound to the two new *InventTable* methods. The properties shown in Table 5-14 are changed on the two controls.

TABLE 5-14 ***PromotionText* and *PromotionImage* Property Settings**

Property	*PromotionText*	*PromotionImage*
Left		Auto (right)
Width	70.00 char	2.0 inch
Height		2.0 inch
DynamicHeight	Yes	
ShowLabel	No	No
Table	*InventTable*	*InventTable*
DataMethod	*PromotionText*	*PromotionImage*

The *ShowLabel* properties are set to No because no headings should be printed. The *PromotionText* control is set to a fixed width of 70 characters with a dynamic height so that the text won't be truncated. The *PromotionImage* has a fixed size of 2 inches by 2 inches and is right-justified on the page.

The last step is to look up an *InventTable* record buffer based on the invoiced item and then send the buffer to the report. You do this with the following new method on the *BodyReference* body section.

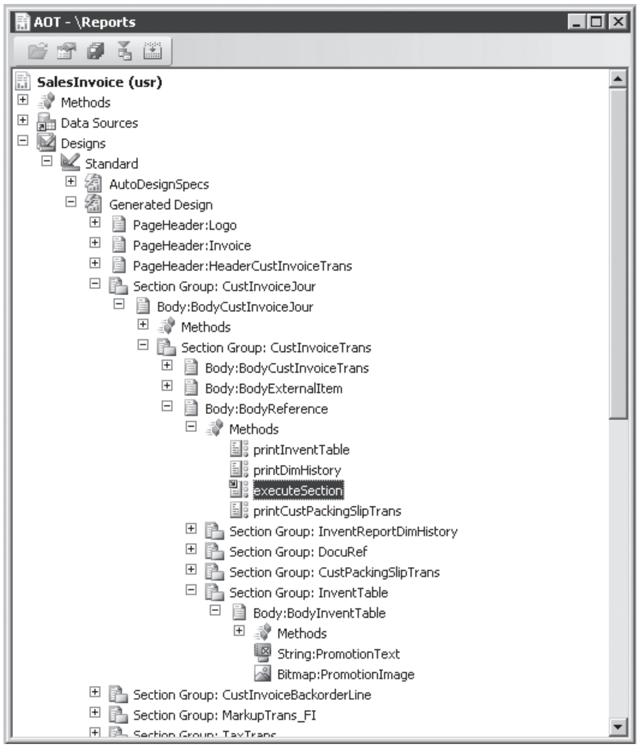
```
void printInventTable()
{
    InventTable inventTable = custInvoiceTrans.inventTable();
    if (inventTable.RecId)
    {
         element.send(inventTable);
    }
}
```

The method uses the *InventTable* lookup method on the *CustInvoiceTrans* table, which returns a record buffer for the invoiced item, which the method subsequently sends to the report.

The preceding method should be called from the *executionSection* method on the same body section. The following method is therefore customized by including the call to the *printInventTable* method.

```
void  executeSection()
{;
    this.printCustPackingSlipTrans();
    this.printDimHistory();
    this.printInventTable();
}
```

The positioning of the body section, report control, and report methods is shown in Figure 5-18.



```
SalesInvoice (usr)
+ Methods
+ Data Sources
- Designs
    - Standard
        + AutoDesignSpecs
        - Generated Design
            + PageHeader:Logo
            + PageHeader:Invoice
            + PageHeader:HeaderCustInvoiceTrans
            - Section Group: CustInvoiceJour
                - Body:BodyCustInvoiceJour
                    + Methods
                    - Section Group: CustInvoiceTrans
                        + Body:BodyCustInvoiceTrans
                        + Body:BodyExternalItem
                        - Body:BodyReference
                            - Methods
                                printInventTable
                                printDimHistory
                                executeSection
                                printCustPackingSlipTrans
                            + Section Group: InventReportDimHistory
                            + Section Group: DocuRef
                            + Section Group: CustPackingSlipTrans
                            - Section Group: InventTable
                                - Body:BodyInventTable
                                    + Methods
                                    String:PromotionText
                                    Bitmap:PromotionImage
            + Section Group: CustInvoiceBackorderLine
            + Section Group: MarkupTrans_FI
            + Section Group: TaxTrans
```

FIGURE 5-18 Position of the new sections, control, and methods in the *SalesInvoice* report

After the completion of all the customizations to the *SalesInvoice* report and the addition of new methods to *InventTable*, the report prints the promotion below each invoiced item on the report, as shown in Figure 5-14.

Preventing Printing of an Empty Body Section

The solution thus far has one flaw: it prints an empty *BodyInventTable* body section if there is no document reference for the *PromoText* and *PromoImage* document types, which causes an empty box to appear below each item on the invoice. You could easily fix this by altering the *printInventTable* method to include a check for text or images, as shown in the following change to the *printInventTable* method.

```
void printInventTable()
{
    InventTable inventTable = custInvoiceTrans.inventTable();
    if (inventTable.RecId &&
        (inventTable.PromotionText() || inventTable.PromotionImage()))
    {
         element.send(inventTable);
    }
}
```

This code ensures that the *InventTable* record buffer is sent to the report only if the *PromotionText* method or the *PromotionImage* method returns a value.

In terms of performance, this change isn't optimal because methods could be executed twice if a promotion were added to the *InventTable* record. This could result in as many as five round-trips to the database for each printed invoiced item: two from the *printInventTable* method, two when printing the values, and one when the report runtime determines the height of the *PromotionText* control.

A better solution is to cache the returned values from the *PromotionText* and *PromotionImage* methods when they are called in the *printInventTable* method and then use the cached values instead of retrieving them from the database when printing the *PromotionText* and *PromotionImage* controls.

The cache variables must be added to the *classDeclaration* of the report, so the following lines are inserted there.

```
   DocuValueFile          promotionImage;
    Notes                  promotionText;
```

The *printInventTable* method is modified to store the returned values from the *PromotionText* and *PromotionImage* methods on the *InventTable* record buffer in the newly created variables, as shown in the following copy of the method.

```
void printInventTable()
{
    InventTable inventTable = custInvoiceTrans.inventTable();
    ;
    promotionImage  = inventTable.PromotionImage();
    promotionText   = inventTable.PromotionText();

    if (inventTable.RecId &&
        (promotionText || promotionImage))
    {
         element.send(inventTable);
    }
}
```

In addition to these two new display methods, *PromotionText* and *PromotionImage* are created to return the values of the variables. The following code samples show these methods, implemented in the *BodyInventTable* body section.

```
display Notes PromotionText()
{
    return promotionText;
}
```

```
display DocuValueFile PromotionImage()
{
    return promotionImage;
}
```

With these two methods named similarly to the *InventTable* methods, you must remove only the value in the *Table* property on the *PromotionImage* and *PromotionText* report controls to enable the report to retrieve the value from the local report methods instead of the *InventTable* methods. You can even remove the display method modifiers from the two *InventTable* methods because they are no longer used as display methods.

When you print the report again, no empty *BodyInventTable* body sections appear, and the printing of this specific section is optimized. The report will never result in more than two round-trips to the database for each invoiced item. The only disadvantages are that return types of the methods on the *InventTable* and the equivalent methods on the report should be kept synchronized, and these return types should again be kept synchronized with the types of the cache variables. This synchronization wasn't necessary earlier in the example, before the values in the report were cached.

Number Sequence Customization

In Chapter 6, "Extending Dynamics AX," the sample X++ code shows that a service order feature must have a *number sequence* to generate a unique identification number. To achieve this, you must customize the number sequence class, setting up the relationship between a module and a number sequence reference, and also associating the number sequence reference with the extended data type in which you want to store a number from the sequence.

When you want to create a new number sequence, you must first create an extended data type. The ID of the type is used as the identifier for the number sequence reference, so it must be unique. Figure 5-19 shows a string data type named *BikeServiceOrderId*.

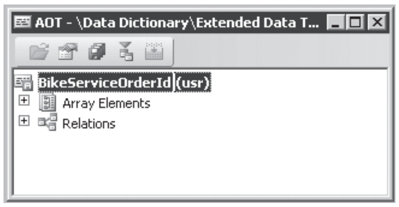


FIGURE 5-19 *BikeServiceOrderId* extended data type

The properties on the extended data type are set to create a type with a maximum length of 20 characters, as shown in Table 5-15.

TABLE 5-15 ***BikeServiceOrderId* Property Settings**

Property	Settings
Type	String
Label	Service order
HelpText	Service order ID
StringSize	20

To implement a number sequence reference for service orders and assign it a specific service order number sequence, you must make changes to a *NumberSeqReference* class. To implement the reference in the Accounts Receivable module, among other references used by the sales order functionality, you add the following lines of X++ code to the *loadModule* method on the *NumberSeqReference_SalesOrder* class.

```
    numRef.DataTypeId            = typeId2ExtendedTypeId(
                                   typeid(BikeServiceOrderId));
    numRef.ReferenceHelp         = "Unique key for the service order table, "+
                                   "used when identification of a service "+
                                   "order is allocated automatically.";
    numRef.WizardContinuous      = false;
    numRef.WizardManual          = NoYes::No;
    numRef.WizardAllowChangeDown = NoYes::No;
    numRef.WizardAllowChangeUp   = NoYes::No;
    numRef.SortField             = 100;
    this.create(numRef);
```

These are the only modifications necessary to set up a new number sequence reference. The reference is available in the Accounts Receivable parameter form, and a number sequence can be created automatically by using the Number Sequence Wizard. You start the Number Sequence Wizard by clicking the Wizard button in the Number Sequences form located in the navigation pane under Basic\Setup\Number Sequences\Number Sequences.

The *numRef* table buffer in the preceding example is of a *NumberSequenceReference* table type. This table contains several fields that can be set depending on the reference you want to create. These fields are described in Table 5-16.

TABLE 5-16 ***NumberSequenceReference*** **Field Explanations**

Field	Explanation
DataTypeId	The ID for the reference. Use the ID of the extended data type.
ConfigurationKeyId	The configuration key that must be enabled for the reference to display. The configuration key should be set only if it is different from the key associated with the extended data type.
ReferenceLabel	The number sequence reference label should be set only if it is different from the label on the extended data type.
ReferenceHelp	The number sequence reference user interface Help field should be set only if the Help text is different from text in the *HelpText* property on the extended data type.
DataTypeSameAsId	Indicates that the reference can use the number from another number sequence. To make this possible, set the ID for the reference to the listed number sequence. This setting is usually applied to voucher references that use the ID of the journal as the voucher number.
GroupEnabled	Indicates that the reference is enabled for use with number sequence groups. This setting should be specified only if the reference can be set up for each number sequence group.
SortField	The position of the reference in the list. Use a sufficiently high number to avoid conflict with other or future references within the same module.
WizardLowest	The default value for the Smallest field when creating the number sequence with the Number Sequence Wizard.
WizardHighest	The default value for the Largest field when creating the number sequence with the Number Sequence Wizard.
WizardManual	The default value for the Manual field when creating the number sequence with the Number Sequence Wizard.

Field	Explanation
WizardContinuous	The default value for the Continuous field when creating the number sequence with the Number Sequence Wizard.
WizardAllowChangeDown	The default value for the To A Lower Number field when creating the number sequence with the Number Sequence Wizard.
WizardAllowChangeUp	The default value for the To A Higher Number field when creating the number sequence with the Number Sequence Wizard.
WizardFetchAheadQty	The default value for the Quantity Of Numbers pre allocation field when creating the number sequence with the Number Sequence Wizard. This field also enables the pre allocation number sequence feature, but it can't be used in combination with a sequence marked Continuous.

Finally, the following method is implemented on the *SalesParameters* table. The method returns the new number sequence reference and should be used in the X++ code that requires numbers from the number sequence.

```
static client server NumberSequenceReference  numRefBikeServiceOrderId()
{
    return NumberSeqReference::findReference(
                           typeId2ExtendedTypeId(typeid(BikeServiceOrderId)));
}
```

Index

About the Authors

Principal Authors

Lars Dragheim Olsen is a software architect on the Dynamics AX team at the Microsoft Development Center in Copenhagen, Denmark. He joined Damgaard Data in 1998 as a software design engineer, shortly after the first version of Dynamics AX was released. His work has focused mainly on the Supply Chain Management modules within Dynamics AX and the integration of these modules with other modules, such as Financials and Project. During the development of Dynamics AX 2009, he worked as a software architect, concentrating primarily on the multisite features within the Supply Chain Management modules. Before working for Damgaard Data, Navision, and Microsoft, he worked for seven years as a system consultant on another ERP product. He lives in Denmark with his four children, Daniel, Christian, Isabella, and Maja, and his girlfriend, Camilla.

Michael Fruergaard Pontoppidan joined Damgaard Data in 1996 as a software design engineer on the MorphX team, delivering the developer experience for the first release of Dynamics AX after graduating from DTU (Technical University of Denmark). In 1999, he became the program manager and lead developer for the Application Integration and Deployment team that delivered on the Load ’n Go vision. For Dynamics AX 4.0, he worked as a software architect on version control, unit testing, and Microsoft’s Trustworthy Computing initiative, while advocating code quality improvements through Engineering Excellence, tools, processes, and training. For Dynamics AX 2009, Michael joined the Developer and Partner Tools team and continued driving high-quality productivity features into the toolsets delivered with Dynamics AX. Michael lives in Denmark with his wife, Katrine, and their two children, Laura and Malte. His blog is at *http://blogs.msdn.com/mfp*.

Hans Jørgen Skovgaard joined Microsoft in 2003 as product unit manager for the Dynamics AX product line. As part of Microsoft's Navision acquisition process, Hans facilitated and managed the introduction of Engineering Excellence initiatives, aligned developer competence, created new teams, and organized training for new developers. Hans joined Microsoft with more than 20 years of professional software development and management experience. Before his engagement with Dynamics AX, Hans was vice president of engineering at Mondosoft, a search engine company, for three years. Before that, he was vice president of CRM development in the ERP company Baan for 10 years, during which time he architected a product configuration technology and associated tools. Hans has an MSc in AI (artificial intelligence) and an MBA from IMD, one of the world's leading business schools. Hans lives in Denmark with his wife, Nomi, and his three lovely daughters, Ristil, Simone, and Mikala. He holds a black belt in karate and is an avid mountain biker.

Tomasz Kaminski joined Microsoft in 2007 as a software design engineer in test on the Developers and Partner Tools team at the Microsoft Development Center in Copenhagen. Before Microsoft, he worked in Poland as software design engineer on data acquisition systems at WINUEL SA and embedded systems at Siemens Sp. z o.o. For Dynamics AX 2009, Tomasz worked on Code Upgrade, the MorphX IDE, and the Rapid Configuration tool. Professionally, he is passionate about software design, architecture, and test-driven development. He holds an MSc in computer science from the Wroclaw University of Technology in Poland. In his free time, he enjoys bird watching and biking. Tomasz lives in Denmark with his wife, Anna.

Deepak Kumar is a program manager at Microsoft working on Dynamics AX Server and Data Upgrade features. He has more than 13 years of industry experience, spending the last eight of those years at Microsoft. Deepak's experience is primarily within the large enterprise, working on ERP, database administration (SQL Server and Oracle), development, performance tuning, and management. He has a master's degree in information management, a bachelor's degree in computer science, and Microsoft (MCP) and Oracle DBA (OCP) certifications. Deepak has been a technical columnist and technology reviewer for national newspapers in India and ran his own small business focusing on consultancy, ERP implementation, and corporate training. Deepak lives in Seattle with his wife, Nupur, and his two young children, Ayan and Arisha. In his spare time, he likes reading research articles, playing tennis, taking short hikes, and doing outdoor activities with his family.

Mey Meenakshisundaram is a principal program manager lead in the Dynamics AX product group who focuses on Enterprise Portal, Role Centers, and search. He has 16 years of experience in software engineering, consulting, and management, the last eight years of which have been spent at Microsoft. Prior to his current role, he led the engineering team that developed and implemented the portal, content management, and Web services for the customer relationship system used by the internal Microsoft Sales team. Before Microsoft, he led software product development teams in Singapore and India for the manufacturing, service, and banking sectors in the Asia Pacific.

Mey is a coauthor of the book *Inside Microsoft Dynamics AX 4.0* and is a highly rated speaker at Microsoft conferences. His self-made mission is to get Enterprise Portal onto every desktop of every Dynamics AX customer. He lives in Sammamish, Washington, with his wife, Amutha, and his children, Meena and Shammu. Mey regularly blogs at *http://blogs.msdn.com/solutions*.

Michael Merz is a program manager focusing on tools for integrating Dynamics AX with other systems. Although his passions include applying innovative patterns and technologies to integrating heterogeneous systems in general, he is currently focused on Software-plus-Services (S+S). Before joining Microsoft, Michael worked in various engineering roles for companies including Amazon.com and BEA Systems as well as for early-stage startup companies; he has also worked as a researcher for the European Union and holds an MSc in computer science from Ulm University, Germany. Michael blogs at *http://blogs.msdn.com/aif.*

Karl Tolgu is a senior program manager for Microsoft Dynamics AX. He is responsible for the delivery of the workflow infrastructure in Dynamics AX. Previously, Karl worked on Project Accounting modules in Dynamics SL and Dynamics GP. He has worked in the software industry in both the United Kingdom and the United States since graduating. He has held various software development management positions at Oracle Corporation and Niku Corporation. Karl resides in Seattle, Washington, with his wife, Karin, and three sons, Karl Christian, Sten Alexander, and Thomas Sebastian.

Kirill Val joined Microsoft in 2005 and has worked as a software design engineer for various Dynamics AX teams, such as Finance, Application Implementation, Server, and Upgrade. He has eight years of experience developing ERP, Web, Financials, and Supply Chain Management applications; before joining Microsoft, he worked for several Microsoft Dynamics partners as a software development lead and program manager.

Kirill's work on Dynamics AX 2009 included contributions to the architecture, design, and development of the data upgrade framework; the batch server processing framework and the electronic signature feature; and integration solutions for enterprise-level businesses. He has an MSc degree in applied mathematics and computer science, and he enjoys playing volleyball, hiking, and traveling in his free time.

Contributing Authors

Srikanth Avadhanam is a development manager for the Dynamics AX product line. He is responsible for overseeing the development of the Dynamics AX application server platform. Before his engagement with Dynamics AX, Srikanth spent a few years designing and implementing various subsystems of the Microsoft SQL Server relational engine; he holds several patents in areas related to SQL query optimization.

Srikanth's technical interests include engineering scalable and performant metadata-driven application servers and database technologies. Srikanth lives in Redmond, Washington, with his wife and two children. Srikanth's team blogs at *http://blogs.msdn.com/daxis*.

Chris Garty is a senior program manager on the Dynamics AX Client team in Fargo, North Dakota. During the Dynamics AX 2009 cycle, Chris was an integral part of the list-page creation effort and continues to work on user experience improvements to the Dynamics AX client. Before his current role, he was a program manager on the Microsoft Business Framework and Dynamics Tools projects focused on business logic and Web services. He has 10 years of experience in software development and consulting, with the last five years spent at Microsoft.

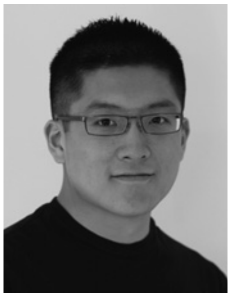
Chris was born and raised in New Zealand, near Hobbiton, and he is lucky enough to visit there almost yearly to see his family. He moved to Fargo to work for the best company in the world and lives there, five winters and one big flood later, with his wife, Jolene. He spends time away from Microsoft slowly working toward an MBA, traveling occasionally, playing soccer and tennis frequently, and relaxing with friends as much as possible. Chris occasionally blogs at *http://blogs.msdn.com/chrisgarty*.

Raghavendran Gururajan is a program manager with the Dynamics AX Server platform team, with direct responsibility for the Data Access stack for the past two releases. Before this role, his experience as a Microsoft consultant in the field included building business applications for the high-tech manufacturing, auto industry, telecom, and financial sectors. He has a total of 17 years, experience in software development. He is a frequent speaker at customer and partner-focused events such as Convergence. He lives in Redmond, Washington, with his wife, Bhuvaneshwari, and son, Nanda.

Thejas Haridev Durgam joined Microsoft in 2005 as a software engineer on the Business Intelligence and Reporting team. During this time, he has worked on X++ reporting, SQL Server Reporting Services ad hoc reporting and SQL Reporting Services report integration with Enterprise Portal, and the Dynamics AX client. Before joining Microsoft, he worked for a year as a software development intern at Websense Inc. in San Diego, California.

Thejas has his master's degree in computer science from the State University of New York at Binghamton and has a bachelor's degree in computer science from R.V. College of Engineering, Bangalore, India. His hobbies include cricket, tennis, Xbox, and the stock market. He currently resides in downtown Bellevue, Washington.

Josh Honeyman is a senior development lead in Microsoft Business Solutions. He joined Microsoft as part of the acquisition of Great Plains Software, Inc., in 2001, after which he continued to work on Microsoft Dynamics GP. He is now responsible for the development of the workflow and business process infrastructure for Dynamics AX.



Wayne Kuo is a software design engineer in test who joined Microsoft in 2006 as a college graduate from the University of Waterloo in Ontario, Canada. He has been with the Business Intelligence and Reporting team in Dynamics AX for three years and has been primarily responsible for building the testing framework for the Microsoft Visual Studio report design experience in Dynamics AX 2009. He also worked on the SQL Server Reporting Services team to help deliver the Report Customization Definition Extension project that was part of the SQL Server 2008 release. During his school years, he held software development internships at a variety of organizations, including Environment Canada, Toronto Star Media, and Embarcadero Technologies. He now lives in Seattle and enjoys outdoor activities, motor racing, and music production at his local church.

Vijay Kurup is a senior program manager on the Dynamics AX product team working on the Dynamics AX Application Object Server (AOS) and in security areas. He is responsible for features such as memory management, session management, licensing and configuration, and security. For Dynamics AX 2009, he worked on designing the new server-bound Batch framework and adding multiple time-zone support and various other features to improve the performance and reliability of the AOS. Vijay has worked in the software industry both in India and the United States. He joined Microsoft in 2005 and has been a program manager on the Dynamics AX server team in Redmond since then. Vijay resides in Sammamish, Washington, with his wife, Anuradha, and two-year-old son, Nikhil.

Tete Mensa-Annan is a senior program manager lead on the Dynamics AX team. He is an 11-year Microsoft veteran focusing on workflow and business process management.

Amar Nalla is a senior software design engineer who has been working on Dynamics AX since version 4.0. He works on the platform team, primarily in the data access layer and other Dynamics AX server-related areas. He has been with Microsoft for the past eight years; before working on Dynamics AX, he worked on SQLXML and other XML technologies on the SQL Server team.

He blogs actively at *http://blogs.msdn.com/daxis*, where he covers a variety of topics related to the Dynamics AX platform.

Saveen Reddy has worked for Microsoft for 13 years on projects including Exchange Server, PhotoDraw, Windows Server, and Forefront Client Security. Currently he manages the Program Management team that builds business intelligence platform components and features into Dynamics AX. He regularly posts to his MSDN blog (*http://blogs.msdn.com/saveenr*) on topics such as program management, graphics and visualization, business intelligence, and software development.

Sri Srinivasan works as PM Architect for Dynamics AX, with responsibilities for driving performance initiatives, architecture changes, and scalability for the solution. Sri has been working with Microsoft Dynamics for four years and has been instrumental in the release of the benchmarks for Dynamics AX 4.0 and Dynamics AX 2009. Sri comes with an extensive ERP product development background, working on performance for other ERP solutions, such as PeopleSoft, JD Edwards, and Oracle before joining Microsoft. He blogs on the Performance team blog at *http://blogs.msdn.com/axperf*.

Satish Thomas is a software design engineer in the Dynamics AX product group with a focus on everything upgrade-related in Dynamics AX. He joined Microsoft in 2006 after graduating from Illinois Institute of Technology in Chicago, Illinois. Satish grew up in Africa (Nigeria, Botswana) before moving to the United States for college, and he currently lives in Sammamish, Washington.

This book substantially builds on content written for the *Inside Microsoft Dynamics AX 4.0* book. That content was written by Arthur Greef, Michael Fruergaard Pontoppidan, Lars Dragheim Olsen, Mey Meenaskshisundaram, Karl Tolgu, Hans Jørgen Skovgaard, Palle Agermark, Per Baarsoe Jorgensen, and Thomas Due Kay.