Inside Microsoft Dynamics AX 2012 R3

The Microsoft Dynamics AX Team
Contents

Foreword .......................................................... xxiii
Introduction ....................................................... xxv

PART I  A TOUR OF THE DEVELOPMENT ENVIRONMENT

Chapter 1  Architectural overview 3

Introduction ....................................................... 3
AX 2012 five-layer solution architecture ..................... 4
AX 2012 application platform architecture .................... 6
  Application development environments ....................... 6
  Data tier ................................................................ 7
  Middle tier ......................................................... 7
  Presentation tier ................................................ 8
AX 2012 application meta-model architecture .................. 9
  Application data element types ................................. 9
  MorphX user interface control element types ................. 11
  Workflow element types ....................................... 12
  Code element types ............................................. 13
  Services element types ........................................ 13
  Role-based security element types ............................ 14
  Web client element types ..................................... 14
  Documentation and resource element types .................. 16
  License and configuration element types .................... 17

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can continually improve our
books and learning resources for you. To participate in a brief online survey, please visit:
microsoft.com/learning/booksurvey
Table Browser tool .................................................... 53
Find tool .................................................................... 54
Compare tool ............................................................... 57
  Starting the Compare tool ........................................ 57
  Using the Compare tool .......................................... 59
  Compare APIs ....................................................... 61
Cross-Reference tool .................................................. 62
Version control .......................................................... 64
  Element life cycle .................................................. 66
  Common version control tasks .................................. 68
  Working with labels .............................................. 69
  Synchronizing elements ........................................... 69
  Viewing the synchronization log ............................... 70
  Showing the history of an element ............................ 71
  Comparing revisions ............................................. 72
  Viewing pending elements ..................................... 72
  Creating a build .................................................... 73
  Integrating AX 2012 with other version control systems ........................................ 74

Chapter 3  AX 2012 and .NET  ........................................ 75

  Introduction ................................................................ 75
  Integrating AX 2012 with other systems .................... 76
    Using third-party assemblies .................................. 76
    Writing managed code .......................................... 79
    Hot swapping assemblies on the server .................... 87
  Using LINQ with AX 2012 R3 .................................... 89
    The var keyword ................................................. 89
    Extension methods ............................................. 90
    Anonymous types .............................................. 90
    Lambda expressions .......................................... 91
    Walkthrough: Constructing a LINQ query .................. 91
    Using queries to read data .................................... 95
    AX 2012 R3–specific extension methods .................... 98
    Updating, deleting, and inserting records .................. 99
Contents

Limitations .................................................. 101
Advanced: limiting overhead .................................. 101

Chapter 4  The X++ programming language  105

Introduction .................................................... 105
Jobs ........................................................................ 106
The type system .................................................. 106
  Value types ..................................................... 106
  Reference types .............................................. 107
  Type hierarchies .............................................. 107
Syntax .................................................................... 110
  Variable declarations ........................................ 111
  Expressions ...................................................... 112
  Statements ....................................................... 114
  Macros ................................................................ 130
  Comments ......................................................... 132
  XML documentation .......................................... 132
Classes and interfaces ........................................... 133
  Fields .................................................................. 134
  Methods ............................................................ 135
  Delegates .......................................................... 136
  Pre-event and post-event handlers ......................... 138
  Attributes ........................................................ 139
Code access security ............................................. 140
Compiling and running X++ as .NET CIL .................... 142
Design and implementation patterns ....................... 143
  Class-level patterns ........................................... 144
  Table-level patterns .......................................... 146
PART II DEVELOPING FOR AX 2012

Chapter 5 Designing the user experience

Introduction .......................................................... 151
Role-tailored design approach .................................. 153
User experience components .................................... 154
  Navigation layer forms .......................................... 155
  Work layer forms ................................................ 156
Role Center pages .................................................. 156
  Cues ............................................................... 157
  Designing Role Centers ......................................... 157
Area pages .......................................................... 158
  Designing area pages .......................................... 159
List pages .......................................................... 160
  Scenario: taking a call from a customer ..................... 161
  Using list pages as an alternative to reports ............... 162
  Designing list pages .......................................... 163
Details forms ...................................................... 164
Transaction details forms ....................................... 167
Enterprise Portal web client user experience ............... 169
  Navigation layer forms ........................................ 170
  Work layer forms ................................................ 171
  Designing for Enterprise Portal ............................... 171
Designing for your users ....................................... 171

Chapter 6 The AX 2012 client

Introduction .......................................................... 173
Working with forms ............................................... 173
  Form patterns ................................................... 174
  Form metadata .................................................. 176
  Form data sources .............................................. 177
  Form queries .................................................... 183
Chapter 7  Enterprise Portal  207

Introduction ........................................... 207
Enterprise Portal architecture .................. 208
Enterprise Portal components .................... 211
  Web parts ........................................... 211
  AOT elements ...................................... 213
Datasets .......................................................... 213
Enterprise Portal framework controls ..................... 215
Developing for Enterprise Portal ............................. 228
Creating a model-driven list page ............................ 229
Creating a details page ......................................... 231
AJAX .............................................................. 233
Session disposal and caching ................................ 234
Context ............................................................ 235
Data ................................................................. 237
Metadata ............................................................ 238
Proxy classes ..................................................... 239
ViewState .......................................................... 241
Labels ............................................................... 242
Formatting .......................................................... 242
Validation .......................................................... 243
Error handling ..................................................... 244
Security ............................................................. 245
Secure web elements .......................................... 246
Record context and encryption ............................... 247
SharePoint integration ......................................... 248
Site navigation .................................................. 248
Site definitions, page templates, and web parts ........... 249
Importing and deploying a web part page ................. 252
Enterprise Search .............................................. 253
Themes ............................................................. 256

Chapter 8  Workflow in AX 2012 .......................... 257
Introduction ...................................................... 257
AX 2012 workflow infrastructure ........................... 258
Windows Workflow Foundation ............................. 261
Key workflow concepts ....................................... 262
Workflow document and workflow document class ........ 262
Workflow categories .......................................... 262
Workflow types .................................................. 263
Event handlers .................................................. 264
Menu items ...................................................... 264
Workflow elements ........................................... 264
Queues ............................................................... 265
Providers .......................................................... 266
Workflows ......................................................... 267
Workflow instances ............................................ 268
Work items ......................................................... 268
Workflow architecture ........................................ 268
Workflow runtime .............................................. 269
Workflow runtime interaction .............................. 270
Logical approval and task workflows .................... 272
Workflow life cycle .............................................. 275
Implementing workflows ..................................... 276
  Creating workflow artifacts, dependent artifacts, and
  business logic .................................................. 276
  Managing state ................................................ 279
  Creating a workflow category ............................ 280
  Creating the workflow document class ................. 280
  Adding a workflow display menu item .................. 283
  Activating the workflow .................................... 283

Chapter 9 Reporting in AX 2012 289
  Introduction .................................................. 289
  Inside the AX 2012 reporting framework ............... 290
    Client-side reporting solutions ....................... 290
    Server-side reporting solutions ...................... 292
    Report execution sequence ............................ 293
  Planning your reporting solution ....................... 293
    Reporting and users ..................................... 293
    Roles in report development ........................... 294
  Creating production reports ............................. 295
    Model elements for reports ............................ 296
    SSRS extensions ......................................... 299
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX 2012 extensions</td>
<td>300</td>
</tr>
<tr>
<td>Creating charts for Enterprise Portal</td>
<td>303</td>
</tr>
<tr>
<td>AX 2012 chart development tools</td>
<td>303</td>
</tr>
<tr>
<td>Integration with AX 2012</td>
<td>304</td>
</tr>
<tr>
<td>Data series</td>
<td>306</td>
</tr>
<tr>
<td>Adding interactive functions to a chart</td>
<td>308</td>
</tr>
<tr>
<td>Overriding the default chart format</td>
<td>310</td>
</tr>
<tr>
<td>Troubleshooting the reporting framework</td>
<td>311</td>
</tr>
<tr>
<td>The report server cannot be validated</td>
<td>311</td>
</tr>
<tr>
<td>A report cannot be generated</td>
<td>311</td>
</tr>
<tr>
<td>A chart cannot be debugged because of SharePoint sandbox issues</td>
<td>312</td>
</tr>
<tr>
<td>A report times out</td>
<td>312</td>
</tr>
</tbody>
</table>

**Chapter 10  BI and analytics**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>313</td>
</tr>
<tr>
<td>Components of the AX 2012 BI solution</td>
<td>313</td>
</tr>
<tr>
<td>Implementing the AX 2012 BI solution</td>
<td>315</td>
</tr>
<tr>
<td>Implementing the prerequisites</td>
<td>316</td>
</tr>
<tr>
<td>Configuring an SSAS server</td>
<td>317</td>
</tr>
<tr>
<td>Deploying cubes</td>
<td>317</td>
</tr>
<tr>
<td>Deploying cubes in an environment with multiple partitions</td>
<td>321</td>
</tr>
<tr>
<td>Processing cubes</td>
<td>323</td>
</tr>
<tr>
<td>Provisioning users</td>
<td>323</td>
</tr>
<tr>
<td>Customizing the AX 2012 BI solution</td>
<td>324</td>
</tr>
<tr>
<td>Configuring analytic content</td>
<td>325</td>
</tr>
<tr>
<td>Customizing cubes</td>
<td>327</td>
</tr>
<tr>
<td>Extending cubes</td>
<td>335</td>
</tr>
<tr>
<td>Integrating AX 2012 analytic components with external data sources</td>
<td>338</td>
</tr>
<tr>
<td>Maintaining customized and extended projects in the AOT</td>
<td>340</td>
</tr>
<tr>
<td>Creating cubes</td>
<td>341</td>
</tr>
<tr>
<td>Identifying requirements</td>
<td>341</td>
</tr>
<tr>
<td>Defining metadata</td>
<td>342</td>
</tr>
</tbody>
</table>
Generating and deploying the cube ........................................ 345
Adding KPIs and calculations ........................................................... 350
Displaying analytic content in Role Centers .............................. 351
Providing insights tailored to a persona ........................................ 352
Choosing a presentation tool based on a persona ....................... 352
SQL Server Power View ................................................................. 353
Power BI for Office 365 ................................................................. 359
Comparing Power View and Power BI ........................................... 360
Authoring with Excel ................................................................. 360
Business Overview web part and KPI List web part ................. 361
Developing reports with Report Builder ...................... 366
Developing reports with the Visual Studio tools for AX 2012 .... 366

Chapter 11  Security, licensing, and configuration 371

Introduction ........................................................................... 371
Security framework overview .................................................. 372
Authentication ......................................................................... 373
Authorization ........................................................................... 373
Data security ........................................................................... 376

Developing security artifacts ................................................. 377
Setting permissions for a form ............................................... 377
Setting permissions for server methods ................................. 379
Setting permissions for controls .......................................... 379
Creating privileges .................................................................. 380
Assigning privileges and duties to security roles ................. 381
Using valid time state tables .................................................. 383

Validating security artifacts .................................................. 384
Creating users ....................................................................... 384
Assigning users to roles ........................................................ 384
Setting up segregation of duties rules .................................. 385

Creating extensible data security policies ............................... 385
Data security policy concepts ............................................... 385
Developing an extensible data security policy ...................... 386
Debugging extensible data security policies ............................ 389
Security coding ....................................................... 390
  Table permissions framework .................................. 390
  Code access security framework ............................... 392
  Best practice rules ............................................. 393
  Security debugging ............................................. 394
Licensing and configuration ........................................ 396
  Configuration hierarchy ....................................... 398
  Configuration keys .............................................. 399
  Using configuration keys ...................................... 400
  Types of CALs .................................................. 401
  Customization and licensing .................................. 403

Chapter 12  AX 2012 services and integration 405
  Introduction .................................................... 405
  Types of AX 2012 services ........................................ 407
    System services ............................................... 407
    Custom services ............................................... 408
    Document services ............................................ 412
    Security considerations ...................................... 420
  Publishing AX 2012 services .................................... 421
  Consuming AX 2012 services ...................................... 422
    Sample WCF client for CustCustomerService ............... 422
  Consuming system services ...................................... 425
  Updating business documents .................................... 428
  Invoking custom services asynchronously .................... 430
  The AX 2012 send framework .................................... 432
    Implementing a trigger for transmission .................. 432
  Consuming external web services from AX 2012 ............... 435
  Performance considerations ..................................... 435

Chapter 13  Performance 437
  Introduction .................................................... 437
  Client/server performance ...................................... 437
    Reducing round trips between the client and the server .... 438
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing tier-aware code</td>
<td>442</td>
</tr>
<tr>
<td>Transaction performance</td>
<td>447</td>
</tr>
<tr>
<td>Set-based data manipulation operators</td>
<td>447</td>
</tr>
<tr>
<td>Restartable jobs and optimistic concurrency</td>
<td>465</td>
</tr>
<tr>
<td>Caching</td>
<td>467</td>
</tr>
<tr>
<td>Field lists</td>
<td>477</td>
</tr>
<tr>
<td>Field justification</td>
<td>483</td>
</tr>
<tr>
<td>Performance configuration options</td>
<td>483</td>
</tr>
<tr>
<td>SQL Administration form</td>
<td>483</td>
</tr>
<tr>
<td>Server Configuration form</td>
<td>484</td>
</tr>
<tr>
<td>AOS configuration</td>
<td>486</td>
</tr>
<tr>
<td>Client configuration</td>
<td>486</td>
</tr>
<tr>
<td>Client performance</td>
<td>486</td>
</tr>
<tr>
<td>Number sequence caching</td>
<td>487</td>
</tr>
<tr>
<td>Extensive logging</td>
<td>487</td>
</tr>
<tr>
<td>Master scheduling and inventory closing</td>
<td>487</td>
</tr>
<tr>
<td>Coding patterns for performance</td>
<td>487</td>
</tr>
<tr>
<td>Executing X++ code as common intermediate language</td>
<td>487</td>
</tr>
<tr>
<td>Using parallel execution effectively</td>
<td>488</td>
</tr>
<tr>
<td>The SysOperation framework</td>
<td>489</td>
</tr>
<tr>
<td>Patterns for checking to see whether a record exists</td>
<td>494</td>
</tr>
<tr>
<td>Running a query only as often as necessary</td>
<td>495</td>
</tr>
<tr>
<td>When to prefer two queries over a join</td>
<td>496</td>
</tr>
<tr>
<td>Indexing tips and tricks</td>
<td>497</td>
</tr>
<tr>
<td>When to use firstfast</td>
<td>498</td>
</tr>
<tr>
<td>Optimizing list pages</td>
<td>498</td>
</tr>
<tr>
<td>Aggregating fields to reduce loop iterations</td>
<td>499</td>
</tr>
<tr>
<td>Performance monitoring tools</td>
<td>501</td>
</tr>
<tr>
<td>Microsoft Dynamics AX Trace Parser</td>
<td>501</td>
</tr>
<tr>
<td>Monitoring database activity</td>
<td>510</td>
</tr>
<tr>
<td>Using the SQL Server connection context to find the SPID or user behind a client session</td>
<td>511</td>
</tr>
<tr>
<td>The client access log</td>
<td>512</td>
</tr>
<tr>
<td>Visual Studio Profiler</td>
<td>512</td>
</tr>
</tbody>
</table>
## Chapter 14  Extending AX 2012

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>515</td>
</tr>
<tr>
<td>The SysOperation framework</td>
<td>515</td>
</tr>
<tr>
<td>SysOperation framework classes</td>
<td>516</td>
</tr>
<tr>
<td>SysOperation framework attributes</td>
<td>517</td>
</tr>
<tr>
<td>Comparing the SysOperation and RunBase frameworks</td>
<td>517</td>
</tr>
<tr>
<td>RunBase example: SysOpSampleBasicRunbaseBatch</td>
<td>518</td>
</tr>
<tr>
<td>SysOperation example: SysOpSampleBasicController</td>
<td>526</td>
</tr>
<tr>
<td>The RunBase framework</td>
<td>533</td>
</tr>
<tr>
<td>Inheritance in the RunBase framework</td>
<td>533</td>
</tr>
<tr>
<td>Property method pattern</td>
<td>534</td>
</tr>
<tr>
<td>Pack-unpack pattern</td>
<td>535</td>
</tr>
<tr>
<td>Client/server considerations</td>
<td>538</td>
</tr>
<tr>
<td>The extension framework</td>
<td>539</td>
</tr>
<tr>
<td>Create an extension</td>
<td>539</td>
</tr>
<tr>
<td>Add metadata</td>
<td>540</td>
</tr>
<tr>
<td>Extension example</td>
<td>541</td>
</tr>
<tr>
<td>Eventing</td>
<td>543</td>
</tr>
<tr>
<td>Delegates</td>
<td>544</td>
</tr>
<tr>
<td>Pre and post events</td>
<td>545</td>
</tr>
<tr>
<td>Event handlers</td>
<td>545</td>
</tr>
<tr>
<td>Eventing example</td>
<td>547</td>
</tr>
</tbody>
</table>

## Chapter 15  Testing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>549</td>
</tr>
<tr>
<td>Unit testing features in AX 2012</td>
<td>550</td>
</tr>
<tr>
<td>Using predefined test attributes</td>
<td>550</td>
</tr>
<tr>
<td>Creating test attributes and filters</td>
<td>552</td>
</tr>
<tr>
<td>Microsoft Visual Studio 2010 test tools</td>
<td>556</td>
</tr>
<tr>
<td>Using all aspects of the ALM solution</td>
<td>556</td>
</tr>
<tr>
<td>Using an acceptance test driven development approach</td>
<td>557</td>
</tr>
<tr>
<td>Using shared steps</td>
<td>559</td>
</tr>
<tr>
<td>Recording shared steps for fast forwarding</td>
<td>560</td>
</tr>
</tbody>
</table>
Chapter 16  Customizing and adding Help  569

Introduction ...................................................... 569
Help system overview ............................................. 570
  AX 2012 client .................................................. 571
  Help viewer ...................................................... 571
  Help server ...................................................... 572
  AOS ................................................................. 573
Help content overview ........................................... 573
  Topics ............................................................... 573
  Publisher .......................................................... 574
  Table of contents ............................................... 574
  Summary page .................................................... 574
Creating content .................................................. 575
  Walkthrough: create a topic in HTML ....................... 576
  Adding labels, fields, and menu items to a topic .......... 584
  Make a topic context-sensitive ................................ 586
  Update content from other publishers ....................... 587
  Create a table of contents file ............................... 588
  Creating non-HTML content .................................... 591
Publishing content ............................................... 593
  Add a publisher to the Web.config file ................... 594
  Publish content to the Help server ......................... 596
  Set Help document set properties ........................... 597
Troubleshooting the Help system .............................. 598
  The Help viewer cannot display content .................... 598
  The Help viewer cannot display the table of contents .... 599
## PART III UNDER THE HOOD

## Chapter 17 The database layer 603

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>603</td>
</tr>
<tr>
<td>Temporary tables</td>
<td>604</td>
</tr>
<tr>
<td><em>InMemory</em> temporary tables</td>
<td>604</td>
</tr>
<tr>
<td><em>TempDB</em> temporary tables</td>
<td>609</td>
</tr>
<tr>
<td>Creating temporary tables</td>
<td>610</td>
</tr>
<tr>
<td>Surrogate keys</td>
<td>612</td>
</tr>
<tr>
<td>Alternate keys</td>
<td>614</td>
</tr>
<tr>
<td>Table relations</td>
<td>615</td>
</tr>
<tr>
<td>EDT relations and table relations</td>
<td>615</td>
</tr>
<tr>
<td>Foreign key relations</td>
<td>617</td>
</tr>
<tr>
<td>The <code>CreateNavigationPropertyMethods</code> property</td>
<td>618</td>
</tr>
<tr>
<td>Table inheritance</td>
<td>621</td>
</tr>
<tr>
<td>Modeling table inheritance</td>
<td>621</td>
</tr>
<tr>
<td>Table inheritance storage model</td>
<td>623</td>
</tr>
<tr>
<td>Polymorphic behavior</td>
<td>624</td>
</tr>
<tr>
<td>Performance considerations</td>
<td>625</td>
</tr>
<tr>
<td>Unit of Work</td>
<td>626</td>
</tr>
<tr>
<td>Date-effective framework</td>
<td>628</td>
</tr>
<tr>
<td>Relational modeling of date-effective entities</td>
<td>628</td>
</tr>
<tr>
<td>Support for data retrieval</td>
<td>630</td>
</tr>
<tr>
<td>Run-time support for data consistency</td>
<td>631</td>
</tr>
<tr>
<td>Full-text support</td>
<td>633</td>
</tr>
<tr>
<td>The <code>QueryFilter</code> API</td>
<td>635</td>
</tr>
<tr>
<td>Data partitions</td>
<td>638</td>
</tr>
<tr>
<td>Partition management</td>
<td>638</td>
</tr>
<tr>
<td>Development experience</td>
<td>638</td>
</tr>
<tr>
<td>Run-time experience</td>
<td>639</td>
</tr>
</tbody>
</table>
Chapter 18  Automating tasks and document distribution  641

Introduction ................................................................. 641
Batch processing in AX 2012 ........................................ 643
  Common uses of the batch framework ...................... 643
  Performance .......................................................... 644
Creating and executing a batch job ......................... 645
  Creating a batch-executable class ......................... 645
  Creating a batch job ............................................. 647
  Configuring the batch server and creating a batch group ... 654
Managing batch jobs ............................................. 657
Debugging a batch task ....................................... 658
Print management in AX 2012 ............................. 661
  Common uses of print management .................... 662
  The print management hierarchy ....................... 662
  Print management settings ............................... 663

Chapter 19  Application domain frameworks  671

Introduction ................................................................. 671
The organization model framework ...................... 672
  How the organization model framework works .... 672
  When to use the organization model framework .... 675
  Extending the organization model framework .......... 677
The product model framework ............................. 681
  How the product model framework works .......... 681
  When to use the product model framework .......... 685
  Extending the product model framework .......... 685
The operations resource framework ..................... 686
  How the operations resource framework works ... 687
  When to use the operations resource framework ... 690
  Extending the operations resource framework ...... 690
  MorphX model element prefixes for the operations resource framework .................................. 692
The dimension framework .................................. 692
  How the dimension framework works ............... 692
Chapter 20  Reflection  707

Introduction ........................................... 707

Reflection system functions ........................... 708
  Intrinsic functions ........................................... 708
  typeOf system function ...................................... 710
  classIdGet system function ................................ 710

Reflection APIs ............................................ 711
  Table data API ............................................. 711
  Dictionary API ............................................. 715
  Treenodes API ............................................. 718
  TreeNodeType ............................................. 721

Chapter 21  Application models  725

Introduction ............................................. 725

Layers .................................................... 726

Models ..................................................... 728
This page intentionally left blank
Foreword

The release of Microsoft Dynamics AX 2012 R3 and this book coincide with the tenth anniversary of my involvement with the development of this product. I’ve had the pleasure to work with a great team of people throughout that period. When I reflect on the modest ambition we set out with a decade ago, I’m excited to see all that we have achieved and am grateful for all the support we received along the way from our customers, partners, and the community around this product.

We set out to build a next-generation line-of-business system that empowered people. We wanted to go beyond traditional ERP in multiple ways:

■ First and foremost was to create a system of empowerment, not a system of records. Microsoft Dynamics AX is designed to help people do their jobs, not to record what they did after they did it.

■ Second, we wanted to maintain an agile system that allowed businesses to change at their own pace and not at the pace of previous generations of electronic concrete.

■ Third, we wanted to provide functional depth and richness while maintaining simplicity of implementation, to allow both midsize and large organizations to use the same system.

The embodiment of our first goal is role-tailored computing and pervasive BI. Those new to the Microsoft Dynamics AX community after AX 2009 can’t imagine a day when that wasn’t a standard part of the product. AX 2012 takes that richness to a whole new level with more than 80 predefined security roles, and Role Centers for more than 40 distinct functions in an organization.

The implementation of our second goal is in the richness of the AX 2012 metadata system and tools, combined with the fact that all of our solutions and localizations are designed to work together. AX 2012 enhances those capabilities even further while adding the organizational model, self-balancing dimensions, date effectivity, and other powerful application foundation elements.

The realization of the third goal came in the form of deep industry solutions for manufacturing, distribution, retail, service industries, and the public sector, along with a comprehensive set of life cycle services for design, development, deployment, and operations.
This book focuses on the enhancements to the Microsoft Dynamics AX developer toolset and is written by the team that brought you those tools. It’s truly an insider’s view of the entire AX 2012 development and runtime environment (now updated for the AX 2012 R3 release). I hope you enjoy it as much as we enjoyed writing the book and creating the product.

Here’s to the next ten years of our journey together.

Thanks,

Hal Howard

Head of Product Development, Microsoft Dynamics AX
Corporate Vice President, Microsoft Dynamics Research and Development
Introduction

Microsoft Dynamics AX 2012 represents a new generation of enterprise resource planning (ERP) software. With more than 1,000 new features and prebuilt industry capabilities for manufacturing, distribution, services, retail, and the public sector, AX 2012 provides a robust platform for developers to deliver specialized functionality more efficiently to the industries that they support. AX 2012 is a truly global solution, able to scale with any business as it grows. It is simple enough to deploy for a single business unit in a single country, yet robust enough to support the unique requirements for business systems in 36 countries/regions—all from a single-instance deployment of the software. With AX 2012 R3, Microsoft Dynamics AX delivers new levels of capability in warehouse and transportation management, demand planning, and retail.

AX 2012 R3 also represents an important step forward in the evolution of Microsoft Dynamics AX for the cloud. As Microsoft Technical Fellow Mike Ehrenberg explains:

*Microsoft is transforming for a cloud-first, mobile-first world. As part of that transformation, with the AX 2012 R3 release, we are certifying the deployment of Microsoft Dynamics AX on the Microsoft Azure cloud platform, which uses the Azure Infrastructure as a Service (IaaS) technology. This opens up the option for customers ready to move to the cloud to deploy the power of Microsoft Dynamics AX to run their business; for customers that favor on-premises deployment, it complements the option to harness the Microsoft Azure cloud platform for training, development, testing, and disaster recovery—all workloads with the uneven demand that the cloud serves so well. One of the most exciting new capabilities introduced with AX 2012 R3 is Lifecycle Services, our new Azure cloud-based service that streamlines every aspect of the ERP deployment, management, servicing, and upgrade lifecycle—regardless of whether AX 2012 itself is deployed on-premises or in the cloud. We are leveraging the cloud to deliver rapidly evolving services to help all of our customers ensure that they are following best practices across their AX 2012 projects. We are already seeing great results in rapid deployments, streamlined support interactions, and performance tuning—and this is only the beginning of our very exciting journey.*
Customers have also weighed in on the benefits of Microsoft Dynamics AX 2012:

*Microsoft Dynamics AX 2012 allows us to collaborate within our organization and with our constituents ... using built-in controls and fund/encumbrance accounting capabilities to ensure compliance with Public Sector requirements ... and using out-of-the-box Business Analytics and Intelligence ... so executives can make effective decisions in real time.*

Mike Bailey  
*Director of Finance and Information Services*  
*City of Redmond (Washington)*

*With AX 2012, developing for and customizing Microsoft Dynamics AX will be easier than ever. Developers will be able to work with X++ directly from within Microsoft Visual Studio and enjoy more sophisticated features in the X++ editor, for example. Also, the release includes more prebuilt interoperability with Microsoft SharePoint Server and SQL Server Reporting Services, so that developers spend less time on mundane work when setting up customer systems.*

Guido Van de Velde  
*Director of MECOMS™*  
*Ferranti Computer Systems*

AX 2012 is substantially different from its predecessor, which can mean a steep learning curve for developers and system implementers who have worked with previous versions. However, by providing a broad overview of the architectural changes, new technologies, and tools for this release, the authors of *Inside Microsoft Dynamics AX 2012 R3* have created a resource that will help reduce the time that it takes for developers to become productive.

**The history of Microsoft Dynamics AX**

Historically, Microsoft Dynamics AX encompasses more than 25 years of experience in business application innovation and developer productivity. Microsoft acquired the predecessor of Microsoft Dynamics AX, called Axapta, in 2002, with its purchase of the Danish company Navision A/S. The success of the product has spurred an increasing commitment of research and development resources, which allows Microsoft Dynamics AX to grow and strengthen its offering continuously.
The development team that created AX 2012 consists of three large teams, two that are based in the United States (Fargo, North Dakota, and Redmond, Washington) and one that is based in Denmark (Copenhagen). The Fargo team focuses on finance and human resources (HR), the Redmond team concentrates on project management and accounting and customer relationship management (CRM), and the Copenhagen team delivers supply chain management (SCM). In addition, a framework team develops infrastructure components, and a worldwide distributed team localizes the Microsoft Dynamics AX features to meet national regulations or local differences in business practices in numerous languages and markets around the world.

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

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 its predecessor, XAL. The X association carries over in the name Microsoft Dynamics AX.
Who should read this book

This book explores the technology and development tools in AX 2012 through the AX 2012 R3 release. It is designed to help new and existing Microsoft Dynamics AX developers by providing holistic and in-depth information about developing for AX 2012—information that may not be available from other resources, such as SDK documentation, blogs, or forums. It aids developers who are either customizing AX 2012 for a specific implementation or building modules or applications that blend seamlessly with AX 2012. System implementers and consultants will also find much of the information useful.

Assumptions

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

Who should not read this book

This book is not aimed at those who install, upgrade, or deploy AX 2012. It is also beyond the scope of this book to include details about the sizing of production environments. For more information about these topics, refer to the extensive installation and implementation documentation that is supplied with the product or that is available on Microsoft TechNet, Microsoft Developer Network (MSDN), and other websites.

The book also does not provide instructions for those who configure parameter options within AX 2012 or the business users who use the application in their day-to-day work. For assistance with these activities, refer to the help that is included with the product and available on TechNet at http://technet.microsoft.com/en-us/library/gg852966.aspx.
Organization of this book

Although Inside Microsoft Dynamics AX 2012 R3 does not provide exhaustive coverage of every feature in the product, it does offer a broad view that will benefit developers as they develop for AX 2012.

This book is divided into four sections, each of which focuses on AX 2012 from a different angle. Part I, “A tour of the development environment,” provides an overview of the AX 2012 architecture that has been written with developers in mind. The chapters in Part I also provide a tour of the internal AX 2012 development environment to help new developers familiarize themselves with the designers and tools that they will use to implement their customizations, extensions, and integrations.

Part II, “Developing for AX 2012,” provides the information that developers need to customize and extend AX 2012. In addition to explanations of the features, many chapters include examples, some of which are available as downloadable files that can help you learn how to code for AX 2012. For information about how to access these files, see the “Code samples” section, later in this introduction.

Part III, “Under the hood,” is largely devoted to illustrating how developers can use the underlying foundation of the AX 2012 application frameworks to develop their solutions, with a focus on the database layer, system and application frameworks, reflection, and models.

Part IV, “Beyond AX 2012,” focuses on developing companion apps for mobile devices that allow AX 2012 users to participate in critical business processes even when they are away from their computers. It also describes exciting new techniques and tools, such as Lifecycle Services, that help partners and customers manage every aspect of the application life cycle.
Conventions and features in this book

This book presents information by using the following conventions, which are designed to make the information readable and easy to follow.

- Application Object Tree (AOT) paths use backslashes to separate nodes, such as `Forms\AccountingDistribution\Methods`.
- The names of methods, functions, properties and property values, fields, and nodes appear in italics.
- Registry keys and T-SQL commands appear in capital letters.
- User interface (UI) paths use angle brackets to indicate actions—for example, “On the File menu, point to Tools > Options.”
- Boxed elements with labels such as “Note” provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, “Press Alt+Tab” means that you hold down the Alt key while you press the Tab key.

System requirements

To work with most of the sample code, you must have the RTM version of AX 2012 installed. For the Language-Integrated Query (LINQ) samples, you must be using AX 2012 R3. For information about the system requirements for installing Microsoft Dynamics AX 2012, see the Microsoft Dynamics AX 2012 Installation Guide at


You must also have an Internet connection to download the sample files that are provided as supplements to many of the chapters.

Note Some of the features described in this book, such as data partitioning and the EP Chart Control, apply only to AX 2012 R2 and AX 2012 R3. That is noted where those features are discussed.
Code samples

Most of the chapters in this book include code examples that let you interactively try out the new material presented in the main text. You can download the example code from the following page:

http://aka.ms/InsideDynaAXR3

Follow the instructions to download the 9780735685109_files.zip file.

Installing the code samples

Follow these steps to install the code samples on your computer:

1. Unzip the file that you downloaded from the book’s website.
2. If prompted, review the displayed end user license agreement. If you accept the terms, select the accept option, and then click Next.

   Note  If the license agreement doesn’t appear, you can access it from the same webpage from which you downloaded the file.

Using the code samples

The code examples referenced in each chapter are provided as both .xpo files that you can import into Microsoft Dynamics AX and Visual Studio projects that you can open through the corresponding .csproj files. Many of these examples are incomplete, and you cannot import and run them successfully without following the steps indicated in the associated chapter.
Acknowledgments

We want to thank all the people who assisted us in bringing this book to press. We apologize for anyone whose name we missed.

Microsoft Dynamics product team

Special thanks go to the following colleagues, whom we’re fortunate to work with.

Margaret Sherman, whose Managing Editor duties included wrangling authors, chasing down stray chapters, translating techno-speak into clear English, keeping numerous balls in the air, and herding a few cats. Margaret kept the project moving forward, on schedule, on budget, and with a real commitment to quality content. Thank you, Margaret! This project wouldn’t have happened without your leadership!

Mark Baker and Steve Kubis, who contributed ace project management and editing work.

Margo Crandall, who provided a quick and accurate technical review at the last minute for Chapter 23.

Hal Howard, Richard Barnwell, and Ann Beebe, who sponsored the project and provided resources for it.

We’re also grateful to the following members of the product team, who provided us with the reviews and research that helped us refine this book:

Ned Baker
Ian Beck
Andy Blehm
Jim Brotherton
Ed Budrys
Gregory Christiaens
Ahmad El Husseini
Josh Honeyman
Hitesh Jawa
Vijeta Johri
Bo Kampmann
Vinod Kumar
Arif Kureshy
Josh Lange

Mey Meenakshisundaram
Igor Menshutkin
Jatan Modi
Sasha Nazarov
Adrian Orth
Christopher Read (Entirenet)
Bruce Rivard
Gana Sadasivam
Alex Samoylenko
Ramesh Shankar
Tao Wang
Lance Wheelwright
Chunke Yang
In addition, we want to thank Joris de Gruyter of Streamline Systems LLC. His SysTestListenerTRX code samples on CodePlex (http://dynamicsaxbuild.codeplex.com/releases), with supporting documentation on his blog (http://daxmusings.blogspot.com/), and his collaboration as we investigated this approach for executing SysTests from Microsoft Dynamics AX were valuable resources as we prepared the chapter on testing.

Microsoft Press
Another big thank you goes to the great people at Microsoft Press for their support and expertise throughout the writing and publishing process.

Carol Dillingham, the Content Project Manager for the book, who provided ongoing support and guidance throughout the life of the project.

Rosemary Caperton—Acquisitions Editor

Allan Iversen—Technical Reviewer

Kathy Krause—Project Editor and Copyeditor with Online Training Solutions, Inc. (OTSI)

Errata, updates, & book support
We’ve made every effort to ensure the accuracy of this book. If you discover an error, please submit it to us via mspinput@microsoft.com. You can also reach the Microsoft Press Book Support team for other support via the same alias. Please note that product support for Microsoft software and hardware is not offered through this address. For help with Microsoft software or hardware, go to http://support.microsoft.com.

Free ebooks from Microsoft Press
From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!
We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you’re busy, so we’ve kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

Let’s keep the conversation going! We’re on Twitter:

http://twitter.com/MicrosoftPress

This edition of the book is dedicated to Hal Howard, with many thanks for your leadership.

—The Microsoft Dynamics AX Team
This page intentionally left blank
CHAPTER 2

The MorphX development environment and tools

In this chapter

Introduction .............................................. 19
Application Object Tree ................................ 21
Projects .................................................. 27
The property sheet ........................................ 31
X++ code editor .......................................... 32
Label editor .............................................. 34
Compiler ................................................... 38
Best Practices tool ....................................... 40
Debugger .................................................... 45
Reverse Engineering tool ............................... 48
Table Browser tool ...................................... 53
Find tool ...................................................... 54
Compare tool ............................................ 57
Cross-Reference tool ..................................... 62
Version control .......................................... 64

Introduction

AX 2012 includes a set of tools, the MorphX development tools, that you can use to build and modify AX 2012 business applications. Each feature of a business application uses the application model elements described in Chapter 1, “Architectural overview.” With the MorphX tools, you can 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 indexes, the relations, and the methods, among other things.

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 AX 2012 software development kit (SDK) on the Microsoft Developer Network (MSDN).
Tip  To enable development mode in AX 2012, press Ctrl+Shift+W to launch the Development Workspace, which holds all of the development tools.

Table 2-1 lists the MorphX tools and components.

**TABLE 2-1**  MorphX tools and other components used for development.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Object Tree (AOT)</td>
<td>Start development activities. The AOT is the main entry point for most development activities. It allows the developer to browse the repository of all elements that together make up the business application. You can use the AOT to invoke the other tools and to inspect and create elements.</td>
</tr>
<tr>
<td>Projects</td>
<td>Group related elements into projects.</td>
</tr>
<tr>
<td>Property sheet</td>
<td>Inspect and modify properties of elements. The property sheet shows key and value pairs.</td>
</tr>
<tr>
<td>X++ code editor</td>
<td>Inspect and write X++ source code.</td>
</tr>
<tr>
<td>Label editor</td>
<td>Create and inspect localizable strings.</td>
</tr>
<tr>
<td>Compiler</td>
<td>Compile X++ code into an executable format.</td>
</tr>
<tr>
<td>Best Practices tool</td>
<td>Automatically detect defects in both your code and your elements.</td>
</tr>
<tr>
<td>Debugger</td>
<td>Find bugs in your X++ code.</td>
</tr>
<tr>
<td>Reverse Engineering tool</td>
<td>Generate Microsoft Visio Unified Modeling Language (UML) and entity relationship diagrams (ERDs) from elements.</td>
</tr>
<tr>
<td>Table Browser tool</td>
<td>View the contents of a table directly from a table element.</td>
</tr>
<tr>
<td>Type Hierarchy Browser and Type Hierarchy Context</td>
<td>Navigate and understand the type hierarchy of the currently active element.</td>
</tr>
<tr>
<td>Find tool</td>
<td>Search for code or metadata patterns in the AOT.</td>
</tr>
<tr>
<td>Compare tool</td>
<td>See a line-by-line comparison of two versions of the same element.</td>
</tr>
<tr>
<td>Cross-Reference tool</td>
<td>Determine where an element is used.</td>
</tr>
<tr>
<td>Version control</td>
<td>Track all changes to elements and see a full revision log.</td>
</tr>
</tbody>
</table>

You can access these development tools from the following places:

- In the Development Workspace, on the Tools menu
- On the context menus of elements in the AOT

You can personalize the behavior of many MorphX tools by clicking Options on the Tools menu. Figure 2-1 shows the Options form.
FIGURE 2-1 The Options form, in which development options are specified.

Application Object Tree

The AOT is the main entry point to MorphX and 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 through 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 are located under Data Dictionary, and all web-related elements are located under Web. Figure 2-2 shows the AOT.
You can navigate through 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 arranged alphabetically. Because there are thousands of elements, it’s important to understand the naming conventions and adhere to them to use the AOT effectively.

All element names in the AOT use the following structure:

\(<\text{Business area name}> + <\text{Functional area}> + <\text{Functionality, action performed, or type of content}>\)

With 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. For example, in the name `VendPaymReconciliationImport`, the prefix `Vend` is an abbreviation of the business area name (Vendor), `PaymReconciliation` describes the functional area (payment reconciliation), and `Import` lists the action performed (import). The name `CustPaymReconciliationImport` describes a similar functional area and action for the Customer business area.

**Tip** When building add-on functionality, in addition to following this naming convention, you should add another prefix that uniquely identifies the solution. This additional prefix will help prevent name conflicts if your solution is combined with work from other sources. Consider using a prefix that identifies the company and the solution. For example, if a company called MyCopr is building a payroll system, it could use the prefix `McPR` on all elements added.
Table 2-2 contains a list of the most common prefixes and their descriptions.

**TABLE 2-2** Common prefixes.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ax</td>
<td>Microsoft Dynamics AX typed data source</td>
</tr>
<tr>
<td>Axd</td>
<td>Microsoft Dynamics AX business document</td>
</tr>
<tr>
<td>Asset</td>
<td>Asset management</td>
</tr>
<tr>
<td>BOM</td>
<td>Bill of material</td>
</tr>
<tr>
<td>COS</td>
<td>Cost accounting</td>
</tr>
<tr>
<td>Cust</td>
<td>Customer</td>
</tr>
<tr>
<td>Dir</td>
<td>Directory, global address book</td>
</tr>
<tr>
<td>EcoRes</td>
<td>Economic resources</td>
</tr>
<tr>
<td>HRM/HCM</td>
<td>Human resources</td>
</tr>
<tr>
<td>Invent</td>
<td>Inventory management</td>
</tr>
<tr>
<td>JMG</td>
<td>Shop floor control</td>
</tr>
<tr>
<td>KM</td>
<td>Knowledge management</td>
</tr>
<tr>
<td>Ledger</td>
<td>General ledger</td>
</tr>
<tr>
<td>PBA</td>
<td>Product builder</td>
</tr>
<tr>
<td>Prod</td>
<td>Production</td>
</tr>
<tr>
<td>Proj</td>
<td>Project</td>
</tr>
<tr>
<td>Purch</td>
<td>Purchase</td>
</tr>
<tr>
<td>Req</td>
<td>Requirements</td>
</tr>
<tr>
<td>Sales</td>
<td>Sales</td>
</tr>
<tr>
<td>SMA</td>
<td>Service management</td>
</tr>
<tr>
<td>SMM</td>
<td>Sales and marketing management, also called customer relationship management (CRM)</td>
</tr>
<tr>
<td>Sys</td>
<td>Application frameworks and development tools</td>
</tr>
<tr>
<td>Tax</td>
<td>Tax engine</td>
</tr>
<tr>
<td>Vend</td>
<td>Vendor</td>
</tr>
<tr>
<td>Web</td>
<td>Web framework</td>
</tr>
<tr>
<td>WMS</td>
<td>Warehouse management</td>
</tr>
</tbody>
</table>

**Tip** When creating new elements, ensure that you follow the recommended naming conventions. Any future development and maintenance will be much easier.

Projects, described in detail later in this chapter, provide an alternative view of the information in the AOT.
Creating elements in the AOT

You can create new elements in the AOT by right-clicking the element category node and selecting New <Element Type>, as shown in Figure 2-3.

![Creating a new element in the AOT.](image)

Elements are given automatically generated names when they are created. However, you should replace the default names with new names that conform to the naming convention.

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 2-9, later in this chapter) to inspect or modify properties, and you can use the X++ code editor (shown in Figure 2-11, later in this chapter) 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 appear 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 Arrow or Down Arrow key.

A red vertical line next to a root element name marks it as modified and unsaved, or dirty, as shown in Figure 2-4.
A dirty element is saved in the following situations:

- When the element is executed.
- When the developer explicitly invokes the Save or Save All action.
- When autosave takes place. You specify the frequency of autosave in the Options form, which is accessible from the Tools menu.

**Refreshing elements in the AOT**

If several developers modify elements simultaneously in the same installation of AX 2012, each developer's local elements might not be synchronized 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 force the refresh of an element explicitly. You do this by right-clicking the element, and then clicking Restore. This action refreshes both the on-disk and the in-memory versions of the element.

Typically, the general integrity of what's shown in the AOT is managed automatically, but some operations, such as restoring the application database or reinstalling the application, can lead to inconsistencies that require manual resolution to ensure that the latest elements are used. To perform manual resolution, follow these steps:

1. Close the AX 2012 client to clear any in-memory elements.
2. Stop the Microsoft Dynamics Server service on the Application Object Server (AOS) to clear any in-memory elements.
3. Delete the application element cache files (*.auc) from the Local Application Data folder (located in %LocalAppData%) to remove the on-disk elements.
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 any node.

Here are two facts to remember about actions:

■ The actions that are 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 node as the root. This action was used to create the screen capture of the AccountingDistribution element shown earlier in Figure 2-4. After 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 to be available only for certain nodes, modify the **verifyItem** method on the **SysContextMenu** class.

Element layers and models in the AOT

When you modify an element from a lower layer, a copy of the element is placed in the current layer and the current model. All elements in the current layer appear in bold type (as shown in Figure 2-5), which makes it easy to recognize changes. For a description of the layer technology, see the “Layers” section in Chapter 21, “Application models.”
You can use the Application object layer and Application object model settings in the Options form to personalize the information shown after the element name in the AOT (see Figure 2-1, shown earlier). Figure 2-5 shows a class with the Show All Layers option set. 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 then click Layers to access its versions from lower layers. It is highly recommended that you use the Show All Layers setting during code upgrade because it provides a visual representation of the layer dimension directly in the AOT.

**Projects**

For a fully customizable overview of the elements, you can use *projects*. In a project, you can group and structure elements according to your preference. A project is a powerful alternative to the AOT because you can collect all the elements needed for a feature in one project.

**Creating a project**

You open projects from the AOT by clicking the Project icon on the toolbar. Figure 2-6 shows the Projects window and its *Private* and *Shared* projects nodes.
Except for its structure, a project generally behaves 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 AX 2012 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 form. If specified, the chosen project automatically opens when AX 2012 is started.

Automatically generating a project

Projects can be automatically generated in several ways—from using group masks to customizing project types—to make working with them easier. The following sections outline the various ways to generate projects automatically.

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 providing a regular expression as the value of the GroupMask property. The contents of the group are created automatically and
are 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 2-7 shows the ProjectGroupType property set to Classes and the GroupMask property set to ReleaseUpdate on a project group. All classes with names containing ReleaseUpdate (the prefix for data upgrade scripts) will be included in the project group.

**FIGURE 2-7** Property sheet specifying settings for ProjectGroupType and GroupMask.

Figure 2-8 shows the resulting project when the settings from Figure 2-7 are used.

**FIGURE 2-8** Project created by using a group mask.

**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 Filter on the project’s toolbar. Depending on the complexity of the query, a project can be generated instantly or it might take several minutes.
With filters, you can create projects containing elements that meet the following criteria:

- 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 that 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 that make up the newly created wizard.

You can also use several other wizards, such as the AIF Document Service Wizard and the Class Wizard, to create projects. To access these wizards, on the Tools menu, click Wizards.

**Layer comparison**  You can compare the elements in one layer with the elements in another layer, which is called the *reference layer*. If an element exists in both layers and the definitions of the element are different, or if the element doesn’t exist in the reference layer, the element is added to the resulting project. To compare layers, click Tools > Code Upgrade > Compare Layers.

**Upgrade projects**  When you upgrade from one version of Microsoft Dynamics AX to another or install a new service pack, you need to deal with any new elements that have been 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 code upgrade, see “Microsoft Dynamics AX 2012 White Papers: Code Upgrade” at [http://www.microsoft.com/download/en/details.aspx?id=20864](http://www.microsoft.com/download/en/details.aspx?id=20864).

To create an upgrade project, click Tools > Code Upgrade > Detect Code Upgrade Conflicts.

**Project types**

When you create a new project, you can specify a project type. So far, this chapter has discussed standard projects. The Test project, used to group a set of classes for unit testing, is another specialized project type provided in AX 2012.
You can create a custom specialized project by creating a new class that extends the `ProjectNode` class. With a specialized project, you can control the structure, icons, and actions available to the project.

The property sheet

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

When the Development Workspace opens, the property sheet is visible by default. If you close it, you can open it again anytime by pressing Alt+Enter or by clicking the Properties button on the toolbar of the Development Workspace. The property sheet automatically updates itself to show properties for any element selected in the AOT. You don't have to open the property sheet manually for each element; you can leave it open and browse the elements. Figure 2-9 shows the property sheet for the `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.

![Figure 2-9 Property sheet for an element in the AOT.](image)

Figure 2-10 shows the Categories tab for the class shown in Figure 2-9. On this tab, related properties are categorized. For elements with many properties, this view can make it easier to find the right property.
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. Elements that come from Microsoft all have the same time and user stamps.

The default sort order places related properties near each other. Categories were introduced in an earlier version of Microsoft Dynamics AX to make finding properties easier, but you can also sort properties alphabetically by setting a parameter in the Options form.

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**

All X++ code is written 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 2-11.
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 2-3. For AX 2012, some shortcuts differ from those in earlier versions to align with commonly used integrated development environments (IDEs) such as Microsoft Visual Studio.

**TABLE 2-3** X++ code editor shortcut keys.

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show the Help window</td>
<td>F1</td>
<td>Opens context-sensitive Help for the type or method currently selected in the editor.</td>
</tr>
<tr>
<td>Go to the next error message</td>
<td>F4</td>
<td>Opens the editor and positions the cursor at the next compilation error, based on the contents of the Compiler Output window.</td>
</tr>
<tr>
<td>Execute the current element</td>
<td>F5</td>
<td>Starts the current form, job, or class.</td>
</tr>
<tr>
<td>Compile</td>
<td>F7</td>
<td>Compiles the current method.</td>
</tr>
<tr>
<td>Toggle a breakpoint</td>
<td>F9</td>
<td>Sets or removes a breakpoint.</td>
</tr>
<tr>
<td>Run an editor script</td>
<td>Alt+R</td>
<td>Lists all available editor scripts and lets you select one to execute (such as Send To Mail Recipient).</td>
</tr>
<tr>
<td>Open the Label editor</td>
<td>Ctrl+Alt+Spacebar</td>
<td>Opens the Label editor and searches for the selected text.</td>
</tr>
<tr>
<td>Go to the implementation (drill down in code)</td>
<td>F12</td>
<td>Goes to the implementation of the selected method. This shortcut is highly useful for fast navigation.</td>
</tr>
<tr>
<td>Go to the next method</td>
<td>Ctrl+Tab</td>
<td>Sets the focus on the next method in the editor.</td>
</tr>
<tr>
<td>Go to the previous method</td>
<td>Ctrl+Shift+Tab</td>
<td>Sets the focus on the previous method in the editor.</td>
</tr>
<tr>
<td>Enable block selection</td>
<td>Alt+&lt;mouse select&gt; or Alt+Shift+arrow keys</td>
<td>Selects a block of code. Select the code you want by pressing the Alt key while selecting text with the mouse. Alternatively, hold down Alt and Shift while moving the cursor with the arrow keys.</td>
</tr>
<tr>
<td>Cancel the selection</td>
<td>Esc</td>
<td>Cancels the current selection.</td>
</tr>
<tr>
<td>Delete the current selection or line</td>
<td>Ctrl+X</td>
<td>Deletes the current selection or, if nothing is selected, the current line.</td>
</tr>
<tr>
<td>Start an incremental search</td>
<td>Ctrl+I</td>
<td>Starts an incremental search, which marks the first occurrence of the search text as you type it. Pressing Ctrl+I again moves to the next occurrence, and Ctrl+Shift+I moves to the previous occurrence.</td>
</tr>
<tr>
<td>Insert an XML document header</td>
<td>///</td>
<td>Inserts an XML comment header when you type ///. When typed in front of a class or method header, this shortcut prepopulates the XML document with template information relevant to the class or method.</td>
</tr>
<tr>
<td>Comment the selection</td>
<td>Ctrl+E, C</td>
<td>Inserts comment marking for the current selection.</td>
</tr>
<tr>
<td>Uncomment the selection</td>
<td>Ctrl+E, U</td>
<td>Removes comment marking for the current selection.</td>
</tr>
</tbody>
</table>
Editor scripts

The X++ code editor contains a set of editor scripts that you can invoke by clicking the Script icon on the X++ code editor toolbar or by right-clicking an empty line in the code editor, pointing to Scripts, and then clicking the script you want. Built-in editor scripts provide functionality such as the following:

- Send to mail recipient.
- Send to file.
- Generate code for standard code patterns such as `main`, `construct`, and `parm` methods.
- Open the AOT for the element that owns the method.

**Note** By generating code, in a matter of minutes you can create 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 and setters on classes. Code is generated in accordance with X++ best practices.

**Tip** To add a main method to a class, add a new method, press Ctrl+A to select all code in the editor tab for the new method, type `main`, and then press the Tab key. This will replace the text in the editor with the standard template for a static main method.

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 AX 2012 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 uses. 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 files that are named with three-letter identifiers. In AX 2012, the label files are managed in the AOT and distributed by using model files. Figure 2-12 shows how the `Label Files` node in the AOT looks with multiple label files and the en-us language identifier.
The underlying source representation is a simple text file that follows this naming convention:

\texttt{Ax<Label file identifier><Locale>.ALD}

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

\texttt{Axsysen-us.ALD}
\texttt{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 simple:

\texttt{@<Label ID><Label text>}
\texttt{[Label description]}

Figure 2-13 shows an example of a label file.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{label_file_notepad.png}
\caption{Label file opened in Notepad showing a few labels from the en-us label file.}
\end{figure}
This simple structure allows for localization outside of AX 2012 with third-party tools. The AOT provides a set of operations for the label files, including an Export To Label file that can be used to extract a file for external translation.

You can create new label files by using the Label File Wizard, which you access directly from the Label Files node in the AOT, or from the Tools menu by pointing to Wizards > Label File Wizard. The wizard guides you through the steps of 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. If you have an existing .ald file, you can also create the appropriate entry in the AOT by using Create From File on the context menu of the Label Files node in the AOT.

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 match the layer in which it is used. AX 2012 includes 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. However, AX 2012 doesn’t impose any limitations on the label file name.

Consider the following 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 probably merge two separately developed features into the same installation, a task that will be more difficult if the label file names collide.

- When referencing existing labels, feel free to reference labels in the label files provided by Microsoft, but avoid making changes to labels in these label files because they are updated with each new version of Microsoft Dynamics AX.

Creating a label

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

- On the Tools menu, point to Label > Label Editor.
- On the X++ code editor toolbar, click the Lookup Label > Text button.
- On text properties in the property sheet, click the Lookup button.

You can use the Label editor (shown in Figure 2-14) to find existing labels. Reusing a label is sometimes preferable to creating a new one. You can create a new label by pressing Ctrl+N or by clicking New.
In addition to finding and creating new labels, you can use the Label editor to find out where a label is used. The Label editor also logs any changes to each label.

Consider the following tips when creating and reusing labels:

- When reusing a label, make sure that the label means what you intend it to in all languages. Some words are homonyms (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. Use the description column to note the intended meaning of the label.

- When creating a new label, ensure that you use complete sentences or other stand-alone words or phrases. Don’t construct complete sentences by concatenating labels with one or two 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 be converted automatically 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 is greater than or equal to 1. Placeholders can also be used within labels. The following code shows a few examples:

```csharp
// 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");
```

FIGURE 2-14 The Label editor.
The following are some best practices to consider when referencing labels from X++:

- Always create user interface text by using a label. When referencing labels from X++ code, use double quotation marks.
- 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.

**Compiler**

Whenever you make a change to X++ code, you must recompile, just as you would in any other programming language. You start the recompile by pressing F7 in the X++ code editor. Your code also recompiles whenever you close the editor or save changes to an 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 2-4, later in this section, for a list of example compiler warnings. Compiler warnings can and should be addressed. Check-in attempts with compiler warnings are rejected unless specifically allowed in the version control system settings.
- **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. Be careful when using to-do comments to postpone work, and never release code unless all to-dos are addressed. For a developer, there is nothing worse than debugging an issue and finding a to-do comment indicating that the issue was already known but overlooked.
- **Best practice deviations** The Best Practices tool carries out more complex validations. For more information, see the “Best Practices tool” section later in this chapter.
Note  Unlike other languages, X++ requires that you compile only code you’ve modified, because the intermediate language the compiler produces is persisted along with the X++ code and metadata. Of course, your changes can require other methods that consume 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. Always ensure that you compile the entire AOT when you consider your changes complete, and fix any compilation errors found. If you’re changing the class declaration somewhere in a class hierarchy, all classes deriving from the changed class should be recompiled, too. This can be achieved by using the Compile Forward option under Add-Ins in the context menu for the changed class node.

The Compiler Output window provides access to every issue found during compilation, as shown in Figure 2-15. The window presents one list of all relevant errors, warnings, best practice deviations, and tasks. Each type of message can be disabled or enabled by using the respective buttons. Each line in the list contains information about each issue that the compiler detects, a description of the issue, and its location.

FIGURE 2-15 The powerful combination of the X++ code editor and the Compiler Output window.
You can export the contents of the Compiler Output window. 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 Internet Explorer or reimported into the Compiler Output window in another AX 2012 session.

In the Compiler Output window, click Setup > Compiler to define the types of issues that the compiler should report. Compiler warnings are grouped into four levels, as shown by the examples in Table 2-4. Each level represents a certain level of severity, with 1 being the most critical and 4 being recommended to comply with best practices.

<table>
<thead>
<tr>
<th>Warning message</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break statement found outside legal context</td>
<td>1</td>
</tr>
<tr>
<td>The new method of a derived class does not call super()</td>
<td>1</td>
</tr>
<tr>
<td>The new method of a derived class may not call super()</td>
<td>1</td>
</tr>
<tr>
<td>Function never returns a value</td>
<td>1</td>
</tr>
<tr>
<td>Not all paths return a value</td>
<td>1</td>
</tr>
<tr>
<td>Assignment/comparison loses precision</td>
<td>1</td>
</tr>
<tr>
<td>Unreachable code</td>
<td>2</td>
</tr>
<tr>
<td>Empty compound statement</td>
<td>3</td>
</tr>
<tr>
<td>Class names should start with an upper case letter</td>
<td>4</td>
</tr>
<tr>
<td>Member names should start with a lower case letter</td>
<td>4</td>
</tr>
</tbody>
</table>

### Best Practices tool

Following Microsoft Dynamics AX best practices when you develop applications has several important benefits:

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

- Your learning curve is flattened. When you perform similar tasks in a standard way, you are more likely to be comfortable in an unknown area of the application. Consequently, adding new resources to a project is more cost effective, and downstream consumers of the code can 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 AX 2012, installing service packs, or upgrading to future releases.
You are more likely to ship on time. Most of the problems developers face when implementing a solution in Microsoft 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 in the code base.

The AX 2012 SDK contains an important discussion about conforming to best practices in AX 2012. Constructing code that follows proven standards and patterns can’t guarantee a project’s success, but it minimizes the risk of failure because of late, expensive discovery, and it decreases the long-term maintenance cost. The AX 2012 SDK is available at http://msdn.microsoft.com/en-us/library/aa496079.aspx.

The Best Practices tool is a powerful supplement to the best practices discussion in the SDK. This tool is the MorphX version of a static code analysis tool, similar to FxCop for the Microsoft .NET Framework. The Best Practices tool is embedded in the compiler, and the results are reported in the Compiler Output window the same way as other messages from the compilation process.

The purpose of static code analysis is to detect defects and risky coding patterns in the code automatically. 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. Because an application in AX 2012 is much more than just code, the Best Practices tool also performs static analysis on the metadata—the properties, structures, and relationships that are maintained in the AOT.

The Best Practices tool displays deviations from the best practice rules, as shown earlier in Figure 2-15. Double-clicking a line on the Best Practices tab opens the X++ code editor on the violating line of code or, if the Best Practices violation is related to metadata, it will open the element in an AOT window.

**Rules**

The Best Practices tool includes about 400 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: on the Tools menu, click Options > Development, and then click Best Practices.

**Note** You must set the compiler error level to 4 if you want best practice rule violations to be reported. To turn off best practice violation reporting, in the Compiler Output window, click Setup > Compiler, and then set the compiler error level to less than 4.
The best practice rules are divided into categories. By default, all categories are turned on, as shown in Figure 2-16.

![Best practice parameters dialog box](image)

**FIGURE 2-16** The 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** Following a 95/5 rule for warnings is recommended. 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 (for example, if you assign 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 stop a piece of code from generating a best practice 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:

- **Dangerous API exceptions**  
  When exceptions exist that are impossible to detect automatically, examine each error to ensure the correct implementation. Dangerous application programming interfaces (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 can use some so-called dangerous APIs when you take certain precautions, such as using code access security (CAS). You can suppress the error after you apply the appropriate mitigations.

- **False positives**  
  About 5 percent of all warnings are false positives and can be suppressed. Note that only warnings caused by actual code can be suppressed this way, 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 in the Best Practices list in the Compiler Output dialog box.

Some of the metadata best practice violations can also be suppressed, but the process of suppressing them is different. Instead of adding a comment to the source code, you add the violation to a global list of ignored violations. This list is maintained in the macro named `SysBPCheckIgnore`. This allows for central review of the number of suppressions, which should be kept to a minimum. For more information, see “Best Practice Checks” at [http://msdn.microsoft.com/en-us/library/aa874347.aspx](http://msdn.microsoft.com/en-us/library/aa874347.aspx).

Adding custom rules

You can use the Best Practices tool to create your own set of rules. The classes used to check for rules are named `SysBPCheck<Element type>`. 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:

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

    for (i=scanner.lines(); i>0; i--)
    {
        line = scanner.sourceLine(i);
        lineLen = strLen(line);
        for (j=conLen(devNames); j>0; 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 2-5.

**TABLE 2-5** Best practice errors in `checkUseOfNames`.

<table>
<thead>
<tr>
<th>Message</th>
<th>Line</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Don’t use your name!</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>Don’t use your name!</td>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>Don’t use your name!</td>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>Variable k not used</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>Method contains text constant: 'Don’t use your name!'</td>
<td>20</td>
<td>59</td>
</tr>
</tbody>
</table>

In an actual implementation, names of developers would probably be read from a file. Ensure that you cache the names to prevent the compiler from going to the disk to read the names for each method being compiled.

---

**Note** The best practice check just shown also identified that the code contained a variable named `k` that was declared, but never referenced. This is one of the valuable checks that ensures that the code can easily be kept up to date, which helps avoid mistakes. In this case, `k` was not intended for a specific purpose and can be removed.
Like most development environments, MorphX features a debugger. The debugger is a stand-alone application, not part of the AX 2012 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 following AX 2012 components:

- AX 2012 client
- AOS
- Business Connector (BC.NET)

For other debugging scenarios, such as web services, Microsoft SQL Server Reporting Services (SSRS) reports, and Enterprise Portal web client, see Chapter 3, “AX 2012 and .NET.”

Enabling debugging

For the debugger to start, a breakpoint must be hit when X++ code is executed. You set breakpoints by using the X++ code editor in the Development Workspace. The debugger starts automatically when any component hits a breakpoint.

You must enable debugging for each component as follows:

- In the Development Workspace, on the Tools menu, click Options > Development > Debug, and then select When Breakpoint in the Debug Mode list.
- From the AOS, open the Microsoft Dynamics AX Server Configuration Utility under Start > Administrative Tools > Microsoft Dynamics AX 2012 Server Configuration. Create a new configuration, if necessary, and then select the check box Enable Breakpoints to debug X++ code running on this server.
- For Enterprise Portal code that uses the BCPROXY context to run interpreted X++ code, in the Microsoft Dynamics AX Server Configuration Utility, create a new configuration, if necessary, and select the check box Enable Global Breakpoints.

Ensure that you are a member of the local Windows security group named Microsoft Dynamics AX Debugging Users. This is normally ensured by using setup, but if you did not set up AX 2012 by using your current account, you need to do this manually through Edit Local Users And Groups in Windows Control Panel. This is necessary to prohibit unauthorized debugging, which could expose sensitive data, provide a security risk, or impose unplanned service disruptions.

**Caution** It is recommended 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 the client will stop responding to users. Running the application with debug support enabled also noticeably affects performance.
To set or remove breakpoints, press 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.

To enable or disable a breakpoint, press Ctrl+F9. For a list of all breakpoints, press Shift+F9.

Breakpoints are persisted in the SysBreakpoints and SysBreakpointLists database tables. Each developer has his or her own set of breakpoints. This means that your breakpoints are not cleared when you close AX 2012 and that other AX 2012 components can access them and break where you want them to.

**Debugger user 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 inspecting variables and other aspects of the code. Figure 2-17 shows the debugger opened to a breakpoint with all the windows enabled.

![Debugger with all windows enabled.](image)
Table 2-6 describes the debugger’s various windows and some of its other features.

**TABLE 2-6  Debugger user interface elements.**

<table>
<thead>
<tr>
<th>Debugger element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code window</td>
<td>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.</td>
</tr>
<tr>
<td>Variables window</td>
<td>Shows local, global, and member variables, along with their names, values, and types. Local variables are variables in scope at the current execution point. Global variables are variables on global classes that are always instantiated: Appl, Infolog, ClassFactory, and VersionControl. Member variables are shown on classes. If a variable is changed as you step through execution, it is marked in red. Each variable is associated with a client or server icon. You can modify the value of a variable by double-clicking the value.</td>
</tr>
<tr>
<td>Call Stack window</td>
<td>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.</td>
</tr>
<tr>
<td>Watch window</td>
<td>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. You can use this window to inspect variables without the scope limitations of the Variables window. You can drag a variable here from the Code window or the Variables window.</td>
</tr>
<tr>
<td>Breakpoints window</td>
<td>Lists all your breakpoints. You can delete, enable, and disable the breakpoints through this window.</td>
</tr>
</tbody>
</table>
| Output window      | Shows the traces that are enabled and the output that is sent to the Infolog application framework, which is introduced in Chapter 5, “Designing the user experience.” The Output window has the following views:  
  - **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 form appear on these pages. |
| Status bar window  | Provides 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 this level reaches zero, the transaction is committed. |

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

Table 2-7 lists the most important shortcut keys available in the debugger.

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

Reverse Engineering tool

You can generate Visio models from existing metadata. Considering the amount of metadata available in AX 2012 (more than 50,000 elements and more than 18 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 Reverse Engineering tool is a great aid when you need to visualize metadata.

Note You must have Visio 2007 or later installed to use the 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, in the Projects window, right-click a project or a perspective, and point to Add-Ins > Reverse Engineer. You can also open the tool by selecting Reverse Engineer from the Tools menu. In the dialog box shown in Figure 2-18, you must specify a file name and model type.
When you click OK, the tool uses the metadata for all elements in the project to generate a Visio document that opens automatically. 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 data model contains a class for each table and view in the project and the class’s attributes and associations.

The UML data 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 AX 2012 are generated as UML attributes. All attributes are marked as public to reflect the nature of fields in AX 2012. Each attribute also shows the type. The primary key field is underlined. If a field is a part of one or more indexes, the field name is prefixed with the names of the indexes; if the index is unique, the index name is noted in braces.

Relationships in AX 2012 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 a UML aggregation, represented 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 lifespan of two or more tables and is represented by a black diamond.
Figure 2-19 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.

The name of an association endpoint is the name of the relationship. The names and types of all fields in the relationship appear in braces.

**UML object model**

When generating an object model, the Reverse Engineering tool looks for Microsoft Dynamics AX classes, tables, and interfaces in the project. The UML model contains a class for each Microsoft Dynamics AX table and class in the project and an interface for each Microsoft Dynamics AX interface in the project. The UML model also contains attributes and operations, including return types, parameters, and the types of the parameters. Figure 2-20 shows an object model of the most important RunBase and Batch classes and interfaces in Microsoft Dynamics AX. To simplify the view, some attributes and operations have been removed and operation parameters are suppressed.

The UML object model also contains referenced classes, 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 AX 2012 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 2-20** 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 are placed into the stack when in debug mode. To enable debug mode, on the Development tab of the Options form, select 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 the fields in each table. Figure 2-21 shows an entity relationship diagram (ERD) for the tables HcmBenefit (Benefit), HcmBenefitOption (Benefit option), HcmBenefitType (Benefit type), and HcmBenefitPlan (Benefit plan).

Note  For AX 2012 R2, Microsoft has introduced a website that hosts ERDs for the core tables in AX 2012 R2 application modules. You can use the site to quickly get detailed information about a large number of tables. To access the site, go to http://www.microsoft.com/dynamics/ax/erd/ax2012r2/Default.htm.

FIGURE 2-21 ERD using Crow’s Foot notation.

Fields in AX 2012 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 AX 2012 are generated as entity relationships. The `EntityRelationshipRole` property of the relationship 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, do the following: In Visio, create a new database model diagram, and then on the `Database` tab, point to Import > Import ERwin ERX File. Afterward, you can drag relevant tables from the Tables And Views pane (available from the Database tab) to the diagram canvas.

**Table Browser tool**

The Table Browser tool is a small, helpful tool that can be used in numerous scenarios. You can browse and maintain the records in a table without having to build a 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.

To access the Table Browser tool, right-click any of the following types of items in the AOT, and then point to Add-Ins > Table Browser:

- Tables
- Tables listed as data sources in forms, queries, and data sets
- 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 2-22 shows the Table Browser tool when it is started from the `CustTrans` table. In addition to the querying, sorting, and filtering capabilities provided by the grid control, you can type an SQL `SELECT` statement directly into the form by using X++ `SELECT` statement syntax and see a visual display of the result set. This tool is a great way to test complex `SELECT` statements. It fully supports grouping, sorting, aggregation, and field lists.
FIGURE 2-22 The Table Browser tool showing the contents of the CustTrans table demo data.

You can also 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 standard 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, and the size of AX 2012 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 2-23, 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.
On the Name & Location tab, you define what you’re searching for and where to look:

- In the Search list, the options are Methods and All Nodes. If you choose All Nodes, the Properties tab appears.
- The Named 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.
- If 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 by using the same criterion. You can disable this behavior by clearing the Use Selection check box.

On 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.

On the Filter tab, shown in Figure 2-24, you can write a more complex query by using X++ and type libraries. The code in the Source text box is the body of a method with the following profile:

```java
boolean FilterMethod(str _treeNodeName,
                     str _treeNodeSource,
                     XRefPath _path,
                     ClassRunMode _runMode)
```
The example in Figure 2-24 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.

![Figure 2-24 Filtering in the Find tool.](image)

The Properties tab appears when All Nodes is selected in the Search list. 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 2-25. The search begins when you click Find Now. The results appear at the bottom of the dialog box as they are found.

![Figure 2-25 Search results in the Find tool.](image)

Double-clicking any line in the result set opens the X++ code editor and sets the focus on the code example that matches. 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. AX 2012 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

To open the Compare tool, right-click an element, and then click Compare. A dialog box opens where you can select the versions of the element you want to compare, as shown in Figure 2-26.

![Comparison dialog box](image)

FIGURE 2-26 The 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, FPK, FPP, SLN, SLP, ISV, ISP, VAR, VAP, CUS, CUP, USR, and USP.

■ **Old layered version types (old SYS, old SYP, and so on)** If a baseline model store is present, elements from the files are available here. This allows you to compare an older version of an element with its latest version. For more information about layers and the baseline model store, see Chapter 21, “Application models.”

■ **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 will be overwritten 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.

- **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 classes. A class is conflicting if 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 with the publisher’s changes to the class. MorphX requires access to all three versions of the class—the original version in the baseline model store, a version with your changes in the current layer in the baseline model store, 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. Conflict resolution is shown in Figure 2-27.

![FIGURE 2-27 How the upgraded version proposal is created.](image)

**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 2-28 shows the Advanced tab, on which you can specify comparison options.

![Comparison options on the Advanced tab.](image)

**Figure 2-28** Comparison options on the Advanced tab.

The following list describes the comparison options shown in Figure 2-28:

- **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 during the comparison. 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. This option is cleared by default.

- **Show Line Numbers**  The Compare tool can add line numbers to all X++ code that is displayed. This option is cleared 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, start the comparison by clicking Compare. Results are displayed in a three-pane dialog box, as shown in Figure 2-29. The top pane contains the elements and options that you selected, the left pane displays a tree structure resembling the AOT, and the right pane shows details that correspond to the item selected in the tree.

Color-coded 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 particular version. 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 sign (≠) 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 of any element, including elements in old layers.

Details about the differences are shown in the right pane. Color coding is also used in this pane to highlight differences the same way that it is in the tree structure. If an element is editable, small action icons appear. These icons allow you to make changes to code, 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. If a version control system is in use, an element is editable if it is from the current layer and is checked out.
**Compare APIs**

Although AX 2012 provides comparison functionality for development purposes only, you can reuse the comparison functionality for other tasks. You can use the available APIs 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 present 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 SysCompareContextProvider interface. It provides the context for comparison. For example, it creates a Tutorial_Comparable object for each customer, sets the default comparison options, and handles context menus.

Figure 2-30 shows a comparison of two customers, the result of running the tutorial.

![Figure 2-30](image)

**FIGURE 2-30** The result of comparing two customers by 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.

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

## Cross-Reference tool

The concept of cross-references in AX 2012 is simple. If an element uses another element, the reference is recorded. With cross-references, you can determine which elements a particular element uses and which elements other elements are using. AX 2012 provides the Cross-Reference tool for accessing and managing cross-reference information.

Here are a couple of typical scenarios for using the Cross-Reference tool:

- 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.

You must update the Cross-Reference tool regularly to ensure accuracy. The update typically takes several hours. The footprint in a database is about 1.5 gigabytes (GB) for a standard application.

To update the Cross-Reference tool, on the Tools menu, point to > 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 its information to be reliable. 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 slows down compilation, however. Another option is to update cross-references manually for the elements in a project. To do so, right-click the project and point to Add-Ins > Cross-Reference > Update.
In addition to the main cross-reference information, two smaller cross-reference subsystems exist:

- **Data model** Stores information about relationships between tables. It is primarily used by the query form and the Reverse Engineering tool.

- **Type hierarchy** Stores information about class and data type inheritance.

For more information about these subsystems and the tools that rely on them, see the AX 2012 SDK (http://msdn.microsoft.com/en-us/library/aa496079.aspx).

The information that the Cross-Reference tool collects is quite comprehensive. You can find a complete list of cross-referenced elements by opening the AOT, expanding the System Documentation node, and clicking Enums and then xRefKind. When the Cross-Reference tool is updating, it scans all metadata and X++ code for references to elements of the kinds listed in the xRefKind subnode.

**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 `<Element type>Str` or `<Element type>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. Here is an example:

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

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

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

For more information about intrinsic functions, see Chapter 20, “Reflection.”

To access usage information, right-click any element in the AOT and point to Add-Ins > Cross-Reference > Used By. If the option isn’t available, either the element isn’t used or the cross-reference hasn’t been updated.
Figure 2-31 shows where the prompt method is used on the RunBaseBatch class.

![FIGURE 2-31 The Cross-Reference tool, showing where RunBaseBatch.prompt is used.](image)

When you view cross-references for a class method, the Application hierarchy tree is visible, so that you can see whether the same method is used on a parent or subclass. For types that don’t support inheritance, the Application hierarchy tree is hidden.

**Version control**

The Version Control tool in MorphX makes it possible to use a version control system, such as Microsoft Visual SourceSafe or Visual Studio Team Foundation Server (TFS), to keep track of changes to elements in the AOT. The tool is accessible from several places: from the Version Control menu in the Development Workspace, from toolbars in the AOT and the 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 the change history and retrieve old versions of an element.

- **Code quality enforcement** The implementation of version control in AX 2012 enables a fully configurable quality standard for all check-ins. With the quality standard, all changes are verified according to coding practices. If a change doesn’t meet the criteria, it is rejected.

- **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. A developer can rewrite fundamental areas of the system without causing instability issues for others. Developers are also unaffected by any downtime of a centralized development server.
Even though using a version control system is optional, it is strongly recommended that you consider one for any development project. AX 2012 supports three version control systems: Visual SourceSafe 6.0 and TFS, which are designed for large development projects, and MorphX VCS. MorphX Version Control System (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 process. Table 2-8 shows a side-by-side comparison of the version control system options.

**TABLE 2-8 Overview of version control systems.**

<table>
<thead>
<tr>
<th>Requirements and features</th>
<th>No version control system</th>
<th>MorphX VCS</th>
<th>Visual SourceSafe</th>
<th>TFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Object Servers required</td>
<td>1</td>
<td>1</td>
<td>1 for each developer</td>
<td>1 for each developer</td>
</tr>
<tr>
<td>Database servers required</td>
<td>1</td>
<td>1</td>
<td>1 for each developer</td>
<td>1 for each developer</td>
</tr>
<tr>
<td>Build process required</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Master file</td>
<td>Model store</td>
<td>Model store</td>
<td>XPOs</td>
<td>XPOs</td>
</tr>
<tr>
<td>Isolated development</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple checkout</td>
<td>N/A</td>
<td>No</td>
<td>Configurable</td>
<td>Configurable</td>
</tr>
<tr>
<td>Change description</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Change history</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Change list support (atomic check-in of a set of files)</td>
<td>N/A</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Code quality enforcement</td>
<td>No</td>
<td>Configurable</td>
<td>Configurable</td>
<td>Configurable</td>
</tr>
</tbody>
</table>

The elements persisted on the version control server are file representations of the elements in the AOT. The file format used is the standard Microsoft Dynamics AX export format (.xpo). Each .xpo file contains only one root 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 a 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 has some limitations. For example, MorphX VCS does not provide any tools for maintenance, such as making backups, archiving, or labeling.

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

Figure 2-32 shows a typical deployment using Visual SourceSafe or TFS, 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.
FIGURE 2-32 Typical deployment using version control.

**Note** In earlier versions of Microsoft Dynamics AX, a Team Server was required to assign unique IDs as elements were created. AX 2012 uses a new ID allocation scheme, which eliminates the need for the Team Server. For more information about element IDs, see Chapter 21.

**Element life cycle**

Figure 2-33 shows the element life cycle in a version control system. When an element is in a state marked with a lighter shade, it can be edited; otherwise, it is read-only.

You can create an element in two ways:

- Create a new element.
- Customize an existing element, resulting in an *overlayered* version of the element. Because elements are stored for each 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.
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.

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.

Source code casing
You can set the Source Code Titlecase Update tool, available on the Add-Ins submenu, to execute automatically 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.
Common version control tasks

Table 2-9 describes some of the tasks that are typically performed with a version control system. Later sections describe additional tasks that you can perform when using version control with AX 2012.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check out an element</td>
<td>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. To see which elements you have currently checked out, on the Microsoft Dynamics AX menu, click Control &gt; Pending Objects. 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.</td>
</tr>
<tr>
<td>Undo a checkout</td>
<td>If you decide that you don’t want to modify an element that you checked out, you can undo the checkout. This releases your lock on the element and imports the most recent checked-in revision of the element to undo your changes.</td>
</tr>
</tbody>
</table>
| Check in an element  | 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 later in Figure 2-34 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. The following procedure is recommended for checking in your work:  
  ■ Perform synchronization to update all elements in your environment to the latest version.  
  ■ Verify that everything is still working as intended. Compilation is not enough.  
  ■ Check in the elements. |
| Create an element    | When using version control, you create new elements just as you normally would in a 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 (on the Development Workspace Version Control menu, point to Control > Setup > System Settings). By default, jobs and private projects are not accepted. New elements should follow Microsoft 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. |
| Rename an element    | An element must be checked in to be renamed. Because all references between .xpo files are strictly name-based, all references to renamed elements must be updated. For example, if 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, and thus they are not affected when an element is renamed; in most cases, it is enough to check out the form or report and include it in the check-in to update the .xpo file. You can use the Cross-Reference tool 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 an element is renamed. |
| Delete an element    | You delete an element as you normally would in Microsoft 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. |
| Get the latest version of an element | If someone else has checked in a new version of an element, you can use the Get Latest option on the context menu to get the version of the element that was checked in most recently. This option isn’t available if you have the element checked out yourself. |
Working with labels

Working with labels is 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 checkouts are allowed for label files. This means that others can change labels while you have a label file checked out.

Figure 2-34 shows the Check In dialog box.

![Check In dialog box](image)

**FIGURE 2-34** The Check In dialog box.

If you create a new label when using version control, a temporary label ID is assigned (for example, @$AA0007 as opposed to @USR1921). When you check in a label file, your changes are automatically merged into the latest version of the file and the temporary label IDs are updated. All references in the code are automatically updated to the newly assigned label IDs. Temporary IDs eliminate the need for a central Team Server, which was required for AX 2009, because IDs no longer have to be assigned when the labels are created. If you modify or delete a label that another person has also modified or deleted, your conflicting changes are abandoned. Such lost changes are shown in the Infolog after the check-in completes.

Synchronizing elements

Synchronization makes it possible for 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 Development Workspace. On the Version Control menu, point to Periodic > Synchronize.
Synchronization is divided into three operations that happen automatically in the following sequence:

1. The latest files are copied from the version control server to the local disk.
2. The files are imported into the AOT.
3. The imported files are compiled.

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 2-35 shows the Synchronization dialog box.

![Synchronization dialog box](image)

**FIGURE 2-35** The Synchronization dialog box.

Selecting the Force check box gets the latest version of all files, even if they haven’t 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 available with MorphX VCS.

**Viewing the synchronization log**

The way that you keep track of versions on the client depends on your version control system. Visual SourceSafe requires that AX 2012 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 AX 2012 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 AX 2012.

When synchronization fails, the import operation is usually the cause of the problem. Synchronization failure leaves your system in a partially synchronized state. To complete the synchronization, restart AX 2012 and restart the import. You use the synchronization log to restart the import, and you access it from the Development Workspace menu at Version Control > Inquiries > Synchronization log.
The Synchronization Log dialog box, shown in Figure 2-36, displays each batch of files, and you can restart the import operation by clicking Process. If the Completed check box is not selected, the import has failed and should be restarted.

The Synchronization log is not available with MorphX VCS.

**FIGURE 2-36** The Synchronization Log dialog box.

**Showing the history of an element**

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 2-37.

**FIGURE 2-37** Revision history of an element.
For each revision, 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 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, where you can 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 you can use the standard MorphX toolbox. Clicking View File opens the.xpo file for the selected version in Notepad.

Comparing revisions

Comparison is the key to harvesting the benefits of a version control system. You can start a comparison from several places, including from the context menu of an element, by pointing to Compare. Figure 2-38 shows the Comparison dialog box, where two revisions of the form CustTable are selected.

![Comparison dialog box]

**FIGURE 2-38** Comparing element revisions from version control.

The Compare dialog box contains a list of all checked-in versions, in addition to the element versions available in other layers that are installed.

Viewing 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 2-39, 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.
FIGURE 2-39 Pending elements.

You can access the Pending Objects dialog box from the Development Workspace menu: Version Control > Pending Objects.

Creating a build

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

1. Use the CombineXPOs command-line utility to combine all .xpo files into one. This step makes the .xpo file consumable by AX 2012. AX 2012 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= <FileName.xpo>-MODEL=<Model Name> to Ax32.exe. This step imports the .xpo file and compiles everything. After this step is complete, the new model is ready in the model store.

3. Export the model to a file by using the axutil command-line utility: axutil export /model:<model name> /file:<model file name>.

4. Follow these steps for each layer and each model that you build.

The build process doesn’t apply to MorphX VCS.
Integrating AX 2012 with other version control systems

The implementation of the version control system in AX 2012 is fully pluggable. This means that any version control system can be integrated with AX 2012.

Integrating with another version control system requires a new class implementing the `SysVersion-ControlFileBasedBackEnd` interface. It is the implementation's responsibility to provide the communication with the version control system server being used.
This page intentionally left blank
CHAPTER 22

Developing mobile apps for AX 2012

In this chapter

Introduction ............................................. 745
The mobile app landscape and AX 2012 ............ 746
Mobile architecture ...................................... 746
Developing a mobile app ............................... 752
Architectural variations ................................. 758
Resources .................................................. 759

Introduction

Device form factors and their associated interfaces are changing rapidly. At the same time, AX 2012 customers have an increasing need to interact with AX 2012 across a broad set of devices. In addition, new technologies are facilitating new scenarios, in which apps that provide personalized information in context are becoming more important for improving employees’ productivity. Microsoft is investing in these new scenarios that let customers work with devices in new ways, and Microsoft Dynamics is taking full advantage of this work.

Microsoft Dynamics is creating new mobile device experiences for AX 2012 customers through apps. These apps are designed to help a broad range of employees improve their efficiency while on the go and stay connected to their business processes. Scenarios include native experiences for both smartphones and tablets. In addition to developing its own apps, Microsoft Dynamics is committed to helping developers build their own immersive device experiences for AX 2012.

These apps can interact directly with AX 2012 by using the existing AX 2012 services framework. This approach also uses existing AX 2012 user accounts so that no additional provisioning is required. For more information about AX 2012 services, see Chapter 12, “AX 2012 services and integration.”

This chapter begins by providing an overview of the current mobile app landscape and how it relates to AX 2012. It then describes the AX 2012 mobile architecture and the technical and user interface considerations for developing mobile apps. For more detailed instructions about developing mobile apps for AX 2012, see the sources listed in the “Resources” section at the end of the chapter.
The mobile app landscape and AX 2012

With AX 2012 R3 and its associated investments in mobile solutions, the Microsoft Dynamics AX team expects that the Microsoft Dynamics AX ecosystem will see an increasing investment in mobile apps. There is an early-mover advantage for developers who identify and take advantage of these opportunities, and AX 2012 customers who equip their people with powerful mobile apps will have an advantage in their respective markets. The mobile architecture discussed throughout this chapter has been proven by mobile apps that are already available in the marketplace and by customers who are deploying those apps.

Creating mobile apps requires a different set of skills than developing for AX 2012, and customers and partners might find it advantageous to pair an AX 2012 developer with a mobile app developer. This way, necessary AX 2012 services can be created by experienced AX 2012 developers. Except for needing to understand the content of the services, typical mobile app developers can develop AX 2012 mobile apps without having to understand the intricate details of AX 2012. Thus, partners and customers can take advantage of the vast pool of mobile app developers.

Mobile architecture

The AX 2012 mobile architecture is designed to help developers overcome the following challenges that are inherent in developing mobile apps:

- How to facilitate communication between apps and on-premises installations
- How to authenticate users
- How to develop for multiple platforms

Because AX 2012 most often runs on-premises or is privately hosted, the average phone or tablet device does not have access to AX 2012 services, which are usually available only through a corporate network. To overcome this challenge, the mobile architecture uses the Microsoft Azure Service Bus Relay. The Service Bus Relay solves the challenges of communicating between on-premises applications and the outside world by allowing on-premises web services to project public endpoints. Systems can then access these web services, which continue to run on-premises, from anywhere.

The Service Bus Relay is based on a namespace that is set up for each company that deploys mobile apps for AX 2012. For example, an administrator from the Contoso company can create an Azure subscription and set up a unique namespace, such as contosomobapps. The namespace is a friendly identifier that end users enter into the mobile app to connect to their company's AX 2012 instance.

The second challenge is authenticating users. AX 2012 has existing user accounts that are based on Active Directory accounts. Ideally, device users simply use their corporate identities (user IDs and
passwords) to access AX 2012 from their devices. The mobile architecture accomplishes this by using Active Directory Federation Services (AD FS). AD FS is a Windows Server component that is used by users whose devices are not on the corporate network to authenticate themselves by using their existing corporate credentials.

Another challenge that mobile app developers face is providing apps for various device platforms, such as Windows 8.1, Windows Phone 8, iOS, and Android. By taking advantage of the mobile architecture, developers can create applications for the platform they choose, depending on customer requirements.

Note In some scenarios, you might want to use variations on the architecture described in this section. For more information, see the “Architectural variations” section later in this chapter.

Mobile architecture components

The AX 2012 mobile architecture includes the following key components:

- **AX 2012 services** Apps communicate with AX 2012 through the AX 2012 services framework.
- **Active Directory** Users who access AX 2012 through mobile apps are authenticated against their existing corporate identities.
- **AD FS** AD FS facilitates authentication of Active Directory–based users who are accessing AX 2012 through mobile apps.
- **Service Bus Relay** The Service Bus Relay ferries messages from the mobile app to an on-premises listener through which the app can reach an on-premises instance of AX 2012.
- **On-premises listener** The listener listens for messages from mobile apps that are relayed through the Service Bus Relay. The listener then makes calls to AX 2012 and responds to the mobile app through the Service Bus Relay.
- **Microsoft Azure Active Directory Access Control** Access Control verifies that the user has been authenticated through AD FS and then allows messages to be sent through the Service Bus Relay.
- **Mobile apps** These apps can be for a phone or tablet on any of the prevailing mobile platforms—Windows, iOS, or Android. The architecture provides the means for these apps to communicate with AX 2012. The apps can then interact with AX 2012 as appropriate, based on the business scenario. Apps can also be designed to communicate with other systems, depending on the needs of the business scenario.
Figure 22-1 shows the components of the mobile architecture.

To become familiar with the mobile architecture, we recommend that you install and configure one or more of the AX 2012 mobile apps and the Microsoft Dynamics AX connector (on-premises listener) for mobile apps. Doing so will help you learn the configuration steps necessary for Azure, AD FS, AX 2012, and the listener. That familiarity will help you greatly in debugging and troubleshooting your apps. For more information about these components and their configuration, see the following white papers:


- Configure Microsoft Dynamics AX Connector for Mobile Applications at https://mbs.microsoft.com/downloads/customer/AX/ConfigureAXConnectorforMobileApplications.pdf. To access this paper, you must have CustomerSource or PartnerSource credentials.

You can download the AX 2012 expense app, which is used as an example throughout this chapter, at http://apps.microsoft.com/windows/en-us/app/dynamics-ax-2012-expenses/07aab6f9-c6ce-4b81-b04c-4b43c3f6de67.
Message flow and authentication

The following process outlines the flow of messages and describes how users are authenticated:

1. The user submits credentials to obtain a token from AD FS.

2. If the user’s credentials are verified, a token is returned that is based on Security Assertion Markup Language (SAML). The token contains a claim that indicates the user name and company domain, such as user@contoso.com. The claim is used to verify the user’s company affiliation and identity within the company.

3. The mobile app then presents the claim in the SAML token to Access Control. Access Control controls access to the Service Bus Relay; any app that calls the Service Bus Relay must have a token from Access Control.

4. Access Control validates the SAML token based on the previously established trust relationship between Access Control and AD FS for the Service Bus Relay namespace and company domain. Based on the claim in the SAML token, Access Control verifies that the user is from the company associated with the Service Bus Relay namespace. Access Control then provides a second token to the mobile app in Simple Web Token (SWT) format. The SWT token is then included in the request to the Service Bus Relay to send a message to AX 2012. The SAML token is also included and is used by the on-premises listener to authenticate the user.

5. The mobile app then sends the message (with associated tokens) through the Service Bus Relay to the on-premises listener. The Service Bus Relay ferries the message to the listener.

The listener must also be authenticated to listen to the Service Bus Relay. This authentication is based on a Service Bus Relay namespace credential that is stored on the listener. The listener presents this credential as it starts to listen for messages coming from the namespace.

6. After a message is received from the Service Bus Relay, the listener validates the SAML token. The validation is based on a previously established trust relationship between the listener and AD FS. After verifying the token, the listener uses the claim to determine which AX 2012 user is using the mobile app making the call to AX 2012. The listener calls AX 2012 on behalf of the user identified in the claim.

7. The listener receives a response from AX 2012 and then returns the response message through the Service Bus Relay.

8. The Service Bus Relay sends the response message to the mobile app.
Figure 22-2 illustrates the message flow and authentication process. The numbering in the figure roughly corresponds to the numbers in the list.

**Using AX 2012 services for mobile clients**

To develop mobile apps, you must have an interface with AX 2012. The ideal approach is to use AX 2012 services for communication between mobile apps and AX 2012. For more information about AX 2012 services, see Chapter 12.

The AX 2012 services framework provides multiple types of services—system services, custom services, and document services—each with its own programming model. Custom services are the most likely to provide the exchange of messages that are appropriate for a mobile app.

To design an AX 2012 service that can be consumed by your mobile app, you will need to do the following:

1. Create AX 2012 services and data contracts, and deploy a basic inbound port to expose the service operations for consumption.

2. Implement the service methods in X++ classes to perform data operations in AX 2012.
The mobile client must communicate with the AX 2012 service, which is hosted on an Application Object Server (AOS) instance that is deployed behind a corporate firewall. Instead of configuring changes in the firewall, such as exposing an Internet Information Services (IIS) server externally to expose the on-premises service, the app makes use of a secured Service Bus Relay.

AX 2012 services are typically accessed by using Simple Object Access Protocol (SOAP). This approach might work well for some mobile applications. However, the trend in developing web and mobile applications is to interact by using RESTful services. (Representational state transfer [REST] is an architecture that allows developers to write asynchronous code that connects with cloud-based services more easily.) For information about RESTful services, see the article, “An Introduction To RESTful Services With WCF,” at http://msdn.microsoft.com/en-us/magazine/dd315413.aspx.

Currently AX 2012 does not expose custom services by using a RESTful approach. One option is for the on-premises listener to translate RESTful calls from a mobile app to SOAP calls to AX 2012 (and vice versa). The Microsoft Dynamics AX product group used this approach to develop the Microsoft Dynamics AX Windows Store apps (expenses, timesheets, and approvals). For more information about the Microsoft Dynamics AX Windows Store apps, see “What’s new: Companion apps for Microsoft Dynamics AX 2012 R2” at http://technet.microsoft.com/en-us/library/dn527182.aspx.

Developing an on-premises listener

As described earlier, you’ll need an on-premises service that acts as an intermediary between the Service Bus Relay and AX 2012 services. One approach is to use Windows Communication Foundation (WCF) to build a simple listener.

Beyond the listener’s primary purpose of passing messages between the Service Bus Relay and AX 2012, the listener can provide additional features that facilitate the development of mobile applications. For example, the listener can provide configuration information to the app, such as whether certain features of the app are enabled or disabled. The listener can expose RESTful calls to the client. The Microsoft Dynamics AX product group used WCF features to translate between the SOAP and JavaScript Object Notation (JSON) protocols for messaging to the Windows Store apps. This is a standard feature of WCF. The listener can also capture meaningful information in the event log and provide telemetry information.

Another approach is to use the Azure Service Bus adapter built into AX 2012 starting with AX 2012 R2 cumulative update 6. With this approach, you configure IIS as the listener. Depending on the requirements of your mobile app, you can choose the best approach. For a comparison of these approaches, see “Microsoft Dynamics AX White Paper: Developing Mobile Apps” at http://www.microsoft.com/en-us/download/details.aspx?id=38413.
Developing a mobile app

When you are developing a mobile app, it is helpful to start by prototyping and developing based on simple scenarios. For example, one initial prototype app that the Microsoft Dynamics AX team developed simply updated data to AX 2012; it didn’t read data from AX 2012 or require authentication. Additional features were added as the app moved from prototype to production. Keeping apps as focused and as simple as possible is a good guideline for mobile apps in general.

Platform options and considerations

The mobile architecture described in this chapter is platform-independent and device-independent. You can use this architecture to develop applications across the spectrum of devices, including laptops, tablets, and phones. You can also use it across the landscape of development technologies, such as C# and Extensible Application Markup Language (XAML), HTML5 and JavaScript, Objective-C, and Java.

The underlying technologies used in the architecture, such as AD FS, the Service Bus Relay, and AX 2012 services, are commonly used in cross-platform development.

Developer documentation and tools

One of the goals of the mobile architecture is to solve the basic issues of interacting with AX 2012 so that mobile app developers can focus on their apps. With the problem of communicating with AX 2012 solved, developers can use the wealth of documentation and tools available to them. A large community of mobile app developers builds apps for AX 2012, without needing to understand the underlying AX 2012 framework. To the mobile app, calls to AX 2012 are simply web services calls, which is very common for mobile apps.

Microsoft offers an exhaustive set of samples, tutorials, and tools for developers who are creating apps for the Windows Store and for Windows Phone. As of this writing, the Windows 8.1 app sample pack includes more than 330 samples at http://code.msdn.microsoft.com/windowsapps/Windows-8-Modern-Style-App-Samples/view/SamplePack#content. The following are of particular of interest:


These types of samples can be relevant to developing mobile apps for AX 2012.

Third-party libraries

In addition to published samples, an extensive set of third-party libraries are available for developing mobile apps. Examples of functions provided by these libraries include data binding, storage, lookups, currency, controls (such as calendars), and date/time functions. If your organization allows the use of third-party libraries, using these libraries can help boost developer productivity.
Best practices
This section describes some valuable considerations to keep in mind when designing and developing mobile apps.

- **Take advantage of native device capabilities** Taking advantage of the capabilities of a device can greatly enhance mobile apps. For example, the device’s camera can be used to record documents or receipts. The camera can also be used to capture damage or repair situations (for construction, manufacturing, or field services) as part of an AX 2012 transaction. The geographic information from the device can be used to map a location or address known to AX 2012, such as a customer, vendor, or site.

- **Make use of data caching and local storage** Master and lookup data might need to be cached on the device as an alternative to requesting data from the service each time it is needed. For example, expense categories or currency types are relatively static and can be safely cached locally. The mobile app can refresh those on an infrequent basis. Some reference data can be cached but must be refreshed more frequently. For example, projects or customers can be stored locally but might need to be refreshed at app startup or on a regular basis.

  You can store data locally by using available technologies such as indexDB or SQLite. These technologies are supported across platforms and devices. You might need to use the RecVersion field to determine whether records are synchronized. For more information, see “Concurrency When Updating Data” at http://technet.microsoft.com/en-us/library/cc639058.aspx.

- **Consider bandwidth and connectivity in your design** Mobile devices typically have less bandwidth than desktop computers on corporate networks. You’ll need to make appropriate tradeoffs to ensure that your apps work on lower bandwidths. For example, a phone app might impose a size limit on a picture that is sent with a transaction because of bandwidth limitations.

  You should also consider intermittent connectivity. Mobile apps should be designed to expect interruptions in connectivity. This might mean saving appropriate state information and data locally, and using that stored data when connectivity is restored.

- **Allow for offline use** In some scenarios, it is beneficial for users to have some app functionality when the app is offline (not connected). Consider whether you can enable some of the app’s functionality offline. It is important to consider the level of business logic that should reside in the app for offline scenarios. It probably makes sense to enable offline features where extensive business logic (which is available in AX 2012) is not required. If you don’t enable offline features in such cases, your app will need to apply the business logic and have the user react as appropriate when it reconnects.

Key aspects of authentication
Authentication was discussed earlier in this chapter as part of the overall message flow. However, because authentication is a key and potentially complicated part of the process, this section focuses on some of its more complicated aspects.
As mentioned earlier, because the mobile app will likely run on a device that is not on a corporate network, users must be authenticated. The first step in the process is to acquire the users’ credentials (user names and passwords). We recommend that you store those credentials in the app by using a secure storage mechanism on the device. That way, the mobile app can silently authenticate users without prompting them for credentials each time the app runs.

Another complicated aspect of configuring authentication is setting up Secure Sockets Layer (SSL). AD FS uses SSL to ensure that the passing of credentials and tokens between the mobile app and AD FS is protected. The AD FS endpoint has a URL based on the company’s domain. For example, the AD FS endpoint for the Contoso company might end in `contoso.com`. To enable SSL, you must install an appropriate SSL certificate in AD FS. More precisely, you must set up SSL in IIS, which is used as the front end for AD FS.

Setting up SSL in IIS is a common practice because many websites require the use of SSL for interactions that require authentication or share protected information. The SSL certificate must be issued by a recognized certificate authority (CA). In issuing an SSL certificate, the CA verifies that the entity requesting the certificate is associated with the domain for which the certificate is being issued. For example, the administrator for Contoso must prove to the CA that she is associated with the `contoso.com` domain. This is typically done by the CA making contact by using the contact information (phone number and email address) associated with the contoso.com domain in the Domain Name System (DNS) records. Whether you are setting up AD FS for demonstration, testing, or production purposes, it is necessary to have a CA-issued SSL certificate for the domain used by AD FS. For more information, see “Obtain an SSL Certificate” at http://msdn.microsoft.com/en-us/library/gg981937.aspx.


**User experience**

Developing mobile apps for AX 2012 creates an opportunity for developers to reimagine user experiences. Across the landscape of devices, mobile apps are generating a wave of exciting new user experiences. We encourage you to be inspired by these immersive experiences when envisioning mobile apps for AX 2012. The expense app created by the Microsoft Dynamics AX product group provides some vivid examples.

The following illustrations show examples of the existing AX 2012 experience compared with the new mobile app experience in the expense app. The top half of Figure 22-3 shows the expense report experience in the AX 2012 Employee Services portal. The bottom half of Figure 23-3 shows the corresponding experience in the AX 2012 expense mobile app. Both user interfaces display views of a list of...
expense reports. Note that the mobile app takes advantage of cards to represent expense reports and uses color to represent status. The app is designed to be touch-friendly and hides commands until needed. However, the mobile app also works well with a mouse and keyboard.

FIGURE 22-3 Screenshots of expense reports in the AX 2012 Employee Services portal and in the expense mobile app.
Figure 22-4 shows the edit experience within an expense report in the AX 2012 Employee Services portal and in the expense app. Again, the mobile app uses cards to represent expenses, and color and icons to show categories. The mobile app also uses a calendar to show the expenses during the week based on the transaction date.

![Expense report interface]

**FIGURE 22-4** Screenshots of the edit expense reports experience in the AX 2012 Employee Services portal and in the expense mobile app.

Globalization and localization

Mobile apps should support globalization and localization, as described here:

- Globalization entails using the correct formats for numbers, dates, times, addresses, and phone numbers for different locales. For the AX 2012 apps developed by Microsoft, globalization is based on the country or regional settings of the device.

- Localization entails displaying the user interface in the language of the user. For the AX 2012 apps developed by Microsoft, the app language is based on the user’s language in AX 2012.

We recommend that you design apps so that resources, such as strings and images, are separate from code. This enables them to be independently localized into different languages.

App monitoring

Having information about the usage of your mobile apps is generally valuable, so we suggest that you develop your apps to capture and save monitoring information. This approach is also referred to as telemetry or instrumentation. Windows Store apps and Windows Phone apps inherently capture some information, including downloads, basic usage, and errors. In addition, it is useful to capture more detailed information about how the apps are used. Application insights help you find out what users are doing with the app and help you diagnose performance or exceptions with the app. Application Insights for Microsoft Visual Studio Online is a cloud-based service designed for monitoring mobile or web apps. For more information, see http://msdn.microsoft.com/en-us/vstudio/dn481095(v=vs.98).aspx.

Web traffic debugging

The mobile app architecture relies heavily on calls to web services. These include calls to authenticate the user of the app and calls to AX 2012 (through the Service Bus Relay). It is very useful to capture and analyze the content of these calls. The Fiddler web debug proxy is an excellent tool for viewing and analyzing the web services calls. You can configure Fiddler to work with mobile devices.

The following blog posts describe how to use Fiddler to develop apps for Windows 8 phones:


And the following blog post describes how to use Fiddler with iPhone5:

Architectural variations

So far, this chapter has focused primarily on a specific mobile architecture that solves the key challenges of allowing remote mobile apps to work with an on-premises deployment of AX 2012. This architecture is also optimized for apps that are distributed by app stores and installed on local devices. However, certain variations of this architecture might be appropriate in specific scenarios.

On-corpnet apps

In some scenarios, mobile apps are used exclusively within a corporate network environment. An example is a warehouse app that has corporate network connectivity throughout the facility. In such a case, the user is already authenticated when accessing the corporate network, so further authentication isn’t necessary. Also, the app can access AX 2012 through the corporate network, so the Service Bus Relay might not be necessary.

Web apps

Another approach to mobile apps is the use of web apps targeted for mobile devices. In this case, the app is available through the web browser on the device. This approach has some tradeoffs. Some scenarios might be most appropriate for web apps, and other scenarios might be most appropriate for mobile apps that are installed on the device. Many popular websites today also deliver mobile apps that users install, even though the experience is also available through browsers.

Web apps have the following advantages:

- They work across devices, although they need to be tested and refined to handle device-specific nuances.
- They are more adaptable to AX 2012 customizations and extensions. Because they are rendered from the AX 2012 server, the apps can include the customizations and extensions that have been applied to the server.
- They do not require users to install or configure the apps.

Web apps have the following disadvantages:

- They don’t provide the high-fidelity, immersive experiences available in modern mobile apps.
- They rely completely on the server to deliver the app and content. The server might not perform as well or might be less responsive than an app that is installed on the device.
- They typically don’t have local caching or offline capabilities.
- They probably can’t take advantage of device features such as a camera or GPS.

You can use these tradeoffs to decide which approach is most appropriate for your scenario.
Resources

This section contains a list of recommended sources of information to assist developers with troubleshooting and debugging.

- **CustomerSource and PartnerSource**  The “Mobile Apps for Dynamics AX” page on CustomerSource and PartnerSource has a wealth of information for deploying the existing apps or developing mobile apps:
  
  

  **Note**  You must register to be able to access these sites.


- **Support**  The normal Microsoft Dynamics AX support channels are available to support the mobile apps released by Microsoft for AX 2012. This includes support for the on-premises listener and its associated configuration.
This page intentionally left blank
This page intentionally left blank
Index

A
abstract tables, in inheritance hierarchy, 622
acceptance test driven development (ATDD), 557, 558, 566
access control
defined, 373
security roles, managing through, 375
access operators, X++ expressions, 113
accessing data with queries, 299
accounting
distribution process, 704
events, documenting, 702
requirements, deriving, 698
accounting framework, 698–702
action controls
Action pane strips, 188
Action panes, 188
buttons, 187
action icons, 60
action menu items, 264
Action pane
described, 188
designing for ease of use, 169
details forms, displaying in, 165
Enterprise Portal, using with, 170
organizing by activity, 163
strips, 188
web part in Enterprise Portal, 211
actions, 26
activating a workflow, 283–287
activity entity, 688–692
address bar, 155
address book framework, integrating, 675
ad hoc mode, using, 482, 483
ad hoc reports, 293
aggregates, 96
AJAX, 233
alert notifications, displaying on webpages, 213
ALM. See application life cycle management (ALM)
alternate key columns, 52
alternate keys, 614
analytic content
configuring, 325, 326
Role Centers, displaying in, 351
anonymous types, 90
anytype
reference type, 108
variable declaration, 111
AOS. See Application Object Server (AOS)
AOT. See Application Object Tree (AOT)
APIs
Batch, 652–654
document services, provided by, 412
model store, 740
one-way messages, using to send, 433
QueryFilter, 635–638
reflection, 711–724
securing, 392
application data element types, 9–18
application development environments, 6
Application Integration services, 8
application life cycle management (ALM)
benchmarking, 773
deploying customizations, 768
life cycle phases, 761
Lifecycle Services (LCS), 762–768
solution, tracking with, 556
application model elements, updating with MorphX, 19
Application Object Server (AOS)
client configuration, 486
configuration, 486, 655, 659
Help system, and, 573
improving performance of, 486
report execution sequence, in, 293
system services, 407, 408
Application Object Tree (AOT)
assemblies, referencing, 77
autorefresh, enforcing, 25
chart controls, managing, 304
dirty elements, 25
document set properties, setting, 597
elements, 20, 22, 24–26
elements, Enterprise Portal, 213
Application Object Tree (AOT) (continued)
  jobs, creating with, 106
  Label Files node, 34
  layers and models, 26
  manual resolution, 25
  modeling capabilities, 326
  navigating, 21
  opening, 21
  projects, maintaining in, 340
  publishing services, 421
  subnodes, changing the order, 24
  synchronizing elements, 25
application platform architecture
  data element types, 9
  development environments, 6
  layers, 5
  tiers, 7, 8
approvals workflow element, 264, 277
architecture
  application meta-model, 9
  application platform, 6
  client-side reporting solutions, 290, 291
  design principles, 3
  DIXF, components of, 772
  Enterprise Portal, described, 208
  layers, 4
  mobile, 746–751, 758
  security, in AX 2012, 372
  server-side reporting solutions, 292
area pages, 158–160
Arithmetic operators, expressions, 113
artifacts
  ALM solution, tracking with, 556
  custom services, 408
  document services, 412, 413
  security, validating, 384
  workflow, 278
ASP.NET controls
  Chart Control, 290
  User control web part, hosting with, 213
  user input, validating, 243
ASP.NET webpages, creating with AJAX, 233
assemblies
  coding against, in X++, 78
  hot swapping on servers, 87, 88
  references, adding, 77
  strong-named, using, 76, 77
  third-party, 76
assigning security roles, 376, 384
ATDD (acceptance test driven development), 557, 558, 566
attributes
  classes and methods, using with, 139
  creating, 552
  predefined, test, 550–552
product, 684
SysObsoleteAttribute, 139
SysOperation framework, 517
authentication
  described, 373
  mobile apps, 749, 754
Authenticate, signing models with, 734
authorization, 373, 391
auto design reports, 297, 298
Auto variables, 200
auto-inference, 378
automated decisions and tasks, in workflows, 265
autorefresh, 25
avg aggregate functions, in select statements, 119
AX 2012
  batch processing, 643, 644
  chart development tools, 303
  client, Help actions, 571
  consuming services, 422–432
  custom services, 408, 411
  data connection queries, 299
  designing new transaction details forms, 169
  Development Workspace, launching, 20
  Enterprise Search, 253
  Excel templates, building, 203
  extending transaction details forms, 169
  extensions, 300
  IDs, assigning, 66
  integrating with other systems, 74, 76–88
  labels, 34
  localization, allowing for, 36
  managed DLLs, referencing from, 77
  metadata layers, 727
  mobile architecture, 746–751
  mobile clients, services for, 750, 751
  MorphX development tools, 19
  print management, 661–669
  publishing services, 421
  reporting development tools, 296
  reporting framework architecture, 293
  security framework, 372–376
  send framework, 432–434
  services framework, 407–420, 750
  solution, as, 3
  SQL Server Power View, 353–358
  system services, 408
  Trustworthy Computing, 390, 393
  unit testing features, 550–553
  value type conversions, 128
  version control systems, 65
  Word templates, building, 204
AX 2012 R3
  extension methods, specific to, 98, 99
  LINQ, using with, 89
  Axd<Document> classes, 414
Ax<Table> classes, 415
AXUtil, models and, 731, 734
Azure demo environment topology, 766

B
backing entities, creating, 695, 696
base enumeration
  types, 9, 111
  values, adding, 677, 679
basic integration ports, 421
batch bundling, 488
batch framework
  Batch API, using, 652–654
  common uses, 643, 644
described, 641
  requirements, 516
batch groups, 643, 656
batch jobs
  batch-executable classes, creating, 645–647
creating, 647–654
described, 643
managing, 657, 658
task dependencies, 650–653
batch processing, 643, 644
batch servers
  AOS instances, configuring as, 655
described, 643
operations, running on, 517
batch tasks
  debugging, 658–660
described, 643
batch-executable classes, creating, 645–647
benchmarking, 773
best practices
  mobile apps, 753
  rules, 41, 43
  washed version, 57
Best Practices tool
  application logic, validating with, 393
  benefits of, 40
  checking customized tables, 417
custom rules, adding, 43
deviations, 38
  suppressing errors and warnings, 43
BI. See business intelligence (BI)
Bitwise operators, expressions, 113
boolean variable declaration, 111
BoundField controls, 227
BPM (Business Process Modeler), 764, 765
breakpoints, 46, 47
browsers, interactions with Enterprise Portal, 209
build process
  creating, 73
  executing tests, 563–566
business documents. See also workflow, document
class
document hashes, 430
transmitting, 432–434
updating, 428–430
workflow documents, 262
business intelligence (BI)
analytic content, configuring, 325, 326
components of, 314
customizing solutions, 324
displaying information, 211
external data integration, 338–340
implementing, 315–324
prebuilt solutions, customizing, 324–340
business logic, referencing in classes, 201
Business Overview web part, 211, 361
Business Process Modeler (BPM), 764, 765
business processes, 257, 258. See also process cycles
business unit, 673
button controls, 187, 188

C
CacheLookup values, 441
caching
  client vs. server tiers, 471, 474
declarative display method, 439
  EntireTable, 474
global variables, 477
indexing, and, 441, 468
number sequencing, 487
records, 467, 468
set-based, enabling, 474
unique index join, 441
calculations, moving to AX 2012, 351
calendar, dates dimensions, 330
call stack, 106
calls, grouping into chunks, 445
CALs (client access licenses), 401
capability, 687, 688
CAS (code access security), 140
case, specifying for source code, 67
Case Sensitive comparison option, 59
categories
  product, 684
  reporting functions, 294
  workflows, using in, 280
chart controls
  binding to datasets, 306
ED Chart Control, adding, 304
markup elements, 305
Cross-Reference tool, 62, 63
CRUD operations, 408

cubes
  creating, 341–351
  customizing, 327–335
  data, exposing to users, 351
  deploying, 317–323
  extending, 335–337
  field-level properties, defining, 344
  processing, 323
  table-level properties, defining, 344

cue element types, 12
cue group element types, 12
cue groups, 195
cues, 157
Cues web part, 212
currency conversions, 333, 346–350
CustName, 110
Customer Details form, 152
customers, creating new, 152
Customization Analysis, 766
customizations, deploying, 768
customizing
  business intelligence solutions, 324–340
  defaulting logic for table fields, 419
  document services, 417
  Help system, 569, 570
  custom lookups, 201
  custom rules, adding, 43
  custom services
    adding, 418
    artifacts, 408
    invoking asynchronously, 430
    registering, 411

data
  business documents, updating in, 428
dimensions, querying, 696
ingoing and exporting, 769–772
data access logic, using datasets, 213
data binding
  controls, 186, 306
  field values, displaying, 227
  strategies, 306
data caching, mobile apps and, 753
data connections, supported in SSRS, 299
data contracts
  parameters, using in service operations, 410
  X++ collection types, using in, 411
  Data Dictionary, 21
  Data Import/Export Framework (DIXF), 771, 772
data mash-ups, 354
data models
  entity relationship, 52
  UML, generating, 49
data partitions, 638, 639
data security, 376, 377
data series, 306–308
data source view (DSV), 337
data sources
  derived, 179, 180
  forms, 177
  metadata properties, 182
  Office Add-ins, making available to, 202
  OLAP, 306
  OLTP, 306
  saving records in, 181
  services, making available, 202
  table inheritance and, 178–180
data tier architecture, 7
data warehouses, 339
data-aware statements, syntax, 116
database transactions, rolling back, 123
datasets
  binding chart controls to, 306
  data access logic, defining, 213
  dynamic series, 308
  initialization with init method, 214
  methods, 214, 215
  multiseries, 307
  single series, 307
  temporary record buffers as pointers to, 605–608
  views, 214
data effectivity, 181
data variable declaration, 111
date-effective framework, 628–633
debugging. See also errors; troubleshooting
  batch tasks, 658–660
  breakpoints, 46
  Debugger tool, 20
  enabling, 45
  extensible data security policies, 389
  Help system, 598, 599
  multiple concurrent updates, 430
  security constructs, 394
  shortcut keys, 48
  user interface elements, 46, 47
  web traffic, 757
  X++ in batches, with Visual Studio, 660
declarations section, in HTML, 576–578
declarative display method caching, 439
decoupling
  base and derived classes, 539, 540
events, 544
  Default Dimensions, 693
defaulting logic, 419, 420
delegates

delegates
  declaring, 136
  members of a class, adding, 544
  subscribing to, 137
delete_from transaction statement, 122
department operating unit, 673
dependencies, batch job tasks, 650–653
deploying customizations, 768
derived data sources, 179, 180
derived tables, creating, 621
design, workflow phase, 275
Design node properties, 186
design phase
  described, 761
  hierarchies, creating, 764
  LCS tools and services, 764
design principles of AX 2012, 3
design time, defining tables, 610, 611
designing new transaction detail forms, 169
details forms, 164–166
details pages, creating, 231
DetailsFormMaster template, 175
DetailsFormTransaction template, 175
develop phase
  customizations, deploying, 768
described, 761
  LCS tools and services, 765
development environments
  MorphX, 6
  Visual Studio, 6
development tools
  accessing, 20
  layer comparison, 30
  mobile apps, 752
  wizards, 30
Development Workspace, launching, 20
Dialog template, 175
dictionary API, type-safe reflection, 715
digital signatures, adding to models, 734
Dimension Attribute Sets, 693
dimension framework, 692–697
Dimension Sets, 693
dimension-based configuration technology, 685
dimensions
  concepts, physical table reference mapping, 697
  product, 681
  querying data, 696
  storage, 683, 693, 694
  tracking, 683
dirty elements, 25
display menu items, 264
distribution policies, controlling with tokens, 667–669
DIXF (Data Import/Export Framework), 771, 772
DLLs. See managed DLLs
document
  body section, in HTML, 581
  files, in Help, 572
  hashes, 430
  head section, in HTML, 578–580
  services, 412–420, 424, 428
  sets, 572, 597
documentation and resource element types, 16
documenting XML methods and classes, 132
DropDialog template, 175
DSV (data source view), 337
duties
  security roles, assigning to, 381
  segregating, 375, 385
duty element types, 14
dynalinks, 178
dynamic role assignment, 376
dynamic series datasets, 308
E
editing Power View reports, 357
editor scripts, 34
EDTs (extended data types)
  schemas, restricting, 414
  table relations, and, 615–617
element types, 9–18
elements
  actions, accessing in the AOT, 26
  AOT, 213
  browsing, 20
  creating, 66
  customizing existing, 66
dirty, 25
hierarchies of, 712, 713
IDs, in models, 730, 740
intrinsic functions, referencing with, 708, 709
layers and models, in, 26, 726, 727
life cycle, 66
modifying in the AOT, 24
naming syntax, 22
overwriting, in models, 735
pending, 72
referencing by name, 709
reflecting, using APIs, 711–724
refreshing, 25
revision history, 71
synchronizing, 25, 69
types, 730
upgrade conflicts, 30
versions, 57
workflow, 264
Enterprise Portal
  Action pane, 170
AOT elements, 213
architecture, 208
charts, creating, 303
components of, 211
deploying as SharePoint features, 250
described, 207
design considerations, 171
developing applications, 228
exceptions, 244
feature definitions, 250
framework, described, 8
framework controls, 215–228
master pages, 250
metadata, APIs for accessing, 238
navigation paths, 170
page processing, 210
proxies, predefined, 240
search bar, 170
security, role-based, 245
SharePoint, integrating with, 248, 256
SharePoint navigation elements, using, 248
style sheets, 256
top navigation bar, 170
web client, described, 8
web client user experience, 169
web part pages, creating, 252
web parts, 211
workspace components, 170
Enterprise Portal Chart Control. See EP Chart Control
enterprise resource planning (ERP), 153
Enterprise Search, 253, 254
EntireTable caching, 474
entity relationship data model, 52
enumeration types, 106
EP Chart Control
creating, 304
data binding strategies, 306–308
described, 289
ERP (enterprise resource planning), 153
error handling, 244
errors. See also debugging; troubleshooting
   best practice rules, 42
   compiler, 38
   Help system, 598, 599
   suppressing, 43
event handlers, 264, 545, 546
eventing, 543–548
Excel
data mash-ups, 354
Power BI, 359
reports, sharing with users, 360
templates, building, 203
exception data type enumerations, 124
exception handling
   best practice, 122
duplicate key, 125
exceptions
   dangerous API, 43
   throwing, 122
exists method, 147
export/import file (XPO), 58
exporting
data, 769–772
model stores, 726
expressions
   lambda, 91
   operators, 113
extended data type element types, 10
extended data types, 106, 110
extending transaction detail forms, 169
extensibility patterns
   eventing, 543–548
   extension framework, 539
extensible
classes, 541
data security policies, 385–389
extensible data security (XDS) framework, 376, 675
extension framework, 539, 540
extension methods, 90, 97. See also AX 2012 R3
extensions
   AX 2012, 300
   creating, 539
data processing, 302
disabling, 301
RDL transformations, 301
SSRS, 299
external data integration, 338–340
external data sources, data mash-ups in Excel, 354
external web services, consuming from AX 2012, 435

**F**
FactBox pane, 156
FactBoxes, 164, 165
factory method, 145
false positives, 43
FastTabs
described, 165
organizing fields into, 166
FastTabs TabPage layout control style, 191
feature definitions, 250
Fiddler web debug proxy, 757
field lists, 477–479
field-bound controls, 192
field-level properties, in cubes, 344
fields
   aggregating within code, 499
   declaring, 134
   entity relationship columns, as, 52
fields (continued)
  hiding, using TabPage, 190
  justifying, 483
  methods, overriding, 199
  organizing into FastTabs, 166
  table properties, 616

filters
  applying programmatically, 221
  custom time periods, adding, 364, 365
  projects, using in, 29
  tests, creating for, 552
  financial dimensions, 329, 675
  find method, 147
  Find tool, 54
  firstfast hint, using, 498
  flexible authentication, 373
  foreign key columns, 52
  foreign keys
    CreateNavigationPropertyMethods, 618–620
    relations, 617
    surrogate, 181
forms
  Action panes, 188
  ad hoc mode, using on, 482
  Auto variables, 200
  batch jobs, creating, 648–654
  business logic, adding calls to, 201
  controls, 186, 189, 192–194
  creating, 174
  customizing with code, 197
  data sources, 177
  dynalinks, 178
  element types, 11
  hiding fields, 190
  input controls, 192
  layout controls, 189
  lookups, custom, 201, 202
  metadata, 176, 182
  methods, overriding, 198
  navigation items, adding, 195, 196
  .NET button, adding, 193
  parts, 194, 195
  patterns, 174
  permissions, setting, 377
  queries, 183–186
  referencing parts, 195
  table data information, 712
  TabPage, hiding fields using, 190
  templates, 175
  workflow, enabling in, 284, 285

foundation layer, 5
frameworks
  accounting, 698–702
  address book, integrating, 675
  application modules, integrating, 675, 676
  batch, 641–644, 652–654
  date-effective, 628–633
  dimension, 692–697
  Enterprise Portal, 8, 215–228
  extensible data security, 675
  extension, 539
  operations resource, 686–692
  organization model, 672–680
  policy, 675
  product model, 681–686
  RunBase, 517, 533
  source document, 702–705
  SysOperation, 516
  SysTest, 550–553, 566
full update mode, applying for document services, 428
full-text search queries, 633, 634
functions
  interactive, adding to charts, 308, 309
  reflection system, 707–711

G
Generic Record Reference, 148
global variables, 47
globalization, 757. See also localization
grids, displaying input controls in, 191
group by sort, in join conditions, 120
group masks in projects, 28
grouping, input controls, 189
guid variable declaration, 111

H
handling exceptions, 122, 125
handshake, secure, 140
hashes, documents, 430
Help documentation set element types, 16
Help system
  described, 570–576
  document files, 572
  errors, 598, 599
  Help server, 572, 573
  Help viewer, 571
  HTML metadata file, creating, 591, 592
  publisher ID, 574
  publishing content, 593–597
  summary page, 574
  table of contents, 574, 588–590
  topics, creating, 573–588
  troubleshooting, 598, 599
  web service, 8, 572, 573
  helper threads, 487
hierarchies
   Business Process Modeler (BPM), creating with, 764
   elements, 712, 713
   organization model framework, 673, 674
   print management, 662, 663
   hierarchy designer, extending, 680
history
   batch jobs, reviewing, 658
   elements, 71
hosting ASP.NET controls, 213
hot swapping assemblies, 87, 88
HTML
   Help topics, creating in, 576–588
   metadata file, creating, 591, 592
human workflows, defined, 259–260

I
IDs
   elements, in models, 730
   treenode type, 722
importing
   data, 769–772
   model files, 735, 736
included columns, in indexing, 497
indexing
   caching, and, 441, 468
   included columns, optimizing with, 497
IndexTabs TabPage layout control style, 190
indicator definitions, 365
individual task modeling, 488
info parts, 11, 194
Infolog web part, 212
information messages, best practice rules, 42
infrastructure callback, 273
Infrastructure Estimation, 764
inheritance
   metadata, 177
   RunBase framework, 533
   tables, 178–180, 621–626
InMemory temporary tables, 442, 448, 604–609
input controls, 192
Inquiries menu grouping, 196
Inquiries section, area pages, 159
insert method transaction statements, 122
insert_recordset transaction statements, 122
int variable declaration, 111
int64 variable declaration, 111
Integrated Windows Authentication, 373
integration ports
   basic, 421
   processing messages, sequence of, 435
   publishing services, 408
selecting, 433
integration with Microsoft Office client, 202
interaction patterns, implementing, 232
interactive functions, adding to charts, 308, 309
interface types, 107
inter-form dynalinks, 178
Internet services queries, 299
interoperability
   CLR, 126
   COM, 129
intra-form dynalinks, 178
intrinsic functions
   referencing elements, 708, 709
   using, 63
inventory closing, improving speed of, 487
inversion entry column, 52
Issue Search, 767

J
job element types, 13
jobs
   model elements, 106
   restarting, 465
joined data sources, 178
joins
   group by sort, 120
   joining tables, 120
   operators, 121
   set-based operations, and, 455
   TempDB temporary tables, using with, 609
Journals section, area pages, 159

K
keys
   alternate, columns, 52
   configuration, 399, 400
   foreign, surrogate, 181
keywords, for select statements, 117, 118
KPI List web part, 361
KPIs
   adding, 350, 363
   modeling in AOT, 337

L
label editor, 34, 36, 37
label files
   element type, 16
   Label Files node, 36
   uses for, 242
labels

labels, 34
 checking out, 69
 creating, 36
 Help, referencing from, 584–586
 reusing, 37
 X++, referencing from, 37
 lambda expressions, 91
 Language-Integrated Query. See LINQ (Language-Integrated Query)
languages, changing in prebuilt solutions, 331, 332
 layers
 comparing, 30
 element definitions, 726, 727
 five-layer solution, 4–6
 logical partitions, 5, 6
 metadata, 727
 models, and, 728
 layout controls, 189
 LCS (Lifecycle Services), 762–768
 Ledger Dimensions, 693
 Left navigation web part, 212
 legal entity organization type, 672
 license codes
 described, 397, 398
 element types, 17
 License Sizing Estimator, 764
 licensing
 customizing, 403
 models, 396
 life cycle
 elements, 66
 phases, 761
 Lifecycle Services (LCS), 762–768
 line-item workflows, 265
 linked data sources, 178
 LINQ (Language-Integrated Query)
 anonymous types, creating, 90
 components of, 89
 constructing, 91–94
 data access problems, solving, 95–98
 defined, 76
 extension methods, 90
 lambda expressions, 91
 overhead, limiting with C# compiler, 101
 records, managing, 99, 100
 var keyword, using to omit variable types, 89
 list definition element types, 16
 list pages
 alternative to reports, 162
 Analyze Data button, adding to, 358
 described, 160
 designing, 163
 displaying default columns, 164
 FactBoxes, 164
 interaction classes, defining, 230
 model-driven, creating, 229
 optimizing for performance, 498
 performing bulk actions, 164
 List web part, 212
 ListPage template, 175
 literals, supporting upgrade scenarios, 452
 local variables, 47
 localization, 36, 242, 757
 Logical operators, expressions, 113
 logical partitions, 5
 logs, client access, 512
 lookups, customizing, 201, 202, 222
 M
 macro library, 130, 131
 macros
 class declarations, defining in, 535
 element types, 13
 supported directives, 130
 using parameters, 131
 managed code
 debugging, 84
 proxies, 85
 writing, 79–84
 managed DLLs, referencing from AX 2012, 77
 ManagedHost control, 193
 managing state, 279
 manifest, models described in, 732
 map element types, 10
 map record types, 107
 master data sources, 177
 master pages, 250
 master scheduling, improving speed of, 487
 maxOf aggregate function, in select statements, 120
 measuring performance, 773
 member variables, 47
 memory heap, 106
 menu element types, 11
 menu item element types, 11
 menu items
 labels, in Help topics, 586
 operating unit types, creating for, 678
 role associations, changing, 403
 security properties of, 380
 workflows, in, 264, 283
 menus, grouping, 196
 messages
 flow and authentication, 749
 sending, unsolicited, 432, 433
 metadata
 accessing through managed code, 238
 associations, in forms, 176
 crawling with Enterprise Search, 254
definitions, selecting, 328
derived classes, adding to, 540
element IDs, 740
form data source properties, 182
Help topics, HTML, 578–579
Help topics, non-HTML, 591
inheritance, in forms, 177
layers, 727
model store, in, 737–739
perspectives, defining, 343
queries, retrieving, 425
WSDL, publishing in, 407–420
Method invocations, expressions, 113
methods
decorating with attributes, 139
extension, 90
invoking on objects, 707
main, adding to a class, 34
modifiers, 135
object behavior, declaring, 135
overriding with code, 197, 198
pack-unpack, 535–538
purposes, adding, 679
QueryBuildDataSource, 184
QueryRunQueryBuildDataSource, 184
RunBase overrides, 516
static new, characteristics of, 533
X++, exposing as a custom service, 408
Microsoft Azure Active Directory Access Control, 747
Microsoft Azure Service Bus adapter, 422
Microsoft Azure Service Bus Relay, 746
Microsoft Dynamics AX
Infolog messages, displaying on webpages, 212
Report Definition Customization Extension (RDCE), 300
Reporting Project, 296
Trace Parser, 501. See also tracing
Microsoft Dynamics AX 2012, See AX 2012; AX 2012 R3
Microsoft Dynamics Enterprise Portal configuration, 245. See also Enterprise Portal
Microsoft Dynamics Public configuration, 245
Microsoft Office client, integrating with, 202–205.
See also Office Add-ins
Microsoft SQL Server Reporting Services (SSRS), displaying reports, 212
Microsoft XML Core Services (MSXML), 129
migrating customizations, 768
minOf aggregate function, in select statements, 120
mobile apps
architecture variation considerations, 758
AX 2012 service design for, 750
best practices, designing and developing, 753
data storage, 753
described, 747
developer resources, 752, 759
developing, 752–757
message flow, 749
offline use, allowing for, 753
on-premise listener, 751
usage, monitoring, 757
user authentication, 749, 754
mobile architecture
components of, 747
developer resources, 752, 759
platform options, 752
variations, 758
mobile clients, using AX 2012 services, 750, 751
model element prefixes, MorphX, 692, 701, 705
model file, generating from .xpo, 73
model stores
API, 740
defined, 725
deploying, 739
element IDs, 740
exporting, 726
model-driven list pages, 212, 229
modeling
capabilities in the AOT, 326
functional scenarios, 676
models
categories, 733
conflicts, resolving with push, 736
creating, 731
described, 728, 729
element IDs, 730, 731
exporting, 733
importing, 735, 736
layers, 726
manifest, describing in, 732
overwriting elements, 735
production, deploying to, 739
publishing, 732–736
signing, 734
staging, 737–740
test environment, 738
upgrading, 737
modifications, rolling back, 465
module-specific navigation links, 212
MorphX
Application Object Tree (AOT), 21
Best Practices tool, 38
classes, upgrading, 58
Compare tool, 57
compiler, 38
Cross-Reference tool, 62
datasets, creating, 212
data storage, 753
data storage, 753
developer resources, 752, 759
developing, 752–757
message flow, 749
offline use, allowing for, 753
on-premise listener, 751
usage, monitoring, 757
user authentication, 749, 754
mobile architecture
components of, 747
developer resources, 752, 759
platform options, 752
variations, 758
mobile clients, using AX 2012 services, 750, 751
model element prefixes, MorphX, 692, 701, 705
model file, generating from .xpo, 73
model stores
API, 740
defined, 725
deploying, 739
element IDs, 740
exporting, 726
model-driven list pages, 212, 229
modeling
capabilities in the AOT, 326
functional scenarios, 676
models
categories, 733
conflicts, resolving with push, 736
creating, 731
described, 728, 729
element IDs, 730, 731
exporting, 733
importing, 735, 736
layers, 726
manifest, describing in, 732
overwriting elements, 735
production, deploying to, 739
publishing, 732–736
signing, 734
staging, 737–740
test environment, 738
upgrading, 737
modifications, rolling back, 465
module-specific navigation links, 212
MorphX
Application Object Tree (AOT), 21
Best Practices tool, 38
classes, upgrading, 58
Compare tool, 57
compiler, 38
Cross-Reference tool, 62
datasets, creating, 212
data storage, 753
data storage, 753
developer resources, 752, 759
developing, 752–757
message flow, 749
offline use, allowing for, 753
on-premise listener, 751
usage, monitoring, 757
user authentication, 749, 754
mobile architecture
components of, 747
developer resources, 752, 759
platform options, 752
variations, 758
mobile clients, using AX 2012 services, 750, 751
model element prefixes, MorphX, 692, 701, 705
model file, generating from .xpo, 73
model stores
API, 740
defined, 725
deploying, 739
element IDs, 740
exporting, 726
model-driven list pages, 212, 229
modeling
capabilities in the AOT, 326
functional scenarios, 676
models
categories, 733
conflicts, resolving with push, 736
creating, 731
described, 728, 729
element IDs, 730, 731
exporting, 733
importing, 735, 736
layers, 726
manifest, describing in, 732
overwriting elements, 735
production, deploying to, 739
publishing, 732–736
signing, 734
staging, 737–740
test environment, 738
upgrading, 737
modifications, rolling back, 465
module-specific navigation links, 212
MorphX
Application Object Tree (AOT), 21
Best Practices tool, 38
classes, upgrading, 58
Compare tool, 57
compiler, 38
Cross-Reference tool, 62
datasets, creating, 212
data storage, 753
data storage, 753
developer resources, 752, 759
developing, 752–757
message flow, 749
offline use, allowing for, 753
on-premise listener, 751
usage, monitoring, 757
user authentication, 749, 754
mobile architecture
components of, 747
developer resources, 752, 759
platform options, 752
variations, 758
mobile clients, using AX 2012 services, 750, 751
model element prefixes, MorphX, 692, 701, 705
model file, generating from .xpo, 73
model stores
API, 740
defined, 725
deploying, 739
element IDs, 740
exporting, 726
model-driven list pages, 212, 229
modeling
capabilities in the AOT, 326
functional scenarios, 676
models
categories, 733
conflicts, resolving with push, 736
creating, 731
described, 728, 729
element IDs, 730, 731
exporting, 733
importing, 735, 736
layers, 726
manifest, describing in, 732
overwriting elements, 735
production, deploying to, 739
publishing, 732–736
signing, 734
staging, 737–740
test environment, 738
upgrading, 737
modifications, rolling back, 465
module-specific navigation links, 212
MorphX
Application Object Tree (AOT), 21
Best Practices tool, 38
classes, upgrading, 58
Compare tool, 57
compiler, 38
Cross-Reference tool, 62
datasets, creating, 212
data storage, 753

**MSXML (Microsoft XML Core Services)**

MorphX (continued)
- development environment, 6
- Find tool, 54
- implementing actions, 26
- label editor, 34
- labels, referencing from X++, 37
- model element prefixes, 692, 701, 705
- models, creating, 731
- personalizing tool behavior, 20
- property sheet, 31
- Reverse Engineering tool, 48
- Table Browser tool, 53
- tools and components, 20
- Type Hierarchy Browser, 20, 108
- Type Hierarchy Context, 20, 108
- updating application model elements, 19
- user interface control element types, 11
- version control, 64, 65
- X++ code editor, 32

MSXML (Microsoft XML Core Services), 129
- multiserie datasets, 307

**N**

Name extended data type, 110
Named User license, 396
navigation
- items on forms, 195, 196
- layer forms, 155, 156, 170
- links, 212
- panes, 155, 156
- SharePoint sites, elements, 248
- .NET AJAX. See AJAX
- .NET buttons, adding to forms, 193
- .NET CIL (common intermediate language), running X++ as, 142
- .NET CLR interoperability statements, 114
- number sequence caching, 487
- numeric information, displaying on webpages, 212

**O**

Object creation operators, expressions, 113
object models, UML, 50
object types
- reference type, 109
- variable declaration, 111
objects
- methods, invoking on, 707
- Query, 185
- QueryRun, 185
OCC (optimistic concurrency control), 430
Office 365, Power BI, 359
Office Add-ins, 202–205

Office clients, 8
OLAP database, providing access to, 324
old layered version types, 57
on-corpnet mobile apps, 758
on-premise listener, developing for mobile apps, 751
operate phase
- described, 761
- LCS tools and services, 767
operating unit organization types, 672, 673, 677
operations
- batch servers, running on, 517
- downgrading, 454, 456
- requirements for defining, 516
operations resource framework, 686–692
operators
- join, 121
- set-based data, manipulating, 447
- table hierarchies, 448
optimistic concurrency, 466
optimistic concurrency control (OCC), 430
OptionalRecord, 181
ordered test suites, 562
organization
- hierarchies, 673, 674
- types, 672, 674
organization model framework, 672–680
over-layering, 727
overriding
- controls, 186
- default chart formats, 310
- form methods, 198

**P**

pack-unpack pattern, 535–538
page definition element types, 16
Page title web part, 212
pages
- Enterprise Portal, processing in, 210
- pop-up browser window, opening in, 225
- SharePoint templates, 249
- standard interaction patterns, implementing, 232
- parallel activities, in workflows, 265
- parallel processing, 433, 435, 488
- parameter method, implementing, 144
- parameters
  - AXUpdatePortal utilities, 252
  - data contract, using in service operations, 410
- Parentheses, in expressions, 113
- parent pages, passing data to, 226
- Parm methods, 34
- partial update mode, applying for document services, 429
- partitions, 638, 639
- parts, 194, 195. See also web parts
passing information
  Context data structure, 235
  record context interface, 247
patterns
  checking for existing records, 494
  class-level, 144–146, 145
  extensibility, 539
  for performance, 487–500
  pack-unpack, 535–538
  property method, 534
  storage, dimensions, 693, 694
  table-level, 147, 148
pending elements, viewing, 72
performance
  AOS configuration settings, 486
  benchmarking, 773
  caching, 438, 477
  coding patterns, optimizing for, 487–500
  configuration options, 483–487
  declarative display method caching, 439
  field justification, 483
  field lists, limiting, 479
  list pages, optimizing, 498
  monitoring tools, 501–513
  parallel processing, using, 435
  table inheritance, 626
  transactions, optimizing for, 447
Usage Profiler, 764
Periodic menu grouping, 196
Periodic section, area pages, 159
permissions
  auto-inference, using, 378
  code, 381
  defined, 374
  forms, setting for, 377
  privileges, creating, 380
  property values, 381
personas, defined, 352
personName, 112
perspective element types, 10
physical tables, mapping to concepts, 697
platform architecture
  application development environments, 6
tiers, 7, 8
platforms, and mobile architecture, 752
policies
  context information, 386
  extensible data security, 385
policy framework, 675
policy query, 386
polymorphic
  associations, 147
  queries, 178, 179, 624, 625
pop-up browser windows
  opening pages in, 225
parent page, passing data to, 226
Power BI
  Office 365 and, 359
  Power View, comparing to, 360
Power View, 353–358, 360
pre and post events, 545
prebuilt BI solutions, customizing, 324–346
precision design reports, 298
predefined variant configuration technology, 685
pre-event and post-event handlers, 138
prefixes
  business area name, as, 22
  commonly used, list of, 23
  MorphX, model elements, 692, 701, 705
pre-processing data in reports, 299
presentation tier architecture, 8
primary entity, creating and extending, 165
primary table security policy, 386
primitive types, 106
print management
  applying, 662
  automating tasks, described, 642
  hierarchy of, 662, 663
  settings, 663–669
privilege element types, 14
privileges
  creating, 380
  defined, 374
  security roles, assigning, 381
  using, 246
process cycle element types, 14
process cycles, 375
process states
  accounting distribution, 704
  accounting framework, 700, 701
  subledger journalizing, 704
product masters, 681
product variants, 681
product model framework, 681–686
production
  models, deploying to, 739
  reports, 293, 295
profiles, associating with users, 323
projects
  AOT, maintaining in, 340
  creating, 27
  customizing, 327–335
  development tools, 30
  filters, 29
  generating automatically, 28
  group masks, 28
  layers, comparing, 30
  property sheet, 31
  type, specifying, 30
  upgrading, 30
properties
AOSAuthorization, 391
CopyCallerQuery, 185
Design node, 186
document sets, setting in Help, 597
property method pattern, 534
property sheets, 31
provider callback, 273
proxies
creating new, 240
described, 85
proxy classes, 239
publisher ID, and Help, 574
publishing
AX 2012 services, 421
Help content, 593–597
models, 732–736
Purchase Order form, 167
purposes
base enum values, creating for, 679
creating custom, 678–680
organization types and, 674
Q
quality checks, 67
queries
Axd, guidelines for creating, 416
data, reading through LINQ, 95
data processing extensions, using, 302
dimension data, 696
document services, generating, 412, 416
Enterprise Search, using, 253
filters, applying, 185
forms, 183
Internet services, 299
LINQ, constructing, 91–94
metadata, retrieving, 425
Office Add-ins, making available, 203
polymorphic, 178
reducing execution of, 495
SSAS OLAP, accessing data, 299
T-SQL, accessing data, 299
using, vs. joins, 496
query element types, 11
Query objects, 185
query string parameters, passing record context, 248
QueryBuildDataSource method, 184
QueryFilter API, 635–638
QueryRun objects, 185
QueryRunQueryBuildDataSource method, 184
Quick launch web part, 212
Quick links web part, 212
R
RapidStart Services, accessing LCS, 764
RDCE (Microsoft Dynamics AX Report Definition
Customization Extension), 300
RDL transformations, 301
RDP (report data provider), 299
real variable declaration, 111
record buffer
pointers to datasets, 605–608
turning off checks, 451, 456
record context interface, 247
Record type variable declaration, 111
record types, 107
record-based operations
downgrading to, 450
transferring to set-based, 463
records
caching, 467, 468
date-effective framework modes, 632
existing, checking for, 494
form templates and, 175
inserting multiple, 457
joins and, 182
LINQ, managing with, 99, 100
locate, using record context, 247
polymorphic creation of, 179
saving, in form data sources, 181
selecting optimistically, 466
specifying types users can create, 180
updating multiple, 453
RecordViewCache, 475, 476
reference element types, 13
reference layer, 30
reference types, 107
references, dimension concepts and physical tables, 697
referential integrity, Unit of Work, 626
reflection
APIs, 711–724
described, 707
methods on objects, invoking, 707
Reverse Engineering tool, 707
system functions, 707–711
tables, 714
views, 714
refreshing elements, 25
Relational operators, expressions, 113
relationships between tables, 615–620
released products, 683
rendering extensions, disabling, 301
Report Builder, 366
report data provider (RDP), 299
Report model, in Visual Studio, 297
Report web part, 212
reporting framework, troubleshooting, 311, 312
reports
AX 2012 development tools, 296
categorizing based on roles, 294
client-side solutions, 290
controls, using, 297
creating, 295
datasets, 294, 298
designs, 294, 297, 298
development roles, 294
display content, controlling, 298
edits by users, 357
element types, 12
execution sequences, 300, 302
extensions, 8
layout design, 297
Microsoft Dynamics AX Reporting Project
template, 296
Model Editor, using, 297
Power View, 354, 357
pre-processing data, 299
Report Builder, using, 366
server-side solutions, 292
solutions, planning, 293
SSRS elements, 296
SSRS extensions, 299
static, creating, 302
troubleshooting, 311, 312
Visual Studio tools, using, 366–369
Reports menu grouping, 196
Reports section, area pages, 159
resource element types, 16
resources, identifying for activities, 690
RESTful services, 751
retail channel operating unit, 673
Reverse Engineering tool
described, 48
entity relationship data model, 52
UML models, generating, 49, 50
revision history, 71
RFP Responses, 764
Role Centers
analytic content, displaying, 351–369
OLAP reports, 326
pages, 156, 157
reports, adding, 355, 356
user profiles, 323
role-based security, 14, 246
roles
access control, based on, 373
assigning to users, 384
categories of, 352
element types, 14
menu item associations, changing, 403
privileges, assigning, 381
reporting needs, based on, 294
security, assigning to users, 376
security artifact associations, changing, 403
role-tailored design, 153
rolling back modifications, 465
root data sources, 177
root tables, creating, 621
round trips, reducing, 438–441, 445, 447, 458, 477
RowCount method, 121
rows, deleting multiple, 455
rules, segregating duties, 385. See also best practices, rules
RunBase
inheritance, 533
round trips, handling, 440
SysOperation, comparing to, 517, 518
RunOn, instantiating objects, 442
run time, and temporary tables, 611, 612
runtime modifications, 187

S
Sales Order form, 167
scenarios, modeling, 676
search bar
Enterprise Portal, in, 170
workspace component, 155
searching
Enterprise Search, using, 253, 254
Find tool, using, 54
full-text support, 633, 634
Help viewer, 571
Issue Search, 767
ranges, specifying, 56
Windows Search Service, 573
securing APIs, 392
security
APIs, 392
authorization, role-based access, 373
coding, 390–394
controls, permissions for, 379
data policies, assigning, 376
debugging, 394–396
duties, assigning to roles, 381
Enterprise Portal, 245
exposing web controls, 246
forms, setting permissions for, 377
hash parameters, using, 248
hierarchy and user types, 402
policy concepts, 385
privileges, 380, 381
role-based, 246
roles, 375, 376
security artifacts

security (continued)
server methods, setting permissions for, 379
service operations, and, 420
security artifacts
developing, 377–381
menu options, 395
role associations, changing, 403
validating, 384
security framework, 372–376
security policy element types, 14
select forUpdate transaction statements, 122
select query, ordering and grouping, 119
select statements
aggregate functions, 120
joining tables, 120
keyword options, 117, 118
syntax, 117
separation of concerns and processes, 3
serializing with the pack and unpack methods, 146
Server Configuration form, 484
server methods, setting permissions for, 379
server-side reporting solutions, 292
service contracts
custom services, in, 409, 410
document services, in, 412
service element types, 13
service implementation
custom services, 409
document services, 413
service contracts, 409
service operations, and security, 420
services
AX 2012, consuming, 422
making available, 202
sessions, disposal and caching, 234
set-based operations
caching, 474
code, transferring into, 459
downgrading, 451, 452, 454
InMemory temporary tables, and, 448
joins, using, 455
manipulating data, 447
record-based, downgrading to, 450
table hierarchies, and, 448
Setup menu grouping, 196
Setup section, area pages, 159
SGOC (SysGlobalObjectCache), 477
shared steps, 559–561
SharePoint
developing web part pages and lists, 228
sites, 249
SQL Server Power View, and, 355
themes, integrating with Enterprise Portal, 256
Shift operators, expressions, 113
shortcut keys
debugging, 48
X++ code editor, 33
Show Differences Only comparison option, 59
Show Line Numbers comparison option, 59
signing models, 734
SimpleList template, 175
SimpleListDetails template, 175
single series datasets, 307
site navigation, 248
solution architecture, five-layer, 4–6
source code casing, executing, 67
Source Code Titlecase Update tool, 67, 111
source document framework, 702–705
SQL Administration form, 483
SQL Server Analysis Services. See SSAS (SQL Server Analysis Services)
SQL Server Power View. See Power View
SQL Server Reporting Services. See SSRS (SQL Server Reporting Services)
SSAS (SQL Server Analysis Services), 4, 315, 317, 335, 336
described, 8
OLAP queries, 299
SSRS (SQL Server Reporting Services)
components of, 296
data connections, supported, 299
described, 4
report element types, 12
reporting extensions, 8, 299
reports, displaying, 212
standard layered version types, 57
Standard TabPage layout control style, 190
state model, 279
statements, 114–116
states, managing, 279
static file element types, 16
static RDL reports, 302
static schema, 326
storage, dimensions, 683, 693, 694
str variable declaration, 111
string, as Name extended data type, 110
String concatenation, expressions, 113
string literals, expressing, 112
strong names, 76, 77, 734
style sheets, 256
subledger journalizing process, 704
subnodes, changing the order of, 24
subworkflows, 264
sum aggregate function, in select statements, 120
summary page, in Help, 574
suppressing
errors and warnings, 43
whitespace, during file comparison, 59
surrogate keys, 181, 612–614
synchronizing elements
  sequence of operations, 69
  viewing the log, 70
SysAnyType, 108
SysGlobalCache, 477
SysGlobalObjectCache (SGOC), 477
SysObsoleteAttribute, 139
SysOperation
  attributes, 517
  classes, 516
  execution modes, 489, 490
  RunBase, comparing to, 440, 517, 518
System Diagnostic Service, 767
system documentation element types, 16
system navigation, role-tailored, 153
system services, 407, 408, 425
system workflows, defined, 259
SysTest framework, 550–553, 566

T
Table Browser tool, 53
table collection element types, 11
table hierarchies, and set-based operations, 448
table of contents, in Help, 574, 588–590, 593–599
table permissions framework, 377, 390
table-level patterns, 147, 148
table-level properties, in cubes, 344
TableOfContents template, 175
tables
  alternate keys, 614
  Ax<Table> classes, accessing with, 415
  behavior, specifying in hierarchy, 622
  buffers, 445
  caching contents, 441, 467, 468
  configuration time, 611
  customizing, 417
  data consistency, and run-time support, 631
  data retrieval, 630, 631
  data storage, mapping to, 623
data-effective entities, relational modeling of, 628–630
defaulting logic for fields, 419, 420
design time, 610, 611
  EDT relations, 615–617
  field labels, adding to Help topics, 585
  field properties, 616
  field states, tracking, 420
  foreign keys, 617, 618
  index clause, 118
  inheritance, 178–180, 621–626
  InMemory, 442, 604–608
  joins, 609, 613, 618, 625, 626, 635, 636
  keys, 612–614
model store, 725
multiple records, inserting, 449, 457
physical, mapping to concepts, 697
record types, 107
records, inserting and modifying, 121
references, custom lookups, 201
reflection, 714
relationships between, 615–620
rows, manipulating, 465
run time, 611, 612
surrogate keys, 612–614
TempDB, 609–612
TempDB vs. inMemory, 443
temporary, 442, 443, 448, 449, 604–612
type-safe method, and, 109
unique index join cache, 473
valid time state, using, 383
TabPage layout controls, 190
tasks
  batch, debugging, 658–660
  batch jobs, and, 650–653
  compiler, 38
  workflow element, 264
Team Foundation Build, 563–566
TempDB temporary tables, 443, 444, 609–612
templates
  adding for users, 205
  Excel, building for, 203
  form patterns, 174
  Help topics, 576
  Word, building for, 204
temporary tables
  creating, 610, 611
  InMemory, 442, 604–609
  multiple records, inserting, 449
  set-based operations and inMemory, 448
  TempDB, 609–612
  TempDB vs. inMemory, 443
test cases, developing in phases, 562
Test Data Transfer Tool, 769, 770, 772, 773
test environments, models, 738, 739
Test project type, 30
test suites, using for long scenarios, 562
testing
  ALM solution, 556
  ATDD (acceptance test driven development), 557, 558, 566
  attributes, creating for tests, 552
  build processes, executing in, 563–566
  environments, described, 566, 567
  evolutionary case development, 562
  filters, creating, 552
  integration, 552–555
  ordered test suites, 562
  predefined attributes, 550–552
TFS version control system

testing (continued)
  shared steps, 559–561
  Team Foundation Build, 563–566
  Visual Studio 2010 tools, 556–562
TFS version control system, 65
themes, integrating with SharePoint, 256
third-party
  assemblies, 76–79
  clients, 8
  libraries, mobile app developer resources, 752
tiers
  Business Intelligence solution, 314
caching, client vs. server, 471
client vs. server, 538
instantiating objects using RunOn, 442
time periods, 365
TimeOfDay variable declaration, 111
TODO comments, 38
tokens, print management, 667–669
Toolbar web part, 213
toolbars
  AxToolbar control, using, 224
displaying on webpages, 213
top navigation bar, in Enterprise Portal, 170
top picking, 489
Trace Parser, 501, 502. See also tracing
tracing
  analyzing results, 506
code instrumentation, using, 505
database activity, monitoring, 510
importing, 506
starting, 502, 503
Tracking Cockpit options, 503
troubleshooting, 509
Visual Studio Profiler, 512
Windows Performance Monitor, using, 504
Tracking Cockpit options, 503
transaction details forms
  described, 167
designing new, 169
extending existing, 169
header view, 168
line view, 168
Purchase Order, 168
toggling between views, 168
transaction statements, 122
transactions, optimizing for performance, 447
Transact-SQL (T-SQL) queries, 299
translations, 331, 332
transmitting business documents, 432–434
treenodes, 718–723
triggers, implementing for message transmission, 432
troubleshooting. See also debugging; errors
  Help system, 598, 599
  reporting framework, 311, 312
  reports, 312
  Server Process ID (SPID), using, 511
  systems, using System Diagnostic Service, 767
  tracing issues, 509
  usage issues, 764
trusted code, defining, 140
T-SQL (Transact-SQL) queries, 299
ttsAbort transaction statements, 121
ttsBegin transaction statements, 121
ttsCommit transaction statements, 121
type hierarchies, 107, 621
Type Hierarchy Browser tool, 20, 108
Type Hierarchy Context tool, 20, 108
U
unbound controls, 192
under-layering, 727
Unified work list web part, 213
unique index join cache, 441, 473
Unit of Work, 181, 626–628
unit testing, 550–553
Update Installer for Microsoft Dynamics AX 2012 R3, 767
update_recordset transaction statements, 122
updating
  business documents, 428–430
  Help topics, 587, 588
Upgrade Analysis, 766
upgraded class versions, 58
upgrading projects, 30, 766
URLs, Power View parameters, 356
Usage Profiler, 764
user authentication
  mobile apps, 749, 754
  SSL, setting up, 754
user context information, 47
User control web part, 213
user experience
  Enterprise Portal, 169
  mobile apps, and, 754
  navigation layer, 154
  role-tailored design, 153
  simplifying with FactBoxes, 166
  work layer, 154
user input, validators, 243
user interface
  control element types, MorphX, 11
debugger elements, 46
default, creating from definitions, 516
labels, referencing from Help, 584–586
wizard-driven, 326
user templates, adding, 205
user types, and security hierarchy, 402
user-defined class types, 107
users
access to OLAP database, providing, 324
assigning roles to, 384
CAL role mapping, 402
creating, 384
cube data, exposing data to, 351
Power View reports, editing, 357
profiles, associating with, 323
provisioning, 323
tracking activities, 512
USR layer, 726
utcDateTime variable declaration, 111

V
valid time state tables, 383
validating
logic, with Best Practices tool, 393
security artifacts, 384
validation code elements, 418, 419
validation logic, 418
value stream operating unit, 673
value types, 106
values
combinations, constraining, 694
expressions, 113
Ledger Dimensions, creating, 695
var keyword, 89
variables
Auto, form-specific, 200
declarations, 111
declared as reference types, 107
expressions, 113
grouping in debugger, 47
reference, declared as record types, 107
viewing in debugger, 47
VendName, 110
version control systems
build process, 73
code quality, enforcing, 64
common tasks, 68
element life cycle, 66
history of elements, showing, 71
integrating AX 2012, 74
isolated development, 64
labels, working with, 69
MorphX VCS, 65
pending elements, viewing, 72
quality checks, 67
revisions, comparing, 72
source code casing, 67
synchronization log, viewing, 70
synchronizing elements, 69
TFS, 65
Visual SourceSafe, 65
Version Control tool, 64
VerticalTabs TabPage layout control style, 190
view element types, 10
view record types, 107
views
dimensions, creating, 677
reflection, 714
ViewState, 241
Visio models, generating, 48
Visual SourceSafe 6.0 version control system, 65
Visual Studio
details pages, creating, 231
EP Chart Control markup, 305
Microsoft Dynamics AX Reporting Project
template, 296
Report model, 297
reports, creating with, 366–369
Team Foundation Build, 563–566
test tools, 556–562
X++, debugging in a batch, 660
Visual Studio Profiler, 512

W
weak type systems, avoiding, 108
web apps for mobile devices, 758
web client element types, 15–16
web elements, securing, 246
web menu items, 264
web parts
Action pane, 211
business intelligence information, 211
Business Overview, 361
integrating into webpages, 211
KPI List, 361
linking information, 212
page framework, using from SharePoint, 208
Page Viewer, 356, 357
pages, 251–253
Power View, exposing reports, 355–357
SharePoint sites, 249
web service calls, analyzing, 757
web services
consuming from AX 2012, 435
debugging traffic, 757
Web Services Description Language (WSDL), 407
web traffic, debugging

web traffic, debugging, 757
Web.config files, adding publishers, 594–596

webpages
alert notifications, 213
ASP.NET, creating with AJAX, 233
ASP.NET controls, hosting, 213
Infolog messages, displaying, 212
page-specific navigation, 212
toolbars, displaying, 213
web parts, integrating, 211
workflow actions, displaying, 213
WF (Windows Workflow Foundation), 261
Windows client, 8
Windows Performance Monitor, using to trace, 504
Windows Search Service, 573, 598
Windows Workflow Foundation (WF), 261
WindowMode settings, 232
Word, building templates, 204
work layer forms, 154

workflow
action menu items, 264
actions, displaying on webpages, 213
activating, 283–287
architecture, explained, 268
artifacts, 275–277
automating, 276
categories, 262, 280
conditions, creating, 280
display menu items, 264
document class, 262, 280, 282
documentation, 132, 133
datasets, 214
delegates, adding, 544
design and implementation patterns, 143
document class, 262, 280, 282
documentation, 132, 133
design and implementation patterns, 143
elements, 12, 264
event handlers, 264
forms, enabling in, 284, 285
implementing, 276
infrastructure, 258–260, 276
instances, 268
key concepts, 262–268
life cycle phases, 275
line-item, 265
menu items, 264, 283
provider model, 266
queues, assigning to, 265
runtime, 269–273
states, managing, 279
type contracts, 258–259, 263
work items, 268

workflow runtime
acknowledgment messages, 272
activation messages, processing, 270
API, to expose functionality, 269
application code, invoking, 269

X

X++

APIs, reflecting on elements, 711–724
collections, using in data contracts, 411
datasets, 214
delegates, adding, 544
describing, 407
expression operators, 113
interoperability, allowing, 125
jobs, 106
macro capabilities, 130
methods, exposing as a custom service, 408
primitve types, converting from and to CRL
objects, 128
reflection, system functions, 707–711
statements, 114
syntax, 110, 111
variable declarations, 111

X++ code

class-level patterns, 144
compiling and running as .NET CIL, 142
design and implementation patterns, 143
executing as CIL, 487

X++ code editor

text editor script, 34
opening, 32
recompiling, 38
shortcut keys, 33

XDS (extensible data security) framework

creating policies, 385
described, 376

XML

documentation, 132, 133
messages, sending asynchronously, 431
serialization, implementing for data objects, 414
XPO (export/import file), 58