



# Infor Epiphany Sales and Service Implementation Guide

**Copyright © 2015 Infor**

## **Important Notices**

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

## **Trademark Acknowledgements**

The word and design marks set forth herein are trademarks and/or registered trademarks of Infor and/or related affiliates and subsidiaries. All rights reserved. All other company, product, trade or service names referenced may be registered trademarks or trademarks of their respective owners.

## **Publication information**

Release: 10.0.1

Publication Date: July 14, 2015

# Contents

<b>Chapter 1: Overview.....</b>	<b>11</b>
About Infor.....	11
Purpose of this Guide.....	11
Product Documentation.....	12
Installation and Configuration Guide.....	12
Implementation Guide.....	13
Reference Guide.....	13
Administrator's Guide.....	13
Computer Telephony Integration Guide.....	13
Integration Guide.....	13
Online Help.....	14
Viewing Release Notes and Manuals.....	14
Printing This Document.....	14
Contacting Customer Support.....	14
Location of the Platform Support Matrix.....	15
<b>Chapter 2: Application Implementation and Customization.....</b>	<b>17</b>
High Level Steps.....	17
Customization Checklist.....	20
Delta Modules.....	21
Guidelines.....	22
To Undo a Change.....	22
Best Practices.....	22
Delta modules and EDialogs.....	22
CreateDelta Tool.....	23
Checking Existing User Roles in Studio.....	23
Checking Delta Module Permission.....	24
Creating a New User Role in the Target Environment.....	25
Using Infor Studio for Application Customization.....	25
Data source.....	25
Recordset.....	26
Relationships Between Existing Recordsets.....	26
BIO.....	26
Form.....	26
Integrating a form to the Sales Detail Shell and the Sales List Shell.....	27
Creating a Data Source.....	27
Creating a Recordset.....	28

Defining a Relationship Between Recordsets.....	29
Creating a BIO.....	31
Defining A Form and Integrating The Form With The Sales Detail Shell.....	31
<b>Chapter 3: Best Practices.....</b>	<b>33</b>
Application Implementation/Customization.....	33
Alerts.....	36
Configuration.....	37
Data Migration.....	37
Databases.....	39
SQL Repl.....	41
Deployment.....	41
Dialogs.....	42
Knowledge Management System.....	43
Other.....	43
Upgrade.....	45
Bulk Import.....	45
Setting Defaults for Import Columns.....	46
Lookups.....	47
Creating and Running the Import Job.....	48
Extensions.....	49
BIOs with too many Relationships.....	50
Preserving Relationships.....	50
Sub Classing Issues.....	50
Sales vs. Service Behavior.....	51
<b>Chapter 4: Using Eclipse.....</b>	<b>53</b>
Configuring Eclipse.....	53
Creating New Extensions.....	62
<b>Chapter 5: Source Control.....</b>	<b>69</b>
Overview.....	69
Using CVS.....	69
Setting up Eclipse.....	70
Preparing a Development Machine.....	71
Checking Out a File.....	73
Updating/Uploading Infor Modules.....	75
Updating/Uploading Infor Delta Modules.....	76
Checking In a Module.....	77
Canceling a File Check Out.....	78
<b>Chapter 6: Custom Build Process.....</b>	<b>81</b>



---

Build Process.....	81
Build Process Phases.....	81
Build Types.....	82
Build Source Components.....	82
Sample Build Process (local).....	83
Sample Build Script.....	84
Sample Build Structure.....	89
Dist: retrieve: Retrieve Source Components.....	91
Dist:local.....	91
Dist:compile.....	91
Dist: stage: Create Distribution Tree.....	91
Dist: SQL: Update Operational Database.....	91
Dist: load module: Load Out-of-the-box and Custom Modules.....	92
Dist: deployment: Build and Deploy the Service Enterprise Archive.....	93
Test and Release the New Build.....	93
Sample Build Process (remote).....	93
Build Process after an Upgrade.....	95
Ongoing Development Environment.....	95
<b>Chapter 7: Performance.....</b>	<b>99</b>
Overview.....	99
Performance Issues Related To Application Design.....	99
Extension Development Guidelines.....	100
Performance Issues Related to Deployment.....	100
Process for Improving Application Performance.....	101
Verifying Application Settings and Behavior.....	101
Documenting Configuration.....	102
Identifying Areas for Performance Optimization.....	103
Making Modifications to Improve Performance.....	104
Sample Task List.....	104
Performance Tuning in Deployments.....	105
Quick Checks for Performance Problems.....	105
Configuring the System for Performance Optimization.....	106
Analyzing System Performance.....	107
System Optimizations Undertaken in Implementations.....	108
<b>Chapter 8: Using Out-of-the-box User Interface Templates.....</b>	<b>113</b>
User Interface Template Forms.....	113
List Template.....	113
Detail Template.....	113
Tabgroup Template.....	114
Toggle Template.....	114

---

Making an Object Available from Template Forms.....	114
Adding an Object to a Menu.....	120
Naming Guidelines.....	121
<b>Chapter 9: Bulk Import.....</b>	<b>125</b>
Overview.....	125
Terminology.....	125
Import Process.....	126
Creating an Import Map.....	126
Creating an Import Map using the Import Service Wizard (CSV File Data Source).....	126
Creating an Import Map using the Bulk Import Form.....	127
BIO Transformation.....	128
Creating a Transformation Map.....	129
Creating and Running an Import Job.....	131
Trouble Shooting.....	132
Performance Issues.....	132
Example.....	133
Import Map Details.....	133
Setting up Filter on Address Mapping.....	133
Import Job Details.....	134
Database Import.....	135
Creating an Import Map using the Import Service Wizard (Database Data Source).....	136
Error Handling.....	136
Debugging Bulk Import.....	137
Bulk Importing with Special Characters.....	138
<b>Chapter 10: BIO Externalization.....</b>	<b>139</b>
General Strategies for Re-mapping Recordsets and BIOs.....	139
RecordSets for which Externalization is supported.....	139
Determining Dependencies when changing Field/Attribute Type or Attribute Mapping Type.....	140
Out-of-the-box Fields/Attributes with no Corresponding Column in Legacy Table.....	141
Non-filterable/Sortable Widgets based on Attributes Path Mapped to Externalized BIOs.....	142
Configuring Locking Strategy on Externalized RecordSets.....	142
Configuring Non-usage of Logical Delete on Externalized RecordSets.....	143
'Recent Activity' may not work for Externalized BIOs.....	143
Impact on Polymorphic Relationships where Primary Keys are changed from GUID to String/Integer.....	143
BIOs with Composite Primary Keys cannot be Audited.....	145
Externalization of BIOs subject to ACLs will cause ACLs to no longer work on those BIOs.....	145
Re-mapping Customer BIOs.....	145
Re-mapping Customer/Individual/Organization RecordSets and BIOs.....	145
Non-filterable/Sortable Account Widget on Forms if Individual/Organization is Re-mapped.....	146

---

Removing Multiple Relationships between Two Records.....	147
Session Caching and RecordSets.....	147
RecordSets that do not have FETCH/QUERY Methods Exposed.....	147
Handling Errors Returned by an External System.....	148
Handling Errors when displaying a List in a List View.....	149
Handling Errors when displaying Related Records in a Sub-Tab Form.....	150
<b>Chapter 11: BIO Transformation.....</b>	<b>155</b>
BIO Mapping.....	155
Creating a Transformation Map.....	155
Creating a Unique Map Name.....	156
Linking Source and Target.....	157
Choosing type of Transformation.....	158
Setting Masked Attributes for Map.....	159
Creating Attribute Maps for Transformation.....	159
Setting Properties on Path Elements.....	164
Handling Attribute Domains.....	169
Creating a Path Traversal Map.....	170
Creating a Map.....	170
Setting up Path for Traversal.....	171
Setting Path Element Properties.....	174
<b>Chapter 12: Dialogs.....</b>	<b>175</b>
Overview.....	175
Launching Dialogs from Customer Service.....	175
OpenDialogInFrame.....	175
Refresh.....	180
OpenDialogInWindow.....	181
Refresh.....	184
OpenDefaultDialogInWindow.....	185
Launching Dialogs from a Sub Form (on an Iframe).....	185
Launching Dialogs from a Toolbar (launching Default Dialog on a Separate Window).....	189
IBR and EDialogs.....	190
Best Practices.....	190
<b>Chapter 13: The Replication Service.....</b>	<b>191</b>
Overview.....	191
Mapping BIO types.....	192
Initial Replication.....	192
Batch Replication.....	192
Deletions.....	193
Mapping Keys.....	193

---

Replication State BIO.....	194
Transaction Handling.....	194
Error Handling.....	194
Replication State Process Flow Diagram.....	196
Replication Events.....	196
<b>Chapter 14: Using Access Control Lists.....</b>	<b>199</b>
Overview.....	199
BIO Row Permissions and Access Control Lists.....	200
Permission Inheritance.....	201
Objects Permissions in Out-of-the-box Infor Epiphany Sales.....	202
Upline and Downline.....	204
Implementing Access Control Lists.....	205
Integrating Project Teams.....	213
ACLManagerExtension.....	215
<b>Chapter 15: Translating Application Strings using Infor Studio.....</b>	<b>219</b>
Translating Application String Steps.....	219
<b>Chapter 16: Integrating a Request with Attachments.....</b>	<b>225</b>
Overview.....	225
Integrating Attachments.....	225
<b>Chapter 17: Customizing Help.....</b>	<b>233</b>
Overview.....	233
Customizing Help Roadmap.....	233
Selecting an Authoring Tool.....	234
Locating Source Files.....	234
Setting Up a Project.....	235
Context-Sensitive Help Conventions.....	236
Form Help Files.....	236
Help Customization.....	237
Modifying a Help Topic.....	237
Deleting a Help Topic.....	238
Adding a Help Topic.....	239
RoboHelp Files Customized by Infor.....	242
<b>Chapter 18: Time Zone.....</b>	<b>243</b>
UI Display.....	243
Widget Timezone.....	243
BIO Attribute.....	244
User Preference.....	244

---

Database Storage.....	245
Data Field Timezone.....	245
Out-of-the-Box Behavior.....	245
Use Cases.....	246
<b>Chapter 19: Internationalization.....</b>	<b>249</b>
User Locale.....	249
Default Locale.....	249
Browser Locale.....	249
Locale User Preference.....	250
Date and Time.....	250
Format.....	250
Locale.....	251
Time zone.....	251
Daylight Saving.....	252
Numbers.....	252
Format.....	253
Locale.....	253
Currencies.....	253
Format.....	254
Currency Locale.....	255
Default Currency Locale.....	255
User Locale.....	256
<b>Chapter 20: Enhanced User Experience with Infor SoHo (UX 3.0).....</b>	<b>257</b>
Overview.....	257
Enabling Infor SoHo (UX 3.0) Features.....	258
'Placeholder' for Input Fields.....	258
Hover Image for Toolbar Button and Dropdown Menu.....	259
Configurable Display Options for Toolbar Button and Dropdown Menu.....	260
Application Top Banner.....	260
Application Top Toolbar.....	261
Application Navigation Menu.....	263
Message Dialog.....	265
Widgets and Styling.....	267
Date and Datetime Picker.....	270



## About Infor

Infor delivers business-specific software to enterprising organizations. With experience built-in, Infor's solutions enable businesses of all sizes to be more enterprising and adapt to the rapid changes of a global marketplace. With more than 70,000 customers, Infor is changing what businesses expect from an enterprise software provider. For additional information, visit [www.infor.com](http://www.infor.com).

## Purpose of this Guide

The Infor Epiphany Sales and Service Implementation Guide provides procedural information relevant to individuals involved in implementing and customizing the Infor core and the Customer Service applications built on the core.

Use this book in conjunction with the reference material available in the Reference Guides. The Architecture Reference Guide provides reference material related to the Infor core. The Application Reference Guide provides reference material related to the Infor Epiphany®, powered by Epiphany®, Infor Epiphany Sales and Service applications.

This book consists of the following chapters:

"Application Implementation and Customization" on page 17 provides information on customization high level steps, customization checklist, Delta modules, and how to use Infor Studio for a typical Infor Epiphany Service customization.

"Best Practices" on page 33 provides best practices information.

"Using Eclipse" on page 53 covers how to configure Eclipse for development.

"Source Control" on page 69 describes how to use CVS to manage your source code.

"Custom Build Process" on page 81 describes how to manage the post-installation custom build process.

"Performance" on page 99 covers performance issues related to Infor Epiphany Service implementation and customization.

"Using Out-of-the-box User Interface Templates" on page 113 describes how you can use out-of-the-box user interface templates to give the Infor look and feel for your customizations.

"Bulk Import" on page 125 covers bulk import and describes how you can create import maps and BIO transformation maps.

"BIO Externalization" on page 139 covers BIO externalization which enables you to map one or more out-of-the-box recordsets to tables in custom/legacy databases.

"BIO Transformation" on page 155 describes how to perform BIO transformation using transformation maps.

"Dialogs" on page 175 describes how to configure the Dialogs service to view dialogs from the Infor Epiphany Service application.

"The Replication Service" on page 191 describes the Infor Open Architecture Replication service.

"Using Access Control Lists" on page 199 covers how Access Control Lists are implemented in Infor Epiphany Sales and offers implementation guidelines and best practices.

"Translating Application Strings using Infor Studio" on page 219 covers how to use Infor Studio to translate application strings.

"Integrating a Request with Attachments" on page 225 describes how requests can be integrated with attachments.

"Customizing Help" on page 233 describes how to customize the application online help.

"Time Zone" on page 243 covers how to handle time zone in Infor Epiphany Service.

"Internationalization" on page 249 addresses internationalization issues in Infor Epiphany Service.

## Product Documentation

The Infor Epiphany Sales and Service product documentation includes the manuals and online help systems described in this section. For a summary of features that are new for this release, late-breaking information about installation and upgrade, and information on fixed or outstanding product issues, see the Infor Sales and Service Release Notes. For information on supported platforms, see the Documentation section of the Infor Support Portal, <http://www.inforxtreme.com>.

## Installation and Configuration Guide

The Infor Epiphany Sales and Service Installation and Configuration Guide includes all of the Infor-specific information required to get the Infor Sales and Service applications running. When special configuration is required or recommended for other platform-support software, it is also included. The audience for this book should be familiar with installing and configuring sophisticated enterprise software. They are expected to be experts in their own corporate network configurations and knowledgeable about security topics such as proxy servers and firewalls. General familiarity with database management and maintenance is also assumed.



## Implementation Guide

The Infor Epiphany Sales and Service Implementation Guide provides procedural information relevant to individuals involved in implementing and customizing the Infor Open Architecture and the Infor Epiphany Sales and Service applications built on it. Implementers typically possess expertise in lightweight Java development, HTML, DHTML, JavaScript, and SQL. They primarily work with the Infor Studio configuration tool (and to some extent with the Designer tools). The Infor Epiphany Sales and Service Mobile Wireless Implementation Guide provides additional procedural information relevant to individuals involved in implementing and customizing the Infor Epiphany Sales and Service Mobile Wireless application.

## Reference Guide

The Infor Epiphany Sales and Service Architecture Reference Guide and the Infor Epiphany Sales and Service Application Reference Guide contain technical reference information relevant to implementors involved in implementing and customizing Infor Sales and Service at customer sites. These books provide the reference context for the procedural information available in the Infor Epiphany Sales and Service Implementation Guide. The Infor Epiphany Sales and Service Mobile Wireless Architecture Reference Guide provides additional technical information relevant to individuals involved in implementing the Infor Epiphany Sales and Service Mobile Wireless application.

## Administrator's Guide

The Infor Epiphany Sales and Service Administrator's Guide provides supervisors, managers, and executives with the information to use the Customer Service and Sales Force Automation and Admin Console functionality to manage the work of their agents and salespeople. Instructions for day-to-day maintenance of the system are included in this book.

## Computer Telephony Integration Guide

The Infor Epiphany Sales and Service Computer Telephony Integration Guide (CTI) provides the overview and configuration information needed to implement and customize an Infor CTI-enabled product. Its target audience is the implementors who deploy Infor Sales and Service at the enterprise.

## Integration Guide

The Infor Epiphany Sales and Service Integration Guide provides overview and configuration information for the set of tools used to exchange data with a variety of back-end data sources, including generic

SQL sources, Java and EJB-based sources, Web services, and other database types (using Infor Enterprise Application Integration, or EAI).

## Online Help

Online help documentation for Infor Sales and Service is:

- Admin Console
- Sales
- Self-service
- Service
- IBRDesigner
- WorkflowDesigner
- DialogDesigner
- Studio
- Logviewer

These are available for users of the Infor Contact Center, Infor Sales, Infor Service, Infor Workflow Designer, and Infor Dialog Designer.

## Viewing Release Notes and Manuals

Product release notes and manuals are part of the Sales and Service package. To view product documentation open the document of interest in Acrobat Reader.

## Printing This Document

To print this document at the highest quality resolution, print to a Post-Script driver. Other drivers may not reproduce the screen shots as accurately. This document is designed to be printed on two sides of the page. If your printer is not configured for duplex printing, you may find a blank page at the end of some chapters. This is normal.

## Contacting Customer Support

You may contact the Infor Customer Support center by submitting your incident via the web 24x7 at <http://www.inforxtreme.com>, or by placing a call during our scheduled business hours. For a complete listing of our support centers with web addresses and phone numbers, access our support site at <http://www.inforxtreme.com>.

## Location of the Platform Support Matrix

For more information on platform support, including hardware and software requirements, see the <http://www.inforxtreme.com> web site.



---

# Application Implementation and Customization

## 2

This chapter addresses application implementation and customization. This is done in relation to high level steps, customization checklist, Delta modules, best practices, and using Infor Studio for a typical Customer Service customization.

This chapter assumes that you are familiar with application development and event processing and customization. For more information on these areas, see the *Infor Sales and Service Architecture Reference Guide*.

## High Level Steps

This section describes the high-level steps involved in implementing and customizing the Infor Epiphany Sales and Service applications. These steps are described in relation to an implementation for Acme Corporation - a fictitious hardware manufacturer and “acme” is used as the preferred prefix. Substitute as appropriate.

- 1 Install the Infor Epiphany Service application. For installation instructions, refer to the Installation and Configuration Guide.
- 2 Start the application server.
- 3 Start Infor Studio and point to the correct metadata database. From the Infor Studio **Go** menu, select **Configuration Properties** and setup the **Application Root URL** and the **SSO URL**, and depending on development preferences, the other properties on the screen.
- 4 Create a new module to store all security mapping information. Name this module `acme_permissions.ebm`. Set the `acme_permissions.ebm` as the Current module. Create appropriate user roles/map user profiles, and make sure that authorized users can log into the application. At this point, the Infor Epiphany Service installation is ready for customization.

All of the Infor out-of-the-box modules are marked for **Save Delta**. When modifying an existing piece of functionality/module, all of the changes get recorded as deltas to the out-of-the-box modules and can be saved as such. You can save the changes to out-of-the-box modules as delta modules. The delta modules are written out as `.ebmd` files on to the file system where they can be saved and exported using Infor Studio. For more information, see "Delta Modules" on page 21.

- 5 On the development machine, create a new folder called “acme” in the <InstallDir>\service\applications folder. The <InstallDir> in a Windows environment is typically C:\or so the above folder will be C:\Infor\service\applications\acme. (This directory is called <Acme Home> and you can set this to any location.)
- 6 Under the “acme” folder create an appropriate directory structure. Note that this is a recommended guideline to standardize organization of code in customizations and for naming conventions, directory structure, and so on. For a large scale development, write a common batch script to accomplish the following. The batch script can then be re-used by all developers on the project.

**a** Application functionality level - create folders

When extending the out-of-the-box application, create folders such as

<Acme Home>\common

<Acme Home>\sales

<Acme Home>\service

When building a new application, create folders such as

<Acme Home>\inventory

<Acme Home>\helpdesk

**b** Module functionality level - create sub-folders

For existing modules - all additional metadata for customizations to the out-of-the-box modules can be stored in modules placed in the following folder:

<Acme Home>\modules

For New Modules - all metadata for the new modules can be stored in modules placed in the following folders:

<Acme Home>\modules

Basically all additions and new modules are to go to one directory under <Acme Home>\modules

All additive changes to the out-of-the-box modules should go into the custom module such as acme\_customer\_ui.ebm. This way, the majority of the additions to the metadata can be saved and loaded as a part of the customized moduleset. So, the <Acme Home>\modules can contain:

acme\_ticket\_bio.ebm - for all bio-related metadata added for the ticket module

acme\_ticket\_ui.ebm - for all ui-related metadata added for the ticket module

All the non-additive changes (modifications made to BIO/attribute/form/widget properties, layout etc) will be recorded as ‘deltas’ and should be saved in the above created directory structure.

So, the <Acme Home>\modules folder can contain:

acme\_customer\_bio.ebm - for all additive BIO-related metadata added to the customer module.

acme\_customer\_ui.ebm - for all additive ui-related metadata added to the customer module.

customer\_bio.ebmd - for all changes made to the out-of-the-box customer\_bio.ebm module.

customer\_ui.ebmd - for all changes made to the out-of-the-box customer\_ui.ebm module.

**c** Extension functionality level - Create sub-folders such as

<Acme Home>\com\common\acme\_customer\extension\bio

<Acme Home>\com\common\acme\_customer\extension\helper

<Acme Home>\com\acme\_customer\extension\ui

<Acme Home>\com\helpdesk\acme\_ticketextension\bio

<Acme Home>\com\helpdesk\acme\_ticket\extension\ui

<Acme Home>\com\helpdesk\acme\_ticket\extension\helper

- 7** Use Infor Studio to create new modules for all the various application development entities that are identified during the requirements/design phase. It is a good idea to prefix custom modules with some kind of identifier. For example, for the ACME Corp customization, create new modules such as:

acme\_customer\_bio, acme\_customer\_ui, for appending additional metadata to existing out-of-the-box modules

acme\_ticket\_bio and acme\_ticket\_ui and so on, for adding new modules into the application.

See the section "Delta Modules" on page 21 for information on updating or deleting from existing out-of-the-box modules. Setting up the folder name for each module according to above created directory structure helps avoid the directory structure prompt each time you try to save/export a module.

- 8** Use Infor Studio to make the appropriate metadata changes such as creating new BIOs, creating new forms, and so on. Save/export those modules to the appropriate folders in the directory structure that you created. To assist with an easier upgrade to a future version of the application, make all additive changes into the custom module such as acme\_customer\_ui.ebm. That way, majority of the additive changes to the metadata can be isolated and tracked and this helps with an easier upgrade path.
- 9** Run scrutiny and scrutiny\_fkchecks from the command line or Infor Studio to verify that the metadata changes are valid and all modules have been loaded. Fix any errors that may be reported.
- 10** From the command line or Infor Studio, run reloadmeta.
- 11** Test your application to verify that all customizations work as you intended.
- 12** Save all modified modules to their respective folders.
- 13** Check in the .ebm and .ebmd files into the source code control system.
- 14** For more advanced customizations involving extensions, check to see if the out-of-the-box extension provides any parameters to customize the out-of-the-box behavior. If the answer is yes, change the appropriate parameters to achieve the required results. If there are no such parameters, then
- a** Disable the out-of-the-box extension, for example, `SomeOOBExtension`
  - b** Register a renamed and prefixed copy of the extension in Infor Studio with the exact same Event Category and attach this new extension to the relevant object for the same Event/Event Category. For example, register `ACMESomeMoreExtension` and attach it to the same bio/form/object for the exact event/category with the same priority of the disabled `SomeOOBExtension`.  
  
The package name is determined from the directory structure created above. If the package name for `SomeOOBExtension` is `com.epiphany.common.extension.bio`, then the package name for the new extension would be `com.acme.common.customer.extension.bio`
  - c** In your chosen IDE, create the new Java extension and place it in the appropriate folder and assign the appropriate package name. In the class path for the application server, include the destination for the compiled files. For more information on setting up Classpath, see the Infor Sales and Service Architecture Reference Guide.
  - d** If you install in Development Mode, `<InstallDir>\service\lib\custom` is included in the class path by default and all classes compiled into the custom folder will be loaded as long as the path and the package name match.

**15** Check in any extension source code into a source code control system.

- a** Disable the out-of-the-box extension, for example, `SomeOOBExtension`
- b** Register a renamed and prefixed copy of the extension in Infor Studio with the exact same Event Category and attach this new extension to the relevant object for the same Event/Event Category. For example, register `ACMESomeMoreExtension` and attach it to the same bio/form/object for the exact event/category with the same priority of the disabled `SomeOOBExtension`.

The package name is determined from the directory structure created above. If the package name for `SomeOOBExtension` is `com.epiphany.common.extension.bio`, then the package name for the new extension would be `com.acme.common.customer.extension.bio`

- c** In your chosen IDE, create the new Java extension and place it in the appropriate folder and assign the appropriate package name. In the class path for the application server, include the destination for the compiled files. For more information on setting up Classpath, see the Infor Sales and Service Architecture Reference Guide.
- d** If you install in Development Mode, `<InstallDir>\service\lib\custom` is included in the class path by default and all classes compiled into the custom folder will be loaded as long as the path and the package name match.

**16** As a part of unit testing, on each developer machine, do the following:

- a** Stop the application server.
- b** `dbinit -replacedb -metaAdminUser <adminuser> -metaAdminPwd <adminPwd> -srcAdminUser <sourceAdminUser> - srcAdminPwd <sourceAdminPwd>`
- c** `loadmodule -application Service -application Sales -application Self-Service`
- d** `loadmodule -module <Acme Home>`
- e** Start the server

**17** Deploy the application into a test/staging environment. For more information on deployment, see the Infor Sales and Service Architecture Reference Guide.

## Customization Checklist

Use the following checklist to assist the customization process.

- 1** When you perform `dbinit`, make sure that no user is connected to the database and stop the application server.
- 2** For WebSphere, undeploy `service.ear` (uninstall the deployed application from the WebSphere console).
- 3** Backup files if needed.
- 4** Cleanup files (installedApps, temp, and so on).
- 5** For WebSphere, verify to see leftover objects (for MQSeries or connection pool) in WebSphere repository.
- 6** If required, update `config.xml` or `deployment_config.xml` or `sso_config.xml`
- 7** Run `servicegen`



## 8 For WebSphere, run admin\_deploy

# Delta Modules

The Open Architecture (OA) metadata is permanently represented in a module file. A module file is an XML document that contains all the rows in the metadata tagged as belonging to a particular module.

When a change is made to rows in the metadata that are a part of the module, the entire module file is written to disk. When the module is loaded, all rows that belong to the module are first deleted, and then all rows are inserted.

The delta module file allows you to write only the changed rows into a separate file. The delta file exists as an amended version of the original module file. This helps solve issues such as

- Simplify preservation of customer changes (customizations) after an upgrade. After an upgrade, customers who have modified the out-of-the-box modules will be able to load the new, Infor modified modules, and then apply their delta modules.
- Module load/save time. Small changes to a module (for example, a few labels) will produce a very small delta module file which will load and save faster.
- Deployment unit size. Any change to a module requires that the entire module is packaged into a deployment unit. Most of the time, a user only changes a small number of items in the module relative to the number of total objects stored in the module. The delta reduces this significantly.

Modules marked with the “Save Delta” attribute automatically save the changes to the module in delta format, and not the entire module. By default, all Infor provided out-of-the-box modules are marked with this attribute at each release point. This means that in Infor Studio, you save delta modules by default.

**Note:** Module deltas, if saved using the command line, are cumulative. There can only be one delta module for each module and that delta module will contain all changes based from the base module. It is important that in a development environment restrict the delta module from being modified by more than one user at the same time. Using a source control system is the best way to achieve this.

For Delta modules saved using the command line (cumulative save), each save writes all the changes to the module. This includes changes made in the previous delta files up to the point from the base module itself. Once you save your delta module, you no longer need the old delta modules as all cumulative changes are made.

An object in the metadata owned by a module marked “Save Delta” cannot have its module changed (the app\_module\_id). Infor Studio prevents the user from doing this if either the module that you are changing from or the module you are changing to is marked as “Save Delta.”

Attempting to save a delta file via the command line when the Save Delta flag is not set is disallowed: in other words, if you want to save out an .EBMD file, the save delta flag for that module must be set. The module loader will not load an .EBMD file with the Save Delta flag set in Studio (where .EBMD options are not even shown in the file selector WHEN the Save Delta flag is set). However, the command line version will allow you to load an .EBMD file even though the Save Delta flag in Studio is set to true.

## Guidelines

If you make changes to the out-of-the-box modules then use delta modules. Continue to create all completely new objects in a new, custom module.

- Do not mark your new (custom) modules as “Save Delta.” Continue to save your custom modules as normal, full modules.
- When you load modules, first load the out-of-the-box modules, and then load your changed delta modules as well as the custom modules.
- You have two choices as to how to capture your custom changes to the out-of-the-box modules.
  - If your changes to the out-of-the-box modules are small and can be re-implemented, re-implement your changes using Infor Studio. This way, you will have good delta modules to use from this point forward.
  - You can use the application diff tools to perform a complete upgrade. Note that the application diff (application upgrade) tools are only available (supported) starting with 6.5.3.

## To Undo a Change

To undo a change to the metadata that you made on a module that is marked as “Save Delta,” load the original module (available through the right mouse menu on the Modules list or through the command line).

## Best Practices

- Keep the out-of-the-box modules of your implementation in a known location.
- Do not mark your custom modules as “Save Delta.”
- Create new objects in new custom modules.
- Check-in the delta modules for modules marked “Save Delta” and not the original (out-of-the-box) modules.

## Delta modules and EDialogs

The Delta module does not contain any changes or additions to EDialogs. Infor Studio generates the Delta module based on the transactions recorded in the `tsy_transactions` table. EDialogs does not have the capability to write entries to the `tsy_transactions` table because EDialogs does not use the BIO layer for read or write operations. Therefore, no additions/changes to EDialog are included in the Delta module generation.

## CreateDelta Tool

The CreateDelta tool can be used in any environment where customization is involved. This tool compares an unmodified module with a modified module (ebm files), and, based on the difference, creates a Delta module (ebmd file) which contains only performed modifications. CreateDelta tool is a command utility which should be run from <Installed Dir>\shared\bin. To see the usage of this command type createDelta -?

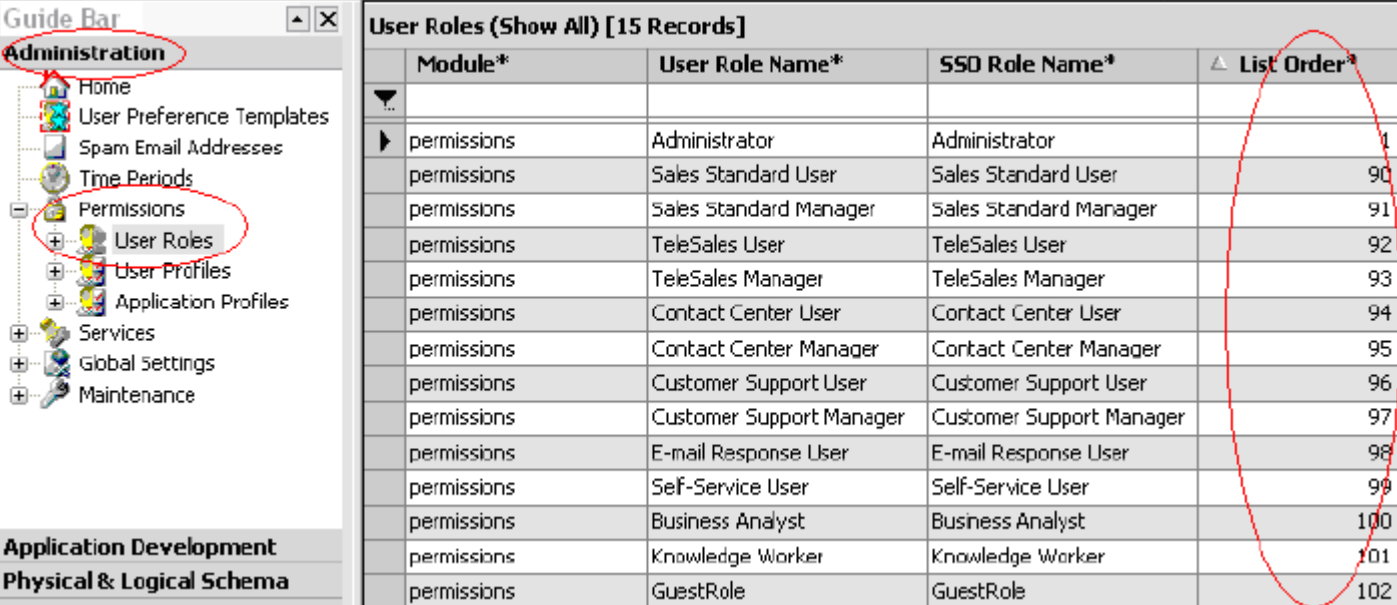
This section outlines the steps required before loading the delta module, which is created by using CreateDelta tool before upgrading to the latest S&S version.

A given customer would most likely customize the installed CRM environment. In a situation where the customer did not follow best practices (where changes made to OOTB modules were not saved as delta modules), the createDelta tool can be used to extract the customizations out of the OOTB modules as delta modules. This extraction will allow the upgrade process to run more smoothly.

**Note:** Loading Delta modules created in an upgraded environment has a limitation which could mainly affect the permission module. The User Role's List Order should not have data that conflicts with the target environment. Use the following procedure below to avoid this problem.

## Checking Existing User Roles in Studio

Open Infor Studio to check the current User Roles in the target environment.



Module*	User Role Name*	SSD Role Name*	List Order*
permissions	Administrator	Administrator	1
permissions	Sales Standard User	Sales Standard User	90
permissions	Sales Standard Manager	Sales Standard Manager	91
permissions	TeleSales User	TeleSales User	92
permissions	TeleSales Manager	TeleSales Manager	93
permissions	Contact Center User	Contact Center User	94
permissions	Contact Center Manager	Contact Center Manager	95
permissions	Customer Support User	Customer Support User	96
permissions	Customer Support Manager	Customer Support Manager	97
permissions	E-mail Response User	E-mail Response User	98
permissions	Self-Service User	Self-Service User	99
permissions	Business Analyst	Business Analyst	100
permissions	Knowledge Worker	Knowledge Worker	101
permissions	GuestRole	GuestRole	102

## Checking Delta Module Permission

- 1 Open `permission.ebmd`, which is created by the `createDelta` tool, with any editor.
- 2 Search for `"name="user_role_listorder"`. The following is a sample xml piece:

```
<object type="user_role" primary-key="user_role_id" primary-key-
value="276709B5E5E2414BAB7AFA49B61F9630" >
  <attribute name="masked_attributes" value="0" />
  <attribute name="sso_role_name" value="uid=testuser02,ou=peo-
ple,dc=epiphany,dc=com" />
  <attribute name="system_profile_flag" value="0" />
  <attribute name="user_role_id"
value="276709B5E5E2414BAB7AFA49B61F9630" />
  <attribute name="user_role_listorder" value=" " />
  <attribute name="user_role_name" value="uid=testuser02,ou=peo-
ple,dc=epiphany,dc=com" />
</object>
<object type="user_role" primary-key="user_role_id" primary-key-
value="4E8876B12E7B48D0970AE2B1612E9A30" >
  <attribute name="masked_attributes" value="0" />
  <attribute name="sso_role_name" value="uid=testuser01,ou=peo-
ple,dc=epiphany,dc=com" />
  <attribute name="system_profile_flag" value="0" />
  <attribute name="user_role_id"
value="4E8876B12E7B48D0970AE2B1612E9A30" />
  <attribute name="user_role_listorder" value=" " />
  <attribute name="user_role_name" value="uid=testuser01,ou=peo-
ple,dc=epiphany,dc=com" />
</object>
<object type="user_role" primary-key="user_role_id" primary-key-
value="7631D176069D42FB90D7F5E3EF5DFC2D" >
  <attribute name="masked_attributes" value="0" />
  <attribute name="sso_role_name" value="uid=testuser03,ou=peo-
ple,dc=epiphany,dc=com" />
  <attribute name="system_profile_flag" value="0" />
  <attribute name="user_role_id"
value="7631D176069D42FB90D7F5E3EF5DFC2D" />
  <attribute name="user_role_listorder" value=" " />
  <attribute name="user_role_name" value="uid=testuser03,ou=peo-
ple,dc=epiphany,dc=com" />
</object>
<object type="user_role" primary-key="user_role_id" primary-key-
value="EF40030747B94F3284C286C926FA8D9E" >
  <attribute name="masked_attributes" value="0" />
  <attribute name="sso_role_name" value="uid=demo,ou=people,dc=epiph-

any,dc=com" />
  <attribute name="system_profile_flag" value="0" />
  <attribute name="user_role_id"
value="EF40030747B94F3284C286C926FA8D9E" />
  <attribute name="user_role_listorder" value=" " />
  <attribute name="user_role_name" value="uid=demo,ou=people,dc=
```

```

epiph-
  any,dc=com" />
</object>
</module>

```

- 3 Make sure none of the values for "user\_role\_listorder" in permission.ebmd are in the target environment in Studio.

## Creating a New User Role in the Target Environment

It may happen that the customer would like to test the target environment before loading the delta module. In some cases, a new user role may need to be created. If a new user role is to be created, the default list order assigned by Studio could be 103, or the number next to the current largest list order (i.e., if the largest list order is 105, the auto assigned one will be 106). Make sure the assigned list order number is not used in the permission.ebmd.

If that number is used in the permission.ebmd, change that number to avoid the conflict. One common way is to change to a high end list order like 2000+, or use the numbering gap between 10 to 80.

As shown in the following figure, change the auto assigned 103 to 2003 and save it.

User Roles (Show All) [15 Records]				
	Module*	User Role Name*	SSO Role Name*	▲ List Order*
▼				
	permissions	GuestRole	GuestRole	102
	Perm_test	uid=dan,ou=people,dc=ssainternal,dc=net	uid=dan,ou=people,dc=ssainternal,dc=net	103

## Using Infor Studio for Application Customization

This section describes how you can use Infor Studio for a typical out-of-the-box Customer Service customization. A typical customization involves using Studio to create a data source, create a recordset, define relationships between recordsets, create a BIO, create a form, and integrate the form with the Sales detail or list form. These concepts are defined first and this is followed by procedures.

### Data source

A Data source stores information about the databases and services from which you access business data. Each data source object contains one or more recordsets. Typical data sources include SQL and Enterprise Applications Integrations (EAI) such as EJB, JMS, HTTP, Java and Web Services. Each

data source type has a corresponding data provider implementation that allows the application server to communicate with the data source.

Implementations typically involve adding additional data sources when trying to access (read/write) data from sources other than the default Infor- created source database. The Infor Open Architecture does not support distributed transactions and you have to take this into consideration when doing the database design.

## Recordset

A recordset is the metadata representation of the physical table in a data source. It is equivalent to a physical table in a data source. Each recordset contains one or more fields that contain business data or references to other recordsets in your system.

A recordset can have one or more field sets, one of which must be the primary key field set. The primary key field set has a unique value for every record in the recordset. Primary fieldsets are typically single and automatically-generated by the (application code and not by database), even though composite primary keys (keys that are made up of more than one field) are also supported. Having a read-only recordset on a database view is also supported although having such recordsets on very complex views can lead to performance issues.

## Relationships Between Existing Recordsets

You have to map all relationships defined in the physical database into the metadata. These relationships can be one-to-one or many-to-one. Defining and representing relationships in metadata allows you to reference data belonging to other BIOs easily using relationship-mapped and path-mapped attributes.

## BIO

A BIO is an entity representing a logical application object which can be associated with one or more recordsets directly or indirectly. For more information on BIOs and BIO attributes, refer to the BIO Programming chapter.

## Form

Information is displayed to the user through forms. Forms are of two types - BIO Forms and Special (Other) Forms. A form is uniquely identified by a UI Handler. A UI Handler is either a pointer to a BIO class and perspective or to a Special Form. BIO Forms are associated with a particular BIO class and a perspective. The out-of-the-box application ships with some commonly used perspectives such as

“detail view”, “list view”, “qbe view”, and “push view.” You can add additional perspectives using Infor Studio (**Guide Bar > User Interface > Maintenance**).

On BIO Forms, forms are associated with BIOs and individual widgets are associated with BIO attributes. You can display attributes from different BIOs on a single form by using path-mapped, indirect mapped, query-mapped or computed attributes. Multiple fields from another BIO can be displayed by using form slots within forms and having BIO-Reference Paths defined for the form slot. For example, to display a list of requests for a customer, you have to define a form slot on the **Customer Detail** form. The form slot definition would have a BIO Path of customer::requests and the perspective would be “list view.”

Special forms are used when the form is not associated with any particular BIO. Some examples of special form types are blank form, Modal Footer Form, Error Form, and so on.

To create standalone forms for the application, in Studio, navigate to **User Interface > Forms**. To integrate forms into the Sales and Service shell, see the next section.

## Integrating a form to the Sales Detail Shell and the Sales List Shell

The Sales\_Detail\_Shell and the Sales\_List\_Shell provide an easy-to-integrate framework across various screens in the application to provide an effective user interface (streamlined navigation for common use cases, shorter navigation paths, reduced scrolling, and fewer screen pops/transitions). From a developer perspective, the shell obviates the need to define redundant metadata at multiple locations.

The *Infor Studio Form Wizard* (available from the Infor Studio Home Page) automates the task of integrating new forms (detail/list etc created for new BIOs) into the existing UI setup and helps create new forms within the shell. The *Infor Studio Form Wizard* is a user-friendly guided interface that enables you to build new list/detail forms and integrate them with the Sales List/Detail Shell.

The following section describes the high level procedures associated with a typical customization. These include

- "Creating a Data Source" on page 27
- "Creating a Recordset" on page 28
- "Defining a Relationship Between Recordsets" on page 29
- "Creating a BIO" on page 31
- "Defining A Form and Integrating The Form With The Sales Detail Shell" on page 31

Refer to the Infor Studio online help for complete detail on using these procedures.

## Creating a Data Source

To define a new data source you can use the Data Source wizard or use the following steps:

- 1 From the Infor Studio *Guide Bar*, under **Physical and Logical Schema**, click **Data Sources**. The **Data Sources** form appears.
- 2 In the **Data Sources** form, add a new entry in the row demarcated by an asterisk. In that row, specify a value for the following parameters.

Module	The module to which the data source belongs.
Name	The name of the data source.
Data Source Type	The type of data source that is being defined.
Transaction Methods Class	The data provider class that implements the - begin, commit, and abort transaction methods for this data source. Leave this value blank if the data source does not support transactions.
Code Page	For SQL data sources, the code page of the data in the data source. If this value is not specified, the default code page for your machine is used.
Attributes	A list of all of the attributes that have been set. If the list is blank, none of the following attributes are set: Supports Dialogs, Supports Lookups, Supports PIM Sync, and Primary Data source.

- 3** In the **Properties** area, specify the applicable connection information for the data source and Save. You can skip this procedure when the application does not have to access multiple databases (from legacy systems) to access and update data. For creating Data Sources to access EJB/JMS/HTTP/Java/Web Services Data sources, refer to the Integration Guide .

## Creating a Recordset

From the Infor Studio *Guide Bar*, under **Physical and Logical Schema**, click **Recordsets and Relationships**. The **Recordsets** form appears

- 1** From the Infor Studio *Guide Bar*, under Physical and Logical Schema, click Recordsets and Relationships. The Recordsets form appears
- 2** In the **Recordsets** form, add a new entry in the row demarcated by an asterisk. In that row, specify a value for the following parameters.

Module	The module to which the recordset belongs.
Data Source	The data source to which the recordset belongs.
Physical Name	The name of the physical entity in the data source that corresponds to the recordset. Choose from selected data source's list of tables and views.
Name	The unique name of the recordset as defined in Studio.
Locking Strategy	Specifies the method by which optimistic locking is performed for data in this recordset.
Fetch Method Class	The data provider class that implements the Fetch() method.



Query Method Class	The data provider class that implements the Query() method.
Update Method Class	The data provider class that implements the Update() method.
Insert Method Class	The data provider class that implements the Insert() method.
Delete Method Class	The data provider class that implements the Delete() method.
Auto Primary Key	If selected, a primary key is automatically generated for a new record and users do not have to specify a unique identifier. If selected, you must also specify a value for the Sequence Name property in the Properties area of the Studio application window.
Supports Logical Delete	If selected, this recordset has exactly one field that specifies whether a row has been deleted. This field is specified by selecting the Logical Delete option for one field in the Fields subform.
Logical Delete	This option is only applicable when Supports Logical Delete is selected. If selected, applications can mark rows in this recordset as deleted rather than physically deleting them from the recordset.

- 3 Click **Save**. The fields and fieldsets of this recordset are automatically created. When this process is finished, a dialog box appears prompting you to automatically create a BIO for the recordset. Click Yes to automatically define such a BIO class.
- 4 In the **Properties** area, specify field properties for each field in your recordset and save.

## Defining a Relationship Between Recordsets

- 1 From the Infor Studio *Guide Bar*, under **Guide BarPhysical and Logical Schema**, click **Recordsets and Relationships**.
- 2 Click **Relationships**. The **Relationships** form appears.
- 3 Scroll to the bottom of the **Relationships** form and select the last, empty row.
- 4 Right-click the **Relationship Editor** subform of the **Relationships** form, and choose **Select Left Side Table**. From the **Recordset Names** dialog, choose the recordset that should be on the left side of the relationship.

If this is a many-to-one relationship, this must be the “many” side of the relationship. For example, in a many-to-one relationship of users and groups, the Users recordset must be on the left side.

- 5 Right-click the **Relationship Editor** subform and choose **Select Right Side Table**. From the **Recordset Names** dialog, choose the recordset that should be on the right side of the relationship.
- 6 Select the field set from the left-side recordset that joins with a field set on the right-side recordset. Draw a line from this field set to the appropriate set on the right-side. Make sure that you draw the relationship line from left to right.
- 7 Right-click the line between both recordsets and choose **Relationship Properties**. Choose the appropriate settings for the relationship.

Module	The module to which the relationship belongs.
Role Name	The name of the relationship. This name is <leftRecordset>_<rightRecordset>_rel, such that leftRecordset is the recordset on the left side of the relationship and rightRecordset is the recordset on the right.
Left Unjoined?	If selected, unjoined rows can exist on the left side of the relationship. Data providers require this information to determine whether to perform an inner or right-, left-, or full-outer join (or the equivalent operation for non-relational-database data sources).
Right Unjoined?	If selected, unjoined rows can exist on the right side of the relationship. Data providers require this information to determine whether to perform an inner or right-, left-, or full-outer join (or the equivalent operation for non-relational-database data sources).
Is One_One?	If selected, the relationship is one-to-one. If not selected, the relationship is many-to-one (where the left is always the many side of the relationship).
Enforced Constraints	If selected, Infor Studio performs insert and delete operations for both sides of the relationship in an order that enforces foreign-key constraints in your data source. If not selected, Infor Studio performs these operations in an arbitrary order.

- 8 (Optional) Add one or more constraints to this relationship in the Relationship Constraints subform. For more information, refer to Architecture Reference Guide.
- 9 Save your work.

## Creating a BIO

You can create a BIO automatically when you create new recordset, by using the BIO wizard or by using the following procedure.

- 1 From the *Infor Studio Guide Bar*, under *Physical and Logical Schema*, click **BIOs**. Under **BIO Related Info**, click either **Main** or **All BIOs**. The **BIO** form appears.
- 2 In the **BIO** form, add a new entry in the row demarcated by an asterisk. In that row, specify a value for the following parameters.

Module	The module to which the BIO class belongs.
Name	The unique name of the BIO class.
Data Source Name	The name of the data source that provides information that is stored in this BIO class.
Recordset Name	The name of the primary recordset with which the BIO class is associated. The primary key of this recordset uniquely identifies a BIO within this class. Note: If this class defines a BIO interface, this value must be Null.
Creatable	If selected, applications can create BIOs of this type. Note: If Creatable is selected, the primary recordset associated with this BIO must have an Insert Method specified.
Row Level Audit	If selected, information about updates to instances of this BIO are stored in your database.

- 3 Right-click the new BIO row and select **Create BIO Attributes**. Choose the fields that you want to create and click **OK**. You can also define **BIO attributes** manually.
- 4 In the **Properties** area, set properties for each attribute.
- 5 Define default display attributes for this BIO. Right-click the new BIO class row and select **Create BIO Display** and **Display Attributes**. When you define display attributes, the assigned widget types are based on the BIO attribute types.
- 6 Click the **Extensions** tab. The **Extensions** sub form appears. In the **Extensions** subform, assign one or more extensions to the BIO, if necessary.
- 7 Click the **Permissions** tab. In the **Permissions** subform, configure permissions for the BIO and save.

## Defining A Form and Integrating The Form With The Sales Detail Shell

Use the *Infor Studio Form Wizard* available from the Infor Studio home page (**Home > Create a New Form**) to create new forms that can be added to the application. The Form Wizard is a user-friendly guided interface on how to build new list/detail forms and integrate them with the Sales List/Detail Shell.



This chapter describes best practices related to implementing the Infor core architecture and the applications built on the core.

The chapter addresses best practices in relation to

- "Application Implementation/Customization" on page 33
- "Alerts" on page 36
- "Configuration" on page 37
- "Data Migration" on page 37
- "Databases" on page 39
- "Deployment" on page 41
- "Dialogs" on page 42
- "Knowledge Management System" on page 43
- "Other" on page 43
- "Upgrade" on page 45
- "Bulk Import" on page 45
- "Delta Modules" on page 21

## Application Implementation/Customization

- Save all customized metadata into new module files. Refer to the guidelines provided in "Application Implementation and Customization." on page 17
- Use separate development, test, and production instances, each with separate metadata and operational databases.
- For parallel implementation efforts, each developer/implementer should have his or her own metadata and operational databases and share work with others through module files (.ebm files).
- Plan the division of the project into modules and tasks so that different developers do not need to work on the same module at the same time.

- Do not modify system modules. The core architecture is tightly coupled with this metadata and changes to the metadata will most likely break major functionality in the core. As a result, application functionality will also be broken.
- Document changes that you make to the shipped application modules (for example, if you make any changes to the Customer module labels, document the changes).
- For building new functionality in the application, create new module(s) to represent that functionality. For information on application module guidelines, EpiExtensions, and Java extensions, see the *Architecture Reference Guide*.

**Note:** The use of EpiExtensions is deprecated. Using Java extensions is now the preferred mechanism.

- Keep modules small (under 5 Mbytes) and focused on one subject area. Keeping modules small also helps during upgrade.
- For upgradability, do not modify the behavior of the extensions delivered with the application. If you have to modify the out-of-the-box extensions, copy the extension and re-name, rather than editing the out-of-the box extension.
- When changing or extending the behavior of the application, supplement existing logic by adding additional extensions (either EpiExtensions or Java extensions) to the event handlers rather than modifying or replacing the existing extensions. For more information on extension programming guidelines, see the *Architecture Reference Guide*.
- Keep new EpiExtensions and Java extensions small and single purpose. This promotes reusability and modularity.
- Use EpiExtensions rather than Java extensions for small pieces of logic.
- Provide a descriptive comment on every extension in the module explaining it's behavior and usage.
- If the extension makes API calls, review each of the exceptions and make sure you handle them in an appropriate fashion. For exception handling guidelines, see the *Architecture Reference Guide*.
- For extensions that make references to BIO class names or their attributes, externalize the names by adding the names to the constants directory.
- When you make modifications to BIO extensions, you do not have to re-start the server to see the changes take effect. If you installed in development mode, the auto-reload-extensions flag in config.xml is on and it governs the reloading of extensions. Make sure the path to the custom extensions is added to both the jvm classpath, and to the com.epiphany.shr.extensions.ClassPathParser.class.path variable.
- For internationalization purposes (if the application is to be translated), separate out error text or UI-visible text from the extension code. Add this text to the metadata string tables and use the string retrieval API in the extension code.
- Avoid re-naming the out-of-the-box attributes and BIOs. Instead, change the labels used on the corresponding widgets in the forms.
- For navigations, minimize the use of Screen type navigations for performance reasons. Use Self and FormSlot navigations as often as possible for quicker UI refreshes.
- Use source control to avoid losing some or all of the source code and confusion regarding version. Have a backup strategy during upgrades.
- When you make changes to a .css file, clear your browser cache. Internet Explorer may use the old version of the file even if a newer version exists on the server. To avoid this Internet Explorer issue,

after making changes to a .css file, clear your browser cache in Internet Explorer (**Tools > Internet Options > General > Temp Internet Files > Delete Files**).

- Reload your customization module using the command line command and not using Infor Studio. The command line option upgrades the module to match the schema change. For more information on using the command line command, refer to the *Installation and Configuration Guide*.
- If an extension writes raw HTML into an HTML widget, write it in such a way that it exists as one snippet in a larger HTML page and not assume that it is the entire page.

The HTML widget allows writing almost arbitrary HTML on to a form. There are a few restrictions that are applicable. First, the widget's specified HTML is written out as part of the larger application HTML document. Therefore, the widget's HTML should be an HTML snippet, not a fully-specified document, and should not, for example, include `html` or `body` tags. Second, whenever specifying script tags to include custom Javascript, make sure that you set the `defer` attribute to true, for example, `<script language="Javascript" defer="true">...</script>`.

- When the **Display Count Type** property is set to **Estimated Count**, **max results count** defines how many records are requested to generate the estimate. For example, if the **list window** is 10 and **max results count** is 20, the system asks for the top 21 rows. If 21 rows are returned, the first 10 are displayed and the toolbar will read (1-10 of 20+).
- You can use the `metadata.memory.profiler` and `debug.memory.leaks` properties to identify certain common sources of memory leaks in custom extensions.

You can set the `metadata.memory.profiler` and `debug.memory.leaks` properties in the `installer.properties` (for WebLogic), `installer_<serverName>.properties` (WebLogic cluster and WebSphere), under `<epnyInstallDir>/shared/etc`.

The `metadata.memory.profiler` and `debug.memory.leaks` properties enable you to identify the source of two of the most common types of memory leaks introduced by custom extensions in an Infor application: keeping handles on the Metadata object after a metadata refresh, and BioServices that are not explicitly cleaned up.

Any time a BioService is created, but `BioService.cleanup()` is not called, a message from the `com.epiphany.shr.sf.util.EpnyStatefulClient` class starting with "Memory leak of a BioService instance detected." is logged at the ERROR level. When the property `debug.memory.leaks` is not set, the message code will be

"EXP\_BIOSERVICE\_MEMORY\_LEAK\_NO\_STACK\_TRACE", and the log message will indicate that setting the `debug.memory.leaks` property to true will log the stack trace of the source of the leak. When the property is set to true, the message code will be

"EXP\_BIOSERVICE\_MEMORY\_LEAK" and the log will include the stack trace from when the offending BioService was created. In the stack trace, the top-most non-core class will typically indicate an extension instantiating a BioService and either not calling `BioService.cleanup()` or not calling it in a "finally" block after instantiating it in the corresponding "try" block.

The expected usage is that the property be not set to true, but if a customer sees memory leak error messages in the log, set the property to true to identify the custom code responsible. This is because, like enabling additional logging, leaving the property on true incurs some performance overhead.

Handles on Metadata objects are a much less common problem. Unlike the BioService leak detection, messages are only logged when `metadata.memory.profiler` is true. In that case, the `com.epiphany.shr.metadata.Metadata` class will log INFO messages with the message key "LOG\_METADATA\_MEMORYLEAK", starting with "Possible Metadata Memory Leaks(s):" if a handle to a Metadata object is held for more than five minutes after a refresh. The messages will indicate the stack trace from when the handle was first fetched. Again, there is some performance overhead

from setting this property to true, so it should only be used when debugging possible leaks when refreshing metadata.

- The CSS Classes form (available from Infor Studio **Guide Bar > User Interface > Maintenance > CSS Styles > Classes**) lists the CSS classes that are used by form widgets. A CSS class is a named style defined in a .css file. For example, you can define a class named myRedClass in a style sheet as follows:

```
...myRedClass      { color: red; }
```

You can then use this class in a form widget, form, or form slot property.

Core CSS classes are defined in master.css and related CSS files:

- masterHigh.css (only used for high end applications)
- masterPortal.css (only used for portal applications)
- \*IE.css (used with Internet Explorer 5.5+)
- \*IE4.css (used with Internet Explorer 4 or 5)
- \*NN.css (used with Netscape Navigator)

Do not directly modify master.css. Changes to master.css can result in upgrade issues. Instead, redefine classes that are listed in master.css in custom.css files.

Add new classes to custom.css and overwrite existing classes by copying them from master.css into custom.css and modify it there. Infor also recommends that you keep your custom.css in a source control system and re-apply (copy back into the file-system: `.../shared/webroot/app/css` after an upgrade).

## Alerts

- When an alert event has an associated restriction expression, the values used in the expression must transition from being false to true for the alert event to execute. The BIO keeps track of old values, from before a change was made, and compares each to the new value. A common mis-use of event restrictions is to have an expression along the lines of “if task.notes contain 'Infor'”. This works if the notes initially do not contain the string 'Infor' and then the string is later added. However, this works only once. What is required is an expression such as “if lead.value > 10000000 “then, each individual's subscription would have a filter expression such as “if lead.customer == '<name of company>’”. This way, the alert event transition is when the value crosses a line and then you subscribe to the event and filter on the companies in which you are interested. If this is not noted, alerts can have an impact on performance. Every time a BIO with an attached alert event is saved, it triggers the event mechanism. When an event is executed, all subscriptions on that alert event need to be checked. This many-to-one relationship means that the processing for a single alert event actually expands by the number of subscriptions (assuming the alert event trigger expression evaluates to true). Therefore, it is better to perform the basic logic in the alert event trigger expression so that the user's subscription filter expressions can be simple.



- To create alerts on new BIOs, make sure that you provide a label to the BIO that you create. Once you create a BIO and give it a label, run either **reloadmeta**, or re-start the server. This ensures that the BIO name appears in the drop-down list in the application. You can then proceed to set up an alert on the new BIO.

## Configuration

- Infor stores some of its configuration files as XML files in the <InforInstallDir>/shared/etc folder. The two configuration files are deployment\_config.xml and config.xml. Typical configuration information found in the deployment\_config.xml file includes database connectivity information including the encrypted password to the metadata, settings for auto reload classes, settings for onehostonly, and some settings for the ECS soap router. Config.xml contains information used to manage the Infor core.

Information in deployment\_config.xml overrides information in config.xml. Therefore, changes to configuration parameters in config.xml alone may not have any effect. The installer writes configuration information to the deployment\_config.xml file and does not modify the config.xml file. However, a core only upgrade replaces the config.xml file with a newer version. If configuration changes are to be retained across core-only upgrades, make these changes in the deployment\_config.xml file.

- If you do not have a valid LDAP/ADS server available at installation time and you encounter SSO related error messages during installation, click through the error messages to continue with the install. Later, edit sso\_config.xml to set the correct values. Check sso\_config.xml (shared/etc/sso\_config.xml) to make sure that the values that you entered when you edited the file are correct (for example, values for serverName, serverPort, user\_base\_dn etc).
- To bypass login to the application where the login screen is not displayed, set the "FrontEnd" property in the deployment\_config.xml (shared/etc/deployment\_config.xml) to true.

```
<config:category name="FrontEnd">
  <config:configParam name="sso-passthru-enabled" value="true"/>
</config:category>
```

Setting the FrontEnd property in the deployment\_config.xml ensures that it is not overwritten after a core-only upgrade.

## Data Migration

When you try to migrate data from an operational database of a previous release to a later release, regardless of the approach that you take for data migration (for example, EPNY OP upgrade scripts, EPNY import service plus BIO to BIO transformation, or MS DTS), you may run into some issues. For example, assume that Customer A migrates thousands of knowledge\_document records from an operational database of a previous release to a later release. However, knowledge\_document.title, a required field on Knowledge Document detail form in the previous release does not have a value in

the later release's operational database. Therefore, when the user opens a migrated knowledge document record, the knowledge document title is empty. The user has to populate the title field before saving the knowledge document record to the database again.

To avoid this issue, run the following SQL against your target metadata. This SQL informs you the widgets that are required on the user interface but are nullable in the database. With this information, you know which non-required data fields (in the database) need to be migrated as well. Note that the SQL listed below is generic and can be used in any upgrade.

```
/*-----*/
SELECT a.form_widget_name, b.form_name, dr.data_recordset_name,
       df.data_field_name, am.app_module_name
FROM form_widget a, form b, app_module am, bio_display_attribute bda,
     bio_attribute ba, bio_class bc, data_recordset dr, data_field df
where a.form_id = b.form_id and a.form_widget_id in ( select
object_id from property_value where parameter_id in ( select
parameter_id from parameter where parameter_name = 'input type' and

object_type_id = 'widget_type')
and display_text = 'required'
and object_type_id = 'form_widget')
and b.app_module_id = am.app_module_id
and a.bio_display_attribute_id = bda.bio_display_attribute_id and
     bda.bio_attribute_id = ba.bio_attribute_id and ba.bio_class_id =
     bc.bio_class_id and bc.data_recordset_id = dr.data_recordset_id and

df.data_recordset_id = dr.data_recordset_id and
ba.bio_attribute_name = df.data_field_name and ba.mapping_type =
'DIRECT' and df.required_flag = 0 order by dr.data_recordset_name;
/
*****/
```

After data migration, use the following SQL to verify whether those non-required data fields are null in the source database. This SQL will show the number of rows in a table in which a non-required data field is null.

```
/*-----*/
SELECT 'SELECT COUNT(*) FROM ', dr.data_recordset_name, ' WHERE ',
       df.data_field_name, ' IS NULL ' FROM form_widget a, form b
, app_module am, bio_display_attribute bda, bio_attribute ba,
     bio_class bc, data_recordset dr, data_field df where a.form_id =
b.form_id and a.form_widget_id in ( select object_id from
property_value where parameter_id in ( select parameter_id from
parameter where parameter_name = 'input type' and object_type_id =
'widget_type')
and display_text = 'required'
and object_type_id = 'form_widget')
and b.app_module_id = am.app_module_id
and a.bio_display_attribute_id = bda.bio_display_attribute_id and
     bda.bio_attribute_id = ba.bio_attribute_id and ba.bio_class_id =
     bc.bio_class_id and bc.data_recordset_id = dr.data_recordset_id and
```

```

df.data_recordset_id = dr.data_recordset_id and
ba.bio_attribute_name = df.data_field_name and ba.mapping_type =
'DIRECT' and df.required_flag = 0 order by dr.data_recordset_name;
/
*****/

```

## Databases

- Do not make changes to the metadata database.
- Place all the customer's schema changes in the `business_data_customization.sql` file. This file is a place-holder sql file for customizations to operational database schema. You can find this file in `\service\schema\scripts`. The out-of-the-box file is empty. Placing customer schema changes in this file makes it easier for the customer to create a new operational database.
- Use SQL Repl macros (see "SQL Repl" on page 41) if you have a heterogeneous database environment (if you use different operational databases).
- Do not modify locked tables. Add only columns to restricted tables. For a listing of locked tables, refer to the Architecture Reference Guide.
- Tables that are added as a result of customization are prefixed with a "c\_".
- The CaseSensitiveSearch recordset field property is database dependent. Out-of-the-box, this property applies for Oracle but not for SQL Server. So, if you set CaseSensitiveSearch recordset field property to TRUE, the Infor system allows you to perform case insensitive searches and setting this property to FALSE disables your ability to perform case insensitive searches. SQL Server by default is case insensitive and setting this property will result in unnecessary UPPER around fields. If the SQL server is installed with \_CS (case sensitive) option, then this property is valid and needs to be set to true and it will work similar to Oracle.
- In Oracle, if you set the Actual Data Type property of a data\_field to NCLOB and you have the data provider pointing to Oracle using oracle\_thin driver, this combination cannot handle very large data (>64K). If such a combination occurs and if the data is very large, the data retrieved from the database for that NCLOB by the driver will be null.
- To improve overall Oracle source database performance, use the following approaches
  - Create multiple tablespaces of different sizes and create a table/index in an appropriate tablespace. A tablespace is a logical grouping of tables/indices inside an Oracle database. A tablespace is given in the data definition language that creates a table/index.
  - Partition tables for better query performance. Partitioning a table means dividing the rows in tables into different partitions. Partitions can be determined by date, status, type, category, and customer name. Each partition can have its own (local) index to boost query performance. A customer's operational database can contain millions of rows (including obsolete records). It is awkward to go through the whole table at run time. If you know the criteria or data fields that are mostly employed in search queries, you can partition a table based on those criteria.
  - Use index-organized table for better query performance. If you know that a table has a moderate size and is frequently queried by its primary key, you can make it an index-organized table for

better performance. Querying an index-organized table is faster because all of its rows are wrapped up as indices.

- Oracle Treats Empty Strings as Null. Oracle does not differentiate between null and empty string (""). Hence, Oracle does not allow "" as a input to a string column that is not nullable. This behavior is different from the other databases that Infor supports. Because of this limitation, you cannot default a value "" on a nullable column. As a workaround, use a different default such as a space or another string like "default".
- Using *UseSQLNVarCharLikeQuery* to address SQL Server limitation

To support LIKE queries that may include multibyte characters on SQL Server, the constant search argument must be prefixed by a "N" to indicate that the constant is multibyte. The core does not do this by default and any multibyte LIKE searches will result in 0 rows returned. To address this SQL Server limitation, you can set a parameter called "UseSQLNVarCharLikeQuery." This parameter will add the prefix "N" to the LIKE searches. For example, the query

```
SELECT * FROM individual WHERE individual.full_name
LIKE '<some japanese text>%'
If UseSQLNVarCharLikeQuery is set to "true" then the
above query will be
SELECT * FROM individual WHERE individual.full_name
LIKE N'<some japanese text>%'
```

You can set the *UseSQLNVarCharLikeQuery* at the DataSource level or at the Field level in Infor Studio. If you set it at the DataSource level, then all corresponding recordsets and fields under that DataSource will prefix the LIKE queries. If you set it set at the Field level, then only when LIKE searches on that particular field will be prefixed.

For fields that are NVARCHAR, the performance of these type of queries will be similar to that of the non-multibyte case where SQL Server will attempt to use an index whenever possible. However, if you incorrectly set *UseSQLNVarCharLikeQuery* to true on fields that are of type VARCHAR, then SQL Server will not use any index to perform the search. Since the out-of-the-box recordsets all use NVARCHAR, it is recommended to change the parameter at the DataSource level and for any customizations that you may have created a VARCHAR field, to set to "false" at the field level.

- Using Oracle in 32 bit environment and Windows authentication for production systems.  
Infor recommends that you not use Oracle in a 32 bit environment and Windows authentication for production systems.

When Oracle is run in a 32 bit environment, it cannot access more than 2GB of main memory. Therefore, Infor does not advise Oracle being deployed on 32 bit server environments in production. Windows 2000 and higher versions assume that the default authentication mechanism is Kerberos. While local NT authentication is still supported by these operating systems, the system will always make a call to authenticate through Kerberos (usually using LDAP or ADS or some other authentication provider), and wait for the call to time out before making the call to the local NT authentication subsystem. This will cause long login times. Therefore, Infor does not recommend using local NT authentication for production systems. In addition, the local NT authentication system stores credential information in the Windows SAM, which cannot be backed up and used for restoration in the event of system failures.

## SQL Repl

Use SQL Repl when you write SQL scripts for operational database schema in a heterogeneous database environment and when you write SQL attribute domains.

The syntax for a SQL macro takes the following general form

- Macro references begin with a pair of dollar signs.
- Arguments to SQL macros are enclosed within square brackets. The separator for arguments is a comma surrounded by tilde characters.
- White space between the macro reference and the opening square bracket of the argument list is not allowed.

The following example expands correctly

```
$$NVL [MAX (col_1) ~, ~0]
```

The following example does not expand correctly

```
$$NVL [ MAX (col_1) ~, ~0]
```

The Infor Sales and Service macro interpreter allows white space between arguments and argument separators. However, those white-space characters are passed through and often appear as part of the expanded SQL statement. Make sure that any white-space characters you embed in your arguments list do not adversely affect the resulting SQL syntax.

Use Infor Studio to see SQL Macros (**Guide Bar > Physical & Logical Schema > Maintenance > System Macros**). Select a macro to see the SQL associated with the macro. The SQL appears in the **System Macro Definitions** form at the bottom. Note that the Infor core does not allow you to modify existing System Macros.

The list of system macros available in Infor Studio are also listed in the *Application Reference Guide*.

## Deployment

- A deployment unit import fails if you try to insert an agent with the same name as the one existing in the destination system. Therefore, to add a new agent, export the new agent selectively from the development system and import the agent into the destination system. To modify an existing agent, export the existing agent from the destination system (production or testing system) and import the agent into the source system (development system). Then, modify the agent and re-export from the source system and import into the destination system.
- When you customize deployment unit metadata (such as file groups and associations), save this customization in a separate module file and associate it with the 'Update Deployment Metadata' action.

When you mark a BIO as exportable in the 'Export Info' tab, do not mark the primary key of the BIO as exportable or displayable attribute, or else you will see a duplicate key violation during deployment unit application.

- Deployment units can contain information that is tied to a particular Infor version and to a specific database schema. For example, BIO data contained in a deployment unit can have references to the operational schema specific to the version of the Infor architecture that was used to create the deployment unit. The upgrade process does not upgrade exported deployment unit jar files. For example, if, between Infor versions, a column that was previously nullable is marked as non-nullable, and the deployment unit contains BIO instances that have a null value for that column, importing the deployment unit after an upgrade will fail. Therefore, deployment unit jar files are not portable across different Infor versions.

To preserve deployment units when you upgrade, or when you make schema changes as part of your own development, make sure that all data contained in your deployment units has been applied in your system before doing the upgrade. After upgrading, you can re-export the deployment units as necessary.

- A deployment unit does not have built-in support for migrating lookups from a development to production environment. BIO types to the out-of-the-box list were added to support migration of lookups from development to production environment
- Make sure customized deployments perform some kind of load testing of the product prior to roll out.
- When you export a deployment unit with a compressed file size greater than 30 MB, you may encounter server out of memory errors. To avoid these errors, split the deployment unit's contents across multiple deployment units.

## Dialogs

Use the following guidelines before moving Dialogs from a Development environment to a Production environment. Using these guidelines during development also helps in averting server start related issues.

- Do not use the system EDialogs.ebm module to save customized Dialogs. Instead, create a new module, for example, Custom\_dialogs.ebm.
- Periodically save all the dialogs that you create in a separate module (make sure that you remove all old versions and keep only the latest one).

Use the following steps

- 1 In Studio create a new module (for example, foo.ebm).
- 2 Open Dialog Designer using the following url  
[http://<machinename>.<domainname>.com/<instance\\_name>/DialogDesigner/main.htm](http://<machinename>.<domainname>.com/<instance_name>/DialogDesigner/main.htm)
- 3 Select **File > Open**. The Open Dialog dialog box opens.
- 4 Open each dialog.
- 5 Select **File > SaveAs**. A dialog box appears.
- 6 Select the new module (foo.ebm) in the module field and save. This saves changes to the database.
- 7 Go to Infor Studio and save the foo.ebm to a location for further use.
- 8 Stop using the old custom\_dialogs.ebm

# Knowledge Management System

This section provides steps that you can follow for successful KMS migration. KMS migration involves migrating data from tables, performing re-indexing, and making sure that the data transfer occurred correctly.

Use the following steps for successful KMS migration and re-indexing.

- 1 For a complete KMS transfer, migrate data from the following tables:
  - knowledge\_document
  - content
  - content\_data
  - e\_km\_category
  - e\_km\_membership
  - e\_km\_category\_cat\_rel
  - e\_doc\_user\_profile (to migrate access information for documents)
- 2 After migrating, perform re-indexing. You do this from the Admin Console application.
- 3 To make sure that the data has been transferred properly, check the count in the following tables.

```
select count(*) from knowledge_document
select count(*) from content
select count(*) from content_data
select count(*) from e_km_category
select count(*) from e_km_membership
select count(*) from e_km_category_cat_rel
```

Check the count in the following two tables as well.

```
select count(*) from e_km_search_index
select count(*) from e_km_search_log
```

**Note:** Do not migrate from the e\_km\_search\_index and e\_km\_search\_log tables. After you have migrated and re-indexed in the new environment, use the e\_km\_search\_index and e\_km\_search\_log tables only for comparison.

## Other

- Business schema definition is exported to business\_schema\_def.properties file. Use the business\_schema\_def.properties file to look up table and column definitions. You can find this file in <Install Dir> shared\schema\scripts folder.
- Do not use DATE-type fields in Fieldsets (Alternate Keys/Foreign Keys/Primary Keys).

- Customers should not edit Sync ACL query expressions. If you want to exempt certain SSO roles from ACLs, you can do it by checking those SSO profiles under the exemptions for the ACL query expression.
- The Infor core uses 16 byte keys. As a result, it cannot handle primary keys longer than 16 bytes (especially concerning external BIODs).
- When you use simple computation for date range validation, any values that you enter in the corresponding date properties (for example, Day, Hour, and so on) are considered incremental to the variable used in the Min or Max properties. For example, if you enter Today for Min property and then 1 in the Day property, the validation rule is Today+1.
- The default value called Default\_Sequence\_Common of type Sequence defined in the shared\_bio.ebm auto generates numeric sequence number. You can access this default value in Infor Studio (**Physical & Logical Schema > Validation Rules and Default Values** . To override the auto generated number (that is, populate reference\_code with your own numbers), go to **Physical & Logical Schema > Recordset > Individual > reference\_code** (field name), and remove the Default Value property.
- In Sales and Service, in order to filter in local date format on a list form, set the text message "TXT\_DATEFORMAT\_ORDER" appropriately. For example, set it to DMY for en\_GB locale.
- Using "Do not lazily initialize" form property.

To make sure that a form gets initialized when it is first loaded, in Infor Studio, from the *Properties* sheet of a normal form, set the *Do not lazily initialize* form-level property to true. The default value of this property is false. The *Do not lazily initialize* property is only available for normal forms and is not available for list forms or other kinds of forms.

When you set this property to true, client-side initialization of the form happens immediately and all the form widgets are instantiated. This is useful if a form has blank rows that need to be conditionally hidden when the form is first rendered, or for any form where widget initialization cannot be delayed. Setting this property to true prevents white space from appearing for the few forms that may have empty rows. It also allows you to delay costly initialization of the bulk of the forms that do not need it to happen right away.

Setting this property to true has performance implications. It slows the rendering of the screen that has this form down by a little bit. Therefore, if you can put off initializing it, do not turn this property on.

If you have a form where sometimes there is a row where all the widgets are hidden under some circumstances, then turn this property on. Otherwise, if the form is not initialized right away, the empty row does not disappear and will continue to be present as a blank space. There are a couple of places in the system where you may need to use this (for example, CTI availability and call control panel) but otherwise this property is rarely needed.

See also "Using the Lazy Load option" on page 109.

- Localizing search operators

You can use Infor Studio to localize QBE operators by doing the following:

- 1 In Infor Studio, navigate to **Customization & Business Process > Localization & Text > Text Messages**
- 2 Select *Qbe\_Operators* for the *Module* name. You should see all the QBE operators for the English locale.
- 3 Add entries for the localized operators here. You have to add new entries and change the *Short Text* column with the localized operator.
- 4 Do *reloadmeta*.



- IIS HTTP compression

When you turn on http compression, only static content can be compressed. This is because when you have 'Compress application files' turned on, by default it only compresses content with file extensions asp, dll and exe. To compress other types of content, add the following extensions

- CSCRIPT ADSUTIL.VBS SET W3Svc/Filters/Compression/GZIP/HcFileExtensions
- "htm" "html" "txt" "css" "js"
- CSCRIPT ADSUTIL.VBS SET W3Svc/Filters/Compression/DEFLATE/HcFileExtensions "htm" "html" "txt" "css" "js"
- CSCRIPT ADSUTIL.VBS SET W3Svc/Filters/Compression/GZIP/HcScriptFileExtensions "dll" "exe" "asp" CSCRIPT
- ADSUTIL.VBS SET W3Svc/Filters/Compression/DEFLATE/HcScriptFileExtensions "dll" "exe" "asp"

## Upgrade

Use the following guidelines to facilitate an easy upgrade.

- Save all customized metadata in new module files.
- Make only allowed changes to the shipped application modules as delta modules. The allowed changes are
  - Deletes that don't invoke BIOs or Recordsets.
  - Updates (that is, changing a service configuration value).
  - Adding/deleting linked objects because Infor Studio saves the objects in the same module file as the parent object.

Store all other metadata in new module files.

- Do not modify system modules.
- Do not modify the schema for locked business database tables. Locked tables are used by the core. Add only columns to restricted tables. For a listing of locked tables, refer to the Architecture Reference Guide.

Make all configuration related changes to the deployment\_config.xml file so that the changes are available after core-only upgrade. For more information, see the section "Configuration" on page 37.

## Bulk Import

This section covers bulk import best practices in relation to

- "Setting Defaults for Import Columns" on page 46
- "Lookups" on page 47

- "Creating and Running the Import Job" on page 48
- "Extensions" on page 49
- "BIOs with too many Relationships" on page 50
- "Preserving Relationships" on page 50
- "Sub Classing Issues" on page 50
- "Sales vs. Service Behavior" on page 51

## Setting Defaults for Import Columns

You can set defaults for a target BIO attribute by using the options outlined in this section.

### Post Transformation Extension

You can set a default for a column on the target BIO attribute whose value is blank or null by setting a default value in an extension that is executed as the transformation's post extension.

### EpiExpression On The Target Attribute

You can override a value for an attribute in an EpiExpression on the target attribute in the transformation map. If you set a constant using EpiExpression it overrides any incoming value from the source attribute. This is not exactly a default for the incoming value being null or blank but more of an override, which will always overwrite the incoming value. For example, you can set the status of an imported individual as active irrespective of the incoming data or even if the status column is completely missing from the incoming data.



**Caution:** Do not use IBR unless you have to because IBR over heads affect performance.

### PRe Transformation Extension

You can set a default for a column on the Source BIO attribute or on the target BIO attribute (preferred) before the transformation is invoked. When the default is set on source attribute using pre-extension, the transformation uses this value and sets it on target. When the default is set on the target attribute using a pre-extension, if the external BIO does not have the incoming attribute set due to missing/blank incoming value, then the default set in the extension is used. If the value exists in the incoming data, the transformation overrides the default set by the extension.

### CSV File

You can create a column in the CSV file for the source column and fill the default value in the file so that when the source is mapped to target, the default is set on the target.

## Import Map

For CSV files, set the default value column in the import map (in Infor Studio, navigate to **Physical & Logical Schema > Bulk Import**, click on a Map, and set the value in the lower pane). This value is used when the incoming CSV file value is missing (has two column separator next to each other). For details on varying the default behavior using values set in Infor Studio, see the online help.

## BIO

On a BIO, set the default value parameter for the target attribute's default value property.

## Recordset

On a recordset, set the default value parameter for the target field's default value property.

## Table

Set the default value for the target column at the table level. This cannot be used for a field declared as required in the recordset as the application flags this column for having null values before the database default is set.

Note the following in relation to overrides:

- A post transformation default overrides all other default settings.
- An EpiExpression default overrides a pre transformation default setting.
- A CSV default is always used (the application treats this as a value) unless one of the above explicitly overrides the value.
- The above defaults override an import map level default.
- An import map level default overrides a BIO attribute default.
- A BIO attribute default overrides a recordset field default.
- A recordset field default overrides a table's column level default value.

**Note:** For CSV import, set a default for an incoming attribute in the import map. Setting the default at BIO/ recordset or table level will set the default for all usage of the BIO/recordset /table and not just in the import mode.

For a database import map, set the default value in an extension attached to the pre or post transformation. The 'CSV file level default' and 'Import map' level default options are not applicable for database import.

If you have to set a default for a target attribute for which no source attribute mapping exists, then, set the default in the pre or post transformation EpiExtension or as an EpiExpression.

## Lookups

Populating a lookup with import works only for lookup attribute domain lookups. For all others, you can use the code strings (as static Strings) rather than using the EpiExpression ability to do a lookup on the fly or set a default. For example, if you want to import to e\_opportunity BIO and you want to set

attribute domain opportunity\_type then you cannot set the opportunity\_type lookup. As a work around, set the value for it at the BIO layer as a default.

## Creating and Running the Import Job

This section provides general best practices and considerations for inserts and updates.

### General

- Split a single import file based job into multiple independent jobs based on start and stop index. You can run these jobs simultaneously to improve the overall import time.
- Vary the commit index on a sample (For example, try varying from 10 and 100) to get an optimum value for the index. The optimum will depend on the machine configuration and database configuration. If the value is 1, every record is committed to the database and import can be slow if the database or network traffic is bad. If the value is very high a large memory is required, as processed records remain in memory till the commit threshold is reached.
- Time your extensions to see how long they take to execute. Use suggestions provided in the extension section to tune extensions to reduce the time taken in extensions.
- Try trial run on a small number of records to see if there are any issues. You can also run import on sample data to see if the imported results are as expected.
- If importing a large number of records, don't check the flag for writing remaining records for the error file. Checking this file will write the remaining record to the .rej file on a job failure and this can be time consuming.

### Inserts

- If you are doing an initial load or only inserts then check the 'Create Only' flag in the BIO transformation map. This will prevent unnecessary query to check if there needs to be an update.
- If you are doing an initial load or only inserts and you are importing a single address/ telephone or e-mail, then instead of mapping to the out-of-the-box BIO's collections, map it to a related BIO through a constrained relationship. You can create such a related BIO if one doesn't exist. For example, on an individual BIO import, instead of mapping a single incoming address data to addresses collection on Individual BIO and filtering based on primary flag, you can map the incoming address data directly to the primary\_address BIO on the individual. With this option, you can also disable extensions around checks such as 'at least one primary flag', 'checking for duplicate primary', 'check for alternate keys', and so on.
- If you are doing an initial load with no other application running (that is, if you are populating data before opening up the application for all users), you can disable/delete the indexes on these tables and create them after the insert. Index maintenance overheads are expensive for insert operations.
- You can disable Sync extensions for the import option since Mobile Sync is deprecated as of 7.0.4 FP1.

### Updates

- Out-of-the-box tables may not have index on the columns you pick as AKs for your recordset updates. Index the columns of the alternate key selected in the transformation map, and the columns used in the filtering of related BIO collections.

- Any column lookups, alternate key for related records, BIO lookups can translate into database query and indexing these columns in the database will improve performance.
- If you are updating a primary single address/ telephone or e-mail, then instead of mapping to the BIO's collection such as addresses, you can map it to a related BIO through a constrained relationship such as primary\_address BIO on Individual (similar to the option described for inserts).

## Extensions

This section provides best practices in relation to disabling extensions and code changes.

### Disabling extensions

- You can improve import throughput by disabling extensions that are not required. You can disable extensions from Infor Studio and from code.
- You can disable extensions from Infor Studio (this option is available from 6.5.2). Click on the **Attribute** column in any extension detail screen and check the **Disable Import** option on the pop up. If the extension has a specific purpose, disable it from Infor Studio for cases when it should not run.
- If your special case cannot be controlled through available metadata, change the code to first check to see if you need to run the remaining parts of the extension. Disable the code that need not run for import. For the option of disabling the code in Import Mode, add check for `_ctx.getUserContext().isServiceModeSet(ServiceModes.BULK_IMPORT)`. For more information, see "Bulk Import" on page 125.

### Code Changes

- Minimize queries from extensions. Check if the information is already available. For example, extensions querying for current user when the current user is part of context.
- Check if you are doing repeated get for the same value from a BIO. It may be more efficient to get it once in a local variable and use that.
- Database checks are part of an extension (CheckAlternateKeys). This extension checks for uniqueness on the Alternate keys defined on the recordset in Studio by doing query to the database. You can remove the unused/not required Alternate key definitions on recordsets to reduce the number of checks.
- Don't get BIO/Unit of work from after save extensions. This can mean an additional database trip. BIO ref is provided in the afterSave extension and has all the values that are available before the save. If you still need to use the BIO in after save extensions use the afterSave signature (available from 6.5.1.3 version) that provides BIO as the parameter.
- Queries written from extensions will be slow unless, the where clause that they translate to, has columns that are part of an index. Check your query where clause or create required indexes.
- Run a small sample import enabling DP logging and see which queries are executed. You can tune your table's indices to make these queries use appropriate index rather than doing a table scan. You can also obtain time taken by these queries using sql log.

## BIOs with too many Relationships

Out-of-the-box recordsets and BIOs can have many relationships or alternate key definitions that are not used by the import job. To improve performance, create new recordset and BIOs that have only required relationships and AKs. This will also require you to assign the extensions applicable to the new BIO and set the BIO and UI level defaults (if applicable) directly on the BIO. For example, if you import just individual with no related records such as address or telephone, instead of using the out-of-the-box Individual BIO, you can create a new my\_individual BIO. This new BIO will not have all the relationships defined as the out-of-the-box Individual BIO. Data import can use this BIO as the target BIO and the import process is faster because it has to process fewer/no relationships and do fewer alternate key checks for every record.

## Preserving Relationships

- For database import, the external BIO need not be a flat structure. You can import the whole tree structure for the incoming data using dbimport by creating external BIOs with relationships, and map that to internal BIOs and related BIOs and transform the complete tree structure with relations in one attempt. For example, if the external data has individuals and addresses on different tables, you can create an individual external BIO, for example, ext\_individual with address being a related BIO on that, and map ext\_individual BIO to the out-of-the-box individual BIO and its related record primary\_address, and transform the whole tree structure of the external BIO to the internal BIO structure.
- You can split jobs into logical units and preserve the relationships as long as you have a unique alternate key that you can use to lookup the related record. For example, you can split the import of Individual and address with the alternate key being for example, reference\_code on individual. You can map and import individual first. You can create a view on source that has the reference\_code added to the address that links each address to a reference\_code hence to an individual. You can then import address from a view by creating an external BIO / recordset and import map based on that. You can use the reference\_code for linking the address being imported back to the individual that is already imported.
- When importing from an external source you can preserve the GUIDs if you have them in external source. Infor GUIDs are 32 bit GUIDs that are hexadecimal and can fit into a binary 16. If your GUIDS are different, then you can preserve them as a new column, which you can define as an alternate key. If you want to preserve your non-Infor standard GUIDs as primary key, you can change the primary key's type to be of String (this may require changes in the out-of-the-box application which is based on GUID primary key).

## Sub Classing Issues

- When importing into many-to-many relationship BIOs, splitting the jobs into simpler maps, all executed as a single container job is easier to debug. For example, for importing into organization\_indiv and creating individual, organization and corresponding address, telephone, and so on from a single CSV file, you can create three maps. One for individual with all its related records such as address,

telephone, and so on, and another for organization and its related records. Finally create a map to organization\_indiv and link the individual and organization created in the previous job into this BIO.

Map1 Import organization-- address

Map2 Import individual--address

Map3 Import organization\_indiv-- Link-individual and organization\_indiv-- Link-organization

Create a container job that imports all the maps as a single job in a specified order.

- You can also create a map that directly maps to the Organization\_individual, creates individual, creates organization and related records in one map but this is difficult to debug. You cannot create a map that instead of having organization\_indiv as focus, starts from for example organization, and then navigates to Organization\_indiv and then to individual and so on.

## Sales vs. Service Behavior

- By default, import jobs are run by setting the application mode to Sales application and Sales business logics is applied accordingly. If you need the behavior to be a specific application mode, add the below to deployment\_config.xml

```
<!-- Select the Application mode (Sales or Service) in which import
job
    should run. The territory assignments, routing are created based
on this
    application. Not setting this property will is interpreted as
Sales. -->
<config:category name="BULK_IMPORT">
    <config:configParam name="APPLICATION_NAME" value="Service"/>
</config:category>
```

The above setting will make the import job run in Service application mode. For example, when you import and create an organization BIO, or an individual customer BIO, by default, the BIO is assigned a project team and a sales team according to the Sales-specific business rules. If you want to prevent this from happening, you can change the mode to Service in deployment\_config.xml.

For individuals, all imported data will be visible only to the administrator unless you assign (from existing user agent data) or import user agent data along with individuals. This makes it accessible to those individuals.

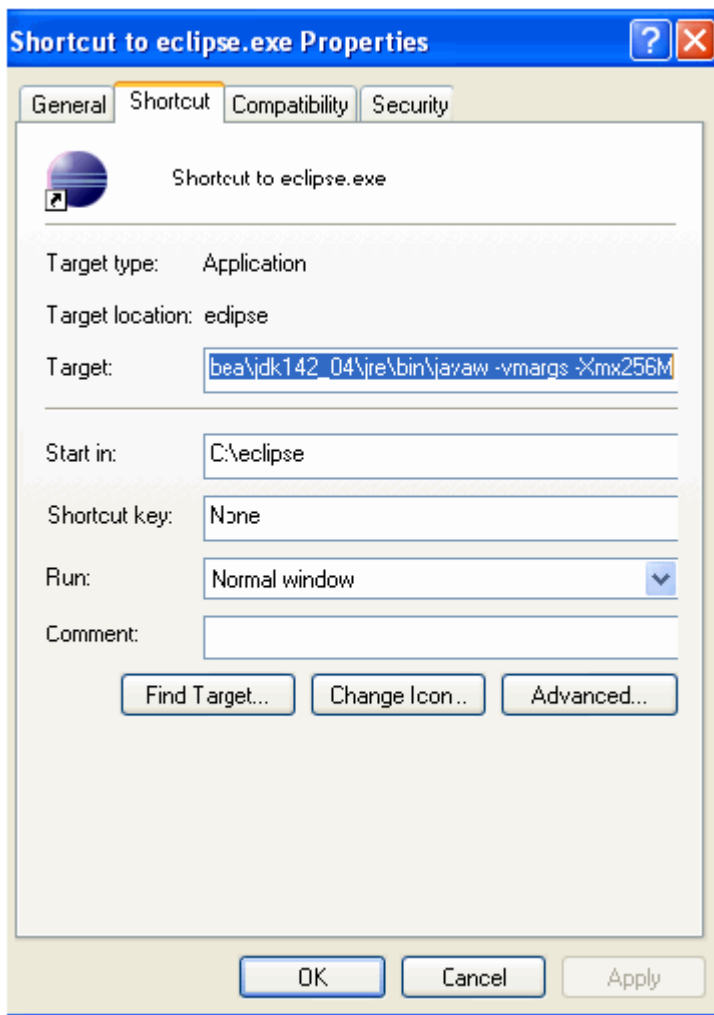




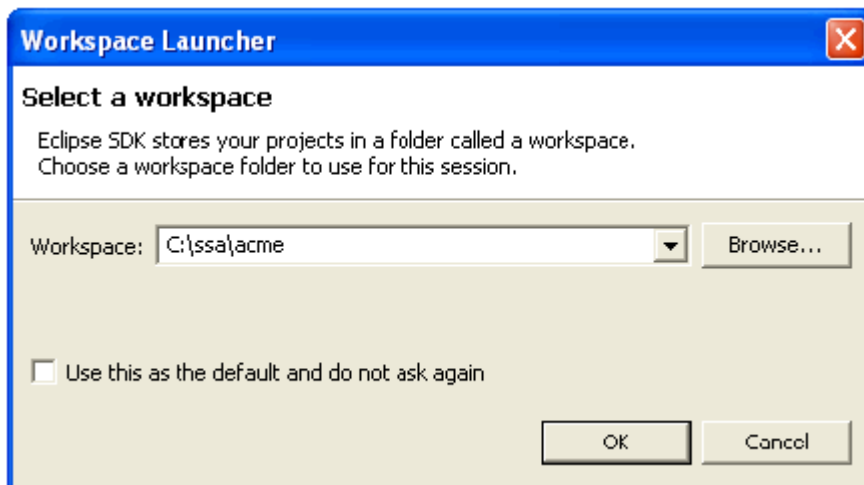
This chapter describes how to configure *Eclipse* for development.

## Configuring Eclipse

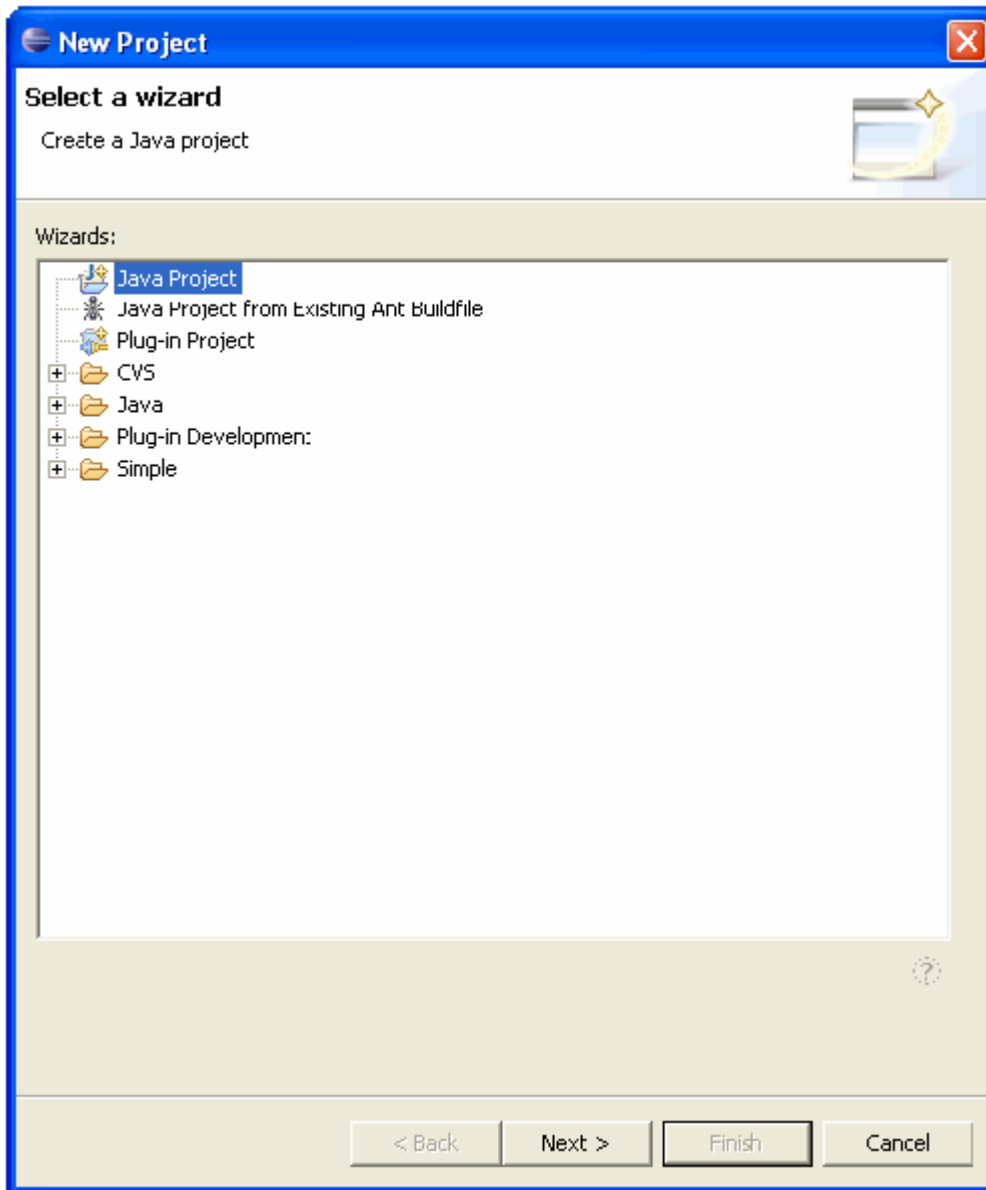
- 1 Unzip the Eclipse files to a folder, for example, `C:\eclipse`.
- 2 Setup a shortcut to launch by doing the following:
  - a Right-click `eclipse.exe` and click **Create Shortcut**.
  - b Drag the `eclipse.exe` shortcut on to the desktop.
  - c Select the short cut, right click, and click **Properties**.
  - d Paste the target path in the Target field. For example:  
`C:\eclipse\eclipse.exe -vm C:\bea\jdk160_05\jre\bin\javaw -vmargs -Xmx256M`  
where `C:\bea\jdk160_05` is where your JDK is installed.
  - e Click **OK**.



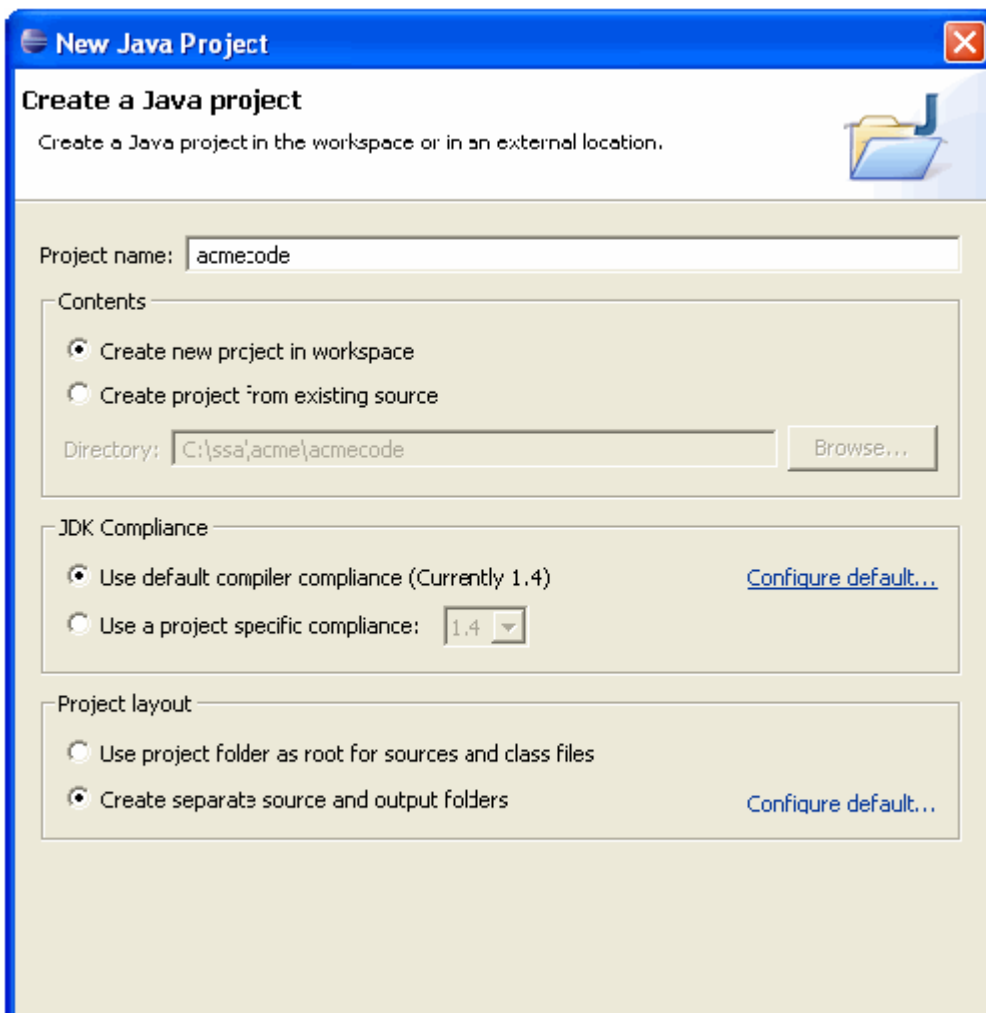
- 3 Double-click the Eclipse short cut to start Eclipse and select a workspace. In Eclipse, projects are stored in a directory called Workspace. In this example, acme is the custom implementation folder and Cassia is the install directory for Infor Epiphany Sales and Service.



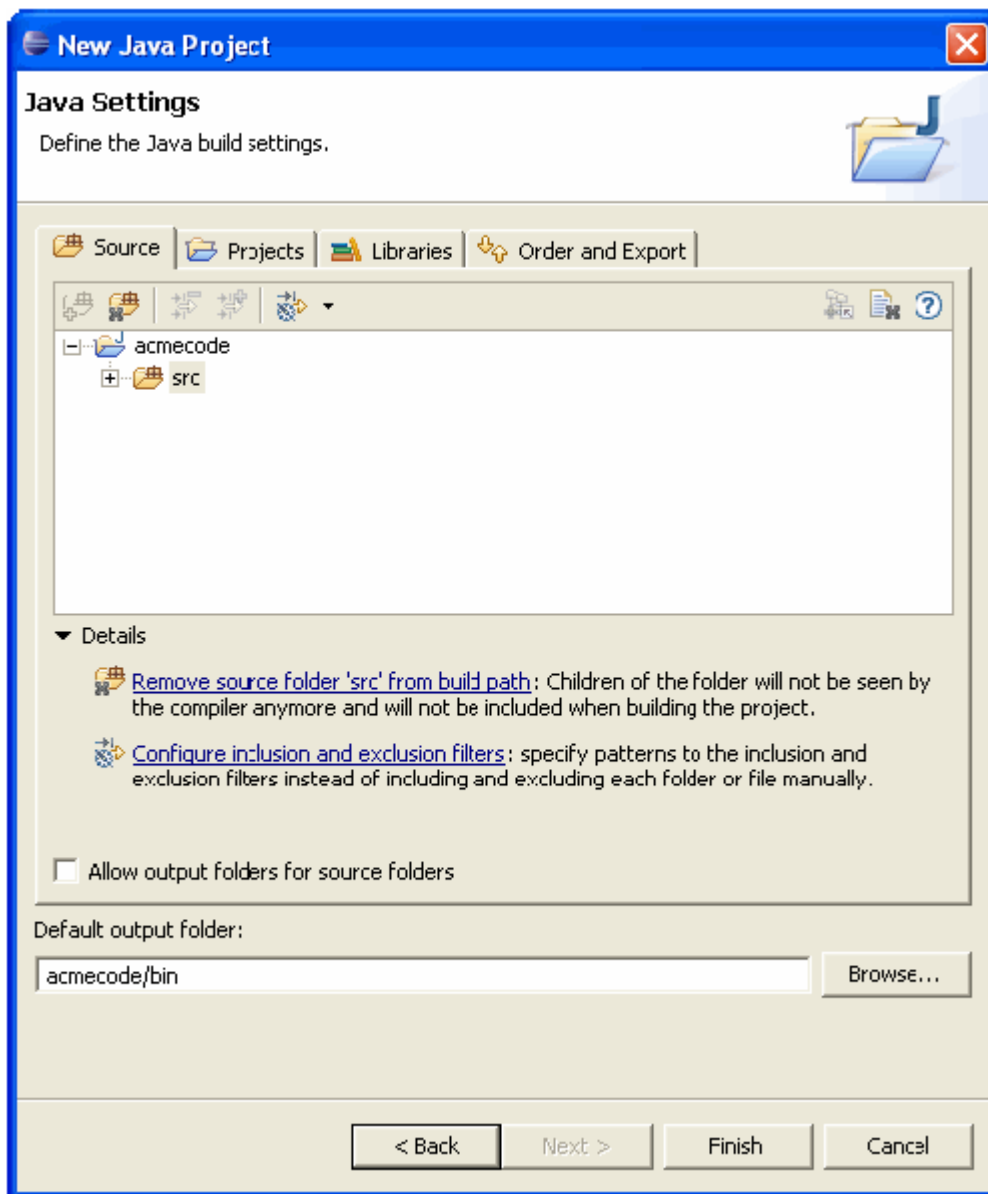
- 4 After launching Eclipse, create a new project (**File > New > Project**) and select the type of the project to be **Java project**. Click **Next**.



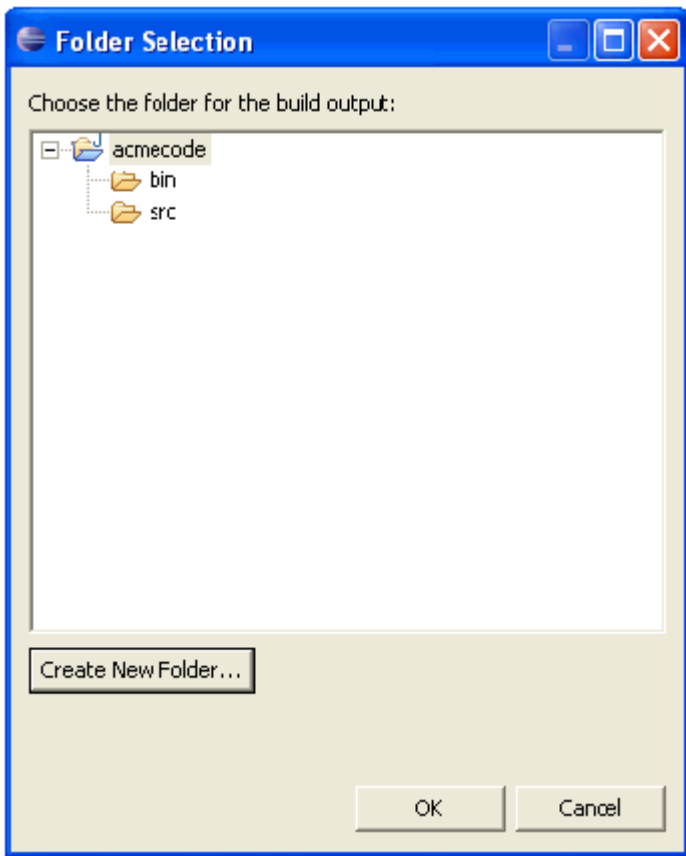
- 5 Provide a name for the project. Under **Project layout**, make sure you select the **Create separate source and output folders** option.



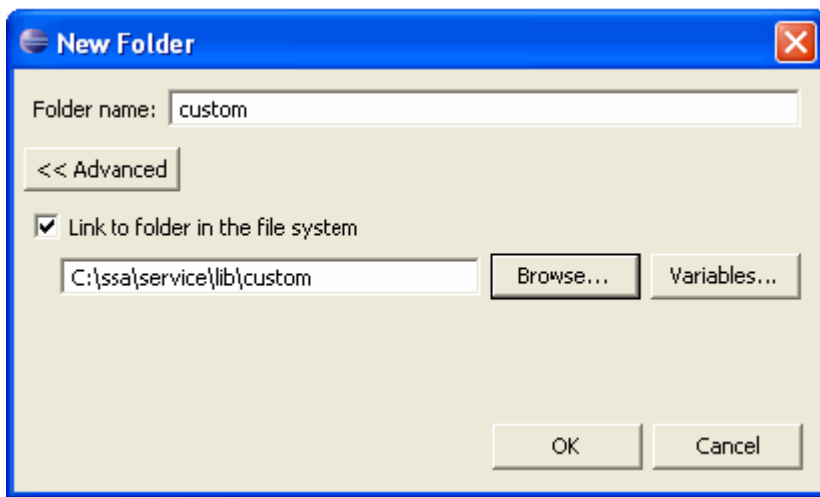
- 6 On the project, for the **Default output Folder**, select **Browse**.



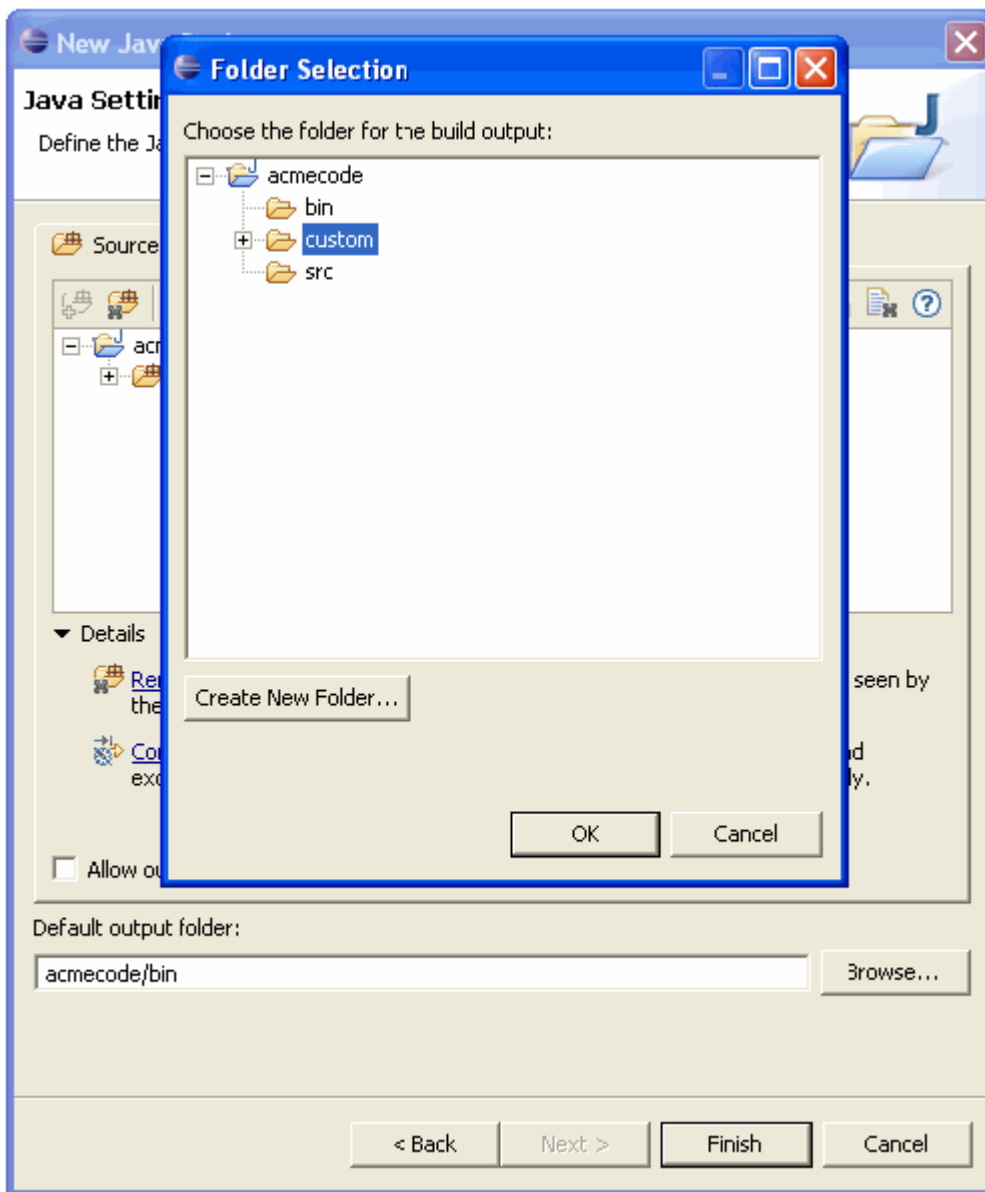
- 7 On the popup window, select the top level folder. In this example, it is acmecode.



- 8 Click **Create New Folder**. The New Folder screen appears. Do the following:
  - a Name the folder **custom** and click the **Advanced** tab.
  - b Check the **Link to folder in the file system** checkbox.
  - c Click Browse to navigate to the folder. Make sure to link the output folder to <installdir>\service\lib\custom. This folder gets automatically picked up as the chosen location for customized Java classes. If you want an alternate location, check the Installation and Configuration Guide for the necessary changes.

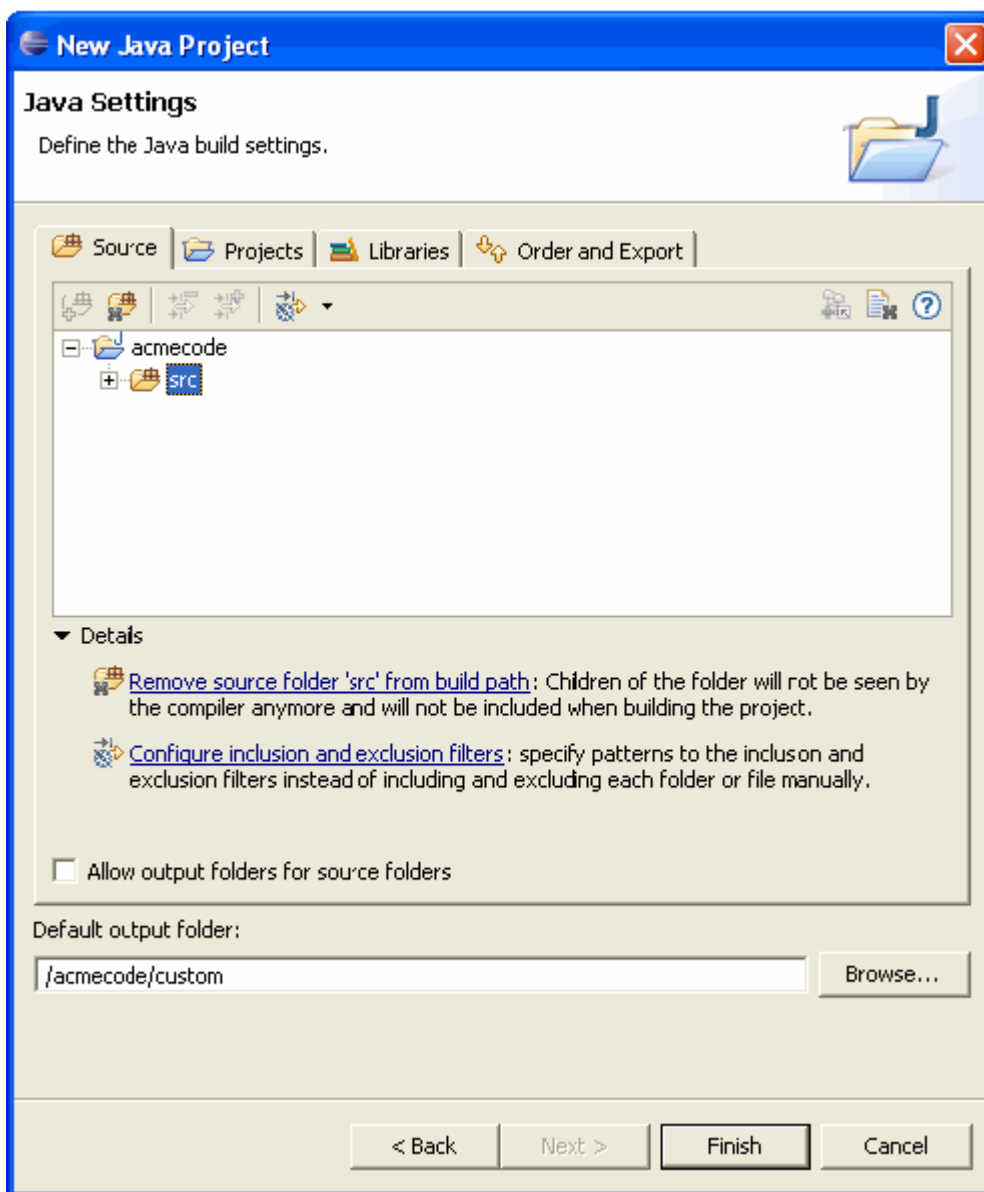


- 9 Click OK. The Folder Selection screen appears.

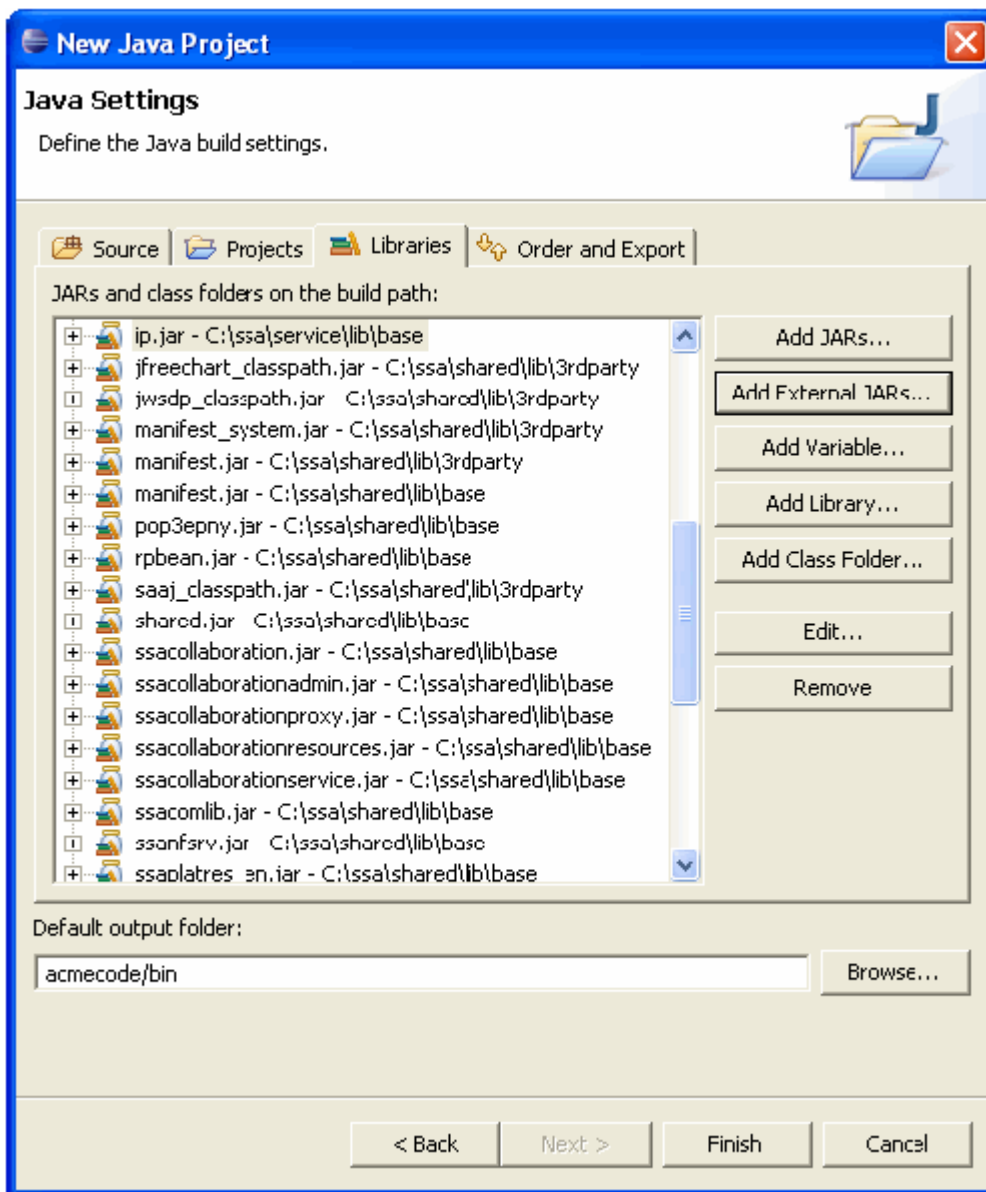


**10** Click OK. This takes you back to the New Java Project screen.





- 11 Click the **Libraries** tab and **Add External JARs**. Add all jars present in <installdir>\shared\lib\base and <installdir>\shared\lib\3rdparty. Also add in <installdir>\service\lib\base\ip.jar. To select the jars, navigate to the folder, select the jar files, and click **Open**.

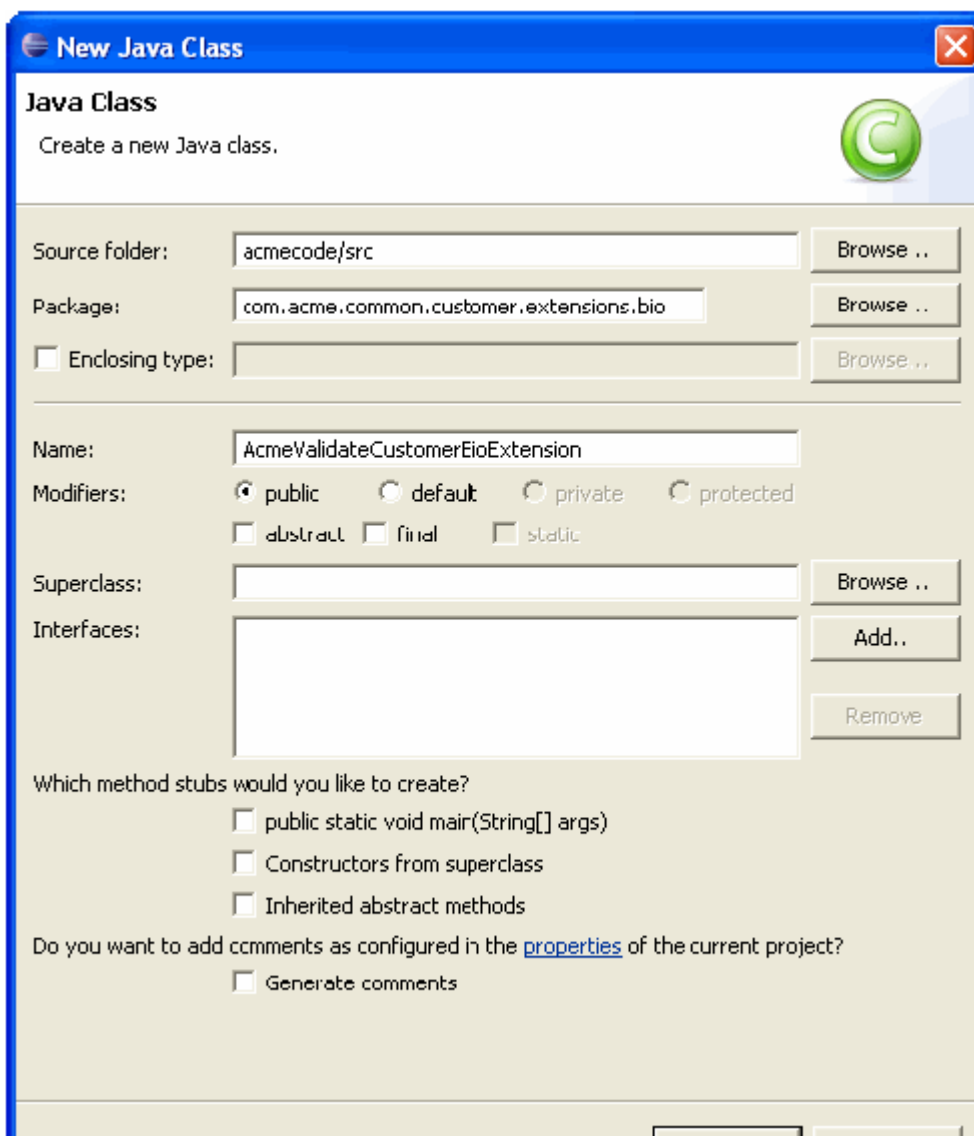


12 Click **Finish** . If **Confirm Perspective Switch** dialog box appears, click **Yes** to the dialog box. You can also select the **Remember my decision** checkbox.

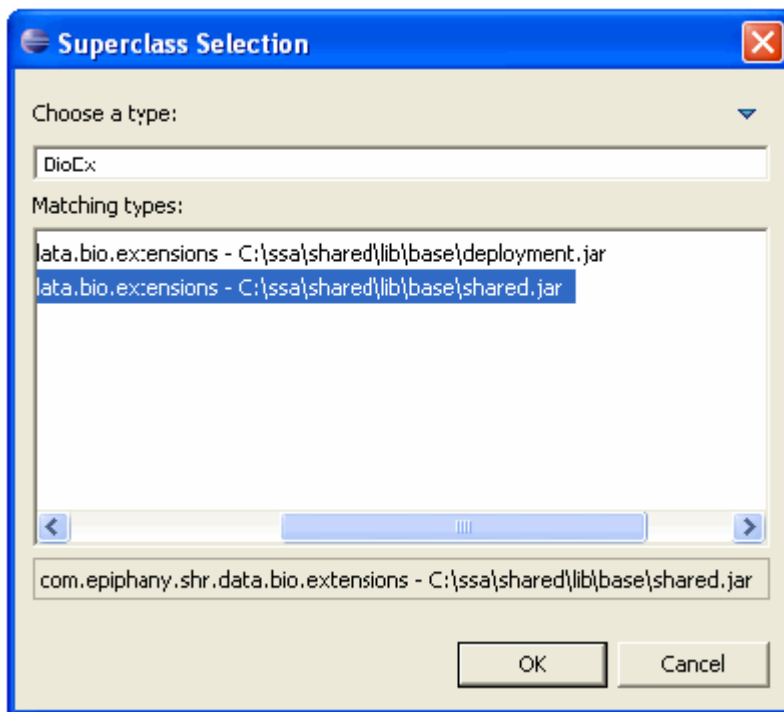
## Creating New Extensions

This section describes how to create a BIO extension. To create a UI extension, you have to extend the `FormExtension` or `ActionExtension` base class.

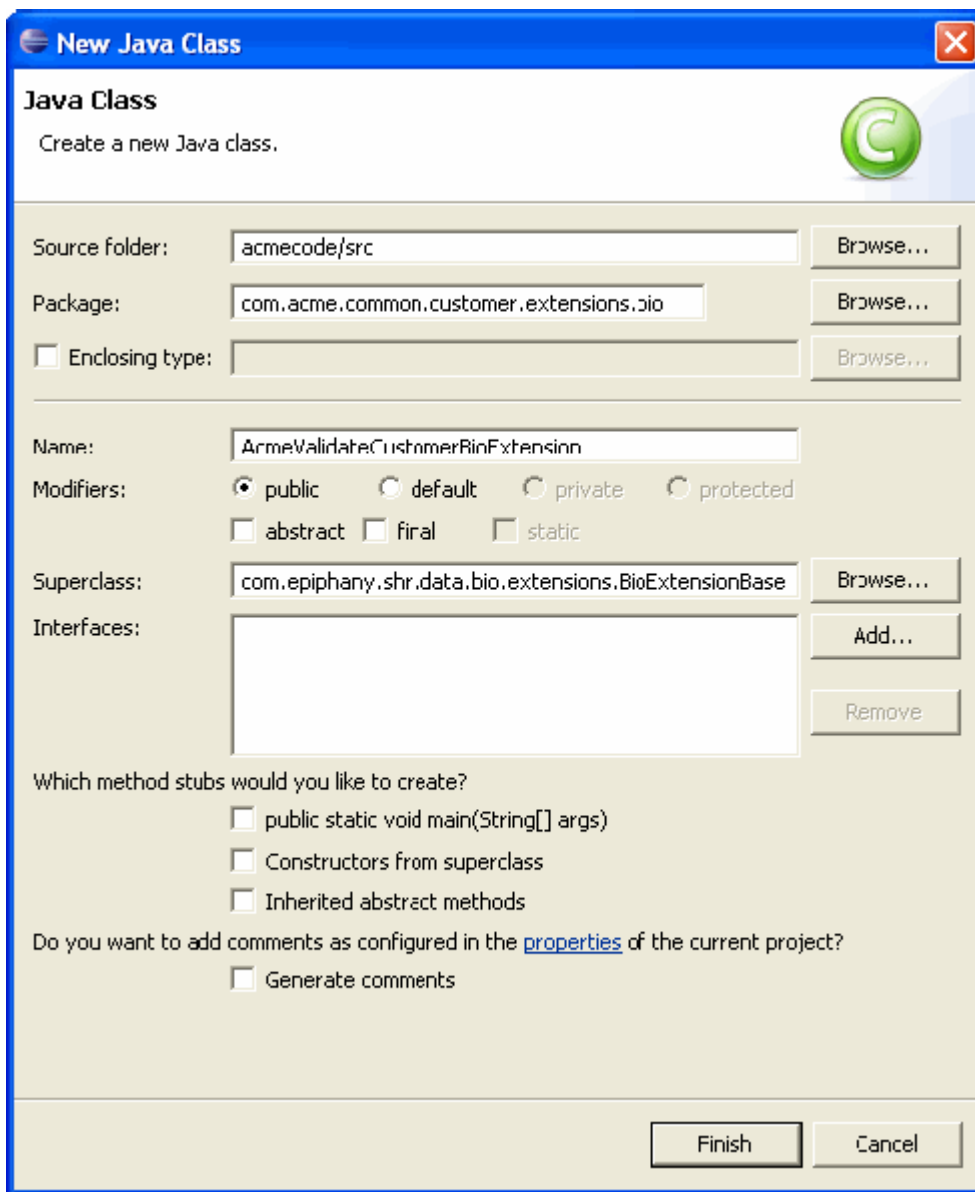
- 1 In Eclipse, create a new class (**File > New > Class**).
- 2 In the New Java Class dialog box, enter the appropriate package name and class name.



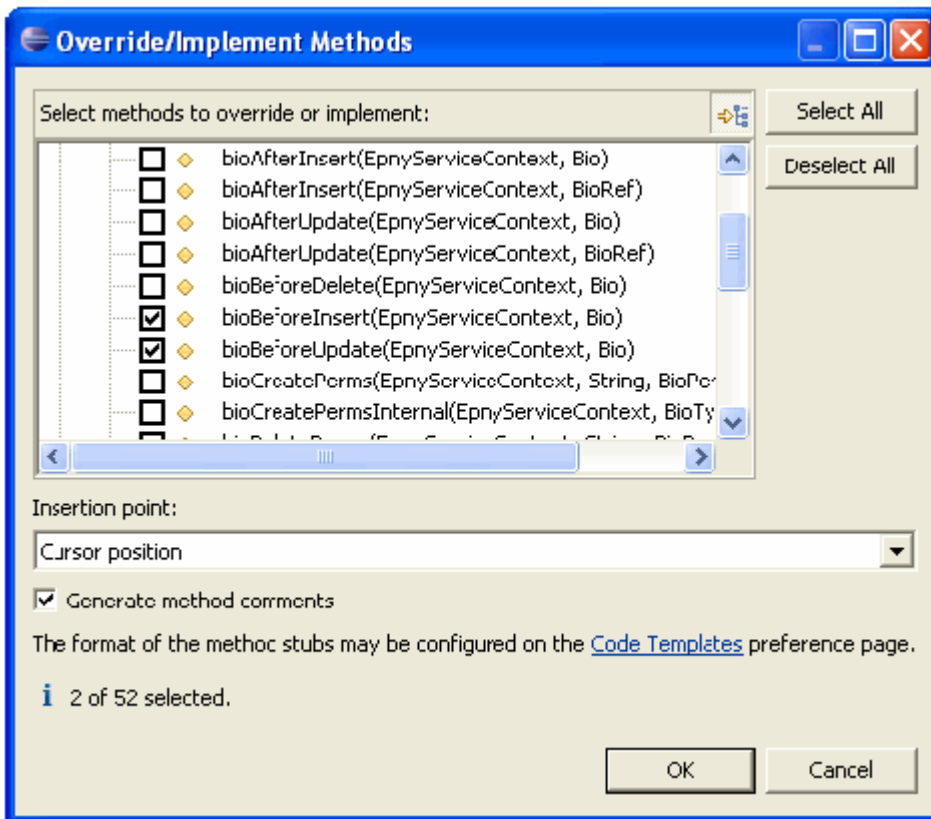
- 3 Click **Browse** next to **Superclass** and select the class you want to inherit from by typing the class name. For example, **BioExtensionBase** class. Click **OK**.



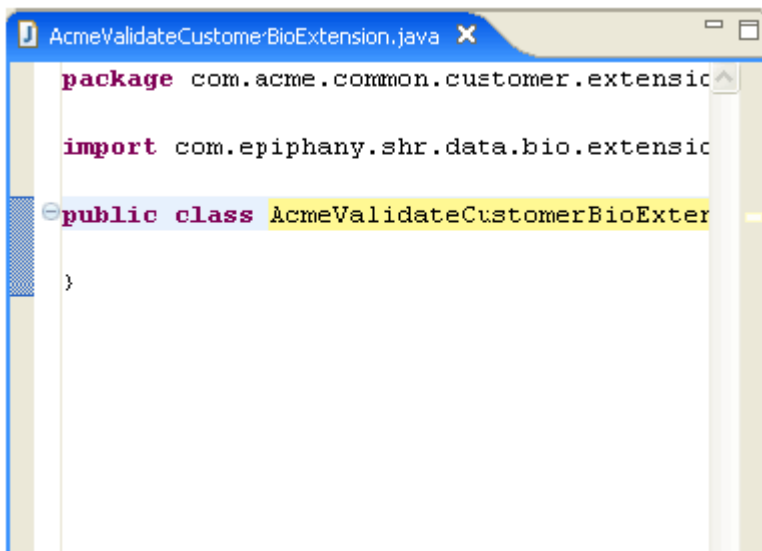
- 4 The New Java Class screen appears. Click **Finish** .



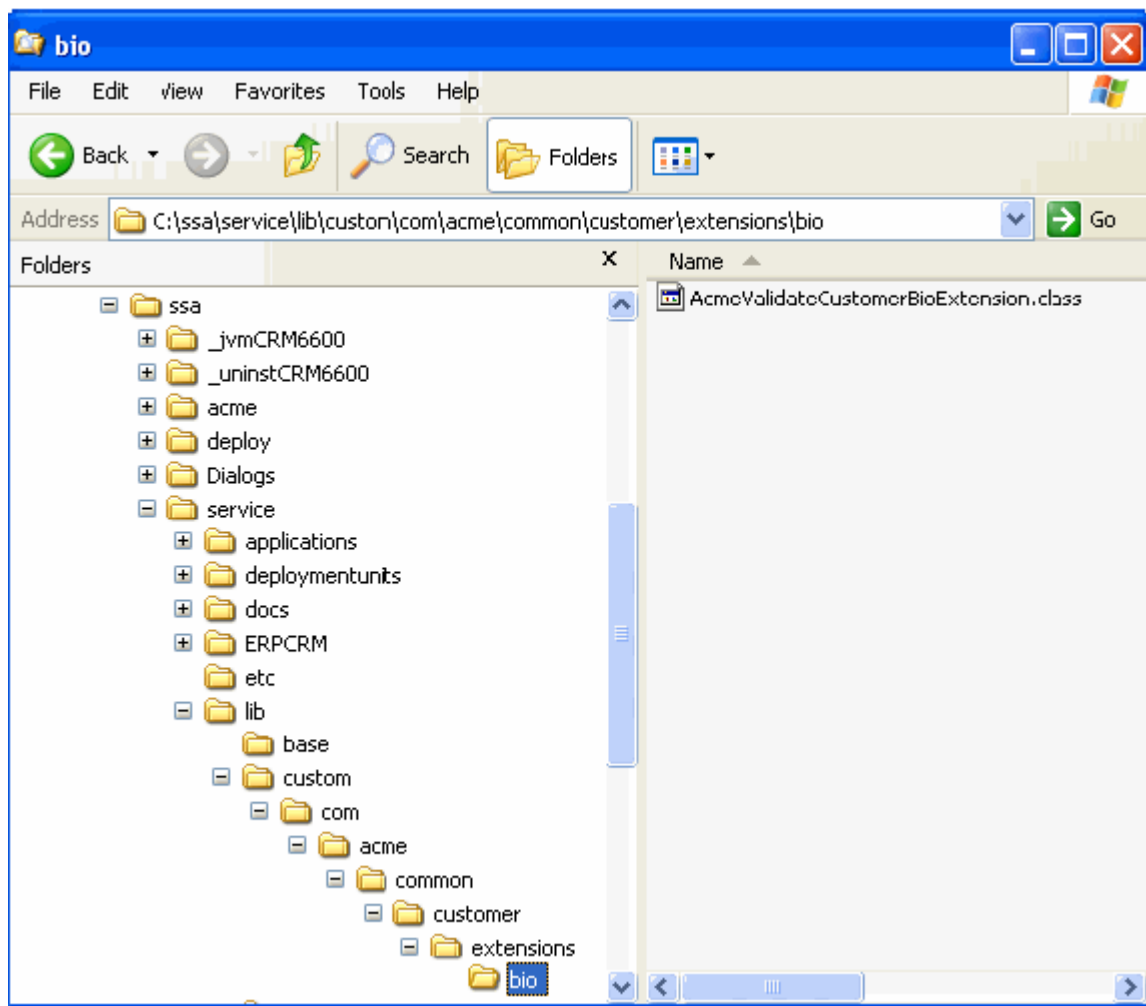
- 5 Go to **Source > Override/Implement Methods** and select the methods you want to override.



- 6 Click **OK**. This should generate the method stubs for your class.
- 7 Write all the necessary custom code and click **Save**. Eclipse by default is configured to build automatically.



- 8 If configured properly, you should see your Java classes appear in `<installdir>\service\lib\custom\com\acme\common\customer\extension\bio`.



For more information on how to write and register extensions, see the Infor Sales and Service Architecture Reference Guide .





This chapter describes how you can use Concurrent Versions System (CVS) for *source control*. CVS is an open-source network-transparent version control system.

## Overview

Application development for the Infor core typically involves using tools such as Infor Studio, Eclipse, and SQL editor. These tools create development files of various types that you have to control throughout the development and testing process. You can use CVS or some other version control software as the source control point for all development related source control file management. The version control structure contains buckets of development files to help manage the consolidation of the overall development effort.

## Using CVS

If you have Eclipse (2.1.3 and higher) installed on a development machine, you can use Eclipse to access the CVS master workspace.

The CVS Repository includes development files of the following types


- Java files
- EBM/EBMD: Infor build modules (sourced from Studio)
- SQL/CTL: Database supporting files
- Static Content: .gif and .jpg files and Infor stylesheet (CSS) files

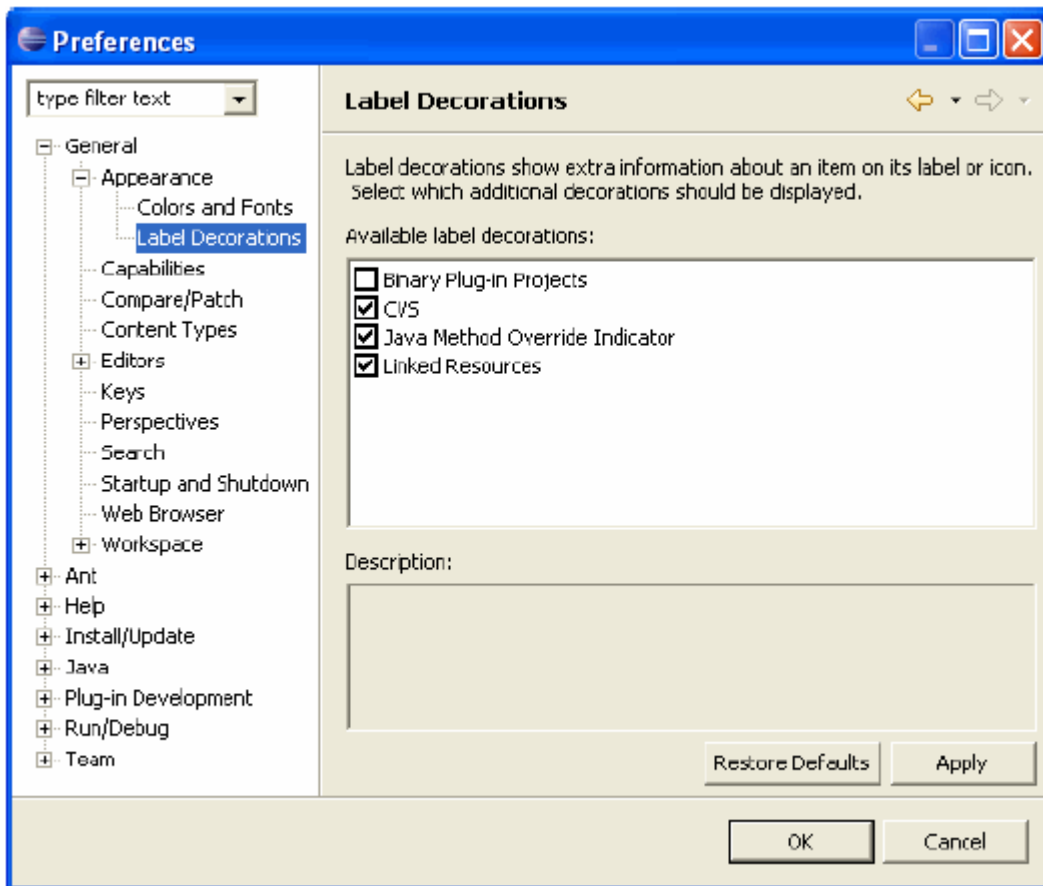
The above files constitute the core source for all development within the Infor Epiphany Service environment.

The rest of this chapter describes procedures that you have to follow to check-in and check-out development work.

## Setting up Eclipse

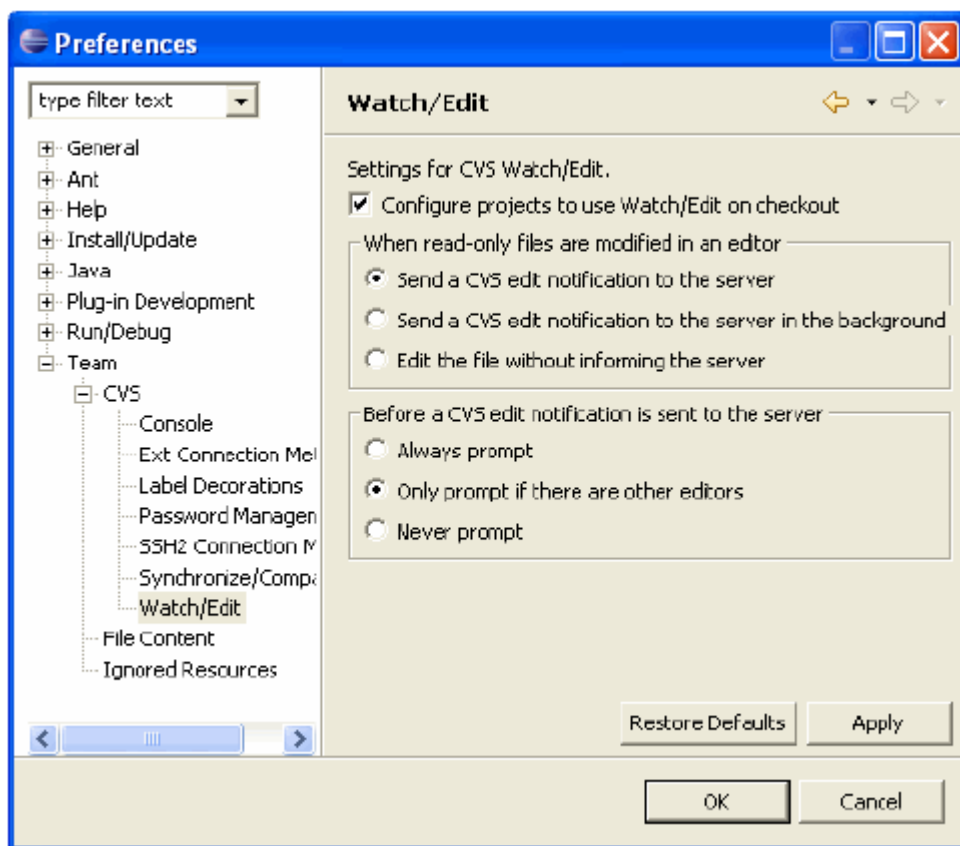
To populate the local CVS workspace with the most updated build components, you have to use Eclipse to access the master CVS repository located on the central CVS server. The Eclipse installation on the developer workstation will already have a perspective that connects into the CVS development repository. The steps below show how to properly populate a CVS workspace with the most updated development modules.

- 1 Start Eclipse (version 2.1.3 and above).
- 2 Open the CVS Repository Perspective by doing the following:
  - a In the upper right-hand corner of Eclipse, click on the  icon and choose Other. The Select Perspective dialog box opens.
  - b Select **CVS Repository Exploring** and click **OK**.
- 3 From the **Window** menu, select **Preferences**. Expand **General > Appearance** and double-click **Label Decorations**. Select the **CVS** checkbox. This will enable CVS label decorations in the Java Perspective. This displays CVS specific information on resources in projects under CVS control. Information includes the revision number, branch or version name.



- 4 Make sure that the **Watch/Edit** option is enabled to receive notification when a file that you wish to edit is being modified by other developers. Under **Window > Preferences**, expand **Team > CVS**

and double-click **Watch/Edit**. Check the **Configure projects to use Watch/Edit on checkout** checkbox and click **OK**.

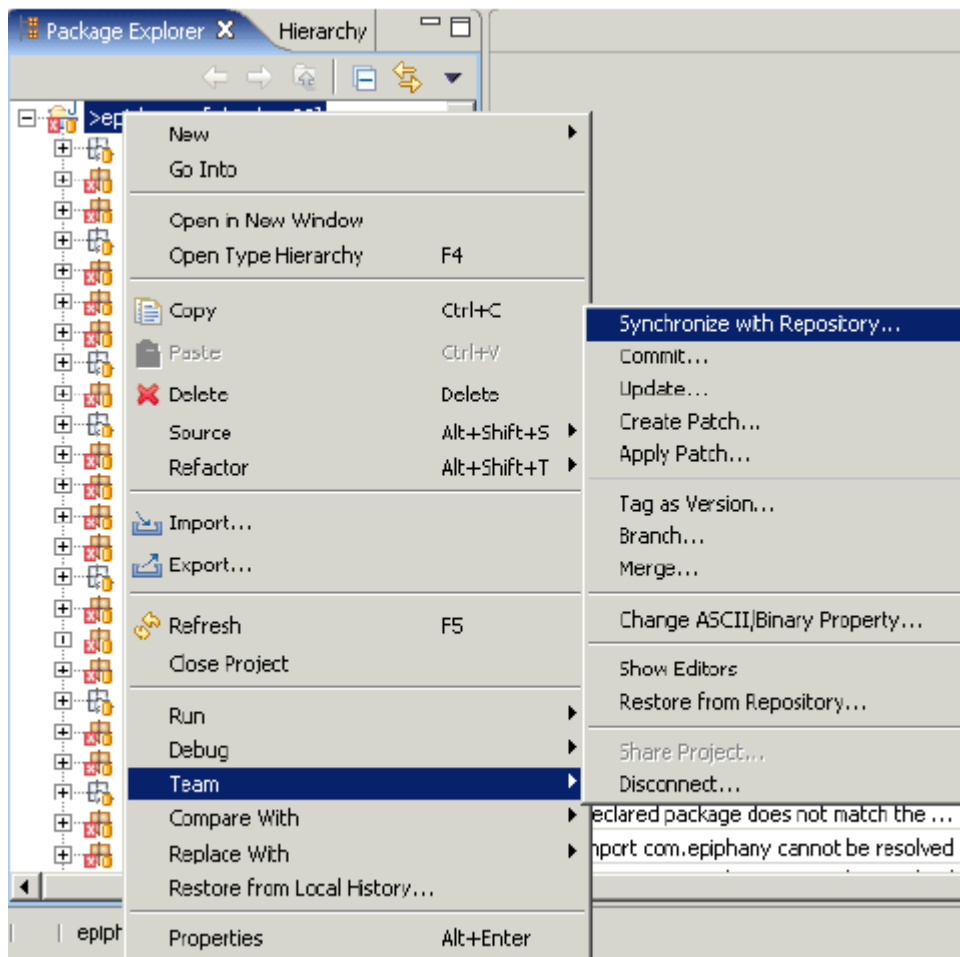


- 5 From the CVS perspective, select a project, right-click, and select **Check Out As Project**. You can also select **Check Out Into...** to specify a particular local workspace folder the project should be checked out into.
- 6 Switch to Java Perspective. From the **Window** menu, select **Open perspective > Java**. From the Java Perspective, validate that all modules exist in the workspace.

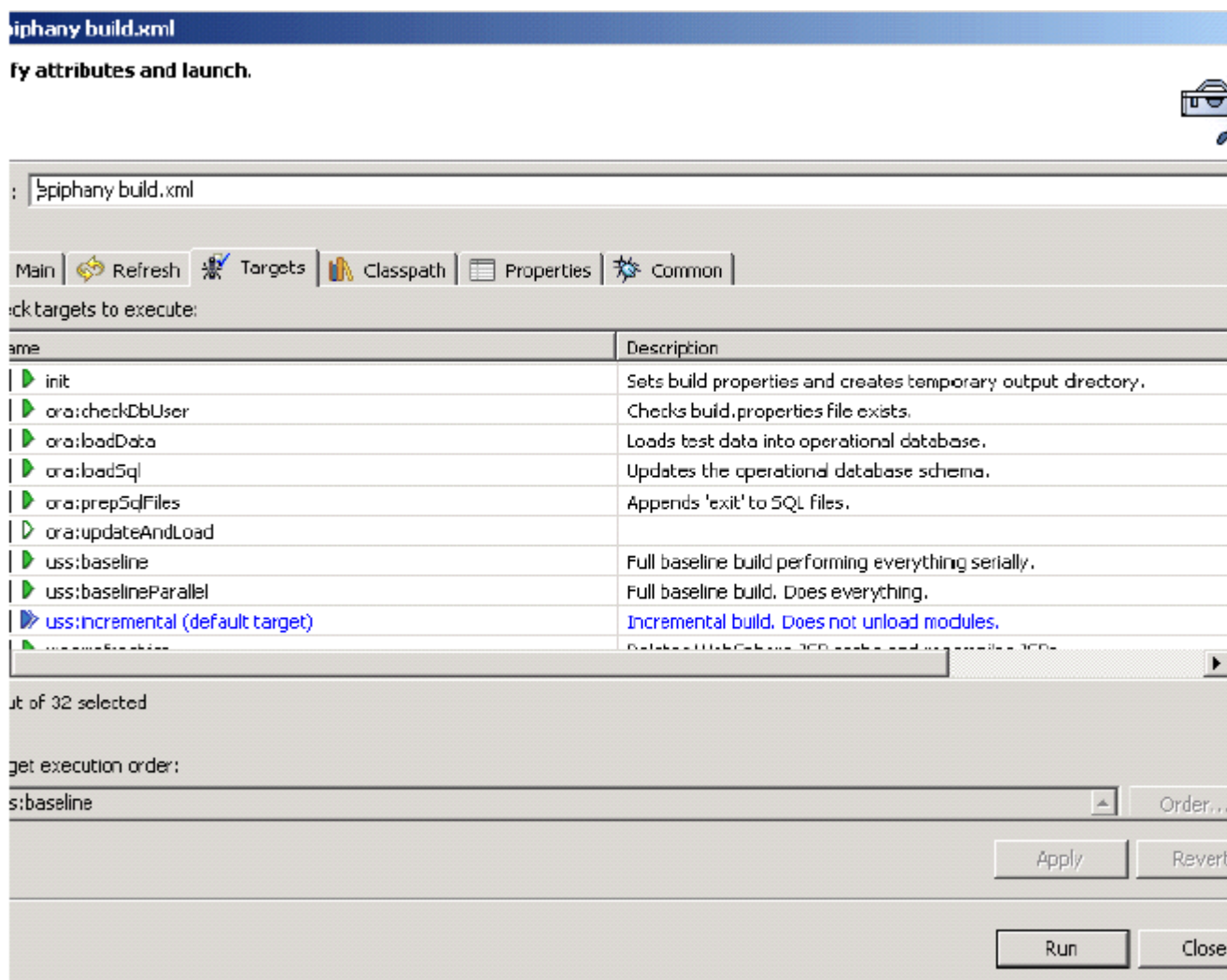
## Preparing a Development Machine

To ensure that all developers are in sync during development, you have to perform a baseline of each developer machine after each successful build has been tagged. This will re-load all out-of-box, custom, and delta modules that are currently in CVS, re-build all custom tables/columns, and re-load all test data. To baseline a development machine, do the following:

- 1 From the Java perspective, in the **Package Explorer**, right click on the top level Infor folder and select **Team > Synchronize with Repository**. Any files that need to be updated will appear in the right window under **Synchronize - Incoming Mode**. Right click on the top level folder again and select **Update from Repository**.



- 2 From the Java perspective, right click on the top level database folder and select the **Team > Synchronize with Repository** option. Any files that need to be updated will appear in the right window under **Synchronize - Incoming** Mode. Right click on the top level folder again and select **Update from Repository**.
- 3 Right click on the build.xml file and select Run Ant. Make sure that only the customer:baseline option is checked and click **Run**.



Once the build.xml script is finished, the developer machine is successfully set up for baseline.

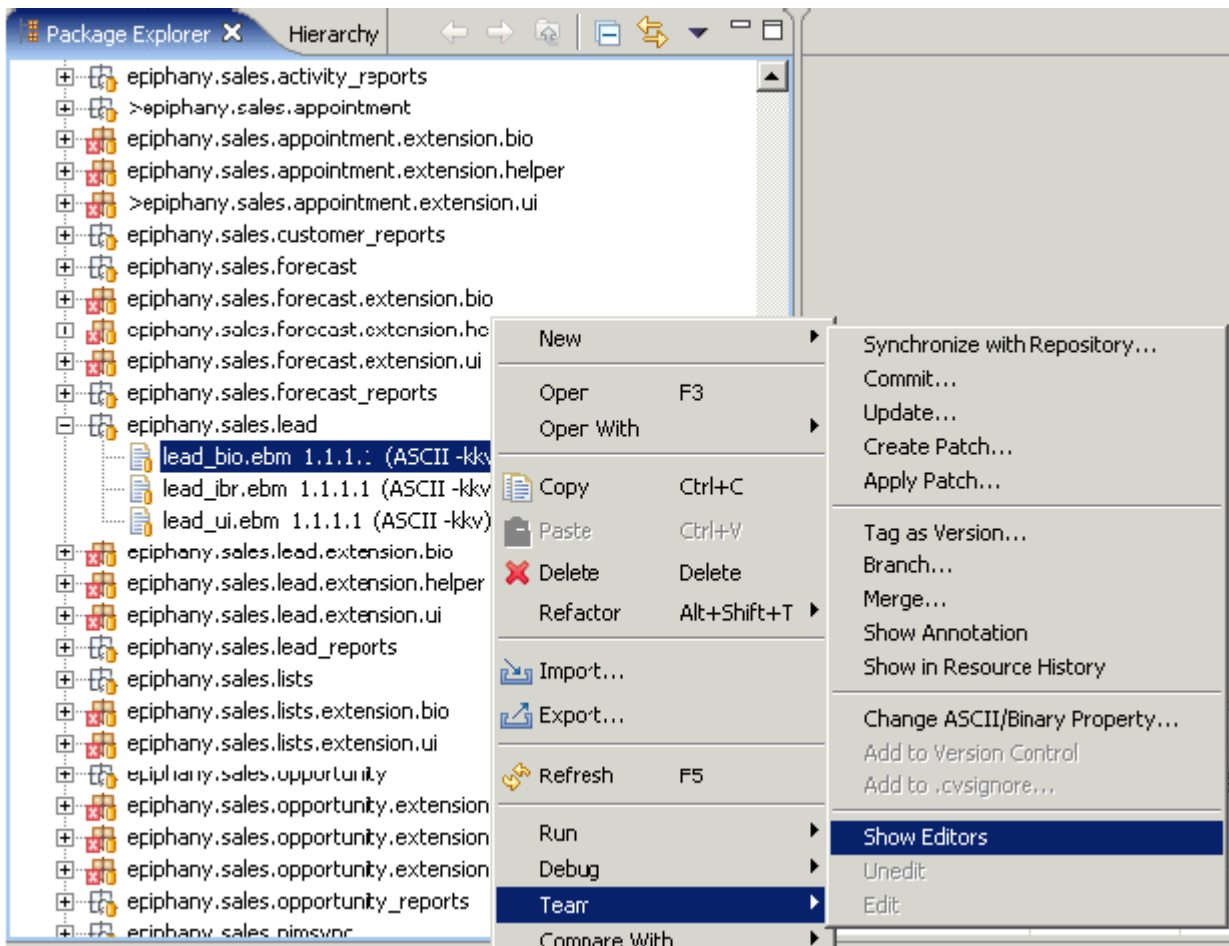
## Checking Out a File

Before you begin any core development work, you have to check out the module that contains that development set of work from the CVS Repository. When checking out modules from the CVS Repository, make sure that you check out only one module at a time. Checkout only those modules for which you know you have to execute development in. Follow this procedure strictly because in many instances development team members have to share modules. Sticking to this procedure ensures the integrity of the developed modules.

To check out an individual file, do the following:

- 1 Start Eclipse.

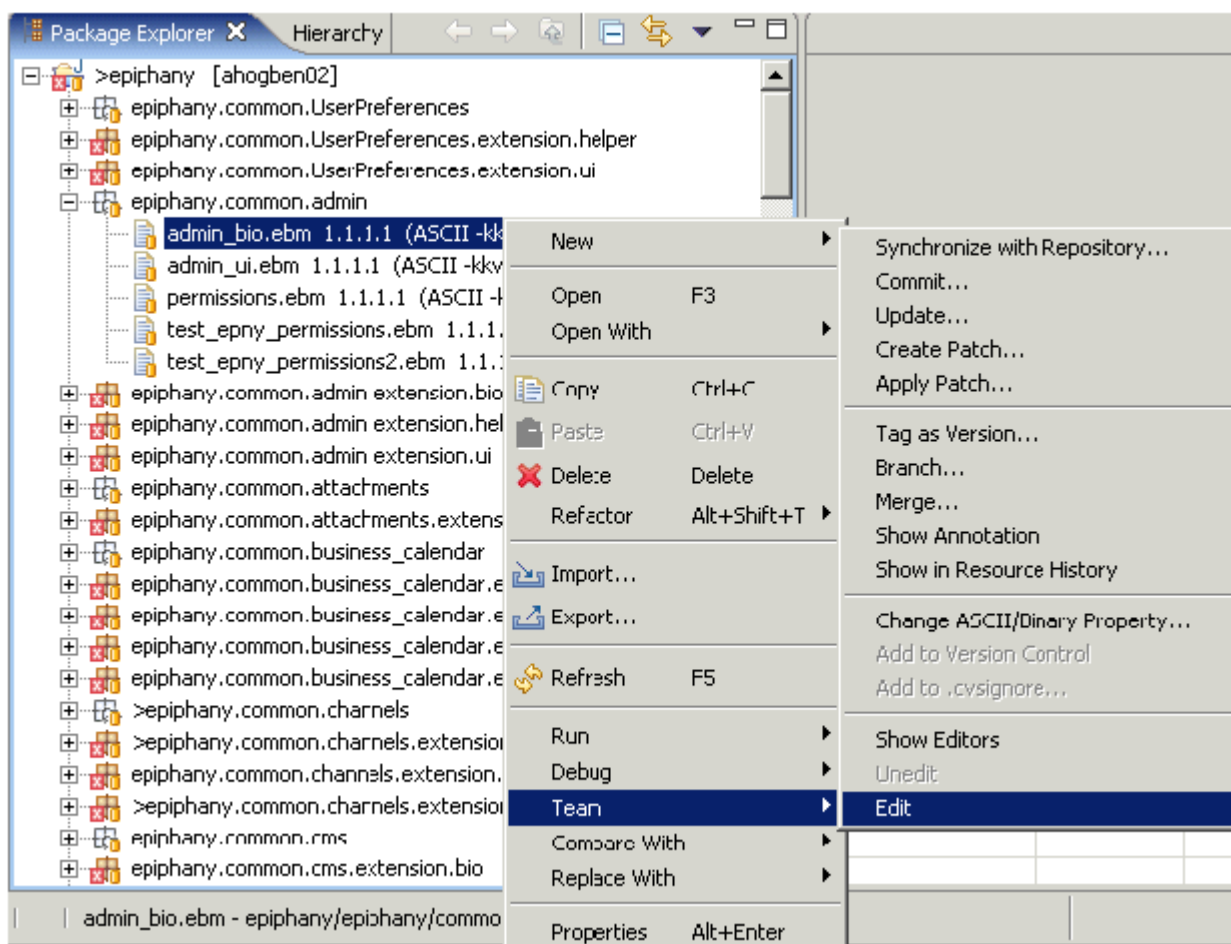
- 2 Open CVS Repository in the Java Perspective.
- 3 Before checking out any files, do the following:
  - a Get the latest files from the repository. Select the file, right click and select **Team > Synchronize with Repository**.
  - b Verify that no other developer is modifying the file that you want to check out. One way to verify that a file has not been checked out is to show the editors of a file. Right click on the selected file and select **Team > Show Editors**. If a file is checked out, a list of editors will be displayed in the bottom right window.



**Caution:** Even if you are editing a file, a second developer can still checkout the same file by ignoring the warning displayed by Eclipse. Therefore, take care to not work on a file that is currently checked out by another developer or plan on merging your changes later.

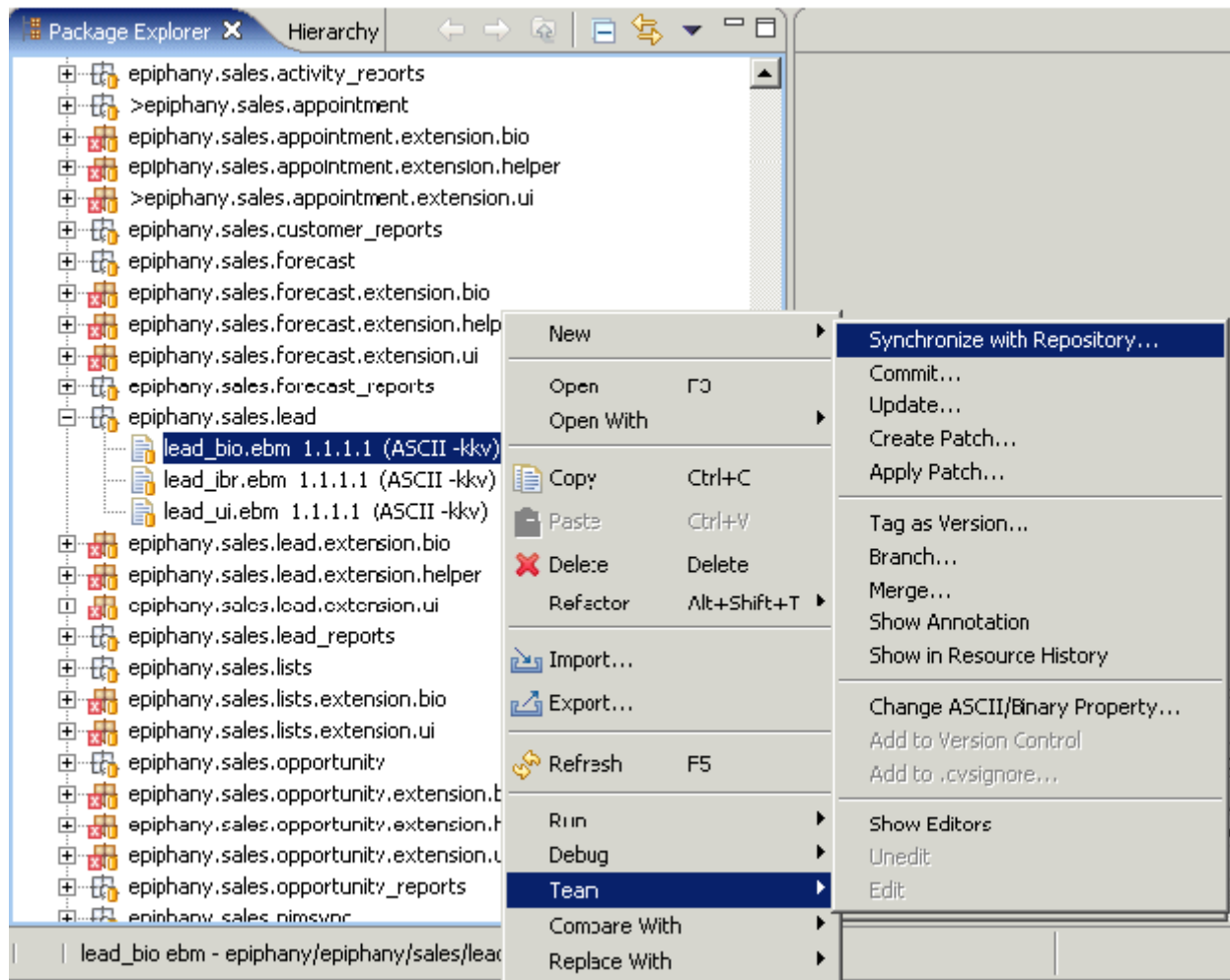
- 4 To check out a file, right click on a selected file and select **Team > Edit** in Java Perspective. Validate that the file has been checked out by using **Show Editors**. Another way you can verify that you

have a file is by clicking on **Team** and making sure that the **Edit** option is grayed out. If it is grayed out, then a developer is currently editing the file.




## Updating/Uploading Infor Modules

Prior to loading the new modules into your own metadata, you first have to validate that the download of all required modules is complete with the most up to date versions. You can do this by right-clicking on each module that needs to be downloaded, and clicking on **Team > Synchronize with Repository**. If there is a recent update to that file, a window on the bottom right entitled: **Synchronize - Incoming Mode** will display an update to the selected file. To update this file, right click on the file and select **Update from Repository**. This will update your local computer with the most recent version on CVS



Once validation of the module is complete, start Infor Studio to connect to your meta database and import the updated EBM files into the metadata for active development.

To correctly load the new modules to develop against, use Infor Studio to do the following:

- 1 From the Guide Bar, navigate to **Application Development > Modules**. A list of modules appears.
- 2 On the right side of the screen, identify the module that needs to be loaded.
- 3 Select the module to be loaded and click the  icon to open a new window.
- 4 Select the module to be loaded and click **Open**. Click **No** when prompted on whether to view the log file or to run the metadata consistency checker.

## Updating/Uploading Infor Delta Modules

Delta modules are modules that contain modifications to an out-of-box Infor module. To load delta modules, use the following steps.

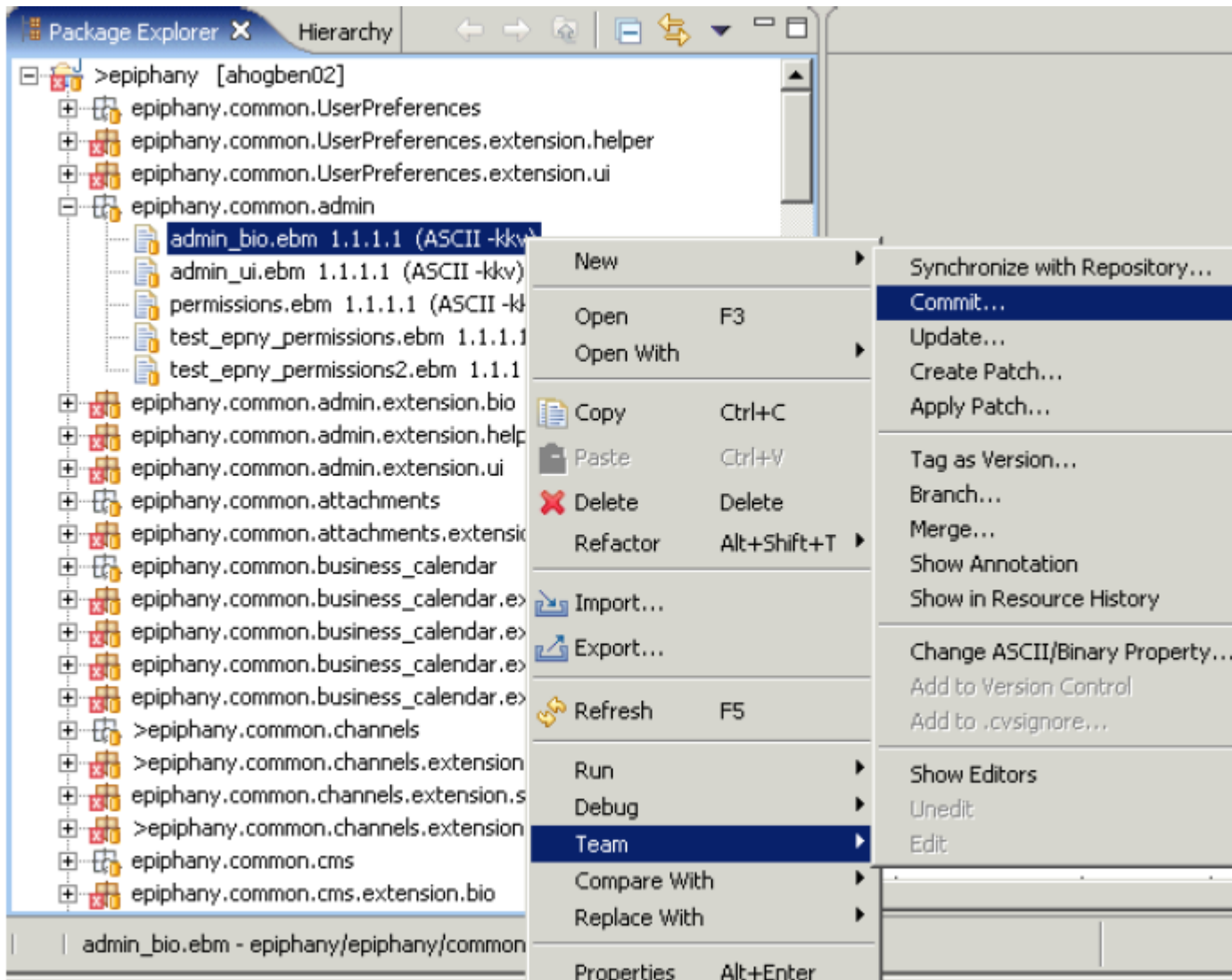


- 1 Check out the delta module from CVS. See the section on "Checking Out a File" on page 73.
- 2 Go to Infor Studio **Guide Bar > Application Development**
- 3 Right click on the out-of-box module that the delta module is modifying and select **Dselete** . Click **Yes** when you are warned that the module will be deleted.
- 4 Open a command window and navigate to the folder where the out-of-the-box module is located. All out-of-box modules are located within a subfolder under the following path:  
C:\Infor\service\modules
- 5 From the directory that the out-of-box module is located, enter the following command:  
**loadmodule -locked -module <module\_name>**
- 6 From the command window, navigate to the folder where the delta module is located. All delta modules are located within a subfolder under the following path:  
C:\eclipse\workspace\Infor\modules\custom
- 7 From the directory that the delta module is located, enter the following command:  
**loadmodule -loadDelta -module <delta\_module\_name>**
- 8 If the delta module has a related custom module, reload the custom module in Infor Studio.

## Checking In a Module

After you test the ebm/ebmd file and export the file using Infor Studio, you have to check the file into CVS repository. To check in a module, do the following:

- 1 Start Eclipse.
- 2 Open CVS Repository.
- 3 Export the ebm/ebmd file from Infor Studio into the proper folder that the file should be exported to (for example, save the customer\_bio.ebm file into the C:\eclipse\...\modules\custom\customer folder). To export a file, do the following:
  - a Navigate to **Application Development > Modules**
  - b Select a module and click **Save Module**. The Browse For folder window appears.
  - c Select the folder that the file should be saved in and click **Ok**.
- 4 In Eclipse, from the Java perspective, select the folder that the module that has been exported from Infor Studio and press F5 to refresh the folder. A ">" symbol appears next to the module if it is modified. In addition, a <module\_name>.xxx file also appears where xxx increments on each save. You can delete the old ones if they are no longer needed.
- 5 From the Java perspective, right click on the selected module and select **Team > Commit**. Fill in your comments as needed. Make sure that you add a brief description in the comments section for each file that you check in. This helps in tracking changes made to a file.



## Canceling a File Check Out

There can be instances during the development cycle when you have to cancel a change that has been done to a file. Make sure that you do not want to keep any changes you made. Changes made to a project file are permanently lost within CVS once a checkout is cancelled.

**Note:** Canceling a checkout on an Infor module does not remove the changes made to a module from a developer's local version of Infor Studio. Therefore, you can recover these changes into CVS by re-exporting the module from Infor Studio into the proper directory in CVS.

To cancel a checkout within CVS, use the following steps for each project that you want to cancel the checkout on.

- 1** In Java Perspective, select the file on the CVS Repository.
- 2** Right click on the unchecked out file and select **Team > Unedit**.
- 3** A confirmation message indicates that the local copy will be overwritten by whatever is currently on CVS. Click **OK** to confirm and CVS will overwrite and delete the file on the local workspace.



This chapter describes how to manage the post-installation custom build process.

## Build Process

A *custom build process* enables you to automatically build all components of an application and easily deploy the application across environments. A build process streamlines the development work performed by multiple developers, facilitates nightly builds, and formalizes application of custom application changes in testing, staging, and production environments.

A build process is relevant because Infor applications are developed in a metadata based environment. It is difficult to version control metadata in which multiple developers work because the environment can be easily corrupted. You therefore require reproducible procedures to constantly integrate developer code while at the same time not create black holes of time that impede development productivity. A build process also emphasizes the need to always have a base-line backup of your environment with the ability to restore or rebuild at any moment.

## Build Process Phases

A build process consists of retrieve, construct, deploy, test, and release phases.

During the retrieve phase, you retrieve source components from version control into the local build environment.

During the construct phase, you execute scripts to build the source components into a deployable artifact. You also mark the source components and update version numbers.

During the deploy phase, you deploy the component on to the application server and perform the required configuration.

During the test phase, you test the deployment and make sure that there were no errors during the build process.

Finally, during the release phase, you make the deployed artifact available for users.

## Build Types

Build types include developer workstation builds, incremental builds, and full builds.

*Development workstation builds* are optionally performed by developers. They are used to synchronize work being performed by multiple developers in one environment. Infor recommends that you have these builds frequently, two to three times weekly during peak development times with many dependent parts.

*Incremental builds* occur on a more regularly scheduled basis. These builds are performed on the master development environment. Incremental builds do not clean the previous build, but instead build on top of the previous build. Infor recommends that you have these builds weekly.

*Full builds* occur during major milestones such as code drops, environment migrations, and during incremental builds when failures cannot be resolved. A full build includes steps that ensure that the environment begins in a stable state so that it is the most accurate representation of current code. You can use full builds to bring both UI and integration components together and for major migrations from one environment to another. Infor recommends that you perform a full build at major project milestones identified in the project plan.

## Build Source Components

A typical Infor application comprises of the following build components:

- Custom modules. Custom modules contain completely new metadata for entities designed specifically for a project.
- Delta modules. Delta modules encapsulate only the changes from the out-of-the-box module. Delta modules are typically used to modify out-of-box modules.
- Operational database scripts. These include scripts related to data definition, data manipulation, stored procedures, and seed data.
- Custom Java extension code. All Java source code files need to be compiled and added to the deployed archive to be accessed by the Infor core.
- Static files. Static files impact the look and feel of the application. These files include cascading style sheets and custom images/js files and so on, and these files are sometimes customized.

## Sample Build Process (local)

This section describes a sample build process and walks you through the retrieve, compile/create, deploy, test, and release phases of the build process. This sample build process assumes that you are using CVS as the version control tool, that Eclipse is used as your Java development environment, and that WebSphere is your application server.

This sample build process is based on the following flow.

**Figure 1: Sample Build Process (local)**

## Sample Build Script

The following script illustrates how you can perform the targets illustrated in the flow.

```
<project name="sample" basedir=". ">
  <!-- ##### Give user a chance to override without editing this file
    (and without typing -D each time the build is executed) ##### -->
  <property file="build.properties"/>
  <!-- ##### Set the properties that control names and versions ##### -->
  >
  <property name="custom.jar" value="custom.jar"/>

  <!-- ##### Set the properties that control various build options -->
  <property name="debug" value="true"/>
  <property name="optimize" value="true"/>

  <!-- ##### Set properties related to the source tree ##### -->
  <property name="connectors.dir" location="${basedir}\..\connectors"/>
  >
  <property name="database.dir" location="${basedir}\..\database"/>
  <property name="database.ddl" value="${database.dir}\ddl"/>
  <property name="database.dml" value="${database.dir}\dml"/>
  <property name="database.sp" value="${database.dir}\sp"/>
  <property name="database.seed" value="${database.dir}\seed"/>
  <property name="eai.dir" location="${basedir}\..\eai"/>
  <property name="eai.dtds" value="${eai.dir}"/>
  <property name="eai.dtds.custom" value="${eai.dtds}\custom"/>
  <property name="extensions.dir" location="${basedir}\..\extensions"/>
  >
  <property name="lib.dir" location="${basedir}\..\lib"/>
  <property name="modules.dir" location="${basedir}\..\modules"/>
  <property name="modules.custom" value="${modules.dir}\custom"/>
  <property name="modules.custom.ebm"
    value="${modules.custom}\ebm"/>
  <property name="modules.custom.ebmd"
    value="${modules.custom}\ebmd"/>
  <property name="modules.oob" value="${modules.dir}\oob"/>
  <property name="modules.oob.ebm"
    value="${modules.oob}\oob.ebm"/>
  <property name="modules.oob.ebmd"
    value="${modules.oob}\oob.ebmd"/>
  <property name="static.dir" location="${basedir}\..\static"/>
  <property name="extensions.src" value="${extensions.dir}\src"/>

  <!-- ##### Set the properties for the build area ##### -->
  <property name="build.dir" location="${basedir}"/>
  <property name="build.classes" value="${build.dir}\classes"/>
```



```

<property name="build.lib" value="${build.dir}\lib"/>
<property name="build.script" value="${build.dir}\script"/>

<!-- ##### Set up properties for the distribution area ##### -->
<property name="distribution.dir" location="${basedir}\..\distribution"/>
<property name="distribution.name" value="custom-dist"/>

<property name="dist.dir" location="${basedir}\..\dist"/>
<property name="dist.lib" value="${dist.dir}\lib"/>
<property name="dist.bin" value="${dist.dir}\bin"/>
<property name="dist.database" value="${dist.dir}\database"/>
<property name="dist.database.ddl" value="${dist.database}\ddl"/>
<property name="dist.database.dml" value="${dist.database}\dml"/>
<property name="dist.database.sp" value="${dist.database}\sp"/>
<property name="dist.database.seed" value="${dist.database}\seed"/>
<property name="dist.eai" value="${dist.dir}\eai"/>
<property name="dist.eai.dtds" value="${dist.eai}\dtds"/>
<property name="dist.eai.dtds.custom"
  value="${dist.eai.dtds}\custom"/>
<property name="dist.modules" value="${dist.dir}\modules"/>
<property name="dist.modules.custom"
  value="${dist.modules}\custom"/>
<property name="dist.modules.custom.ebm"
  value="${dist.modules.custom}\ebm"/>
<property name="dist.modules.custom.ebmd"
  value="${dist.modules.custom}\ebmd"/>
<property name="dist.static" value="${dist.dir}\static"/>

<!-- ##### Set the properties for the Infor install area ##### -->
<property name="Infor.dir" location="C:\Infor"/>
<property name="Infor.service" value="${Infor.dir}\service"/>
<property name="Infor.service.lib"
  value="${Infor.service}\lib"/>
<property name="Infor.service.Web"
  value="${Infor.service}\Web"/>
<property name="Infor.shared" value="${Infor.dir}\shared"/>
<property name="Infor.shared.lib" value="${Infor.shared}\lib"/>
<property name="Infor.shared.etc"
  value="${Infor.shared}\etc"/>
<property name="Infor.shared.etc.eaidtds"
  value="${Infor.shared.etc}\eaidtds"/>
<property name="Infor.shared.src"
  value="${Infor.shared}\src"/>

<!-- ##### Set the paths used in the build ##### -->
<path id="classpath">
  <pathelement location="${Infor.service.lib}\ip.jar"/>
  <pathelement location="${Infor.shared.lib}\shared.jar"/>
  <pathelement location="${Infor.shared.lib}\jdom.jar"/>
  <pathelement location="${Infor.shared.lib}\jakarta-regexp-1.1.jar"/>
</path>
<pathelement location="${build.classes}"/>

```

```
</path>

<!-- ##### Clean build classes and lib directories ##### -->
<target name="clean"
  description="--> Cleans up the intermediate build directories">
  <delete dir="${build.classes}"/>
  <delete dir="${build.lib}"/>
  <delete dir="${dist.dir}"/>
</target>

<!-- ##### Build the code ##### -->
<target name="compile" description="--> Compiles the source code">
  <mkdir dir="${build.classes}"/>
  <mkdir dir="${build.lib}"/>
  <javac srcdir="${extensions.src}" destdir="${build.classes}"
    debug="${debug}" optimize="${optimize}">
  <classpath refid="classpath"/>
</javac>
</target>

<!-- ##### Create the custom.jar ##### -->
<target name="create_jar" depends="compile"
  description="--> Creates the custom jar">
  <jar jarfile="${build.lib}/${custom.jar}" basedir="${build.classes}"
"/>
</target>

<!-- ##### Create distribution tree ##### -->
<target name="dist:stage" description="--> Stage custom.jar and
  static files for custom build distribution">
  <mkdir dir="dist.dir"/>
  <mkdir dir="dist.lib"/>
  <mkdir dir="dist.bin"/>
  <mkdir dir="dist.database"/>
  <mkdir dir="dist.eai"/>
  <mkdir dir="dist.eai.dtds"/>
  <mkdir dir="dist.eai.dtds.custom"/>
  <mkdir dir="dist.modules"/>
  <mkdir dir="dist.modules.custom"/>
  <mkdir dir="dist.static"/>
  <copy todir="${dist.bin}" fromdir="${build.script}"/>
  <copy todir="${dist.lib}">
    <fileset dir="${build.lib}">
      <include name="${custom.jar}"/>
    </fileset>
    <fileset dir="${lib.dir}">
      <include name="custom*.jar"/>
      <include name="patch*.jar"/>
    </fileset>
  </copy>
  <copy todir="${dist.state}">
    <fileset dir="${dist.state}">
      <include name="*.vm"/>
    </fileset>
  </copy>
</target>
```

```

    </fileset>
  </copy>
  <copy todir="${dist.static}" fromdir="${static.dir}"/>
  <copy todir="${dist.eai.dtds.custom}">
    <fileset dir="${eai.dtds.custom}">
      <include name="*.dtd"/>
    </fileset>
  </copy>
  <copy todir="${dist.modules.custom}">
    <fileset dir="${modules.custom}">
      <include name="**\*.*/>
    </fileset>
  </copy>
  <copy todir="${dist.database}">
    <fileset dir="${database}">
      <include name="**\*.sql"/>
    </fileset>
  </copy>
</target>

<!-- ##### Builds, copy distribution to local Infor installation
and deploys them in the local Infor environment##### -->
<target name="dist:local" depends="create_jar,dist:stage,dist:sql,
dist:loadmodule,dist:deployment"
description="--> For development builds: copies custom.jar
and static files to local Infor installation">
  <copy todir="${Infor.service.lib}">
    <fileset dir="${dist.lib}">
      <include name="${custom.jar}"/>
    </fileset>
    <fileset dir="${dist.lib}">
      <include name="custom*.jar"/>
    </fileset>
  </copy>
  <copy todir="${Infor.shared.lib}">
    <fileset dir="${dist.lib}">
      <include name="patch*.jar"/>
    </fileset>
  </copy>
  <copy todir="${Infor.service.web}" fromdir="${dist.static}"/>
  <copy todir="${Infor.shared.src}\web\templates\ui">
    <fileset dir="${dist.lib}">
      <include name="*.vm"/>
    </fileset>
  </copy>
  <copy todir="${Infor.shared.etc.eaidtds}">
    <fileset dir="${dist.eai.dtds.custom}">
      <include name="*.dtd"/>
    </fileset>
  </copy>
</target>

<!-- ##### load the custom tables, stored procedures,

```

```
    custom data ##### -->
<target name="dist:sql" depends="dist:datadefinitionscript,
    dist:datamanipulationscript, dist:storedprocedure, dist:seeddata" >
</target>

<!-- ##### load the schema definition scripts, eg , custom tables#####
-->
<target name="dist:datadefinitionscript" >
</target>

<!-- ##### load the schema modification scripts eg, alter table,
etc, ##### -->
<target name="dist:datamanipulationscript"
    depends="dist:datadefinitionscript" >
</target>

<!-- ##### load the custom stored procedures ##### -->
<target name="dist:storedprocedure"
    depends="dist:datamanipulationscript" >
</target>

<!-- ##### load the custom seed data , like agents, userdata etc #####
-->
<target name="dist:seeddata"
    depends="dist:datadefinitionscript, dist:datamanipulationscript" >
</target>

<!-- ##### load the custom modules and out-of-box modified
modules ##### -->
<target name="dist:loadmodule" depends="dist:removeoobmodules,
    dist:loadoobmodules, dist:loadoobdeltamodules,
    dist:removecustommodules, dist:loadcustommodules" >
</target>
<!-- ##### removes the oob modules corresponding to the
customized oob module (ie delta modules) ##### -->
<target name="dist:removeoobmodules"
    depends="dist:listoobdeltamodules" >
</target>

<!-- ##### loads the oob modules corresponding to the
customized oob module (ie delta modules) ##### -->
<target name="dist:loadoobmodules"
    depends="dist:removeoobmodules" >
</target>

<!-- ##### loads the customized oob module (i.e. delta modules) #####
-->
<target name="dist:loadoobdeltamodules"
    depends="dist:loadoobmodules" >
</target>

<!-- ##### removes the custom modules to prevent conflict
while loading the custom modules later ##### -->
```

```

<target name="dist:removecustommodules"
  depends="dist:listcustommodules" >
</target>

<!-- ##### loads the custom modules ##### -->
<target name="dist:loadcustommodules"
  depends="dist:removecustommodules" >
</target>

<!-- ##### evaluates the list of out of the box delta
modules whcih need to be loaded ##### -->
<target name="dist:listoobdeltamodules" >
</target>

<!-- ##### evaluates the list of custom modules which need to
be loaded ##### -->
<target name="dist:listcustommodules" >
</target>

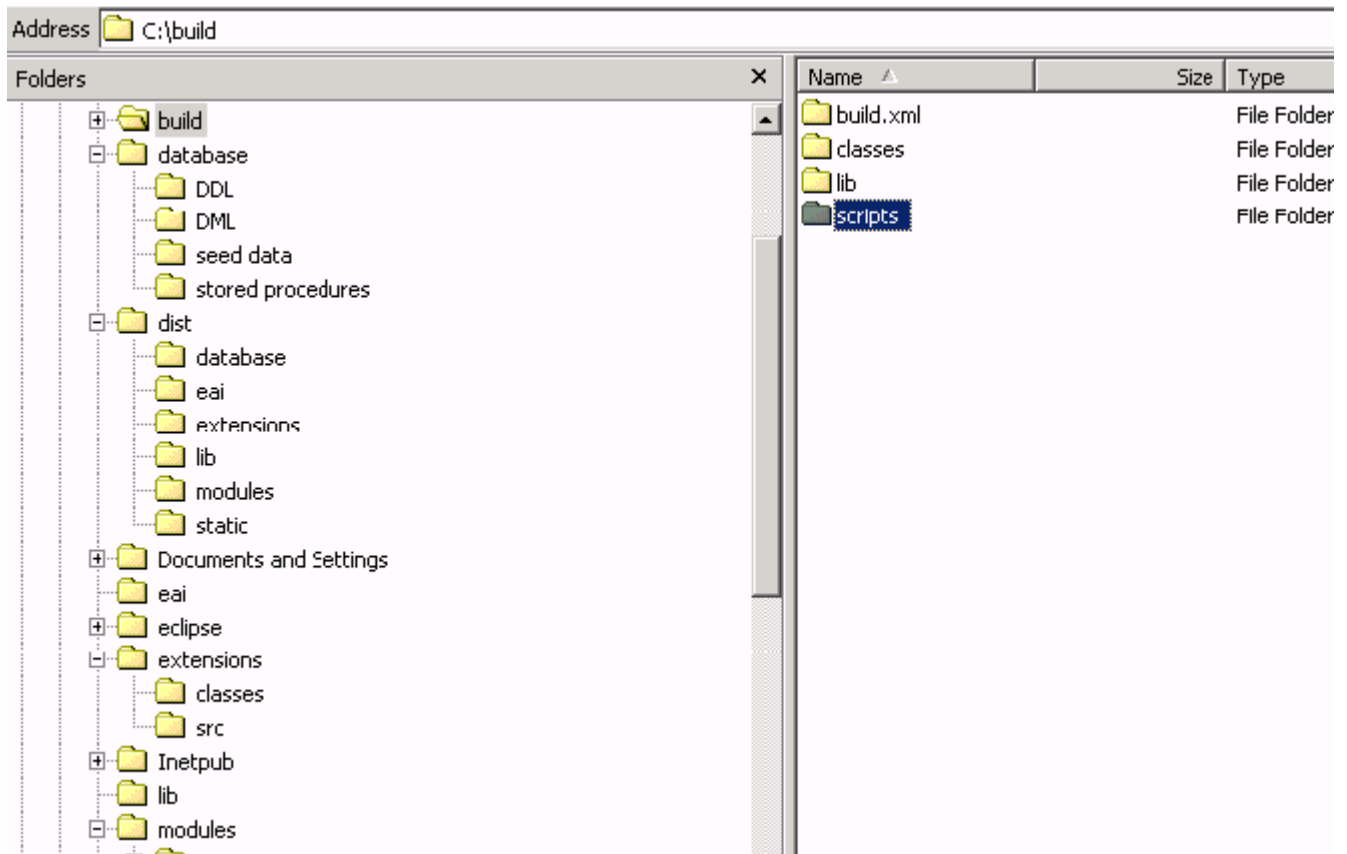
<!-- ##### deploys the application in the current environment ##### --
>
<target name="dist:deployment" >
</target>

<!-- ##### Package distribution and FTP to production server ##### -->
<target name="dist:remote" depends="create_jar,dist:stage"
  description="--> For full/incremental builds:
Uploads custom.jar and static files to remote server">
  <zip destfile="${distribution.dir}\${distribution.name}.zip">
    <zipfileset dir="${dist.dir}" prefix="${distribution.name}"/>
  </zip>
  <ftp server="${ftp.server}"
    remotedir="${remote.dir}"
    userid="${username}"
    password="${password}"> <!-- note: ftp server properties can
        be stored the build.properties,
        which is loaded at the beginning of the
        project -->
    <fileset dir="${distribution.dir}">
      <include name="${distribution.name}.zip"/>
    </fileset>
  </ftp>
</target>
</project>

```

## Sample Build Structure

A sample build structure appears as follows:



The sample version control structure has the following folders

build	Contains customized build scripts and build related files (build.xml, classes, lib, script, and src folders).
database	Contains SQL scripts for updating the Infor operational database (data definition, data manipulation, stored procedures, and seed data).
eai	Contains DTDs and other related files for handling EAI connectivity.
extensions	Contains source files for the Java extensions (classes and src folders).
lib	Contains Java archives for patches that need to be added to Infor installations.
modules	Contains custom (.ebm) and delta (.ebmd) Infor metadata modules in out-of-the-box and custom folders.
static	Contains static images and style sheets (.css) for customizing the look and feel of the Infor application.

---

dist	Distribution tree
------	-------------------

---

The rest of this section elaborates on the steps in the sample build process.

## Dist: retrieve: Retrieve Source Components

Dist:retrieve involves retrieving source components from version control.

To retrieve source components using Eclipse, do the following:

- 1 Start Eclipse and open the CVS Repository Exploring perspective.
- 2 Right-click on the correct version of the *Infor* project and select **Update** .
- 3 Repeat process for additional CVS Modules and all database scripts held in the current branch.

For more information on using CVS, see the chapter "Source Control" on page 69.

## Dist:local

For development builds, dist:local compiles custom classes, stages files, loads SQL tables and modules, and deploys them in the local Infor installation environment.

## Dist:compile

Dist:compile involves compiling and creating all custom extensions and classes and creating the custom.jar.

## Dist: stage: Create Distribution Tree

Dist:stage involves staging the custom.jar, static files, modules, and SQL scripts for custom build distribution. This step creates the distribution tree. The distribution tree includes directories such as lib, bin, database, eai, modules, and so on. This step copies the custom.jar and patch.jar, vm files, modules, database scripts, and static files to specific folders.

## Dist: SQL: Update Operational Database

Dist: SQL updates the operational database by running scripts related to data definition and data manipulation, stored procedures, and seed data.

## Dist: load module: Load Out-of-the-box and Custom Modules

Dist:load module loads the updated out-of-the-box modules and custom modules. This step involves unloading the out-of-the-box modules corresponding to the delta modules, loading the out-of-the-box and delta modules, and unloading and loading the custom modules.

### Updating Out-of-the-box Modules

#### Unloading Out-of-the-box Modules (unloadoob.cmd/.sh)

To resolve conflicts between the out-of-box modules and existing modules with deltas applied, you have to unload every out-of-box module that has a delta module associated with it. The out-of-box modules are found under:

C:\Infor\service\modules

You unload the out-of-the-box modules by calling the *loadmodule* utility with the *-justClear* parameter for each of the modules. For example,

```
C:\eclipse\workspace\Infor>loadmodule -justClear -module C:\Infor\service\modules\customer_
bio.ebm
```

For more information on the *loadmodule* utility, see the *Sales and Service Architecture Reference Guide*.

#### Loading out-of-the-box Modules (loadoob.cmd/.sh)

For all builds, load the out-of-box modules prior to the delta module load. This counters the cumulative effects of incremental delta changes. Load the out-of-box modules only for the modules where deltas have been defined.

Execute the *loadmodule* command with the *-locked* parameter for each of the modules. For example,

```
C:\eclipse\workspace\Infor>loadmodule -locked -module C:\Infor\service\modules\customer_
bio.ebm
```

#### Loading Delta Modules (loaddelta.cmd/.sh)

Delta modules are deployed similar to custom modules except the *loadmodule* command is executed with the *-loadDelta* parameter. For example,

```
C:\eclipse\workspace\Infor>loadmodule -loaddelta -module C:\Infor\service\modules\custom\
customer\customer_bio.ebmd
```

### Updating Custom Modules

#### Unloading Custom Modules (Batch File unloadcustom.cmd/.sh)

Existing custom modules can have conflicts with the out-of-box modules. These conflicts are often resolved by loading a delta. However, since out-of-box modules are reloaded without the deltas during full builds, the custom modules also have to be removed to resolve conflicts.



## Loading Custom Modules (loadcustom.cmd/.sh)

Before building the service archive, deploy the custom modules into Infor to load the JSP forms metadata. You deploy custom modules on the target build machine using the *loadmodule* command-line tool. You have to execute the *loadmodule* command for each module or directory of modules.

After you load the modules into Infor, the JSP forms are stored as metadata in the Infor database. These forms are dynamically rendered as needed.

## Dist: deployment: Build and Deploy the Service Enterprise Archive

Dist: deployment involves building and deploying the service enterprise archive. For more information, see the *Installation and Configuration Guide*.

**Note:** During the course of development, environment variables and properties can be created. These have to set according to the developer specification during each build. This step varies per build and is on request by the development team.

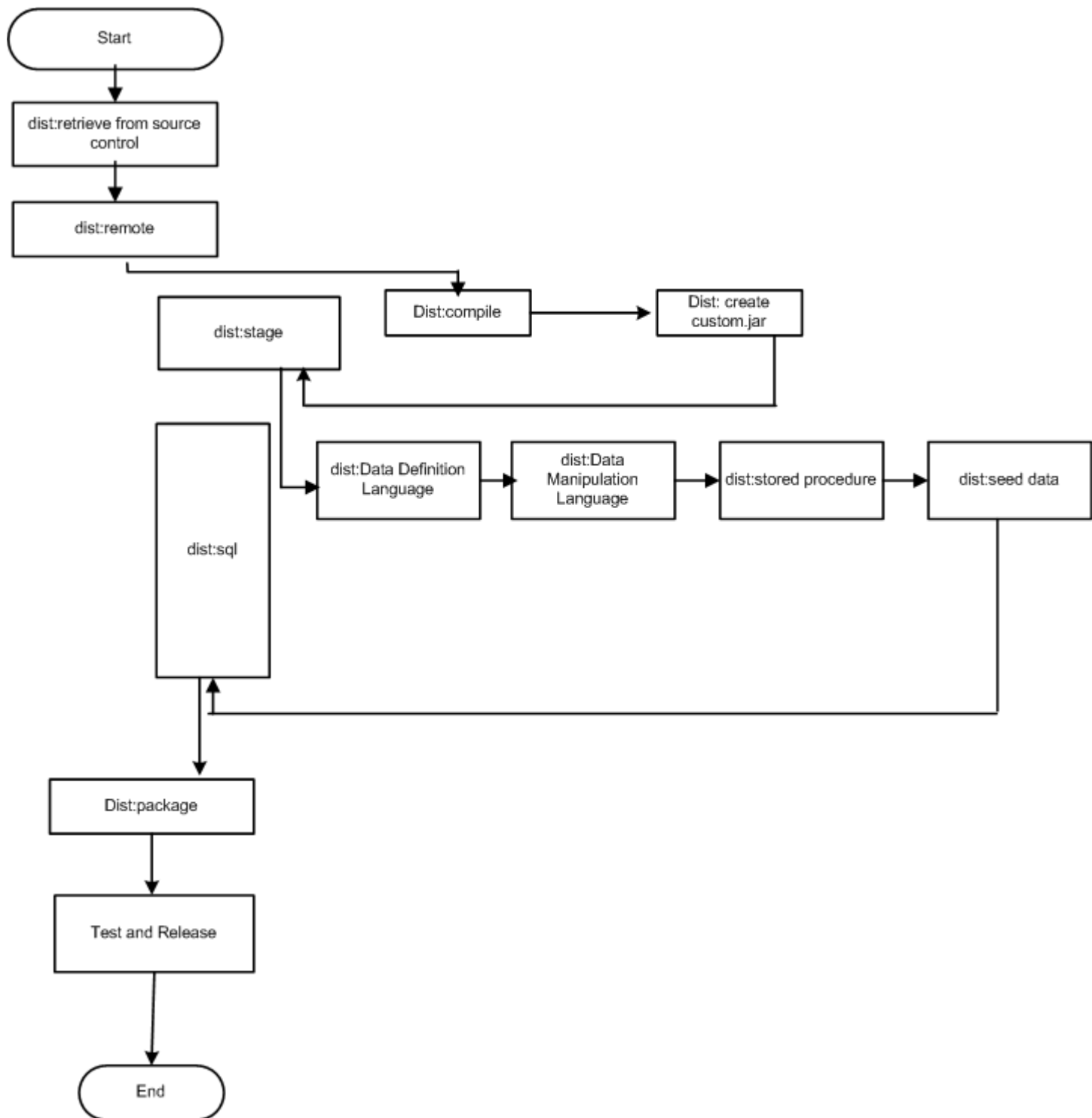
## Test and Release the New Build

The final steps in the custom build process is to test and release the new build. Use the following steps to test and release the new build:

- 1 Load sample test data that supports the new build environment.
- 2 Perform a sanity check.
- 3 Verify that the correct build number is deployed.
- 4 Verify that there are no errors during the build process.
- 5 Start the application server and verify that there are no errors.
- 6 Make sure that all source components are tagged with a specific build number.

## Sample Build Process (remote)

The flow for a sample build process (remote) appears as follows:



**Figure 2: Sample Build Process (remote)**

The sample build process (remote) is relevant to production environments. This flow uses the `dist:remote`, `dist:stage`, and `dist:package` targets. These targets allow you to compile, create, stage, and package

the files for installation on a remote server. Files include custom.jar file along with the latest Infor modules, SQL scripts, and static files.

The following section from the sample build script elaborates the targets further.

```
<!-- ##### Package distribution and FTP to production server ##### -->

<target name="dist:remote" depends="create_jar,dist:stage,dist:package"

  description="--> build for remote  servers">
</target>

<target name="dist:package" depends="create_jar,dist:stage"
  description="--> package the build for installing on remote  servers
">

  <zip destfile="${distribution.dir}\${distribution.name}.zip">
    <zipfileset dir="${dist.dir}" prefix="${distribution.name}"/>
  </zip>
  <ftp server="${ftp.server}"
    remotedir="${remote.dir}"
    userid="${username}"
    password="${password}"> <!-- note: ftp server properties
                                can be stored the build.properties,
                                which is loaded at the beginning of
                                the project -->
    <fileset dir="${distribution.dir}">
      <include name="${distribution.name}.zip"/>
    </fileset>
  </ftp>
</target>
```

## Build Process after an Upgrade

This section describes the build process in an ongoing development environment and how the build process can change after an upgrade is performed.

## Ongoing Development Environment

This section covers version control requirements and how to add and update a development machine in an ongoing development environment.

## Version Control Requirements

Make sure that you do the following:

- 1 Check in ebm customization.
- 2 Check in extension customization.
- 3 Check in sql scripts for operational database customization.
- 4 Check in any patches required for the Infor setup.

## Adding a Development Machine

To add an additional development machine in an ongoing development environment, do the following:

- 1 Install the Infor software and select that you want to create and initialize the database.
- 2 Apply the latest feature pack deployer.
- 3 Execute any custom sql scripts on the operational database if they are not already integrated into previous step.
- 4 Load the application modules (Loadmodule -application)
- 5 Load any test data.
- 6 Add customization.
  - a Load new custom ebms and ebmds.
  - b For out-of-the-box module customization, load out-of-the-box ebms from ClearCase and then load their ebmds.

Infor recommends that you load the ebms from ClearCase before loading all the corresponding ebmds from ClearCase. This extra loading of ebms enables you to avoid issues when people hijack / different local version of ebms on their machine.
  - c Add custom extensions.
- 7 Add the Infor patches (if any).

## Updating a Development Machine

To keep the development machine updated with ongoing updates (assuming no Infor software upgrade), do the following:

- 1 Add new/ update customized ebms.
- 2 Add new/ update customized extensions.
- 3 Add any patches from Infor.
- 4 Update the Operational database with custom sql scripts for any schema changes.

Prerequisite changes:

  - a Remove patches from Infor.
  - b Check-in upgraded ebms for all modules (out-of-the-box and customized) into a single clearcase directory.

- c** (Optional). Make a backup of the operational database.



This chapter addresses performance issues related to the design and implementation of Customer Service.

## Overview

A number of factors influence performance of an Customer Service implementation. These include database, application, and Web server set up, network set up, customizations made to database objects (such as tables, views, and indexes), and customizations made to extensions.

The first part of this chapter covers performance issues related to application design and deployment, process for improving application performance, and provides a sample performance improvement task list. The second part of this chapter addresses performance tuning and provides information on how to diagnose, analyze, and troubleshoot performance issues in deployments.

## Performance Issues Related To Application Design

The choices that you make during application design can impact performance. The following table lists questions about the design of the application and possible solutions to improve performance.

Question	Solution
How much data is displayed in the Home screen? This screen appears when users log into the application and return to it while using the application.	Consider reducing the amount of data displayed in the Home screen.
Are the “List Window” and “Max Result Count” settings high?	Reduce the number of records displayed in list forms and the number of records read while rendering the screen.
Is the display count type for list forms set to “Exact Count”?	Use Estimated Count (along with a lower Max Result Count).

Question	Solution
How many editable drop-downs are used in list forms that are performing slowly?	Limit the number of editable drop-downs in a list form.
Which drop-downs use large attribute domains?	Change drop-downs with large attribute domains to pop-ups to defer the cost of loading the domain.
What extraneous data is displayed in detail forms?	Move lesser-used data elements to other detail forms that can be accessed only when required.
How heavy is the form in the first tab?	Use a lighter form in the first tab or reduce the amount of data displayed in the current form.

## Extension Development Guidelines

When you write extensions, make sure that you follow extension programming guidelines. For information on extension programming guidelines, see the Sales and Service Architecture Reference Guide.

In addition, consider the following examples of best practice.

### Fetching User Preference values

To minimize the queries during application extension processing, extension code does not have to re-query user\_data BIO to access preferences. Preferences can be accessed from the UIRenderContext object using the following API call

```
context.getServiceManager().getUserContext().getPreference(...)
```

In this example, context is of type UIRenderContext.

### Fetching user\_data BIO for current user

To minimize the queries during application extension processing, extension code does not have to re-query user\_data BIO for current user information. Instead, that information can be obtained from the UIRenderContext using the Service Manager API Call as follows:

```
context.getServiceManager().getAgentUserBio()
```

In this example, context is of type com.epiphany.shr.ui.action.UIRenderContext.

## Performance Issues Related to Deployment

Performance issues can emerge at customer deployments because customizations can alter application performance in significant ways. Three forms of performance problems can emerge.



- All screens are slow during single user testing. These problems occur due to environment problems. For example, if the firewall between the Web server and the application server is incorrectly configured, it can cause network delay issues across the entire application.
- Only certain screens or actions are slow during single user testing. These are usually related to a customization or database (or external source) optimization. For example, an extension can take a long time or issue a query that uses an inefficient query execution plan (scanning the entire table, full index scan, etc).
- Screens are slow only when many users simultaneously access the server (scalability issues).

Performance optimization efforts in deployments typically fall into two areas

- Improving performance of some or all of the functionality for a single user. The focus here is on the experience of a single user accessing the application and optimization effort is typically geared towards specific functionalities.
- Improving how the application manages large numbers of users. In this context, optimizing efforts are geared towards improving scalability.

## Process for Improving Application Performance

The process of improving application performance in existing implementations involves

- 1 Verifying application settings and behavior.
- 2 Documenting configuration
- 3 Identifying areas that impact performance
- 4 Identifying activities where performance should be improved
- 5 Modifying the activities that impact performance
- 6 Evaluating the results of modifications

You perform steps 4-6 iteratively. You identify an area that impacts performance, modify that area, and test the application to determine the impact of the modification. You repeat these steps until performance is sufficiently optimized.

The next section elaborates the above steps. This section assumes that you have installed Customer Service and that you have used the Hardware Sizing Guidelines document available from Infor Customer Support to size servers, and so on.

### Verifying Application Settings and Behavior

Application settings determine how functionality is loaded and executed. Therefore, use the Infor recommended settings for production environments. For information on performance impact of turning user preferences on or off, see Application Reference Guide.

Evaluate application behavior to ensure that objects are loaded and executed properly. Before performance testing, do the following:

- Set logging to production levels. You can temporarily change these settings to identify areas where performance can be improved. Return to the production settings to test changes and continue performance testing.
- Disable auto reloading of classes.
- Make sure that all JSP classes are compiled and that they are not recompiled each time they are accessed.
- Make sure that the application does not encounter errors. Even benign errors can have a cost due to the logging. They can also represent situations where parts of the system are blocked on a time-out for something that is not available. Correct programming errors. When integration issues cause errors, resolve the errors or disable the functionalities encountering the problems.
- Make sure that extensions use proper logging, and do not write errors or developer information to the console.

## Documenting Configuration

Document your environment settings. This can help you to identify areas where performance can be improved and develop recommendations for optimizations. Gather the following information.

- Overall configuration - which elements are included?
  - Infor Epiphany Sales
  - Infor Epiphany Service
  - Infor Epiphany Inbound Marketing
  - CTI integration
  - PIM synchronization
  - Email integration
  - Integration with other external applications (list these)
- Hardware configuration - document for each database, application, and Web server as well as the typical client machines
  - System type (Intel, IBM, Sun, HP, and so on)
  - Number of CPUs
  - Amount of RAM
  - Number of I/O Controllers
  - RAID Level
  - Disk layout information
- Operating system configuration - document for each database, application, and Web server as well as the typical client machines
  - Operating system (NT, Windows 2008, UNIX, and so on)
  - Kernel parameters
  - Number and size of file systems

- Memory allocation for pagefiles
- Database server configuration - document for each database server
  - Database type (SQL Server, Oracle)
  - Version
  - Number of instances
  - Number of databases
  - Tablespace/Filegroup layout
  - Current parameter settings
- Web Server
  - Web server type (IIS, Apache)
  - Compression settings
  - Current parameter settings
- Network configuration - describe the network topology, including the number of users accessing the network through a LAN, WAN, and VPN

## Identifying Areas for Performance Optimization

Identify areas that impact performance by answering the following questions:

- How does application performance vary when a single user accesses the application from performance when many concurrent users access the application?
- When the number of concurrent users does not impact performance, will optimization efforts be focused on specific functionalities of the application, or will overall performance for a single user be examined?
- When specific areas will be optimized, what are the specific activities to be optimized?

When you have to optimize all functionalities for a single user, consider one or more of the configurations of the application server, Web server, database server, network, and client machines for modification. Review the information in the section "Documenting Configuration" on page 102 to determine discrepancies with recommended and default settings.

When you have to optimize the performance of a large number of concurrent users accessing the system, examine the configuration of the servers, network, and client machines. Settings that impact the volume of the memory available and bandwidth used are frequently the focus of optimization efforts. Configuration of the state server database also frequently impacts scalability. For more information on server configuration and scalability, refer to the Installation and Configuration Guide.

When you have to optimize performance of specific areas of the application, focus on database objects and extension programming.

Use the following steps to monitor and log application processing.

- When analyzing performance in a development environment, make sure that only one instance of the Web and application server is running. This step may not be practical in production environments.
- Enable logging, run the use cases, and review the logs to find the queries and activities where optimization efforts will be focused.
- Use Web server logs to determine the time required to complete requests.

Use the information that you gather to document areas that can be optimized.

## Making Modifications to Improve Performance

Once you identify the areas that need to be optimized, document recommendations for ways to improve performance. Review the recommendations to determine which ones can be used and the order in which changes will be made.

Use the following criteria to determine the changes that will be made and the sequence in which the changes are implemented:

- Cost and time required to make the changes
- Potential impact on other areas of the application
- Potential success of the change
- Impact on future upgrades
- Impact on external applications
- Impact on users

Areas where changes are typically made include

- Configuration settings (application, web, and database servers; network; client machines). For more information on server configuration, refer to the Installation and Configuration Guide.
- Database object changes (tables, views, indexes)
- Metadata
- Extension programming. For more information on how you can use Java extensions to customize the application, see the Sales and Service Architecture Reference Guide.

Once you make changes to specific areas, test the application to determine the impact of the modifications. Continue with modifications and testing until performance is sufficiently optimized.

## Sample Task List

Use the following sample task list to improve application performance when a single user accesses the application. You can use this list as a starting point for developing a plan for a specific implementation.

- Document the specific areas/use cases where performance optimization will be performed.
- Disable auto reloading of classes.

- Make sure that all JSP classes are compiled. Make sure they are not recompiled each time they are accessed.
- Review logs and verify that the application is not experiencing errors. Eliminate each error by correcting the problem or disabling the functionality that is encountering the error.
- Review the console log to locate any messages written by extensions. Correct any extension that is logging messages to the console.
- Compare the configuration of the application, web, and database servers to the recommended settings, and make any required changes. When your optimization effort is focused on a small number of specific activities, you can postpone this step.
- Document configuration of server and client machines.
- Make sure that no other applications are running or accessing the servers.
- While a single user works through use cases for the activities where performance will be optimized, identify where time is being spent.
- Document areas where performance can be improved.
- Document recommendations for improving performance.
- Evaluate the recommendations, prioritizing the modifications in the sequence in which they will be made.
- Make one or more modifications. Limiting the changes made through any pass allows you to more precisely determine the appropriate modifications for improving performance.
- Repeat the execution of use cases. Determine the impact of the modifications made.
- Continue iterating through the steps beginning with working through use cases until performance is optimized to the appropriate level.
- Log all activities used to identify areas where optimizations can be made and what modifications were implemented.

## Performance Tuning in Deployments

This section covers performance tuning in deployments in relation to

- "Quick Checks for Performance Problems" on page 105
- "Configuring the System for Performance Optimization" on page 106
- "Analyzing System Performance" on page 107
- "System Optimizations Undertaken in Implementations" on page 108

## Quick Checks for Performance Problems

- Check logging levels to make sure that they are at production levels. Excessive logging can slow the system down by requiring excessive disk I/O to write log messages to the log file.

- Make sure that the auto-reload classes setting is off because this can substantially affect the performance of an application server. When the auto-reload classes setting is enabled, the extension manager makes frequent checks to detect changes to the extension and these checks can be expensive. In a production environment, these extensions do not change and therefore the checks are unnecessary. Disable this setting in the deployment\_config.xml by setting auto\_reload\_classes = false (<Installation Directory>/shared/etc/deployment\_config.xml).

## Configuring the System for Performance Optimization

If you encounter performance problems, use the following process to determine the cause of the problem.

- 1 Shut down all but one instance in a cluster to force one application server to process all requests.
- 2 Turn debug logging on by switching the log4jconfiguration.xml file to use the debug version of the log configuration (<Installation Directory>/shared/etc/log4jconfiguration.xml). The debug logging level will decrease system performance, but will help isolate where the relative slowdown is located. The steps that increase logging are for diagnostic purposes only and are not intended for long term use.
- 3 Edit the log4jconfiguration.xml file to enable DEBUG level logging for the following categories:
  - DeveloperLog.Shared.BioLayerLog.PerfLog
  - DeveloperLog.Shared.BioLayerLog.DataProviderLog
  - com.epiphany.shr.util.sql.SelectStatement
  - com.epiphany.shr.util.sql.InsertStatement
  - com.epiphany.shr.util.sql.UpdateStatement
  - com.epiphany.shr.util.sql.DeleteStatement

For example, editing the first entry will result in a section of the log4jconfiguration.xml file to read

```
<category name="DeveloperLog.Shared.BioLayerLog.PerfLog">
  <priority value="DEBUG"/>
  <appender-ref ref="BioPerfFile"/>
</category>
```

You can also configure the application server to log in production mode and use the Admin Console to enable the selected categories.

Enabling DEBUG level logging will cause all the database queries to be executed, and the time taken by the database to process each query, to be logged to the log file.

- 4 Shut down the application server, remove all log files from the service\etc\log directory, and re-start the server.
- 5 Repeat the poorly performing use case two or three times to make sure that all JSP pages are compiled and all caches are populated. Typically, the first execution of a screen is slower than subsequent executions.

The log files will now include information that can be used to assess round trip performance in single user mode.



**Caution:** If you plan to send the logs to Infor Support or Engineering, collect all of the files in the Infor log folder and also the Web server and the application server logs for the same time period. Before sending the file, compress all these files into a zip file. Compression decreases the size of the file and makes it easier to handle and transfer the logs.

## Analyzing System Performance

Once clean log files are generated, use the Log Viewer to analyze the logs (**Start > Programs > Infor > Infor Studio > Logview**). The Log Viewer is installed with Infor Studio. Apply filters in the Log Viewer to identify pertinent information. Filters specifying a predetermined set of ID's are useful to analyze application server runs. Some of the key ID's to filter on are

- **LOG\_FINISH\_HTTP\_REQ\_PROC:** Logged when the controller servlet has processed and sent all data back to the browser. This provides a good indication of how long the request took. Since this time includes the full data transfer time, most of the browser rendering time is included. Each LOG\_FINISH\_HTTP\_REQ\_PROC brackets a screen rendering and includes the time spent in sending the bytes over to the browser. The category for this ID is `com.epiphany.shr.ui.controller.EpnyServlet`
- **LOG\_SSOSESSIONIDIMP\_FOUND\_EPI\_COOKIE:** Logged at the beginning of the processing of a HTTP request. This message provides a good marker to indicate when the request processing started.
- **LOG\_STATE\_SERVER\_SAVED\_BYTES:** Logged when the controller servlet has saved state server data back into the database. Identifying the time consumed in saving state is a good indication of problems with state server database configuration. Note that state server problems typically do not appear unless there are many records (generated by many users) in the state server tables.
- **LOG\_STATE\_SERVER\_RET\_ARR:** Logged when controller servlet has read state from the state server. Identifying the time consumed in reading state provides a good indication of problems with state server data-base configuration.
- **LOG\_SelectStatement.executeQuery:** Logs the Select SQL statement being executed. There are similar log statements for the Insert and Update cases as well. This SQL statement can be used to replay the statement outside the server (through SQL\*Plus or isql for example) to tune the database and determine if indexes are configured correctly.
- **LOG\_SelectStatement.executeQuery.Returned:** Logs how long the Select SQL statement took to execute. Use this to determine which SQL statements are candidates for performance tuning analysis.

Use the logs to determine which aspect of the request processing consumes the most time and target those areas for optimization. Different areas to consider are

- A database query that consumes a lot of time. This is the most common use case. Most database queries should return on the order of 10 milliseconds.
- State server optimizations. Make sure that the state server guidelines as specified in the Hardware Sizing Guidelines document available from Infor Customer Support have been followed, especially around I/O bandwidth to the disk drives. These typically become issues with multi-user scenarios. State server inserts and queries should return on the order of 30 milliseconds.

- Network connectivity issues
- Browser rendering time issues
- JSP compilation issues
- Background service

## System Optimizations Undertaken in Implementations

This section covers performance issues identified in different implementations and the optimizations undertaken to address the issues.

- Time is consumed in sending data to the browser.

This is the case when the time between the LOG\_STATE\_SERVER\_SAVED\_BYTES and the LOG\_FINISH\_HTTP\_REQ\_PROC is significant. For example, in one deployment, this difference was about seventy five percent of the time reported in LOG\_FINISH\_HTTP\_REQ\_PROC. This implies that seventy five percent of the round trip time was spent on returning the data to the browser. You can also check the Web server logs to determine processing time. The Web server plug-ins that are distributed with the WebLogic or WebSphere application server have tracing abilities that can help identify potential bottle necks. Once the issue is identified, standard networking troubleshooting methods can identify an invalid network configuration of a firewall causing the delays.

- Time is consumed in saving or reading state.

This is the case when you observe that the times reported in LOG\_STATE\_SERVER\_SAVED\_BYTES or LOG\_STATE\_SERVER\_RET\_ARR is a significant fraction of the time reported in LOG\_FINISH\_HTTP\_REQ\_PROC. The effect of this is that performance degrades slowly and eventually the system becomes unusable. To address this issue, make sure that the state server database configuration follows the guidelines identified in the Hardware Sizing Guidelines document, specifically that the tablespaces are laid out appropriately in order to maximize disk I/O capacity.

- One query takes a long time.

This is a common use case and can be identified by observing the times reported in the LOG\_SelectStatement.executeQuery.Returned. If one or more queries stand out as consuming a long time, you can address those queries in a number of ways:

- Using the SQL statement logged into the log file, you can optimize the database execution of the SQL statements at issue by adding or re-configuring indexes. Infor does not provide a tool for performing this process because there are database specific tools and utilities and third party products that can be used to help with the analysis. For example, an Oracle DBA would typically use EXPLAIN PLAN to see potential query performance enhancements.
- Some open ended queries can result in large amounts of data being processed. For example, extensions can specify optimization hints to the BIO layer that help with the query optimization. In some cases, either these are left unspecified or specified incorrectly resulting in open ended queries. To address issues such as these, analyze the extensions and modify them to be more efficient. For information on extension programing guidelines, see theSales and Service Architecture Reference Guide.
- An extension can also issue too many queries. To address this, analyze the extension and modify it to be more efficient. For information on extension programing guidelines, see theSales and Service Architecture Reference Guide.



- Determine why the core generates the query the way it does. Often, a configuration can cause the data provider layer to issue a query in a form that skips all the indexes that have been created. For example, the core can use a UPPER in the where clause resulting in a full table scan when there is an index on the regular column. You can also improve other poorly structured queries by changing BIO sub-classing, audit configurations, or record set configurations.
- Execution of an extension takes significant time.

This can occur because the extension implements a sub-optimal algorithm. You can determine this by analyzing the log files. Once you identify the slow extensions, analyze the code for optimization possibilities.
- JSP's are compiled every time.

Sometimes, a configuration parameter causes the dates of the JSP files to be updated which forces the application server to always re-compile the JSP files. Check the application server logs to make sure that the JSP files are not compiled on every page hit.
- The browser is slow.

This can be the case where the times reported in LOG\_FINISH\_HTTP\_REQ\_PROC are good but the user always experiences substantial delays. Using the Java 2 plugin with some versions of Internet Explorer can significantly slow down rendering times. The CPU speed of the browser is also important to consider.
- Background processing affects performance

Background processes such as alert subscription processing or state server cleanup can consume resources and temporarily affect the UI. Check if the background service log statements are present in the middle of the request processing log statements. If this is a persistent issue, target these services to a dedicated partition within the cluster. This will limit the impact of this activity on the interactive users.
- Third party processes affect performance

Sometimes, unrelated processes can impact performance. For example, an unrelated Oracle background process can cause delays. You can identify this by monitoring database performance and looking for activity that is not caused by the Infor application.
- Unusually high application server CPU and local disk usage

Production systems are not expected to have logging at debug levels. Logging can consume a fair amount of CPU and I/O resources. If you encounter performance issues at high user loads, check the amount of logs that are being generated. Even simple System.out statements can slow down the performance of a request measurably.

## Using the Lazy Load option

Mark Blobs/ Clobs (ntext, text, image) non-required columns as lazy loaded in the data field by default unless there is a specific reason not to do so. Mark the **Lazy load** field in Infor Studio in the recordset **Fields** subform of a recordset (**Guide Bar > Physical and Logical Schema > Recordsets and Relationships > Recordsets** ). Mark all large data fields added by the customer also as lazy loaded. Not marking large fields as lazy loaded can lead to performance issues that are hard to track. For example, if a large data column is not marked as lazy loaded, then a list form on such a BIO reads all blobs/clobs associated with it even to display the list. This is expensive especially if only one or two of the items in the list are expanded to actually see the details of where the large data is displayed. In

addition, the printed SQL timing does not account for the time it takes to load large data. It only informs when the query is returned. So you can miss large queries which are slow due to large blobs.

### Other Factors Affecting Performance

The following factors can also have an impact on performance.

- CPU on the application server machine. Faster CPU contributes to better performance. Do not run application servers on machines with 375 MHz CPUs because this can impact performance.
- Network performance. Monitor the number of hops it takes to ping the application servers. If the response times here are a problem, contact network engineers to help improve the response time.
- Database queries. Based on the extent of customizations, there can be queries that need to be tuned or optimized. You can check for these queries by turning on SQL-logging. For more information, see "Viewing SQL Queries In Logs" on page 110 in the next section.
- Removing System.out.println messages from extensions.
- Removing unhandled exceptions and other such application run-time errors by looking at the epny.log4j files.
- Web server settings. Check the settings on Web servers to make sure that there are not many bottlenecks related to the maximum number of concurrent HTTP requests. This becomes an issue when you use the Push service because you create persistent connections on the push port from the Web server.

For additional information, refer to the performance tuning documents available from Infor Customer Support.

### Viewing SQL Queries In Logs

All logging levels are controlled by the log4j configuration. The <Installation Directory>/shared/etc/log4jconfiguration.xml specifies what the settings should be upon server startup. You can also change the settings at runtime using the Admin Console, or through <serverurl>/Infor/soap/setPriority.jsp. (This is called the instance name. The default is Infor, but you can change it during installation while entering values for the Web server.)

To see the SQL queries used by the application in logs, when the application performs any action against the database, change the following log category to DEBUG:

DeveloperLog.Shared.BioLayerLog.DataProviderLog

You can also change the log category to DEBUG for the following:

com.epiphany.shr.util.sql.SelectStatement

com.epiphany.shr.util.sql.InsertStatement

com.epiphany.shr.util.sql.UpdateStatement

com.epiphany.shr.util.sql.DeleteStatement

If you use DeveloperLog.Shared.BioLayerLog.DataProviderLog level logging, you get SQL queries with all the parameter values filled in. You get only the queries/operations done on the BIOs (mostly on the operational database).

If you use SQL (statements) level logging, you do not get all the parameters in the query filled in. However, you get how much time each query takes. You also get all queries/operations run against both meta and operational database. Depending on your need, you can enable one or both.



---

# Using Out-of-the-box User Interface Templates

## 8

This chapter describes the templates on which the Infor Epiphany Service user interface is based, and provides an example of how you can make the forms that you build conform to the Infor Epiphany Service user interface look and feel.

## User Interface Template Forms

The Infor Epiphany Service user interface is based on the list, detail, tabgroup, and toggle templates.

### List Template

The **list template** form contains navigation for all objects that have a menu option on the navigation bar. When a user selects a menu option, a list form is shown in the top slot of the list user interface and a tab group opens when the user previews the record. If the list is associated with any hierarchy, then it is shown to the left of the list.

### Detail Template

The **detail template** form appears when the user chooses to edit an object in the list template. The detail template holds definitions for all objects (opportunities, leads, and so on) that are editable. The template has two parts, a top slot and a tabgroup slot. The top slot contains the focus object's detail form (for example, the Request detail form or the Opportunity detail form). The tabgroup slot contains all the information related to the object in the detail slot.

The contents of the detail slot are different if the business object supports customer/interaction summary. In this case, another form called the summary detail form is used. This form has two slots, a slot for the summary form and another slot for the detail form.

## Tabgroup Template

The **tabgroup template** form contains definitions for all the tabs that can be shown for a business object; the same template form is used for both the list and detail user interface. You define the difference between the list and detail views using form placements and extensions. All business objects have at least one difference between the two views; the detail view of the object in focus is available only in “List Template” tab group view.

## Toggle Template

The **toggle template** form is used in cases where the list form is in the tab-group. In this case, the user can add an item to the list, or view or edit an item from the list in the tabgroup.

The above templates exist in a Sales shared module and are named the Sales\_List\_Shell, Sales\_Detail\_Shell, Sales\_Tabgroup\_Shell, Sales\_Toggle\_Shell, and Sales\_Summary\_Detail\_Shell. These template forms define the Infor Epiphany Service user interface.

The following section describes how you can customize an existing form to conform to the out-of-the-box Infor Epiphany Service user interface. This is described in relation to making an object available from the List and Detail template form, and adding an object to a menu.

## Making an Object Available from Template Forms

Making a business object available from List and Detail Templates involves

- 1 Setting up the toggle view (Sales\_Toggle\_Shell)
- 2 Setting up the tabgroup view (Sales\_Tabgroup\_Shell)
- 3 Setting up the summary detail view (Sales\_Summary\_Detail\_Shell)
- 4 Setting up the detail view (Sales\_Detail\_Shell)
- 5 Setting up the list view (Sales\_List\_Shell)

### Prerequisites

This procedure assumes that you are

- Familiar with form building using Infor Studio
- Loaded the module with the template definition. Loaded the out-of-the-box Sales or Service application.
- Built the BIO Layer components (all components for the object must exist; this includes recordsets, BIO, relationship with other BIO's, and so on).
- Forms for the object, such as list and detail forms are available.

## Setting up the Toggle View (Sales\_Toggle\_Shell)

Follow this step only for tabs that toggle. To set up the toggle view, you require a list form and a detail form. The Sales\_Toggle\_Shell template has only one slot of type toggle. This slot has two form-slot-definitions, one for the first form (typically the list form), and the other to hold the blank form. The blank navigation is pre-defined and you only need to define the navigation for the form that is rendered first, that is, the form with a lower list order.

To set up the contents of the first slot, use the following steps.

- 1** UI Handler: For your new object, add a UI handler to the Sales\_Toggle\_Shell form. Open the form and switch to the UI Handler tab and add a row with the following values.  
Bio Class: Bio class name of the new object that you are introducing (for example, Opportunity or Lead or Quote).  
Perspective: Sales\_Toggle\_Shell
- 2** Form Placement: Since the form is used by several business objects a form placement is required for each one of them.  
Name: Follow the naming guidelines.  
Bio Class: New bio class name.
- 3** Toggle\_slot: Define a form slot definition for your list form.  
Name: Use naming guidelines.  
Bio Path: Null.  
Form Placement: Placement that you created in step 2.  
Tab identifier: Select from a list, First\_tab.  
Content: Bio Class - List form's perspective.  
List Order: Null  
Perspective: Populated by Infor Studio.  
Special Form Type: populated by Infor Studio.
- 4** Navigation: You do this from a widget on the list form that navigates to a detail form. Create a navigation definition from the click event of that widget.  
Container: Sales\_Toggle\_Shell.toggle\_slot.  
Tab Identifier: select from a list, second\_tab.  
Bio Source: Select from a list, From Extension.  
Content: Bio Class plus perspective of the detail form.  
Perspective: Provided by Infor Studio.  
Special Form Type: Provided by Infor Studio.  
Tab List order: Accept the default provided by Infor Studio, or else set to zero.

## Setting up the Tabgroup View (Sales\_Tabgroup\_Shell)

The Sales\_Tabgroup\_Shell form is used by both the list and the detail view. The contents of the tabgroup can be different in each view, so the description accounts for these differences.

Use the following steps to define the tabs:

- 1 **UI Handler:** For your new object (for example, Opportunity, Lead, or Quote), add a UI handler to the Sales\_Tabgroup\_Shell. Open the Sales\_Tabgroup\_Shell form, click the UI Handler tab, and add a row with the following values.

Bio Class: Bio class name of the new object that you are introducing.

Perspective: Sales\_Tabgroup\_Shell

- 2 **Form Placement:** Create three placements for each bio class. The first placement is for tabs visible in the list and detail views. This placement is based on the bio class name. The second placement is for forms visible in list view. This placement is based on the bio class name and the name of the tab group slot in the list shell. The third placement is for tabs that are visible only in the detail view. This placement is based on the bio class name and the name of the slot that holds the tab group in the sales\_detail form.

Name: Follow the naming guidelines. (Placement for both tabs in List and Detail forms).

Bio Class: New bio class name.

Name: Follow the naming guidelines. (Placement for tab in Detail form only).

Form Slot Form: Sales Detail Shell

Form Slot: Tabgroup slot

Bio Class: New bio class name

Name: Follow the naming guidelines. (Placement for tab in List form only)

Form Slot Form: Sales List Shell

Form Slot: Tabgroup slot

Bio Class: New bio class name.

- 3 **Tab Group Slot:** Define one form slot definition for each item in the tab group. Number (list order) the tabs from 1 through n without any gaps using tab-identifiers. Use the proper form placement based on where the tab is visible (list, detail or both forms). If the tab does not support toggle, then the form-slot-def should be as follows:

Name: Form slot definition name. Use the naming guidelines.

Bio Path: Focus Bio Class Name.Related Attribute Name.

Form Placement: Select one of the three placements that you created above depending on the availability of this tab.

Tab identifier: Select from a list of available identifiers (generic\_tab\_identifier\_nnn). Each name identifier is associated with a list order.

Content: Bio Class - List/Detail form's perspective.

List Order: Null

Perspective: Populated by Infor Studio, based on the value of the content attribute.

Special Form Type: Populated by Infor Studio, based on the value of the content attribute.

If a tab supports toggling, then the form slot definition appears as follows:



Name: form slot definition name. Use naming guidelines.

Bio Path: Focus Bio Class Name.Related Attribute Name

Form Placement: Select one of the three placements that you created above based on the availability of this tab.

Tab identifier: Select from a list of available identifiers (generic\_tab\_identifier\_nnn). Each name identifier is associated with a list order.

Content: Bio Class - Sales\_Toggle\_Shell perspective.

List Order: Null

Perspective: Populated by Infor Studio, based on the value of the content attribute.

Special Form Type: Populated by Infor Studio, based on the value of the content attribute.

## Setting up the Summary Detail View (Sales\_Summary\_Detail\_Shell)

You have to setup the Sales\_Summary\_Detail\_Shell template if you want to show customer summary for the business object. This template has two slots, one for customer/interaction summary, and the other for the detail form. You do not have to do anything for the first slot because the contents of the slot are managed by a pre-render extension.

To setup the metadata for this form, do the following:

- 1 UI Handler: For your business object add a UI handler to the Sales\_Summary\_Detail\_Shell form.

Open the form and switch to the UI Handler tab and add a row with the following values:

Bio Class: Bio class name of the new object that you are introducing (for example, Opportunity or Lead or Quote).

Perspective: Sales\_Summary\_Detail\_Shell

- 2 Form Placement: Since the form is used by several business objects, a form placement is required for each one of the objects.

Name: Follow the naming guidelines.

Bio Class: New bio class name.

- 3 Detail\_slot: This slot hosts the detail form, so create a form slot definition for the detail form.

Name: form slot definition name. Use the naming guidelines.

Bio Path: Null.

Form Placement: Placement that you created in the previous step.

Tab identifier: Null.

Content: Bio Class - Detail form's perspective.

List Order: Accept Infor Studio's default value.

Perspective: Populated by Infor Studio.

Special Form Type: populated by Infor Studio.

## Setting up the Detail View (Sales\_Detail\_Shell)

The Sales\_Detail\_Shell form has four slots, the Tabgroup\_Slot, Detail\_Slot, Toolbar\_Slot and the header\_slot. Use the following steps to set up the detail view.

- 1** UI Handler: For your new object add a UI Handler to the Sales\_Detail\_Shell form. Open the form and switch to the UI Handler tab and add a row with the following values.  
Bio Class: Bio class name of the new object that you are introducing, for example, Opportunity or Lead or Quote.  
Perspective: Sales\_Detail\_Shell
- 2** Form Placement: Since the slots in this form hold form-slot-definition for several bio objects, you have to create a form placement for each one of the objects. Add a new placement record with the following values:  
Form Placement Name: Use the form placement naming guidelines.  
Bio Class: Select new Bio type class name.
- 3** Detail\_Slot: The detail form is shown in this slot. Add a form slot definition with the following values:  
Name: Use form slot definition naming guidelines.  
Perspective: detail view  
Form Placement: Select the form placement created above.
- 4** Toolbar Slot: This slot holds the detail form's toolbar. Create a new toolbar for your detail form and add standard widgets (add, delete, refresh, and so on) Add a new row with the following values:  
Name: Use form slot definition naming convention.  
Perspective: Perspective name of the new toolbar.  
Form Placement: Select the form placement created above.
- 5** Tabgroup Slot: This slot holds the tabs for the detail form. The slot type is normal. By default the slot contains a blank form but in this case, it points to another template that contains the tabgroup definition. Add a new row for the new bio type and set the following values.  
Name: Use the form slot definition naming conventions.  
Perspective: Select "Sales\_Tabgroup\_Shell"  
Bio Reference: New Bio's class name.  
Bio Reference Type: Normal  
Form Placement: Select the form placement created in step 2.
- 6** Header\_slot. You have to add headers to the forms in the detail shell. This slot displays a special toolbar with the name of the object that you edit (for example, opportunity name or customer name or a brief description of the request), and a static message in new mode.  
To display the Toolbar with the name of your object:
  - a** Create a new toolbar with one caption widget, the sales\_detail\_shell\_header\_toolbar perspective, and your bio class name. The bio class name should be same as that of your detail form.
  - b** The default toolbar created by the Infor Studio has one widget of type 'Caption.' Change this widget's "Widget Class: property value" to "epnyDetailShellTitle".  
The caption widget normally displays a bio attribute's value in the edit mode and a static string in new mode. The out-of-the-box application follows this standard for the static text. New <Business

Object Class Name>, for instance New Request or New Opportunity. You have to define this static text.

- c Attach the text string to the extension that resolves the caption value, that is `ResolveTitle`. In Infor Studio, navigate to the Extensions list (**Customization & Business processes > Extensions > ResolveTitle** ). To the `ResolveTitle` extension, add a text string under the Text tab. Provide a name, display text, and proper locale for the text message.
- d Go back to the definition of the toolbar that you created and attach a `prerenderWidget` extension (`ResolveTitle`) with the caption widget in your toolbar. This extension has two required parameters. You need to provide a value for both the parameters. These values are used to resolve the title value.

Edit mode, header in edit mode.

New mode, header in new mode.

The parameter values are resolved as follows:

If a text string is surrounded by `$$`, then the extension tries to resolve a global string using the `getTextMessage()` method.

If a text string is surrounded by `$`, then the extension tries to resolve a text string attached to the extension using the `getTextMessage()` method.

If a text string is surrounded by `#`, then the extension treats this string as a property name and finds its value from the focus bio. This should only be used in the `edit_mode`.

All other strings remain as is.

## Setting up the List View (Sales\_List\_Shell)

The `Sales_List_Shell` form has three slots: `Tabgroup_Slot`, `List_Slot` and `Toolbar_Slot`.

- 1 **UI Handler:** For the new object, add a UI handler to the `Sales_List_Shell`. Open the `Sales_List_Shell` form, switch to the UI Handler tab, and add a row with the following values. .  
Bio Class: Bio class name of the new object that you are introducing, Opportunity or Lead or Quote.  
Perspective: `Sales_List_Shell`
- 2 **Form Placement:** Create a form placement for the new bio type. This form placement is used only with this form. Add a new row with the following values:  
Form Placement Name: Use the form placement naming guidelines.  
Bio Class: Select Type of the new Bio from the list.
- 3 **List Slot:** This slot holds the list form, add a new form slot definition with the following values:  
Name: Use the form slot definition naming convention.  
Perspective: list view.  
Bio Reference: New Bio's class name.  
Bio Reference Type: Normal.  
Form Placement: Select the form placement created above.

- 4** **Toolbar Slot:** This slot holds the toolbar of the detail form. Create a new toolbar for your detail form and add standard widgets (add, delete, refresh, and so on). Add a new row with the following values:  
Name: Use the form slot definition naming guidelines.  
Perspective: Perspective name of the new toolbar.  
Form Placement: Select placement created above.
- 5** **Tabgroup Slot:** This form displays a blank form initially; the tab group appears when the user clicks on the preview button in the list.

## Adding an Object to a Menu

After defining all the navigation for the item, to make the item available from a menu, do the following:

- 1** Add a menu item to the appropriate group.
- 2** From this menu item you can navigate to the object's list form using the Sales list form. To enable this navigation, use the following steps.
  - a** Switch to the Action tab for the menu item and find the "menuClickEvent".
  - b** Add new row to the extension list with the following values:  
Extension = QueryAction  
Priority = 100  
For the QueryAction extension set the following property values:  
Perspective=Sales\_List\_Shell  
Type=<Your Bio Class Name>
  - c** Add a navigation using the following values.  
Navigation Name =<Enter any name>  
Navigation Type = Screen  
State Handling = confirm  
Form Placement = <Leave it empty>  
Title = <Leave it empty>
  - d** Add a navigation definition / target using the following values.  
Container = Mainslot  
Bio Source = From Extension.  
Leave all other fields empty.
- 3** When you navigate to the list form (using sales\_list\_shell), the tabgroup slot will be empty. A form appears only after the user clicks on the preview button (a magnifying glass) from the list.  
To implement this, define a navigation as follows:  
Event Type = listSelectEvent  
Navigation Type = FormSlot  
Navigation Name = <Any Name>

Create a navigation definition or use the navigation that you created. Create using the following guidelines.

Container = Sales\_List\_Shell.Tabgroup\_Slot

BIO Source = From Extension

Content = <Your Objects Bio Class Name> - Sales\_Tabgroup\_Shell

## Naming Guidelines

This section describes guidelines for naming objects in the metadata.

### Perspective Name

Toolbar's Perspective Name: Toolbars are closely associated with a form. Make up a name from the parent form's Bio class name and perspective name.

<Bio Class Name>\_<Perspective>\_ "Toolbar"

#### Examples

Request\_detail\_toolbar

Request\_list\_toolbar

opportunity\_qbe\_toolbar

Form Perspective Name: Most forms use built in perspectives (list, qbe, detail, and so on). These perspectives also broadly identify the form's user interface. For example, the list form is used to edit/view multiple rows of data, and qbe is associated for forms used for searching. The perspective name includes the Bio Class Name, keyword summarizing the forms behavior, and any of the built in perspectives. In this case, the form name should be same as the perspective name.

<Bio Class Name>\_<New Behavior>\_<Layout Perspective>

#### Examples

content\_fileupload\_detail

content\_html\_detail

### Exceptions

If a perspective is associated with multiple bio classes, then the perspective name should not have any bio class name.

### Form Name

When naming a form use the bio class name and the perspective name.

<Bio Class Name>\_<Perspective Name>

#### Examples

opportunity\_detail view

opportunity\_qbe view

opportunity\_list view

## Exceptions

If you are not using one of the built in perspectives (for example, qbe,detail, list) then make the form name same as the perspective name.

## Toolbar Form Name

Add the word “toolbar” to the name of the form the toolbar is associated with.

### Examples

Opportunity Detail Toolbar  
Opportunity Qbe Toolbar  
Opportunity List Toolbar

## Form Slot Definition Name

When naming a form slot definition, use the bio class name of the parent bio, bio class name of the child bio (if you are displaying information related to the parent bio), slot where the form is being displayed and the perspective or form (if special form) name.

<Parent Bio Class Name> \_ Slot Name \_ <Child Bio Class Name> \_ <Perspective/ special form name>

### Examples

Opportunity\_TabGroup\_Slot\_Opportunity Detail  
Opportunity\_TabGroup\_Slot\_Related\_Contacts\_List  
Request\_TabGroup\_Slot\_Orders\_List

## Exceptions

If a form slot definition is the default navigation, then you can name the form slot definition as default\_tabgroup\_slot\_blank.

## Form Placement Name

When naming a form placement, use the form name and all the conditions on which the placement is based. A form placement can be based on the bio class name or the slot name. Create a new name as follows:

<Form Name>\_ <Concatenated List of Conditional Statements>

### Examples

Sales\_Detail\_Shell\_Request\_Bio  
Sales\_Tabgroup Shell\_Request\_Bio\_Sales\_Detail\_Shell (Tabgroup Slot)  
Sales\_Tabgroup\_Shell\_Request\_Bio\_Sales\_List\_Shell (Tabgroup Slot)

## Using Static Form Placements

You can also create named form placements that are not based on BIO types or slots and specify the form placement to be used on the content of a navigation definition. You can also specify the form placement name on a target form when defining a form slot definition. This means that you can use multiple forms of the same BIO type within a single template form.

For example, you can have three sets of list forms, all based on the Organization BIO. Each of the three forms has a different perspective, that is, `list_view`, `partner_list_view` and `competitor_list_view`. You can display each of these forms within the `Sales_List_Shell`.

You can have the following three form placements on the `Sales_List_Shell` form:

Form Placement Name: `Sales_List_Shell_Organization_Bio`

BIO: Organization

Form Placement Name: `Sales_List_Shell_Partner`

BIO: <none>

Form Placement Name: `Sales_List_Shell_Competitor`

BIO: <none>

The `List_Slot` of the `Sales_List_Shell` form has the following formslot definitions:

Name: `Organization_List_Slot_List_View`

BIO Path: Organization

Form Placement: `Sales_List_Shell_Organization_Bio`

Content: Organization - List View

Name: `Competitor_List_Slot_List_View`

BIO Path: Organization

Form Placement: `Sales_List_Shell_Competitor`

Content: Organization - competitor\_list\_view

Name: `Partner_List_Slot_List_View`

BIO Path: Organization

Form Placement: `Sales_List_Shell_Partner`

Content: Organization - partner\_list\_view

The menu item navigation to the `organization_competitor_list_view` form has a navigation type of `INTERACTION` whose navigation definition has the content of 'organization - Sales\_List\_Shell'. The form properties are displayed for that content on the navigation definition, and form placement is one of those properties. Set the form placement on that navigation definition's content to 'Sales\_List\_Shell\_Competitor'.

Similarly, for the menu item navigation to the `organization_partner_list_form`, set the form placement on the navigation definition's content to 'Sales\_List\_Shell\_Partner'.

If static form placements are not used on formslot contents, then you run into issues. For example, when you display the `organization_competitor_detail_view` form within the `Sales_Detail_Shell`, the tabgroups for the `organization_detail_view` are displayed because both `organization_competitor_detail_view` and `organization_detail_view` are for the same bio class, `organization`. With the ability to use static form placements on formslot contents, for the `Tabgroup_Slot` of the `Sales_Detail_Shell`, you can specify the following form slot definitions:

Name: `Organization_Tabgroup_Slot_Sales_Detail_Shell`

Bio Path: `Organization`

Form Placement: `Sales_Detail_Shell_Organization_Bio`

Content: `organization - Sales_TabGroup_Shell`

Name: `Competitor_Tabgroup_Slot_Sales_Detail_Shell`

Bio Path: `Organization`

Form Placement: `Sales_Detail_Shell_Competitor` (note that this is a static form placement)

Content: `organization - Sales_TabGroup_Shell`

form placement property on that Content = `Sales_Tabgroup_Shell_Competitor_Sales_Detail_Shell` (note that this is a static form placement defined on the `Sales_Tabgroup_Shell` form and used in the tabgroup slot definitions for the competitor tabs on that form).

Name: `Partner_Tabgroup_Slot_Sales_Detail_Shell`

Bio Path: `Organization`

Form Placement: `Sales_Detail_Shell_Partner` (note that this is a static form placement)

Content: `organization - Sales_TabGroup_Shell`

form placement property on that Content = `Sales_Tabgroup_Shell_Partner_Sales_Detail_Shell` (note that this is a static form placement defined on the `Sales_Tabgroup_Shell` form and used in the tabgroup slot definitions for the partner tabs on that form).



## Overview

*Bulk or external data import enables* you to import data from external business files (for example, CSV files) into Infor data sources where the data is used as a BIO. Specifically, import involves reading data from an external data source, transforming the data into an internal BIO representation, and saving the internal BIO into the Infor system. During import, data can pass through business rules. You can also automate the import process through Scheduler jobs.

## Terminology

**BIO-to-BIO transformation map:** Mapping between an external and internal BIO. The BIO-to-BIO Transformation service transfers an external BIO to an internal BIO based on metadata that defines the mapping.

**External BIO:** A temporary BIO used to map and hold external data (a BIO that matches a file header definition).

**Internal BIO:** Typically, an out-of-the-box BIO to which external data is eventually mapped.

**Import Map:** Mapping between a file and an external BIO.

**New datasource called 'External Import File.'** A datasource where external recordsets based on file headers reside. The datasource is specific for CSV import and you cannot use this datasource for other purpose.

**CSV:** Text file format where each field is separated by a delimiter, usually a comma.

**Admin Console:** An Infor application available to users in the administration group. You assign permission to use the Admin Console in Infor Studio. The Admin Console functionality serves both Sales and Service applications.

## Import Process

The import process consists of creating an import map, creating a BIO-to-BIO transformation map, creating an import job, and running the job.

You create the import map and the BIO-to-BIO transformation map using Infor Studio. Once you create an import map and a transformation map, start the Infor server, log in to the Admin Console, and create an import job. After you create an import job, you can run the job from the Admin Console or use the Scheduler to run scheduled jobs.

The rest of this section deals with the following phases of the import process.

- Creating an import map using Infor Studio
- Creating a BIO-to-BIO Transformation map using Infor Studio
- Creating and running an import job using the Admin Console. Using the Admin Console, you can run the import job as a one time job or as a scheduled job.

## Creating an Import Map

An import map provides the mapping between an external file and an external BIO. You can define an import map in Infor Studio either using the Import Service Wizard or the Bulk Import form. When you use the Import Service wizard to create an import map, a transformation map is automatically created. Note that this transformation map is a placeholder. You still have to do the mapping of the fields in the transformation map.

### Creating an Import Map using the Import Service Wizard (CSV File Data Source)

- 1 From the Infor Studio home page, under **Logical Schema**, click **Create Import map**. The Import Service wizard welcome screen appears.
- 2 Click **Next**. The Module Selection screen appears.
- 3 Choose an existing module, or define a new module by typing it's name in the **New Module** text box.
- 4 Click **Next**. The Type of Data Source screen appears.
- 5 Select **CSV** as the type of data source that will provide the data for import.
- 6 Enter the name and path of the CSV file to be used for import or use the **Browse** button to navigate to a CSV file. Enter delimiter characters (used to separate one field or column from another) for columns and fields.

**Note:** The CSV file must have a header with or without data. The file should also be complete in terms of the columns/column names. The data file that you specify for import in the Admin Console should also match the CSV file in terms of the order of columns.

- 7 Click **Next**. The Import Service Map screen appears.
- 8 Enter a name for the import service map that you want to set up.
- 9 Enter a name for the external BIO that will be created for the external data you import. This BIO is a placeholder that will appear as a recordset in the External Import File data source.
- 10 Enter a name for the transformation BIO that provides the mapping between the external and internal BIO. Click **Next**. The mapping is created after the wizard completes the process.

**Note:** The transformation map that is created in Step 10 is only a place holder. You have to map the fields based on the details of the transformation.

## Creating an Import Map using the Bulk Import Form

You can also use the **Bulk Import** form (**Guide Bar > Physical & Logical Schema > Bulk Import**) to define the mapping for importing data from an external file into an internal BIO. The external file must be a valid CSV file, but it can have any file extension.

Use the **Bulk Import** form to specify the following properties and save your work.

Parameter	Description
Module	The module to which the import map belongs.
Map Name	A name for the import map.
Type	Type of import (CSV or database)
Column Delimiter	Delimiter used to separate one field from another in the CSV file from which you are importing data. Choices: ! # % & , ; ? ^ ` ~
Field Qualifier	A string that can be optionally used to separate out data text in the file from which you are importing data. Choices: NONE ' ! " # ? @.
BIO Transform Map	The Bio transformation map that provides mapping between the external Bio and internal Bio.

In the **Import Map Detail** form, specify details for the following:

Parameter	Description
Module	The module to which the import map belongs.
BIO Class	The Bio whose attributes are being mapped.
BIO Attribute	The Bio attribute the incoming value maps to.
Default	Default value for blank values (like ,, in CSV). If the value of the default field is NULL then Null is set for the external Bio attribute if the .csv file value is blank. If the field is required, then this will be an

Parameter	Description
	error condition. If the value of the default field is a string, then the string value is set for the external Bio attribute if the .csv file value is blank. If the value of the default field is left blank, then nothing is set on the external Bio attribute by default and the Bio or database default is used for the incoming field if the .csv file value is blank. If the field is required, then this could be an error condition.
Tag Name	The name of the CSV file header field.
Tag Order	The order ID is the order of the incoming field in the header of the CSV file. When you manually modify a tag (delete or add a tag), make sure that the tag order numbering matches the ordering of the CSV file header.

## Deleting an Import Map

- 1 In the **Bulk Import** form, select the import map, right-click, and select **Delete** . The details associated with the import map are deleted automatically.
- 2 Verify that the transformation map is deleted as well.
- 3 Manually delete the external record set used by the import map (under datasource External Import file)
- 4 Delete the external BIO (**Physical and Logical Schema > BIO Related Info** ).

## BIO Transformation

*BIO transformation* allows you to transform a BIO of a particular type to a BIO of another type or the same type and traverse path(s) on a BIO or BIO Collection with filtering of collections along the path. To perform BIO transformation, you create a transformation map. This map specifies information on the source and target BIOs and the direction of the transformation. Transformation maps determine whether the transformation is BIO-to-BIO or transversal on a BIO or BIO Collection.

Each transformation map consists of one or more transformation attribute maps. You can create simple direct attribute maps or attribute maps with BIOs in their paths. BIOs in the path can be marked as lookup or create/update.

Transformation attributes have one or more path elements. Path elements are associated with a BIO attribute. If the element is a collection, it can be associated with a filter to retrieve a single element or set up with an alternate key to identify an element. If the element is a BIO, it can be created or updated using a lookup. It may contain 0 or more element filters to narrow a collection to a single element.

**Note:** Every import map has one transformation map associated with it.

## Creating a Transformation Map

When you use the Import Service Wizard to create an import map, a placeholder transformation map is created as part of the mapping process. You can also create a transformation map using the **BIO Transform Maps** form.

- 1 In Infor Studio, navigate to *BIO Transform Maps* (**Guide Bar > Physical & Logical Schema > BIO Related Info**).
- 2 In the **BIO Transform Maps** form, add an entry in the first empty row. Provide a name for the transformation map, specify the source and target BIO, and the direction of the transformation. Make sure that you provide a value for the parameters in the following table and save your entry. The table lists only those fields that are relevant to the import process. For more information on creating transformation maps, see "Chapter 11" on page 155.

Parameter	Description
Module	The module to which the Bio transformation belongs.
BIO Transform Map Name	The unique name of the Bio transformation.
Source BIO	The source Bio for the Bio transformation.
Source BIO Key	A primary or alternate key of the recordset of the source Bio. This is required only for maps that do not have the Create Only masked attribute.
Target BIO	The target Bio for the transformation.
Target BIO Key	A primary or alternate key of the recordset of the target Bio. This is required only for maps that do not have the Create Only masked attribute.
Transform Direction	The direction of the transformation. Select Source to Target .
Transformation Type	The type of transformation. Select Transformation (transformation of a Bio of any type to a Bio of another or the same type).
Masked Attributes	Attributes that can be used to enable properties on the transformation. Check the Create only checkbox. When you use this option, the target is always created and the key translation mechanism is not used.

**Note:** Once you use the Create only masked attribute, the transformation process assumes that there are no Target and Source BIO keys. If you do not use the Create only masked attribute, then, the Target and Source BIO keys are required. If you check the Insert Only, No Update field in the Admin Console, then the Source and Target BIO keys are required. When you check the Insert Only, No Update field in the Admin Console, the import process checks to see that a record does not exist in the database and throws an error if it does. To do this, the import service needs the Source and Target BIO keys in the transformation map.

- 3 In the **BIO Transformation Attribute Maps** subform, create attribute maps to map the source attributes to the target attributes. You can create a direct attribute map or a map with BIOs in its path. Define an attribute map by entering values for the following

Parameter	Description
Module	The module of the transformation map.
Source Attribute Mapping	The source Bio attribute for the attribute map. Click in the field and use the popup window to select a source attribute. Save and close the popup.
Target Attribute Mapping	The target Bio attribute for the attribute map. Click in the field and use the popup window where you select a target attribute. Save and close the popup.
Map Direction	The direction of the attribute map. Select Source to Target .

- 4 In the lower pane of the **BIO Transformation Attribute Maps** subform, create path elements for source (left panel) and target (right panel) for the selected attribute map. Save your work.

Parameter	Description
BIO Attribute	The source or target Bio attribute.
EpiExpression	An EpiExpression used to perform computations using IBR on the source to set on target or vice versa. Use EpiExpression to set constants, fields, defaults, change incoming values, or provide a lookup value from source.
Lookup Attribute Domain	If a Bio attribute has an associated attribute domain, Infor Studio automatically populates this with the name of the attribute domain.
Element Key	An alternate key on the Bio used to identify an element of the BioCollection. Used only when a single element of the collection is inserted or updated.
Masked Attribute	Select Lookup BIO with Source Attributes . When you select this option, an attribute in the source can be mapped to the target in such a way that a Bio in the target path can be looked up using the single source attribute. Use this when looking up a Bio.
Module	The module to which the path element belongs.

- 5 Re-start the server to see the transformation map modifications.



**Caution:** Once you create an import and a transformation map, start the Infor server. Verify that the server is up and running, and then create an import job using the Admin Console.

**Note:** If the import source is a database, the source object that is passed to the transformation extensions is a BIO. If the import source is a CSV file, the source object created by the Import Service is not an instance of a BIO, but rather an instance of a HelperBIO. Thus, your extension code has to treat the source object as a BIO or a HelperBio depending on the import type. If you do not cast the source object to the correct type, you will receive a runtime null pointer exception.

## Creating and Running an Import Job

You create an import job using the Admin Console

([http://<server.organization.com>\[:port#\]/<instance>/app/admin.ctrl](http://<server.organization.com>[:port#]/<instance>/app/admin.ctrl)). The import job or session defines the files that are to be used for import and when it should be imported. For instructions on using the Admin Console to create and edit an import job, refer to the Manager's Guide.

Once you create an import job, you can run the job as a one-time job or as a scheduled job using the Scheduler. For instructions on using the Admin Console to run a one-time job and a scheduled job, refer to the Manager's Guide.

When you create jobs, use the following flags to assist you in troubleshooting:

Name	Description
Write Failed Records to Reject File	Check this flag so that failed record details are logged into the Reject/Error File Directory. Make sure that you pick a location for the Reject/Error File Directory Otherwise you will encounter an error.
Write Error Details to Error file	Check this flag so that error details are logged into the Reject/Error File Directory .
Write Remaining Records on Import Failure	Check this flag so that in case of job failure, records that are not imported are written to reject file. This can be time consuming for failed large import jobs.

If your import exits with an error, all the data that is not imported is in the reject file. If the input file is long and the *Error Threshold* is small, the reject file can be large.

## Trouble Shooting

- To trouble shoot import jobs, always run the trial run to identify metadata errors on new maps.
- In case of metadata errors, the error messages appear in the Console or EpnY.log4j file in the server location on which the job runs. To proceed, fix the explained issue by fixing the metadata in Infor Studio. Then, re-load the meta in a non-production mode or re-start the server before re-trying the job.
- If you see errors in data, read <ImportFileName>\_timestamp.err file from the Reject/Error File Directory for error details. Fix the data that has error in the import file and import by re-executing the job. You can increase the Import File Start Index to start from where the Import Job failed. Alternatively, you can fix the error in <ImportFileName>\_timestamp.rej file (from reject file directory), and edit the import job to submit the file for import.
- Import Service does not give errors if the filename does not match the name defined in the map. For example, if you create an Import job and in the Import File enter the path with a typo or incorrect folder name, and run the job, the History window reports Success and there are no errors in the Rejected directory. This occurs because import does not know there is a typo and only looks for a file by that name. If the file is not found it is not treated as an error condition. This is because there are imports where there could be no files. For example, you can set up a Scheduler job to import from a directory everyday. You don't know the number of files that can be in the directory. You give the directory name or say import\*.csv from the directory. Some days there can be many files matching the pattern. Some days there can be none and this is not treated as an error condition. Therefore, the job is considered a success.

## Performance Issues

- Change *Commit Threshold* to get an optimum number for your import job. This value affects import job performance.
- Change *Error Threshold* for tolerance of error. The import job continues as long as the number of errors encountered is within this threshold.
- If you do not need an extension to run in import mode, you can change the extension code to not run in import mode. The following code is true if you are in Import Mode:  

```
_ctx.getUserContext().isServiceModeSet(ServiceModes.BULK_IMPORT);
```
- You can modify the extension code to run only when the above code is false.
- Every transaction save or 10,000 records whichever comes first is logged into the EpnY Log4j under BkImport category with the current record count.
- Stop index is part of import job definition. This is an optional value. If it is 0 or less, then the value is ignored and the complete file is processed. If the value is greater than 0, then the import will stop after processing the stop index record count.
- Import performance is related to the performance of the BIO extensions that run as part of import Job. To improve performance, you can disable time consuming extensions. Make sure that you disable non-essential extensions only and not extensions required for the import process. You disable extensions using Infor Studio's Extensions form. For more information, refer to Infor Studio online help.



## Example

This example provides high level instructions on importing new individual records from CSV files. Use the sample data provided in the IndividualImport.csv file located at shared\services\example\Bulk Import\Individual. You can also find other examples at shared\services\example\Bulk Import.

## Import Map Details

In Infor Studio, import the Example\_individual\_ImportModule.ebm file. Once you import, the following information appears in Infor Studio:

Module Name: Example\_individual\_ImportModule

Import Map Name: example\_individual\_ImportMap

BIO Name for External Data: example\_individual\_ImportBIO

BIO Name For transformation: individual\_customer

Transformation Map has no Masked Attributes set.

**Note:** The incoming ref\_code is not mapped to any field in the Customer BIO because there is no out of the box reference\_code in the Customer BIO.

**BIO Transform Map**

Module: **Example\_individual\_ImportModule** | BIO Transform Map Name: **example\_individual\_ImportMap**

Source BIO: **example\_individual\_ImportBIO** | Target BIO: **individual\_customer**

Source BIO Item: **PE\_example\_individual\_ImportBIO\_SBOs** | Target SBO Item: **AK\_FOREIGNTRANSFORMATION**

Transformation Type: **Transformation** | Return BIO Type:

Transformation Direction: **Source To Target** | Mapped Attributes:

Link Attributes:

Linking BIO Attributes:

**BIO Transformation Attribute Maps**

Attribute Maps / Path Elements (10 Records)

Module	Source Attribute Mapping	Target Attribute Mapping	Map Direction
Example_individual_ImportModule	example_individual_ImportBIO.birth_date (1)	individual_customer.birth_date (1)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.first_name (2)	individual_customer.first_name (2)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.last_name (3)	individual_customer.last_name (3)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.local_id (4)	individual_customer.local_id (4)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.ref_code (5)	individual_customer.reference_code (5)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.primary_address (6)	individual_customer.addresses.address_1 (6)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.city (7)	individual_customer.addresses.city (7)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.state (8)	individual_customer.addresses.state_code (8)	Source To Target
Example_individual_ImportModule	example_individual_ImportBIO.state (8)	individual_customer.addresses.state_zip (9)	Source To Target

**BIO Attribute** | **BIO Transform Map** | **External Class** | **External Class**

BIO Attribute	BIO Transform Map	External Class	External Class
example_individual_Import...			

## Setting up Filter on Address Mapping

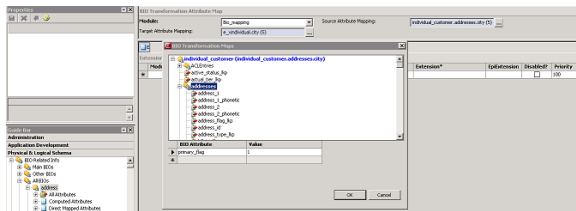
In this example, address information is imported along with individual information. In the Infor system, an individual can have multiple addresses; therefore, address is a BIO Collection. Mapping to a BIO

## Bulk Import

---

Collection requires a filter that identifies the record being imported from the others in the collection. Use `primary_flag` as the unique identifier as you can have only one primary address for an individual. The following step is required on every address attribute to import the address for an individual.

To set up filter on address mapping, for every address attribute being mapped, set the filter. Select the attribute on the destination and click OK to select the attribute. Then, open the Target Attribute Mapping for the specific attribute and click next to address as shown below to get the property window. Select the filter as shown from the available list.



## Import Job Details

- 1 Start the server and log in to the Admin Console.
- 2 Create a new Import Job (**Job Scheduler > Job List > New**)
- 3 Enter a job name and select **ImportJob** for the **Type of Job** input.
- 4 For **Import Map**, use the drop-down to select **example\_individual\_ImportMap**.
- 5 Enter the Import file name as `<Path>IndividualImport.csv`. Enter a valid directory for the **Reject File Directory**.
- 6 Check the **Input File Has Headers** flag.
- 7 Save the job.
- 8 Check the created job from the list and click **Run**.
- 9 Refresh the screen to see the job status. You can also look for job related messages on the server. In case of error, read the `<Importfile_Name_Timestamp>.err` file from the reject directory for details.

The screenshot shows two sections of a web interface. The top section, titled 'Create/Edit Job', contains two dropdown menus: 'Job Name' with the value 'Individual Import Job' and 'Type of Job' with the value 'ImportJob'. The bottom section, titled 'Import Configuration', contains several fields and checkboxes. It includes dropdowns for 'Import Map' (value: 'example\_individual\_importMap') and 'File Encoding' (value: '-----'). It also has text boxes for 'Import File' (value: 'C:\import\IndividualImport.csv') and 'Reject/Error File Directory' (value: 'C:\Reject'). Below these are four numeric input fields: 'Import File Start Index' (0), 'Import File Stop Index' (0), 'Commit Threshold' (1), and 'Error Threshold' (0). At the bottom, there are four checkboxes: 'Input File Has Headers' (checked), 'Rename File After Import' (unchecked), 'Insert Only, No Updates' (unchecked), and 'Trial Run' (unchecked). There are also two checked checkboxes: 'Write Failed Records to Reject File' and 'Write Error details to Error File'. A group of checkboxes labeled 'Write Remaining Records on Import Failure' includes 'Write Remaining Records on Import' (checked) and 'Failure' (unchecked).

You can run the jobs in Insert Only and Insert and Update modes. To set up the csv file for updates, create a reference\_code in customer BIO, identify the individual record you want to update, extract the value for the reference\_code attribute from the database, and replace the value in the csv file provided.

## Database Import

*Database import* enables you to import from any SQL datasource that is created and use the data within the Infor system as a BIO. The import process consists of reading the data from the external data source, transforming the data into an internal BIO representation, and saving the internal BIO into the Infor system. Database import also uses the Transformation service.

Before you can initiate a database import session, you have to define the schema of the external data as BIOs and recordsets. Then you define transformation metadata to transform the attributes of the external BIOs to attributes of internal BIOs. Infor Studio provides a wizard to create the import mapping metadata and the transform metadata.

Once you define the schema and transformation metadata, you can log into the Admin Console and create an import session. You can initiate import sessions immediately or schedule them to periodically import from a particular file location.

Once an import session begins, the mapping metadata is read and the external BIO(s) is chosen for the session. The Import service reads the data for the external BIO from the datasource and calls the Transformation service to map the external BIO to an internal BIO. The transformed BIO is saved in non-trial mode and if applicable, error logging is performed. The entire transaction is committed at specified intervals and also at the end of the session.

If the import is not successful, you have the option to write the reject data (Primary key of the error record) and error details to log files. You also can re-submit a failed import.

**Note:** The input for a database import is a BIO which can be from any SQL external or internal data source. This BIO can get all the relevant details which in turn can come from one or more recordset (each recordset pointing to a table with data in the database). For the BIO to read all the data correctly, you have to correctly define the relationships between the recordsets or create a view on the external system and map the recordset to that view.

## Creating an Import Map using the Import Service Wizard (Database Data Source)

**Note:** Create an external BIO first and then start the Import Service Wizard.

- 1 From the Infor Studio home page, under **Logical Schema**, click **Create Import map**. The Import Service wizard welcome screen appears.
- 2 Click **Next**. The Module Selection screen appears.
- 3 Choose an existing module, or define a new module by typing it's name in the **New Module** text box.
- 4 Click **Next**. The Type of Data Source screen appears.
- 5 Select **Database** as the type of data source that will provide the data for import.
- 6 Click **Next**. The External Import BIO Details screen appears.
- 7 Under **Data Source Types**, select the data source type. Under **BIO**, select your BIO. Click **Next**. The Import Service Map screen appears.
- 8 Enter a name for the import service map that you want to set up.
- 9 Enter a name for the transformation BIO that provides the mapping between the external and internal BIO. Click **Next**. The metadata mapping is created after the wizard completes the process.

**Note:** The transformation map that is created in Step 9 is only a place holder. You have to map the fields based on the details of the transformation. See "Creating a Transformation Map" on page 129.

## Error Handling

Database import error handling is similar to CSV import with the following differences

- For database import, the reject file has only the primary key values and not the entire record. You also cannot use the reject file for re-import. The reject file does not have the records after the failure point copied to it.
- There is no specific sort order for the incoming data collection. To set a sort order, use a view on the external BIO and recordset.
- Re-import works for database import and works when the sort order is consistent. If you cannot guarantee sort order, do not use re-import. Splitting of the job also depends on the sort order.
- Error file and Reject file have Import map name.

- Import cannot provide explicit default value for database import columns at import map level. You have to define default values on the external or internal BIO or through views.

**Note:** If you configure and use a new data source in Infor Studio but do not configure a data source in WebLogic or WAS, you will encounter errors around creating/accessing pooled connections. You have to create a connection pool for the data source in the WebLogic Administration Console before you can use the data source in the application. For instructions on creating a new WebLogic data source, see the Installation and Configuration Guide.

## Debugging Bulk Import

To get the debug log for import category, make sure the section below appears under <!-- Setup Appenders --> (in Log4jConfiguration file - use) and is set to DEBUG

```
<appender name="BKImportFile" class="org.apache.log4j.FileAppender">
  <param name="Encoding" value="UTF-8"/>
  <param name="File" value="${log.dir}/BKImport.log4j"/>
  <param name="Threshold" value="DEBUG"/>
  <param name="Append" value="false"/>
  <layout class="com.epiphany.shr.util.logging.EpnyXMLLayout"/>
</appender>
```

Make sure the section below appears under <!-- Setup Categories --> (in Log4jConfiguration file - use debug version)

```
<category
  name="com.epiphany.shr.dataimport.importservice.ImportServiceBean">
  <priority value="DEBUG"/>
  <appender-ref ref="BKImportFile"/>
</category>
```

You can also get DP logs to see what queries are being run.

Make sure the below section appears under <!-- Setup Categories --> (in Log4jConfiguration file - use debug version) and is set to DEBUG

```
<category name="DeveloperLog.Shared.BioLayerLog.DataProviderLog">
  <priority value="DEBUG"/>
  <appender-ref ref="DataProviderFile"/>
</category>
```

Make sure the below section appears under <!-- Setup Appenders --> (in Log4jConfiguration file - use debug version) and is set to DEBUG

```
<appender name="DataProviderFile" class="org.apache.log4j.FileAppender">
```

```
<param name="Encoding" value="UTF-8"/>
<param name="File" value="${log.dir}/dp.log"/>
<param name="Append" value="false"/>
<layout class="com.epiphany.shr.util.logging.EpnyPatternLayout">
  <param name="ConversionPattern" value="%d [%t] %-5p %c - %m%n"/>
</layout>
</appender>
```

Do not enable any of the above for production mode because it will slow down performance.

In Production mode, use the install version of log4jConfiguration.xml file. If you use the development version of log4jConfiguration.xml file, keep the logging level closer to ERROR for BKImport category and also for most categories such as BIO, DP, SQL to improve import performance. In the Infor log file, if you see a log message printing for every imported record, then it affects performance. You can change the log level setting for that category to a higher level such as from DEBUG to WARN or WARN to ERROR to remove the message from log.

### Best Practices

See "Bulk Import" on page 45.

## Bulk Importing with Special Characters

If the data you are importing contains special characters, for example, the trademark symbol (®), you may observe that the import job completes without any errors, but that no data is imported.

The reason why the job does not fail is because the character encoding discrepancy is translating to an empty file when java reads it. In such cases, the import does not flag this as an error. Periodically, automated imports for some updates in another system can have empty files generated when there is nothing to update.

To properly import the data, save the source data (CSV file or text file) in Unicode or UTF-8 format as described below.

### Saving Your Data in Unicode or UTF-8 Format

If you are using Excel to edit and create data, save it as a CSV file. Open it using editors like textpad or similar text editor. (You will observe the type is ANSI.) Save the file as a unicode file and use the file for the bulk import.

If you create the file by exporting from the database, the database export tool usually lets you pick the encoding type. For example, on Microsoft SQL Server, if you export data as a flat file, you should change the default ANSI encoding type to UTF-8.

This chapter covers *BIO Externalization*. BIO Externalization refers to the scenario where implementers of an Customer Service application re-map one or more out-of-the-box recordsets to tables in their own custom/legacy databases.

## General Strategies for Re-mapping Recordsets and BIOs

This section covers issues related to re-mapping recordsets and BIOs.

### RecordSets for which Externalization is supported

Externalization of the following recordsets is supported:

- Individual
- Customer (certain limitations apply. See "Re-mapping Customer/Individual/Organization RecordSets and BIOs" on page 145)
- Household
- Telephone
- Address
- email\_address
- Product
- Product\_instance

## Determining Dependencies when changing Field/Attribute Type or Attribute Mapping Type

Due to dependencies in the data, when making changes to fields, BIO attributes, and relationships, make sure that other areas in the application are not affected by these changes. To ensure this, Infor Studio provides a tab below Fields, BIO Attributes, and Relationships that lets you view the dependencies.

There are no easy rules-of-thumb about the implications of these dependencies on potential changes or externalization of the data. In general, look through all of the dependencies and sanity-check that the new behavior of the data is consistent with what's expected everywhere it is used (which may mean, for example, checking dependent extension code).

A dependency tab (subform) exists in Infor Studio for the following objects:

- Fields
- BIO Attributes
- Relationships

### Data Field Dependents

The Data Field Dependents tab appears as a subform below any field detail. Double-clicking on any field navigates to its detail. The dependents tab displays the objects in the metadata that have a dependency on the field and that would be affected by externalization. The dependency tab lists objects that have a dependency on the field. These include Direct Mapped BIO attributes, Indirect Mapped BIO attributes, Field sets, Relationships, and Relationship Constraints that use the data field.

### BIO Attribute Dependents

The BIO Attribute Dependents tab appears as a sub-form for any BIO attribute detail. Double-clicking on any BIO Attribute navigates to its detail. The Dependents tab displays the objects in the metadata that have a dependency on the BIO attribute and that would be affected by externalization. The dependency tab lists objects that have a dependency on the BIO attribute. These include Path Mapped attributes, Relation Interface Mapped attributes, Query Mapped attributes, Computed BIORef, Form Widgets, Property Values, and BIO Reference Path.

### Relationship Dependents

The Relationship Dependents tab appears as a sub-form for any relationship detail. Double-clicking on any relationship navigates to its detail. The dependents tab displays the objects in the metadata that have a dependency on the relationship and that would be affected by externalization. The dependency tab lists objects that have a dependency on the relationship. These include Relation mapped Interface Definition, BIO Subclass, Relation mapped BIO, and Indirect field mapped attribute.



## Out-of-the-box Fields/Attributes with no Corresponding Column in Legacy Table

### Re-mapping out-of-the-box non-required Fields/Attributes

When you re-map a recordset to a legacy table, but there is no corresponding column for an out-of-the-box non-required field on that recordset, then, rather than removing that field and the corresponding BIO attribute and all other metadata/extensions dependent on it, use the following approach:

Remove the not required field from the recordset, but do not remove the BIO attribute. Instead, change the BIO attribute mapping from DIRECT to COMPUTED and for the computation class file, use a class file that returns a null object. This means you do not have to be concerned about extensions that reference that attribute. Also, flag the attribute as non-updatable and non-queryable.

The following is example code for the computation class file:

```
package <your custom package name>;

import com.epiphany.shr.data.bio.Bio;
import com.epiphany.shr.data.error.EpiDataException;

public class ComputeNullAttribute

    implements com.epiphany.shr.data.bio.computed.ComputedAttributeSupport
    {
        public Object get(Bio bio, String attributeName, boolean isOldValue)
            throws EpiDataException
        {
            return null;
        }

        public boolean supportsSet(String bioTypeName, String attributeName)
        {
            return false;
        }

        public void set(Bio bio, String attributeName,
                        Object attributeValue, boolean isOldValue)
            throws EpiDataException
        {
            return;
        }
    }
```

## Non-filterable/Sortable Widgets based on Attributes Path Mapped to Externalized BIOs

If a BIO is re-mapped to an external data source, then where other BIOs have attributes path mapped to that BIO, and there are queryable widgets based on those path mapped attributes, in that case the use of those queryable widgets will cause cross database queries which are not supported by the core architecture. Therefore, these widgets should be changed to use a non-queryable widget type, for example, the label widget type or the link widget type.

In some cases where these widgets are queryable, there are two widgets based on the path mapped attribute on the same form, that is, one that is used by the list form itself and one that is used in the filter row of the list form. In this case, rather than changing the type of a widget it is necessary to instead remove the widget used by the filter row.

To reiterate, nothing in the application should generate a query joining records from different data sources. So in addition to making sure not to allow list form filtering on attributes mapped to data sources different from the main BIO being filtered, avoid forming such queries in extension code. Depending on what is being externalized, this may require changing existing extensions and may often require changing the deeper design or functionality.

For example, if customer and address tables exist in different databases, a customer list form can no longer filter on address fields. If extensions implement some batch processing that involves looking for all customers with an address in a particular state, that functionality needs to be completely re-thought and probably dropped. Having customer and address data living in different databases is fundamentally incompatible with the idea of ever getting a list of customers with addresses that match a filter (short of manually “joining” in memory with multiple queries which is almost always not feasible). Note that the idea of displaying all addresses for a given customer still makes sense when they are in different databases. If the reasoning behind this distinction is not clear, think about the SQL necessary to accomplish these tasks. The illegal query would require something like “select \* from customer, address where customer.customer\_id=address.customer\_id and address.state='X'”, where joining customer and address this way is impossible. The related collection requires something like “select \* from address where address.customer\_id = 'X'”, which does not require a cross-database join. Understanding this distinction and the deeper design/functionality implications is vital whenever related data live in different data sources.

## Configuring Locking Strategy on Externalized RecordSets

Out-of-the box most recordsets are configured to use a locking strategy of optimistic revision. The revision\_number field on these recordsets is flagged as the revision field. If you are externalizing a recordset and you do not wish to use the optimistic revision locking strategy, then change the locking strategy on that recordset and do one of the following:

- Delete the revision\_number field for that recordset or
- If you are mapping the revision\_number to some other column but not using it for optimistic revision purposes, then uncheck the **Revision** checkbox.

## Configuring Non-usage of Logical Delete on Externalized RecordSets

Out-of-the box most recordsets are configured for logical delete. The `obsolete_flag` field on these recordsets is flagged as the logical delete field. If you externalize a recordset and do not want to use logical delete then, do the following:

- 1 Un-check the **Supports Logical Delete** and **Logical Delete** check boxes on that recordset.
- 2 Delete the `obsolete_flag` field for that recordset.

## 'Recent Activity' may not work for Externalized BIOs

The 'Recent Activity' information is based on BioRefs, which assume the ability to fetch records based on Primary Key. If the external system does not allow for searches based on the Primary Key of a particular BIO, then the RecentActivity screen will not be able to fetch and display this information.

If fetches based on Primary Keys are supported by the external system, then the **RecentActivity** screen will work.

In order to turn off Recent Activities, do the following:

- If the Agent does not have one of the six slots on the E.piphany Sales home screen set to RecentActivity, then the RecentActivity functionality is "turned off" for that agent.
- To turn off recent activity globally, remove (or disable) the `LogRecentActivityItem.java` extension from all `bioAfter` and `bioBefore` events. If this is done, and the agent sets the home screen to display Recent Activity, the Recent Activity list will always be empty.

## Impact on Polymorphic Relationships where Primary Keys are changed from GUID to String/Integer

In Customer Service, polymorphic tables/recordsets for common objects that are related to many parents are used. For example, the `user_note` table/recordset that is used to record the user entered notes in the Notes tab on many forms.

All such polymorphic tables typically include the following four columns:

- `related_tbl_name` - this holds the table name of the BIO referenced in `related_id`.
- `related_id` - this references the primary key of a BIO where that primary key is a GUID.
- `related_string_id` - this references the primary key of a BIO where that primary key is a string.
- `related_int_id` - this references the primary key of a BIO where that primary key is an integer.

Out-of-the-box, relationships between these polymorphic recordsets and their parent recordsets are based on the `related_id` fieldset and the relationship is further constrained on:

`related_tbl_name` = the name of the parent recordset

The additional `related_string_id` and `related_int_id` columns are provided for the scenario where you want to change the type of the primary key column of some parent table from GUID to integer or string.

The `related_string_id` can also be used in cases when a composite PK is defined (the core architecture internally concatenates the multiple values into a delimited string representation). In the event that you change the primary key of a parent table to type integer, then you have to change the out-of-the-box relationship between the polymorphic recordset and that parent recordset to join using `related_int_id` and not `related_id`. Similarly, in the event that you change the primary key of a parent table to type string, then you have to change the out-of-the-box relationship between the polymorphic recordset and that parent recordset to join using `related_string_id` instead of `related_id`.

This may require that you first define a fieldset on the particular polymorphic recordset for the `related_int_id` or `related_string_id`.

The following is a list of polymorphic recordsets. Unless stated otherwise, these recordsets include the standard `related_id`, `related_string_id` and `related_int_id` fields:

- `audit`
- `e_appointment`
- `e_appointment_invite`
- `e_doc_user_profile`
- `e_pimsync_user_item`
- `e_project_team`
- `e_rel_attachment`
- `e_rel_knowledge_doc`
- `e_related_content`
- `e_related_individual`
- `e_related_product`
- `message`
- `user_note`
- `e_alert_subscription`
- `rp_default_offer` (`offer_related_id`, `offer_related_string_id`, `offer_related_int_id`)
- `rp_offer` (`offer_related_id`, `offer_related_string_id`, `offer_related_int_id`)
- `task_instance`
- `e_alert_action_log`
- `e_alert_push_message`
- `e_alert_subscr_event`
- `e_cal_series` (`entry_related_id`, `entry_related_string_id`, `entry_related_int_id` AND
- `ecur_related_id`, `recur_related_string_id`, `recur_related_int_id`)
- `e_fired_alert`
- `e_push_message`

## BIOs with Composite Primary Keys cannot be Audited

All out-of-the-box business database tables have surrogate primary keys, that is, single column primary keys. Although the core architecture supports composite primary keys, auditing of any BIO based on a table that has a composite primary key is not supported.

## Externalization of BIOs subject to ACLs will cause ACLs to no longer work on those BIOs

The following BIOs are subject to ACLs, and therefore if any of these BIOs are externalized the permissioning on that BIO does not work:

- Customer (and therefore Organization and individual\_customer BIOs as these are subclassed from Customer)
- e\_lead
- e\_opportunity
- Individual
- e\_cal\_occur

To prevent errors when externalizing any of the above BIOs, remove the out-of-the-box ACL Row Permissions from the BIO and disable any of the extensions on that BIO that manage the ACLs (for example, ACLManagerExtension and ConditionalACLExtension).

## Re-mapping Customer BIOs

This section addresses issues related to re-mapping Customer BIOs.

## Re-mapping Customer/Individual/Organization RecordSets and BIOs

In the out-of-the-box Customer Service application, customers are of two different types, that is, 'Individuals' as created through the individual\_customer BIO and used for B2C service/sales, and 'Organizations' as created through the Organization BIO and used for B2B service/sales. Both the individual\_customer BIO and the Organization BIO are subclassed from the Customer BIO. This means that out-of-the-box, an Individual customer is comprised of two records, that is, a record in the Customer table and a record in the Individual table joined through the individual\_id on both tables. Similarly, out-of-the-box, an Organization customer is comprised of two records, that is, a record in the Customer table and a record in the Organization table joined through the organization\_id on both tables. Also,

both the individual\_customer BIO and the Organization BIO inherit all of the attributes of the Customer BIO.

Implementers who wish to re-map the Customer BIOs to their own legacy tables typically only store their Individual data in one legacy table rather than two, or store their organization data in one legacy table rather than two.

Implementers who wish to re-map the individual\_customer BIO to a single table in which they store their Individual data, should map both the Customer recordset and the Individual recordset to that single legacy table. This means that the following fields have to be mapped to the same column:

- customer\_id field on the Customer recordset
- individual\_id field on the Customer recordset
- individual\_id field on the Individual recordset

Implementers who wish to re-map the Organization BIO to a single table in which they store their organization data, should map both the Customer recordset and the Individual recordset to that single legacy table. This means that the following fields have to be mapped to the same column:

- customer\_id field on the Customer recordset
- organization\_id field on the Customer recordset
- organization\_id field on the Organization recordset

Note that a single recordset must be mapped to a single table, and a single recordset field must be mapped to a single column. This means that implementers who re-map the Customer BIOs will not be able to use both the individual\_customer BIO and the Organization BIO as sub-classed BIOs of Customer because the Customer recordset can only be mapped to one table. However, this will typically not be an issue because most implementations either record customers who are Individuals (that is, B2C) OR customers that are organizations (that is B2B), and not both.

## Non-filterable/Sortable Account Widget on Forms if Individual/Organization is Re-mapped

If the Individual or Organization recordset is re-mapped to an external data source, where other BIOs have attributes path mapped to Customer, Organization or Individual, and there are queryable widgets based on those path mapped attributes, in that case, the use of those queryable widgets will cause cross database queries which are not supported by the architecture. Therefore, when the Individual/Organization recordsets are re-mapped to an external data source, these widgets should be changed to use a non-queryable widget type for example, the label widget type or the link widget type. This is basically an application of the more general rules described under "Non-filterable/Sortable Widgets based on Attributes Path Mapped to Externalized BIOs" on page 142.

Note that in some cases where these widgets are queryable, there are in fact two widgets based on the path mapped attribute on the same form, that is, one that is used by the list form itself and one that is used in the filter row of the list form. In this case, rather than changing the type of a widget, it is necessary to remove the widget used by the filter row.

The following is a list of changes necessary to make the front-end 'Account' widgets non queryable:

request\_list\_view form: remove widget 'Copy of customer\_full\_name'

interaction\_list\_view form: remove widget 'Copy of customer\_name'

order\_list\_view form: remove widget 'Copy of customer\_full\_name'

product\_instance\_list\_view form: remove widget 'Copy of customer'

e\_lead\_list form: remove widget 'Copy of customer'

opportunity\_list\_view form: remove widget 'Copy of customer'

quote\_list\_view form: change type of widget customer\_full\_name to label or link

The following is a list of changes necessary to make front-end 'Contact' widgets non queryable -

request\_list\_view form: remove widget 'Copy of individual\_contact\_full\_name'

interaction\_list\_view form: remove widget 'Copy of individual\_name'

e\_lead\_list form: remove widget 'Copy of individual'

opportunity\_list\_view form: remove widget 'Copy of individual'

quote\_list\_view form: remove widget 'Copy of individual'

## Removing Multiple Relationships between Two Records

Currently, the MI data provider allows only one relationship between two recordsets. In the out of box application, there are two relationships between Address and Individual: primary\_address\_individual\_rel and address\_individual\_rel. Remove the primary\_address\_individual\_rel relationship, and change the Mapping Type of related BIO attribute primary\_address from "RELATION" to "COMPUTED," and set the "Mapped Field" to "ComputedPrimaryTouchpoint" which is the same as the BIO attribute "primary\_address\_computed". Apply the same change to the "primary\_email\_address" and "primary\_telephone".

## Session Caching and RecordSets

### RecordSets that do not have FETCH/QUERY Methods Exposed

Externalized recordsets that do not have FETCH or QUERY methods exposed by the external entity need to be cached and retrieved as part of another related recordset. For example, if the address recordset does not have a FETCH/QUERY method exposed, it can be retrieved as part of the Individual recordset that it is related to.

One exception to this limitation is when the SOA connector is enabled using the MI data provider. The SOA connector is capable of inferring a QUERY on the root recordset mapped to a data graph, whenever the BIO layer needs to load any recordset that is part of the graph without needing a map for every a FETCH/QUERY method on every single recordset that is within the graph. For more information, see the SOA connection instructions and examples in the CRMIntegrationGuide.

Note that externalized recordsets that expose FETCH/QUERY methods may also require session caching enabled for performance reasons. This is to avoid going to a Web service or external API repeatedly for data that may change infrequently.

To prevent the BIO layer from invoking a FETCH/QUERY method on a recordset, enable session caching with a time-out value. This will direct the BIO layer to use the data in the Unit of Work cache till the data expires.

In some instances, as part of an extension, you may have to clear the unit of work cache and re-query to make sure a FETCH/QUERY is not issued on a recordset. The following code snippet illustrates how this can be done. For example, to update the cache contents of addresses, you may have to re-query the Individual so that all addresses of that Individual are re-fetched.

```
public class RefetchIndividual extends ActionExtensionBase
{
    protected int execute(ActionContext context, ActionResult result)
        throws EpiException
    {
        if (result.getFocus() instanceof BioBean)
        {
            BioBean _bb = (BioBean) result.getFocus();
            // The following line is to eliminate the individual BIO
            // from the unit of work cache
            context.getState().getDefaultUnitOfWork().removeBio(_bb);
        }
        return RET_CONTINUE;
    }
}
```

## Handling Errors Returned by an External System

When recordsets are mapped to an external system, errors returned by the external system have to be handled, interpreted, and appropriate messages displayed to the user on the forms. You have to do this because often the external system throws different exceptions when a system is down or for various other errors, such as when a customer is not found, when there are data errors, and for security related exceptions.

To distinguish between these various errors and display useful error messages to the user, write an extension to facilitate this. A use case where the user might want to handle errors is while displaying a list in a list view or when displaying the related records in a sub-tab form.



## Handling Errors when displaying a List in a List View

To handle the exceptions, extend the default QueryAction extension. In this extension, you delegate the call to the base class method to do the actual processing and handle any exceptions that might be thrown. Since the actual query will not be invoked to the external system until the first access to any unknown attribute is made, there needs to be a fetch for an attribute that does not belong to the PK field set within the try..catch block. For example, assume that you have a Customer BIO whose PK set is the combination of attributes first\_name and last\_name. To make the error handling work, you have to fetch at least one attribute that is neither first\_name nor last\_name, for instance, date\_of\_birth. The code would be something similar to the following:

```
public class ErrorHandledQueryAction extends QueryAction
{
    private static ILoggerCategory _log
        = ApplicationUtil.getLogger(ErrorHandledQueryAction.class);
    protected int execute(ActionContext context, ActionResult result)
        throws UserException
    {
        int returnCode = RET_CANCEL;
        Object resultObj = null;
        try
        {
            returnCode = super.execute(context, result);
            resultObj = result.getFocus();
            if (resultObj instanceof BioCollection)
            {
                ((BioCollection) resultObj).size();
            }
            else
            {
                BioBean bioBean = (BioBean) result.getFocus();

                // Fetch at least one attribute that does not belong
                // to the PK field set.
                String bioAttribute
                    = BioAttributeTable.getBioAttribute(<bioAttributeName>);
                if (bioAttribute != null)
                {
                    bioBean.get(bioAttribute);
                }
            }
        }
        catch (Exception e)
        {
            _log.error("EXP_EXTERNAL_QUERY_ERROR_TRACE",
                "Full Stack Trace For External Query Error", e);
            Throwable firstEx = e;
            if (e instanceof EpiException)
            {
                firstEx = ((EpiException)e).getDeepestNestedException();
            }
        }
    }
}
```

```
        else if(e instanceof EpiRuntimeException)
        {
            // If it's EpiRuntimeException, try to get the deepest one.

            Enumeration en = ((EpiRuntimeException)e).elements();
            while (en.hasMoreElements())
            {
                firstEx = (Exception)en.nextElement();
            }
        }
        throw new FormException("External System Error : {0} Message:
{1}",
                                new Object[]{firstEx.getClass().
                                                getName(),firstEx.getMessage
                                                ()});
    }
    return returnCode;
}
}
```

## Handling Errors when displaying Related Records in a Sub-Tab Form

To handle errors while displaying the related records in the sub-tab, do the following:

- 1 Create a new extension and attach it to the tabclick event.
- 2 Register a prerender extension on the Sales\_tab\_group shell form only if the tab is the first tab in the tab group. In this case, you cannot use the tabclick event to do the query. If the tab is not the first tab, you do not need a prerender extension.

Within this extension you have to explicitly make the query based on the path, address the exceptions, and display the appropriate error messages. You can also display a blank form in the form slot.

### Sample Extension Code for Sub-classing and Overriding the Default TabClickEvent Extension

```
public class ErrorHandledTabClickAction extends ActionExtensionBase
{
    private static ILoggerCategory
        _log = ApplicationUtil.getLogger(ErrorHandledQueryAction.class);

    private static HashSet BIOS = new HashSet();

    static
    {
        BIOS.add("address");
    }
}
```

```

        BIOS.add("email_address");
        BIOS.add("telephone");
        // Add more BIO type name here
    }

    protected int execute(ActionContext context, ActionResult result)
        throws EpiException
    {
        int returnCode = RET_CANCEL;
        DataBean resultObj = null;
        try
        {
            returnCode = super.execute(context, result);
            if (!BIOS.contains(resultObj.getDataType()))
            {
                return returnCode;
            }
            if (resultObj instanceof BioCollection)
            {
                ((BioCollection)resultObj).size();
            }
            else
            {
                BioBean bioBean = (BioBean)result.getFocus();
                // Fetch at least one attribute that does not belong
                // to the PK field set.
                String bioAttribute
                    = BioAttributeTable.getBioAttribute(<myBioAttribute>);
                if (bioAttribute != null)
                {
                    bioBean.get(bioAttribute);
                }
            }
        }
        catch (Exception e)
        {
            _log.error("EXP_EXTERNAL_QUERY_ERROR_TRACE",
                "Full Stack Trace For External Query Error", e);
            Throwable firstEx = e;
            if (e instanceof EpiException)
            {
                firstEx = ((EpiException)e).getDeepestNestedException();
            }
            else if (firstEx instanceof EpiRuntimeException)
            {
                // If it's EpiRuntimeException, try to get the deepest one
                Enumeration en = ((EpiRuntimeException)firstEx).elements();

                while(en.hasMoreElements())
                {
                    firstEx = (Throwable)en.nextElement();
                }
            }
        }
    }

```

```
        throw new FormException("External System Error : {0} Message:
{1}",
                                new Object[]{firstEx.getClass().
                                getName(),firstEx.getMessage()});
    }
    return returnCode;
}
}
```

### Sample Extension Code for the Sales\_tab\_group Shell Pre-render Extension

```
public class ErrorHandledPreRender extends FormExtensionBase
{
    private static ILoggerCategory _log
        = ApplicationUtil.getLogger(ErrorHandledQueryAction.class);
    protected int preRenderListForm(UIRenderContext context,
                                    RuntimeListFormInterface form)
        throws EpiException
    {
        int returnCode = RET_CANCEL;
        Object resultObj = null;
        try
        {
            returnCode = super.execute(context, result);
            resultObj = result.getFocus();
            if (resultObj instanceof BioCollection)
            {
                ((BioCollection)resultObj).size();
            }
            else
            {
                // The is only one customer returned, go to detail
                // view directly
                BioBean bioBean = (BioBean)result.getFocus();
                // Fetch at least one attribute that does not
                // belong to the PK field set.
                String bioAttribute
                    = BioAttributeTable.getBioAttribute(<myBioAttribute>);
                if (bioAttribute != null)
                {
                    bioBean.get(bioAttribute);
                }
            }
        }
        catch (Exception e)
        {
            _log.error("EXP_EXTERNAL_QUERY_ERROR_TRACE",
                "Full Stack Trace For External Query Error", e);
            Throwable firstEx = e;
            if (e instanceof EpiException)
            {

```

```

        firstEx = ((EpiException)e).getDeepestNestedException();
    }
    else if (firstEx instanceof EpiRuntimeException)
    {
        // If it's EpiRuntimeException, try to get the deepest one
        Enumeration en = ((EpiRuntimeException)firstEx).elements();
        while (en.hasMoreElements())
        {
            firstEx = (Throwable)en.nextElement();
        }
    }
    throw new FormException("External System Error : {0} Message:
{1}",
                                new Object[]{firstEx.getClass().
                                                getName(),firstEx.getMessage()});
    }
    return returnCode;
}
}

```



*BIO transformation* enables you to transform a BIO of any type to a BIO of another or the same type, and traverse path(s) on a BIO or BIO Collection with filtering of collections along the path. To perform BIO transformation, you have to create transformation maps.

## BIO Mapping

Before you create a mapping between two BIO types, address the following questions

- Will the mapping be of type Transformation or Path Traversal?
- If the mapping is of type Transformation, will the mapping be bidirectional? If it is bidirectional, then you have to use a key mapping scheme to create/update the target. You can either use a primary or alternate key of the recordset of the BIO(s) involved. The attributes mapping to the fields of the keys have to be mapped across the two BIO(s).
- If you are not using bidirectional mapping, will the target be always created?
- Will the target be a Helper BIO?
- Is this a default mapping between the two BIO types?

## Creating a Transformation Map

Creating a transformation map involves

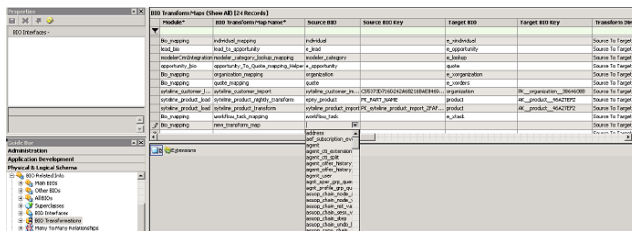
- "Creating a Unique Map Name" on page 156
- "Linking Source and Target" on page 157
- "Choosing type of Transformation" on page 158
- "Setting Masked Attributes for Map" on page 159
- "Creating Attribute Maps for Transformation" on page 159
- "Setting Properties on Path Elements" on page 164
- "Handling Attribute Domains" on page 169

**Note:** This procedure assumes that you are familiar with BIOs, record sets, creating keys on record sets, out-of-the-box BIOs such as Customer and Request, and extensions.

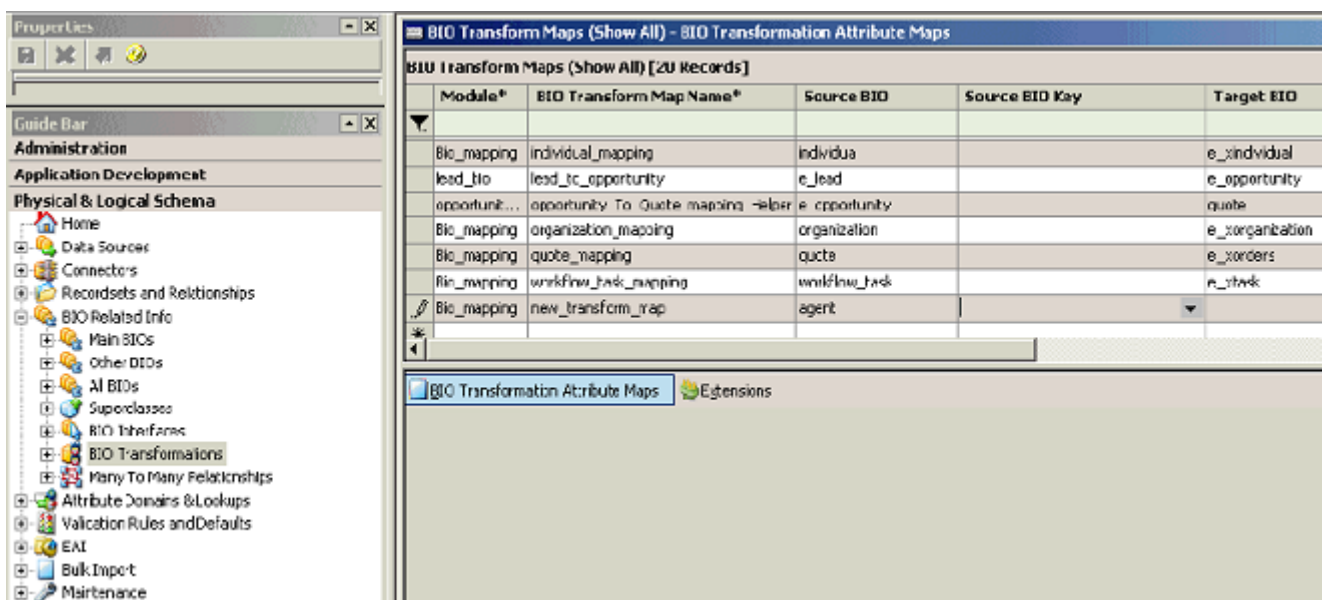
You can also create a transformation map using the BIO Transformation wizard in Infor Studio. For information on using the wizard, refer to the Infor Studio online help.

## Creating a Unique Map Name

- 1 In Infor Studio, navigate to *Bio Transformations (Physical & Logical Schema > BIORelated Info > Bio Transformations)*. The **Bio Transform Maps** form appears.
- 2 In the empty row at the bottom, select a module and provide a unique map name.
- 3 Pick **Source** and **Target** BIO types from the respective drop-downs. Choose Source and Target BIO Keys only if it is a mapping where the target is not created.

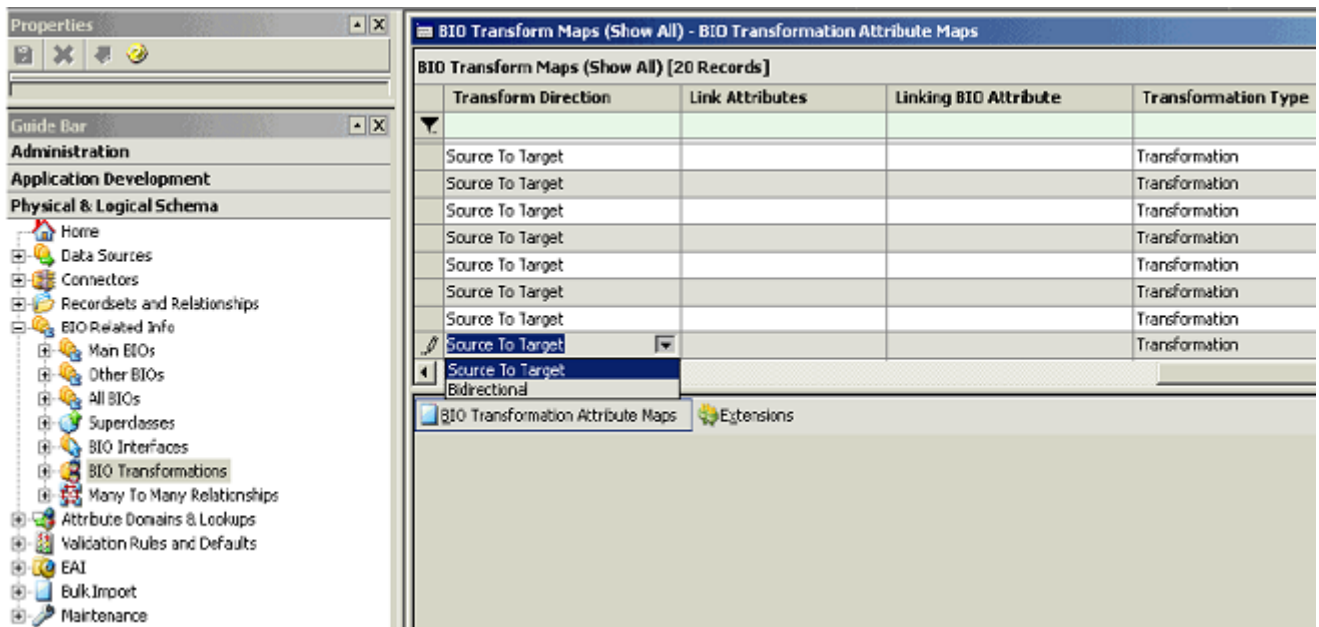


- 4 If key translation is required for the transformation, choose an Alternate Key/Primary Key (AK/PK) on both BIOs. Key translation enables you to search and retrieve the target for update purposes. If a target exists, it is updated, otherwise it is inserted. Key translation is only required for maps that do not have the **Create Only** masked attribute enabled. If you do not define AK/PK on the BIOs being mapped, the drop-down list for the keys will be empty.



- 5 Pick the direction of the transformation. Under **Transform Direction**, select either **Source to Target** or **Bidirectional**





## Linking Source and Target

You set link attributes only if the source and target are to be linked to each other after the transformation is complete. This step is only valid for a transformation where you enable the *Create Only* masked attribute. You can set the following link attributes

Name	Description
Set on Source	Sets the Target Bio on the Source Bio using the Linking Bio Attribute name.
Set on Target	Sets the Source Bio on the Target Bio using the Linking Bio Attribute name.
Set on Source Collection with Role	Adds the Target Bio to the Source Bio collection attribute referred to by the Linking Bio Attribute .
Set on Target Collection with Role	Adds the Source Bio to the Target Bio collection attribute referred to by the Linking Bio Attribute .

If you enable linking, pick the **Linking Bio Attribute** name.

**BIO Transform Maps (Show All) - BIO Transformation Attribute Maps**

**BIO Transform Maps (Show All) [20 Records]**

Transform Direction	Link Attributes	Linking BIO Attribute	Transformation Type
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation
Source To Target			Transformation

**BIO Transformation Attribute Maps**

Set on Source  
Set on Target  
Set on Source Collection with  
Set on Target Collection with

## Choosing type of Transformation

Select the type of transformation to be *Transformation*.

**BIO Transform Maps (Show All) - BIO Transformation Attribute Maps**

**BIO Transform Maps (Show All) [20 Records]**

Target BIO	Target BIO	Transform Direction	Link Attributes	Linking BIO Attribute	Transformation Type
e_xindividual		Source To Target			Transformation
e_xindividual		Source To Target			Transformation
e_opportunity		Source To Target			Transformation
quote		Source To Target			Transformation
e_xorganization		Source To Target			Transformation
e_xorders		Source To Target			Transformation
c_xtask		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation
		Source To Target			Transformation

**BIO Transformation Attribute Maps**

Extensions

**Attribute Maps / Path Elements [0 Records]**

Module*	Source Attribute Mapping	Target Attribute Mapping
* Bio_manning		

## Setting Masked Attributes for Map

Select the type of masked attribute that you want to enable on the transformation. Click ... to open the Masked Attributes dialog box. Select an attribute by checking a check box next to **Create only**, or **Default Mapping between Bio Types**, or **Target is Helper Bio**.

Name	Description
Create only	This attribute is valid only for transformation mappings and the target is always created and does not use any key translation mechanism.
Default Mapping between BIO types	This attribute is useful in recursive mapping. This is used when two Bio(s) or Bio Collections are mapped together in an attribute map.
Target is Helper Bio	This attribute is used to create a HelperBio as the target of transformation.

**Note:** **Create only** and **Target is Helper Bio** override the bidirectional settings on the map.


## Creating Attribute Maps for Transformation

A transformation map is made up of one or more attribute maps. Attribute maps map the source attribute to the target attribute. Each attribute map also contains path elements. You define attribute maps in the **Bio Transformation Attributes Maps** subform.

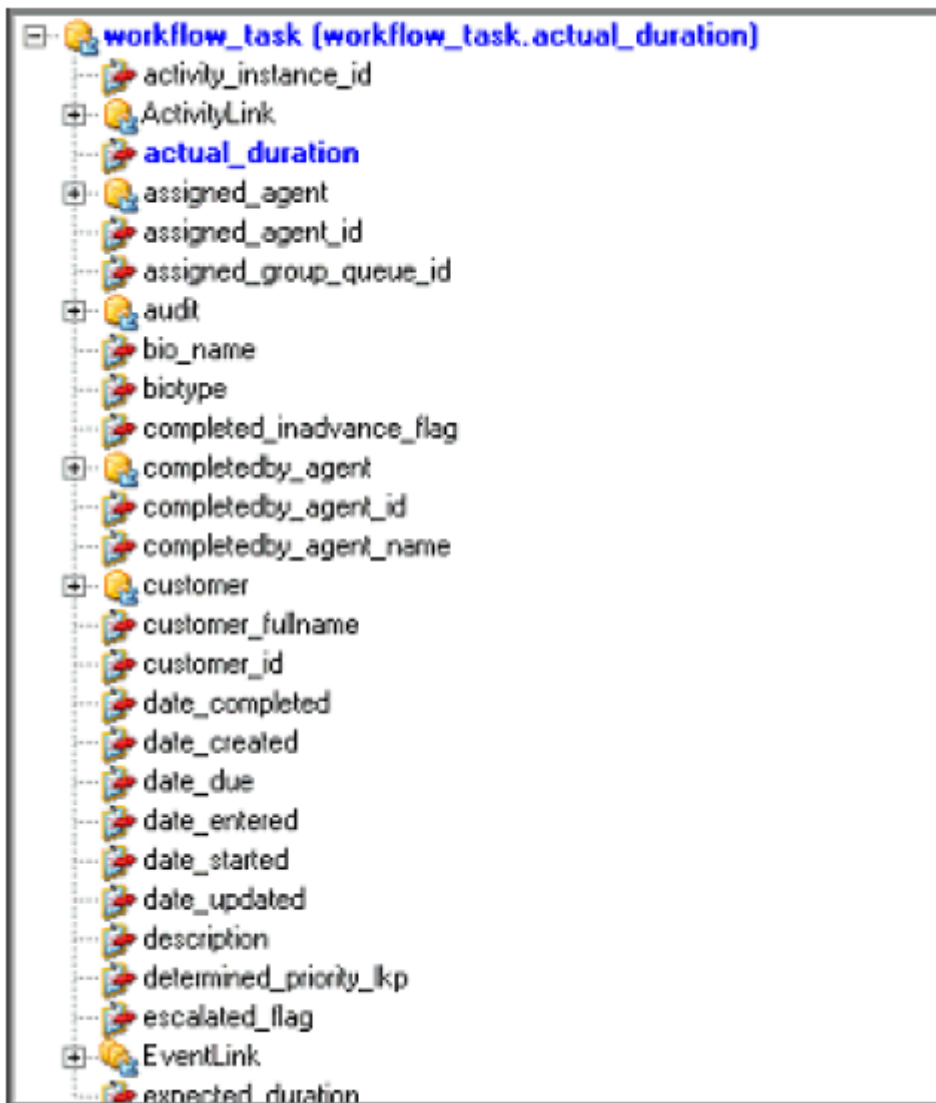
Before defining an attribute map, decide whether the attribute map will be **Bidirectional** or **Source To Target** or **Target To Source** or **Do not set Target, if Source is null**. You set map direction under **Bio Transformation Attribute maps**. If you choose none, the default behavior is bidirectional.

**Note:** Setting the Do not set Target, if Source is null attribute will cause the transformation to not set any target attributes where the source attribute value is null. This includes not evaluating any paths on the target. The transformation process also does not support mapping of Computed or Query attributes.

### Direct to Direct (Simple Direct Mapping of Attributes)

- 1 Begin with the Source BIO. In the **BIO Transformation Attributes** sub form, under **Source Attribute Mapping**, click the  icon. The Bio Transformation Maps window appears. Select an attribute, click Ok to close the pop-up, and save.

The selected attribute is highlighted in the following screen shot.



- 2 Select the Target BIO attribute (similar to how you selected the Source BIO).
- 3 Under **Map Direction**, select the direction of the map.
- 4 Save the attribute map. After your save is successful, a split screen is provided at the bottom. The screen lists all the path elements for the source and target attribute. The left hand side of the screen is for the source and the right hand side is for the target. More than one path element is listed if the attribute is traversed over a BIO or BIO Collection. For direct mapped attributes, you can attach an EpiExpression to perform computations. EpiExpression on the source path is evaluated to set on target and vice versa.

The screenshot displays the Infor Epiphany interface with two main panels. The top panel, titled 'Bio Transform Maps (Show All) [19 Records]', lists various mappings between source and target BIOs. The bottom panel, titled '@ Bio Transformation Attribute Mapping', shows a detailed view of attribute mappings for a specific module.

Module*	Bio Transform Map Name*	Source BIO	Source BIO Key	Target BIO
currency	currency_rate_import	currency_rate_import		e_currency_rate
PIM2sync	donino_appointment_attendee_map	donino_appointment	PIM2sync_AK	e_appointment
PIM2sync	donino_appointment_map	donino_appointment	PIM2sync_AK	e_appointment
PIM2sync	donino_contact_map	donino_contact	PIM2sync_AK	individual
PIM2sync	donino_task_map	donino_task		workflow_task
Bio_mapping	e_lead_mapping	e_lead		e_lead
Bio_mapping	e_opportunity_mapping	e_opportunity		e_opportunity
Bio_mapping	e_related_product_mapping	e_related_product		e_related_product
PIM2sync	exchange_appointment_attendee_map	exchange_webbizer_a...	PIM2sync_AK	e_appointment
PIM2sync	exchange_appointment_map	exchange_webbizer_a...	PIM2sync_AK	e_appointment
PIM2sync	exchange_contact_map	exchange_webbizer_c...	PIM2sync_AK	individual
PIM2sync	exchange_task_map	exchange_webbizer_task	PIM2sync_AK	workflow_task
Bio_mapping	individual_customer_mapping	individual_customer		e_individual
Bio_mapping	individual_mapping	individual		e_individual
lead_bio	lead_to_opportunity	e_lead		e_opportunity
opportunity_bio	opportunity_to_quote_mapping_helper	e_opportunity		order
Bio_mapping	order_mapping	order		e_orders
Bio_mapping	organization_mapping	organization		e_organization
Bio_mapping	workflow_task_mapping	workflow_task		e_task

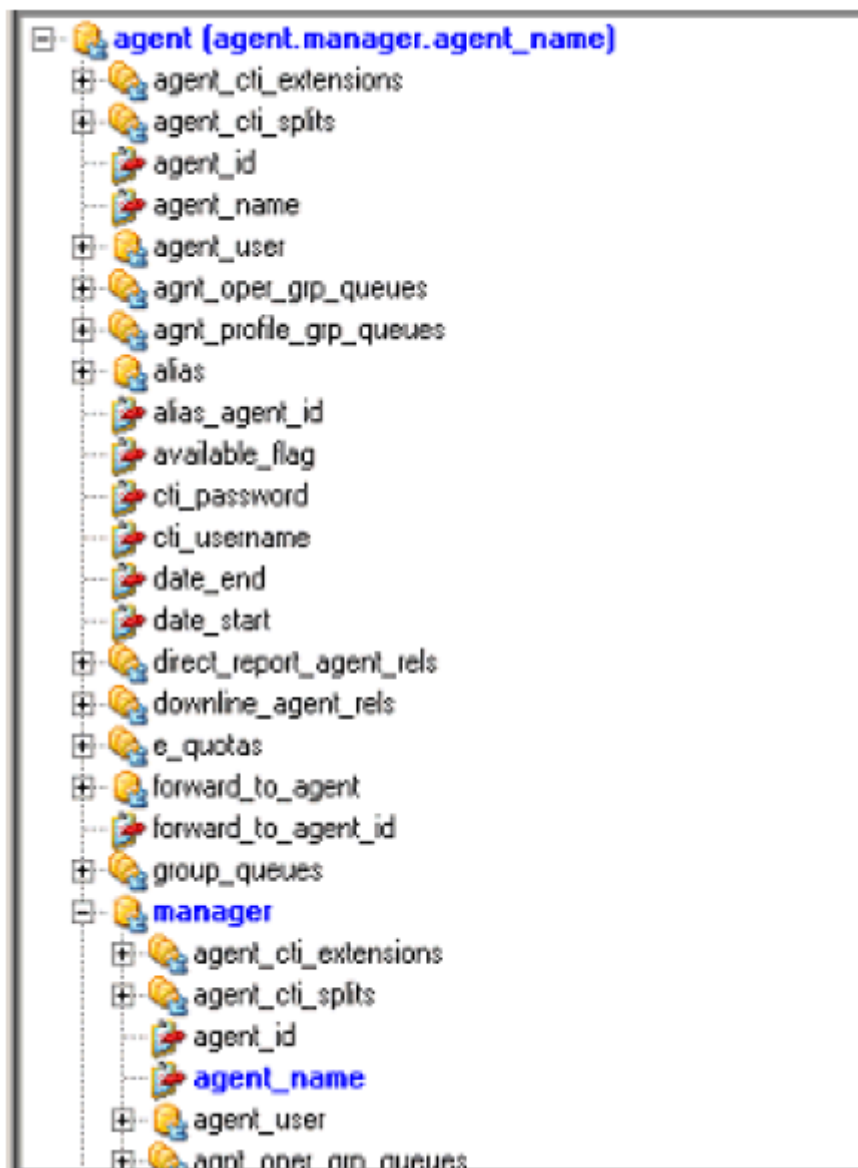
Module*	Source Attribute Mapping	Target Attribute Mapping	Map Direct
Bio_mapping	workflow_task_related_text (15)	e_task_related_text (15)	
Bio_mapping	workflow_task_reminder_date (16)	e_task_reminder_date (16)	
Bio_mapping	workflow_task_progress_status_fp (17)	e_task_status (17)	
Bio_mapping	workflow_task_suspend_duration (18)	e_task_suspend_duration (18)	
Bio_mapping	workflow_task_suspend_resume_date (19)	e_task_suspend_resume_date (19)	
Bio_mapping	workflow_task_task_id (20)	e_task_task_id (20)	
Bio_mapping	workflow_task_task_name (21)	e_task_task_name (21)	
Bio_mapping	workflow_task_task_state (22)	e_task_task_state (22)	
Bio_mapping	workflow_task_task_state (22)	e_task_task_state (22)	

## Mapping Attributes with BIO's in Their Path

Attribute maps can have BIOs in their paths. You can mark the BIOs in the path as lookup or create/update. For example, for the attribute mapping

```
source.agentname < -- > agent.manager.agent_name
```

The manager attribute on agent can be created/updated or looked up (filter using agent\_name with value of agentname). Lookups are based on a single attribute only.



Pick the Source and Target BIO attributes and setup masked attributes if lookup is required for any BIO's in the path. Enable the **Lookup BIO with Source Attributes** option. Once you check the flag, the masked appropriate attribute text is displayed in the Masked Attributes Column.

The screenshot displays the 'BIO Transform Maps' window with a list of 20 records. The columns are: Module\*, Bio Transform Map Name\*, Source BIO, Source BIO Key, and Target BIO. Below this, the 'BIO Transformation Attributes Mapping' window shows a table with columns: Module\*, Source Attribute Mapping, Target Attribute Mapping, and Map Direct. The bottom window shows a table with columns: Module, Bio Attr, Expression, Lookup Attribute Name, Element Key, and Mapped Attributes.

Module*	Bio Transform Map Name*	Source BIO	Source BIO Key	Target BIO
bio_mapping	e_lead_mapping	e_lead		e_lead
bio_mapping	e_opportunity_mapping	e_opportunity		e_opportunity
bio_mapping	e_related_product_mapping	e_related_product		e_related_product
FMSync	exchange_appointment_attendee_map	exchange_webdev_a...	FMSYNC_AU	e_appointment
FMSync	exchange_appointment_map	exchange_webdev_a...	FMSYNC_AU	e_appointment
FMSync	exchange_contact_map	exchange_webdev_a...	FMSYNC_AU	individual
FMSync	exchange_task_map	exchange_webdev_task	FMSYNC_AU	workflow_task
bio_mapping	individual_customer_mapping	individual_customer		e_individual
bio_mapping	individual_mapping	individual		e_individual
lead_bio	lead_to_opportunity	e_lead		e_opportunity
opportunity_bio	opportunity_to_order_mapping_helper	e_opportunity		order
bio_mapping	order_mapping	order		e_order
bio_mapping	organization_mapping	organization		e_organization
bio_mapping	workflow_task_mapping	workflow_task		e_task
bio_mapping	test_mapping	agent		agent
bio_mapping				

Module*	Source Attribute Mapping	Target Attribute Mapping	Map Direct
bio_mapping	agent.agent_name (1)	agent.manager (1)	
bio_mapping	agent.ct_username (2)	agent.agent_user_agent_name (2)	
bio_mapping			

Module	Bio Attr	Expression	Lookup Attribute Name	Element Key	Mapped Attributes
bio_mapping	ct_username				Lookup BIO with Source At...
bio_mapping					

## Extension Support in BIO Transformations

The BIO transformation process supports extensions. You can configure extensions for the following three events: **Before Transformation**, **After Transformation**, and **Custom Attribute Transformation**.

### Before Transformation

Implement the extension from `com.epiphany.shr.data.bioxform.extensions.BioTransformExtensionBase` and implement the `bioAfterTransform(..)` method to set defaults, and so on.

### After Transformation

Use this event to validate the transformation. Implement the extension from `com.epiphany.shr.data.bioxform.extensions.BioTransformExtensionBase` and implement the `bioBeforeTransform(..)` method.

### Custom Attribute Transformation

Use this event to completely customize the attribute transformation without evaluating the attribute map. The extension is provided with the source and target (as defined in the map) and the corresponding attribute names (at the lowest level in the path) and the direction of the transformation (**source > target** or **target > source**) for current execution.

Implement the extension from `com.epiphany.shr.data.bioxform.extensions.BioTransformExtensionBase` and implement the `customAttributeTransform(..)` method.

**Note:**

- Double-click on a map to access the custom attribute transformation event.
- The BIO transformation extensions can get the source object as a BIO or as a Helper Bio depending on what the consumer of the transformation service is creating. If you do not cast the source object to the correct type, you will receive a runtime null pointer exception.

## Setting Properties on Path Elements

You can use the following settings on Source and Target path elements:

- "BIO Attribute" on page 164
- "BIO Transform Map" on page 164
- "Cast BIO Class" on page 164
- "EpiExpression" on page 164
- "Lookup Attribute Domain Name" on page 165
- "Element Key" on page 165
- "Masked Attributes" on page 165
- "Direct Mapped Attribute to a Collection Element Attribute" on page 165

### BIO Attribute

Infor Studio automatically populates the BIO attribute.

### BIO Transform Map

If both source and target attributes are of type BIO or BIO Collection, the attribute map can specify another Bio-to-Bio transformation map for recursive transformation. You have to set this map on the target BIO for source to target transformation and vice versa. If you don't provide a map, the transform service tries to use a default map between the two BIO types. If it does not find a default map, an exception condition occurs during processing.

### Cast BIO Class

Path Elements that are of type BIO can be cast to a specific type as part of the transformation processing.

### EpiExpression

You can use an EpiExpression to perform computations using IBR on the source to set on target or vice versa. Do not set up expressions on both source and target attributes of a bidirectional attribute map. This results in an exception.





**Caution:** Set an attribute map direction to either Source to Target or Target to Source and not Bidirectional. If the direction is Source to Target, you can have the EpiExpression on the Source attribute and vice versa.

Set EpiExpressions only on the last path element (on non BIO or BioCollection attributes) on the source attribute of a map. Common uses include concatenating multiple attributes to one, splitting a single attribute into multiple, setting constant values for attributes, and tokenizing an attribute value using delimiters.

EpiExpression on source path is evaluated to set on target and vice versa.

## Lookup Attribute Domain Name

If a BIO attribute has an associated attribute domain, Infor Studio automatically populates this with the name of the attribute domain. Currently, only the Lookup attribute domain is supported.

## Element Key

Set the element key if a single collection element is being mapped from source to target.

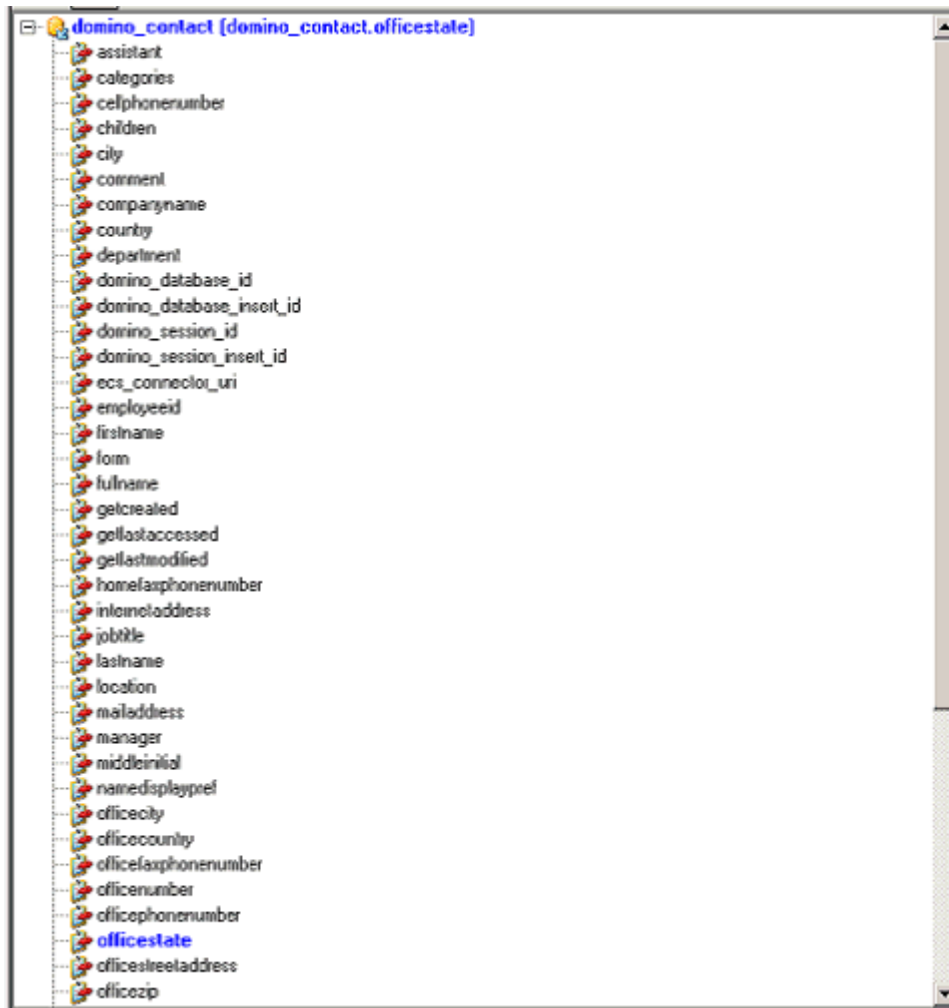
## Masked Attributes

Select either **Delete and Insert Collection Elements** or **Lookup BIO with Source Attributes** .

Name	Description
Delete and Insert Collection Elements	If the path element is a Bio Collection element, the transform service can delete all elements in the collection and recreate the collection. Use this for pivoting a Collection on a Bio to a flat structure (direct mapped attributes) on another Bio.
Lookup BIO with Source Attributes	An attribute in the source can be mapped to the target in such a way, that a Bio in the target path can be looked up using the single source attribute. This is for use only when looking up a BIO.

## Direct Mapped Attribute to a Collection Element Attribute

Use **Identify a Collection element using Filter** if one or more elements in a collection are being added/updated as part of the transformation.

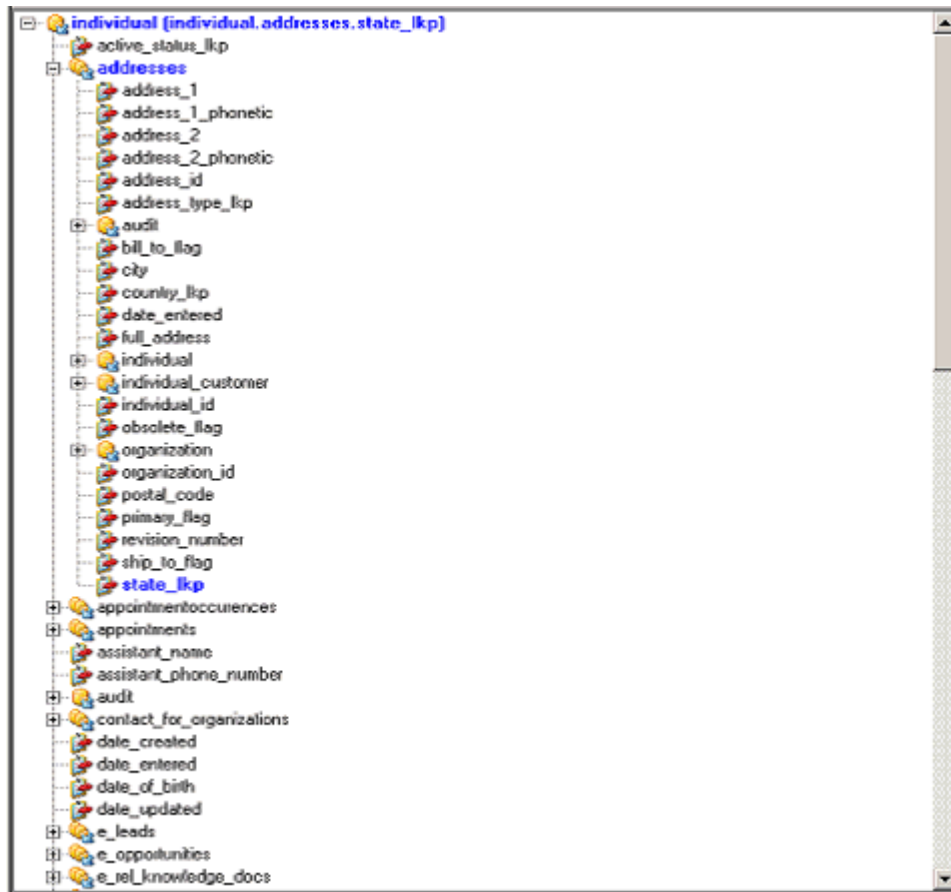


In the above example, source attribute is **officestate** on the **domino\_contact** Bio. The target is **state\_lkp** attribute of an element of **addresses** collection on the **individual** Bio.

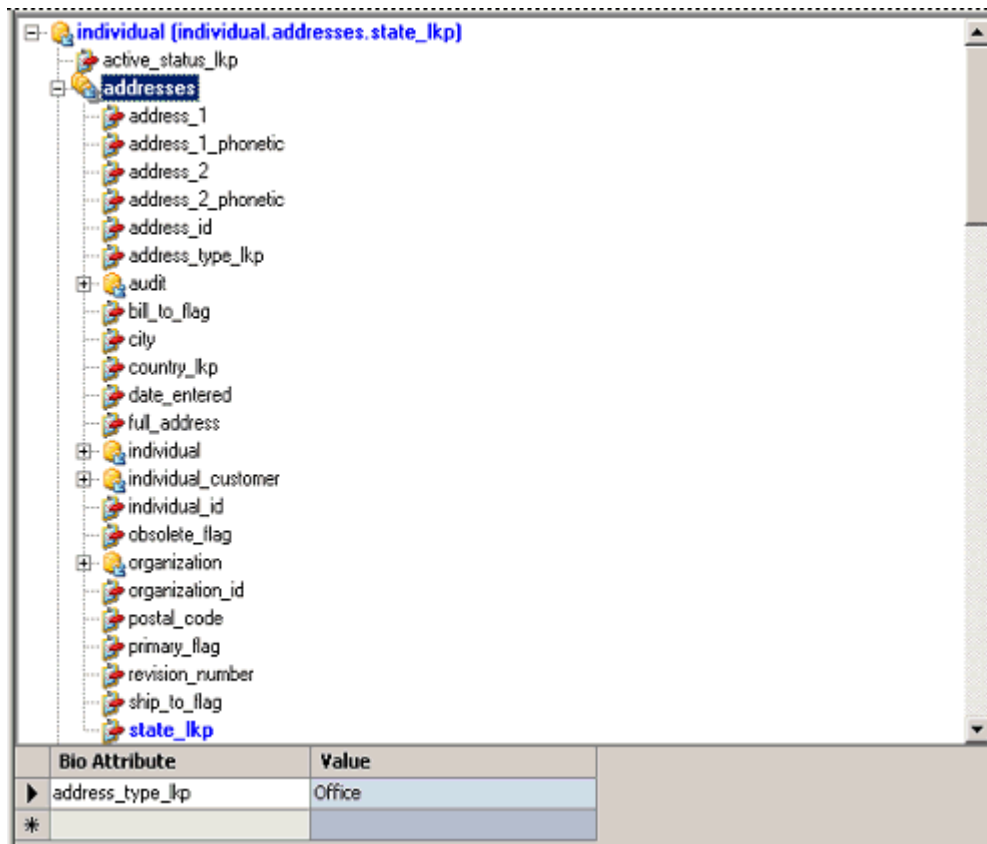


In the above example, the addresses collection on the target is setup with the **Delete and Insert Collection** masked attribute. As part of the transformation, the addresses collection is deleted first and elements are subsequently added to it. This setting is optional.

You can set the filter by clicking on the icon next to the collection attribute, the one to the left of the addresses attribute. The filter can be any number of attributes with default values specified. The filter attributes are used to execute the search. If a single element is found, it is updated, otherwise a new element is created and added to the collection. If more than one element is found for the criteria, it is an exception condition.



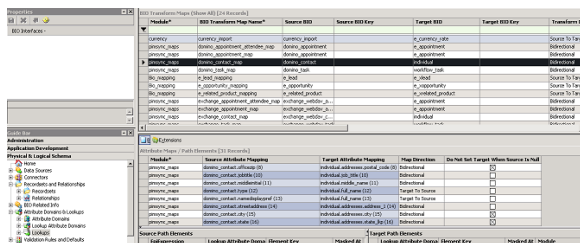
The attributes that have specified values in the filter need not be mapped as those values are used to populate the collection element BIO.



## Identify Collection Element with an Alternate Key

Use this mapping when you have to use a set of attributes on the source BIO to update attributes on a single collection element on the target. In this mapping, a filter is not used. Instead, the Element Key attribute on the path element is used to specify an Alternate Key to be used to uniquely identify the element in the Collection. Only that element is updated/inserted as part of the transformation.

**Note:** Do not use Delete and Insert Collection Elements masked attribute with this type of mapping.



## Handling Attribute Domains

The transformation service handles attribute domains automatically in two ways: by mapping an attribute (no domain) to an attribute with a domain and mapping attributes where both have domains.

### Mapping an Attribute (No Domain) to an Attribute with a Domain

In this case, the value of the attribute without a domain is used as Code String to lookup the attribute domain in metadata. This works in a bidirectional mapping with the following source to target results. Whether a GUID or Code-String is used depends on the type of the Attribute Domain itself.

Code-String > lkp GUID

Lkp GUID > Code-String

Code-String > Code-String

### Mapping Attributes where both have Domains

If both the attributes have the same AD, the value is just copied from source to target. Otherwise, the target is always created and will not use any key translation mechanism.

## Creating a Path Traversal Map


This section provides high level steps involved in creating a Path Traversal map. This section also assumes that you are familiar with creating transformation maps. Path traversal mapping allows a path to be traversed given a BIO. The path may consist of BIO(s) or BioCollection(s) or both. For Bio Collections, you can specify filtering and sorting. A path traversal can result in one of the following being returned


- Direct/Indirect Mapped Attribute of any type
- Relation mapped BIO
- Relation mapped BioCollection

## Creating a Map

- 1 Create a unique transformation map name and choose the mapping type to be of type **Path Traversal**.
- 2 Choose a Source BIO type that is used as the input for the path traversal and save the map.

BIO Transform Maps (Show All) [20 Records]				
	BIO Transform Map Name*	Source BIO	Source BIO Key	Target BIO
▼				
	individual_mapping	individual		e_xindividual
	lead_to_opportunity	e_lead		e_opportunity
	opportunity_To_Quote_mapping_Helper	c_opportunity		quote
	organization_mapping	organization		e_xorganization
	quote_mapping	quote		e_xorders
	workflow_task_mapping	workflow_task		e_xtask
▶	test_path_traversal	agent_user		
*				
◀				

 BIO Transformation Attribute Maps

 Extensions



Attribute Maps / Path Elements [0 Records]				
	Module*	Source Attribute Mapping	Target Attribute Mapping	Map Direction
*	EDialogs			Source To Target

## Setting up Path for Traversal

- 1 Click on **Source Attribute Mapping**

BIO Transform Maps (Show All) [20 Records]					
	Module*	BIO Transform Map Name*	Source BIO	Source BIO Key	Target B
▼					
	Bio_mapping	individual_mapping	individual		e_xindividu
	lead_bio	lead_to_opportunity	e_lead		e_opportun
	opportunit...	opportunity_To_Quote_mapping_Helper	e_opportunity		quote
	Bio_mapping	organization_mapping	organization		e_xorganiz
	Bio_mapping	quote_mapping	quote		e_xorders
	Bio_mapping	workflow_task_mapping	workflow_task		e_xtask
▶	EDialogs	test_path_traversal	agent_user		
*					
◀					

 BIO Transformation Attribute Maps	 Extensions
---	--

Attribute Maps / Path Elements [0 Records]				
	Module*	Source Attribute Mapping	Target Attribute Mapping	Map Direction
*	EDialogs			Source To Target

In the following example, the path traversal results in the return of a BIO. In case the path involves traversing a Bio collection, you can provide a filter so that a single element can be used for further path traversal.



agent\_user (agent\_user.d\_user\_preferences.user\_data)

active\_status\_lkp

agent

agent\_id

agent\_name

agent\_profile\_queues

agent\_user

agnt\_oper\_grp\_queues

agnt\_profile\_grp\_queues

alias

alias\_agent\_id

alias\_downline

available\_flag

cti\_extensions

cti\_password

cti\_splits

cti\_username

d\_user\_preferences

object\_name

obsolete\_flag

revision\_number

user\_data

user\_data\_id

user\_pref\_instance\_id

user\_pref\_tmpl\_name

user\_pref\_value

Bio Attribute	Value
revision_number	20



## Setting Path Element Properties

You can set up the following properties on the path elements. All other properties have no effect on the processing.

Cast Bio Type > Cast related BIOs in the path to specific BIO types.

EpiExpression > Perform computations on non-Bio or BioCollection attributes.

This chapter describes how to configure the Dialogs Service.

## Overview

*Infor Dialogs* is a scripting tool that provides a consistent process for guiding a user through a series of questions and using the answers to determine which subsequent questions and other information to present to the user. EDialogs consists of a metadata repository, the Dialog service, and the Dialog Designer. The metadata repository is a relational database that stores definitions of dialogs. The Dialog Designer is the development tool that you use to create and edit Dialogs. The Dialog service executes the dialogs. It is the engine that manages dialog execution.

## Launching Dialogs from Customer Service

The DialogService provides the ability to view dialogs from an Infor Epiphany Sales and Service application. There are two different ways in which you can do this. You can view a dialog within an iframe of a form or in a separate browser window. The EDialogs module comes with three extensions that enable this capability.

### OpenDialogInFrame

Use the **OpenDialogInFrame** extension to view a dialog in an iframe. In the following example, dialog viewing is enabled in the **individual\_customer\_detail** form. To enable, you have to add a **dropdown** widget (with a list of available dialog definitions) and an **iframe** widget.

- 1 In the **User Interface** tab, click on **Forms**.
- 2 Select **individual\_customer\_detail**.
- 3 Add a new widget of type **form\_widget-iframe**.

- 4 Enter a **Widget Name** as in the following screen and save it.  
You can modify the widget label to be any text in the upper-left corner Properties pane.
- 5 In the Properties pane, set **src** to **about:blank**.

The screenshot shows the Infor Epiphany interface. On the left is the **Properties** pane for a **Form Widget/Form Slots - individual\_custo...**. The **src** property is set to **about:blank**. On the right is the **Forms (Custom Filter) [9 Records]** table.

Module*	Form Name*	Form Type*	BIO
customer_ui	individual_all_people_list_view	LIST	indiv
customer_ui	individual_customer_detail_view	NORMAL	indiv
customer_ui	individual_customer_list_view	LIST	indiv
customer_ui	individual_customer_more_detail_view	NORMAL	indiv
customer_ui	individual_detail_view	NORMAL	indiv
email_ui	individual_email_address_list	LIST	indiv
customer_ui	individual_list_view	LIST	indiv
customer_ui	individual_more_detail_view	NORMAL	indiv
email_ui	individual_recipient_list	LIST	indiv

Below the Forms table is the **Form Widget/Form Slots [26 Records]** table.

Module*	Widget Type*	Attribute Name	Widget Name*
customer_ui	form_widget-dropdown	active_status_flag	active_status_flag
customer_ui	form_slot-normal	address_slot	address_slot
customer_ui	form_widget-dropdown	customer_category_flag	customer_category_flag
customer_ui	form_widget-date picker	date_entered	date_entered
customer_ui	form_widget-date picker	date_of_birth	date_of_birth
customer_ui	form_widget-frame	dialog_viewer_frame	dialog_viewer_frame
customer_ui	form_slot-normal	email_address_slot	email_address_slot
customer_ui	form_widget-text	first_name	first_name
customer_ui	form_widget-dropdown	gender_flag	gender_flag
customer_ui	form_widget-dropdown	generation_flag	generation_flag

- 6 Add a **dropdown** widget and enter a widget name.

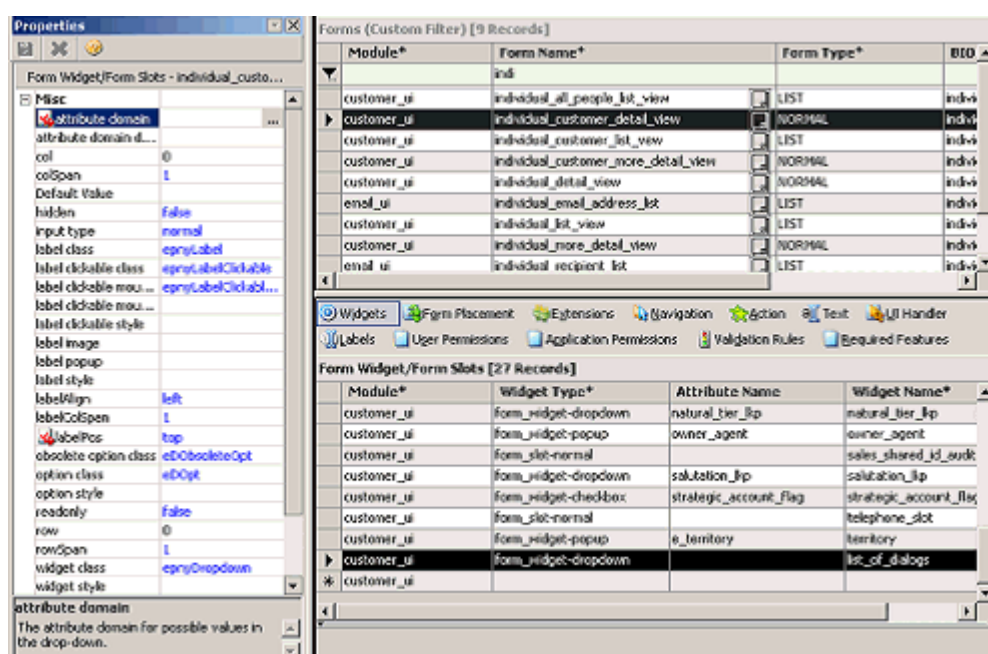
The screenshot shows the Infor Epiphany interface after adding a dropdown widget. The **Properties** pane on the left now shows the **Attribute Domain** property set to **customer\_ui**. The **Forms (Custom Filter) [9 Records]** table is the same as in the previous screenshot.

Module*	Form Name*	Form Type*	BIO
customer_ui	individual_all_people_list_view	LIST	indiv
customer_ui	individual_customer_detail_view	NORMAL	indiv
customer_ui	individual_customer_list_view	LIST	indiv
customer_ui	individual_customer_more_detail_view	NORMAL	indiv
customer_ui	individual_detail_view	NORMAL	indiv
email_ui	individual_email_address_list	LIST	indiv
customer_ui	individual_list_view	LIST	indiv
customer_ui	individual_more_detail_view	NORMAL	indiv
email_ui	individual_recipient_list	LIST	indiv

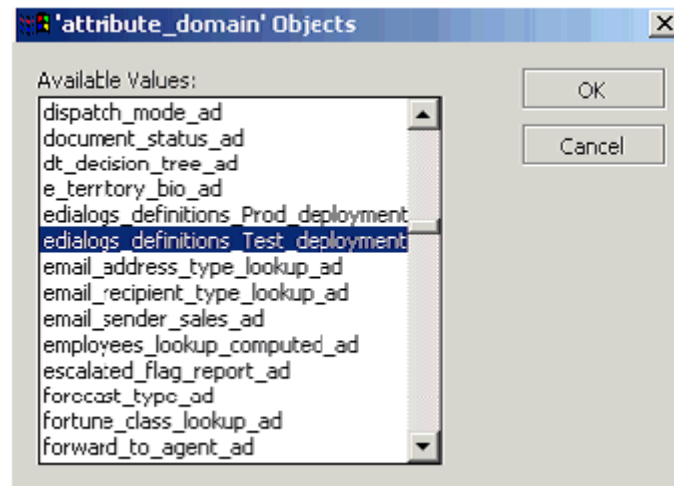
Below the Forms table is the **Form Widget/Form Slots [27 Records]** table.

Module*	Widget Type*	Attribute Name	Widget Name*
customer_ui	form_widget-dropdown	natural_tier_flag	natural_tier_flag
customer_ui	form_widget-popup	owner_agent	owner_agent
customer_ui	form_slot-normal	sales_shared_id_audit	sales_shared_id_audit
customer_ui	form_widget-dropdown	salutation_flag	salutation_flag
customer_ui	form_widget-checkbox	strategic_account_flag	strategic_account_flag
customer_ui	form_slot-normal	telephone_slot	telephone_slot
customer_ui	form_widget-popup	territory	territory
customer_ui	form_widget-dropdown	list_of_dialogs	list_of_dialogs

- a For the dropdown to be populated with dialog definitions, an attribute domain must be assigned in the Properties pane by setting the **Attribute Domain** property.

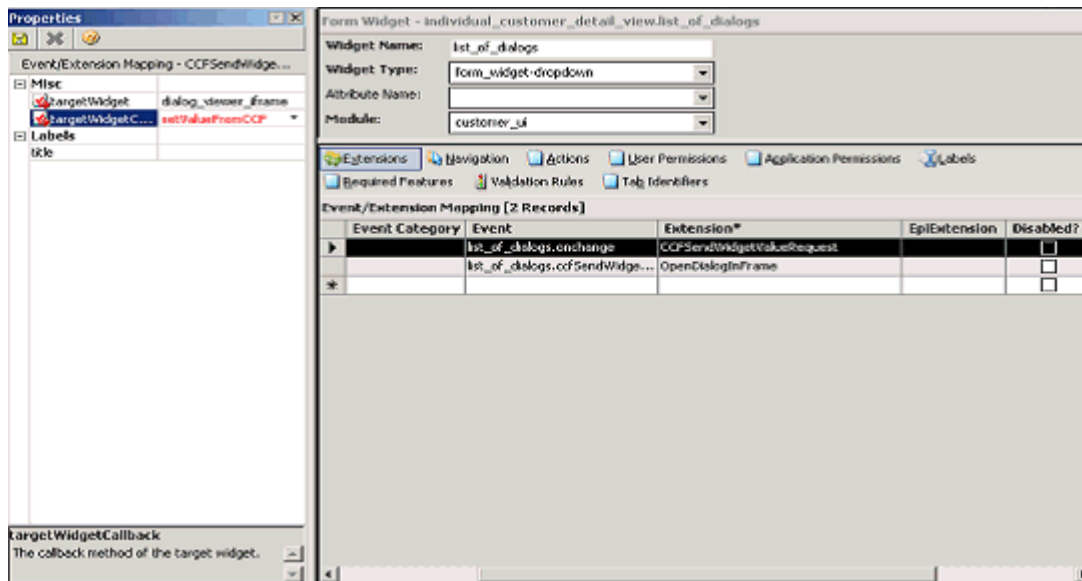


By default, two different attribute domains are available, **edialogs\_definitions\_Prod\_deployment** for all dialogs with Production deployment and **edialogs\_definitions\_Test\_deployment** for all dialogs with Test deployment. Select one and click OK. You can create and use additional customized attribute domains to filter the dialog names.



- b Double-click the **dropdown** widget to assign extensions to this widget.

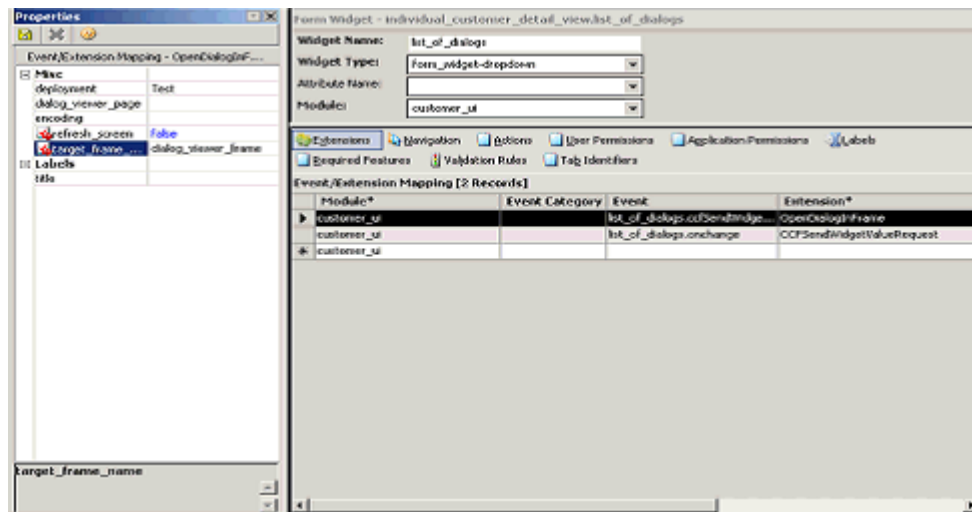
The first extension is for the **onchange** event on the **dropdown** widget. The extension for this is **CCFSendWidgetValueRequest**.



In the Properties pane, the **targetWidget** is the widget name of the frame as defined, and **targetWidgetCallback** is set to **setValueFromCCF**.

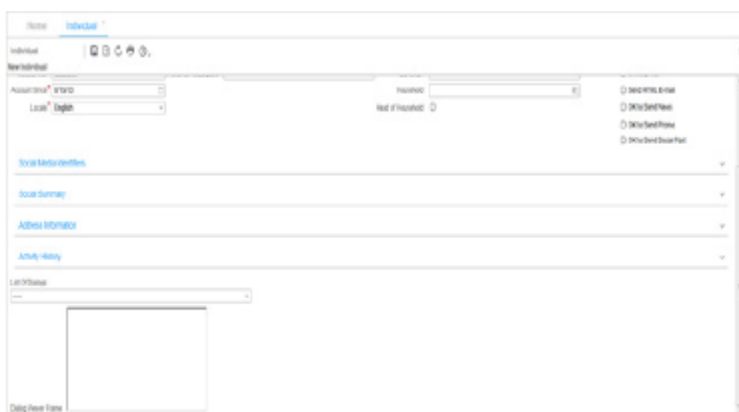
- c Add event **ccfSendWidgetValue** and assign extension **OpenDialogInFrame**

The extension properties with usage details are listed in the next section.



- 7 Add the **dropdown** and **iframe** widgets to the form.

After logging in to the Sales and Service application, the screen appears as follows:



8 Pick a dialog from the **dropdown** and it automatically displays the dialog within the **iframe** as follows:



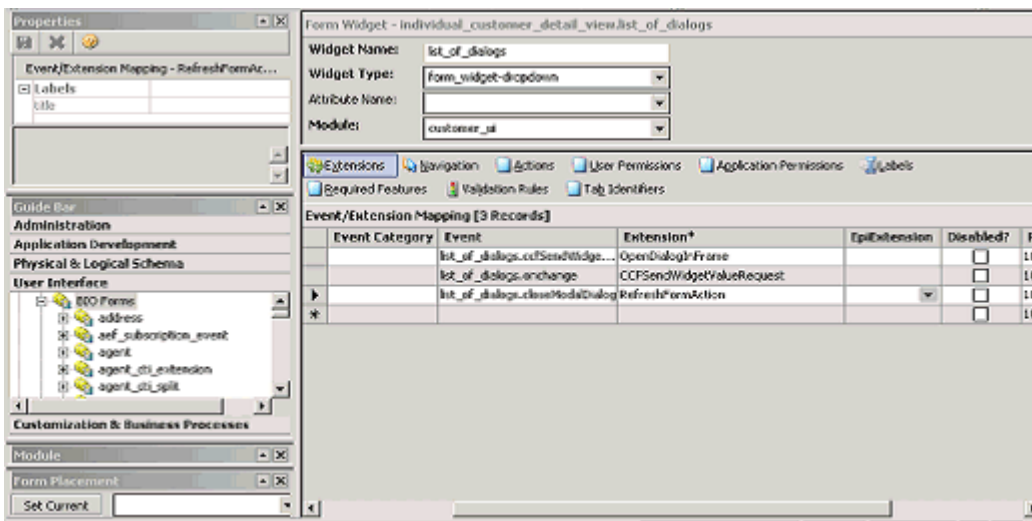
Property Name	Description
deployment	Value should be Test or Production . If none is provided, it is derived from the global user preference ref-code-Dialog-Depl-Domain.
dialog_viewer_page	The default JSP page invoked is InvokeDialog.jsp if none is provided here. Relative paths can be entered here, but not absolute paths.
encoding	Specifies the encoding of dialog names (if non-ASCII) so the viewer can decode them properly.
refresh_screen	If set to True , it enables automatic refresh of application screen at the conclusion of the dialog. It requires additional configuration on the widget—refer the section on Refresh for more details.
target_frame_name	This is the name of the iframe widget.

## Refresh

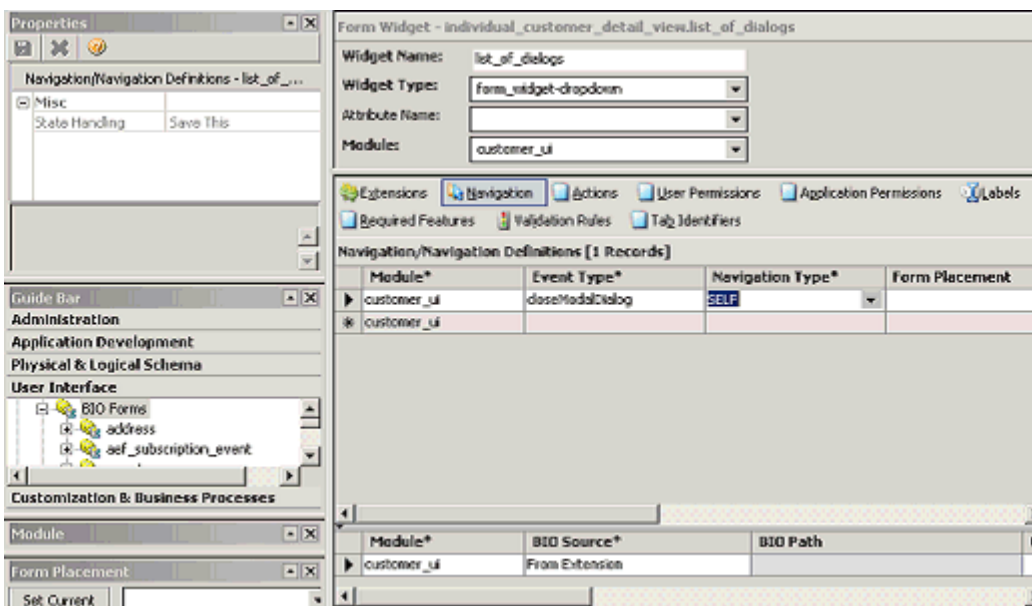
To add the capability to refresh the application screen upon successful completion of the dialog, before the form's JSP is made, do the following:

1 Do one of the following:

- For viewing a dialog in an iframe, add extension **RefreshFormAction** to the **dropdown** widget for **closeModalDialog** event.
- For viewing a dialog in a separate window, add extension **RefreshFormAction** to the **button** widget to **closeModalDialog** event.



2 Add **navigation** to **SELF** for the **closeModalDialog** event using the **Navigation** tab.



3 Place the widgets (if not already done) on the form.



## OpenDialogInWindow

Use the `OpenDialogInWindow` extension to view a dialog in a separate browser window. This example describes how you can enable dialog viewing from the **individual\_customer\_detail\_view** form. To enable, add a **dropdown** widget (with a list of available dialog definitions) and a **button** widget to launch the browser window with the dialog.

- 1 In Infor Studio, navigate to the **individual\_customer\_detail\_view** form (User Interface > Forms > BIO Forms > individual\_customer > individual\_customer\_detail\_view).
- 2 Add a new widget of type **dropdown** and provide a widget name as in the following screen and save it.

You can modify the widget label to any text in the upper left hand corner Properties pane.

The screenshot shows the Infor Studio interface. On the left, the Properties pane is open for a 'Form Widget/Form Slots - individual\_custo...'. The 'attribute domain' is set to 'edialogs\_definitio...'. Below it, the 'attribute domain' is described as 'The attribute domain for possible values in the drop-down.' The 'Guide Bar' on the left shows the 'User Interface' section expanded, with 'BIO Forms' > 'address' > 'eef\_subscription\_event' > 'agent' selected. The main area displays a table titled 'Forms (Custom Filter) [9 Records]'.

Module*	Form Name*	Form Type*	BIO Class
customer_ui	individual_customer_detail_view	NORMAL	individual_cus
customer_ui	individual_customer_list_view	LIST	individual_cus
customer_ui	individual_customer_more_detail_view	NORMAL	individual_cus
customer_ui	individual_detail_view	NORMAL	individual
email_ui	individual_email_address_list	LIST	individual
customer_ui	individual_list_view	LIST	individual
customer_ui	individual_more_detail_view	NORMAL	individual
email_ui	individual_recipient_list	LIST	individual
customer_ui			

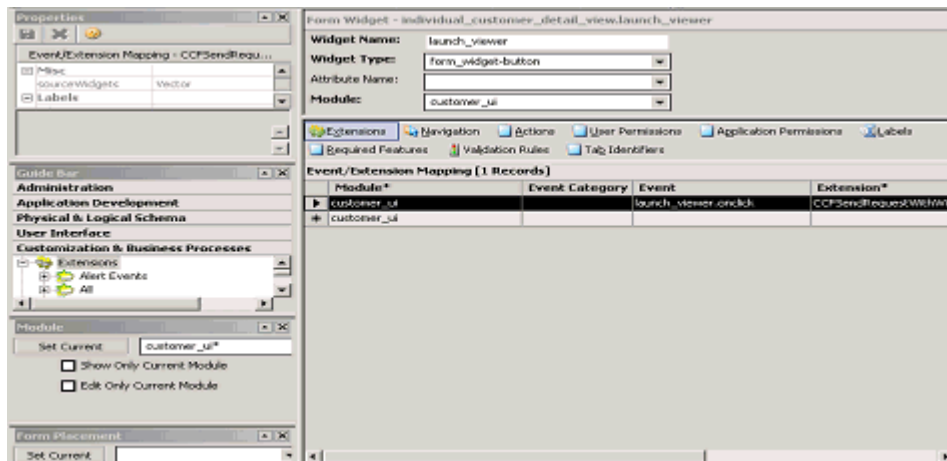
Below the table, the 'Form Widget/Form Slots [28 Records]' table is also visible, showing various widget types and their names.

- 3 In the Properties pane, set the attribute domain for the **dropdown** widget. There are no extensions assigned to this widget.

The screenshot shows a dialog box titled "'attribute\_domain' Objects". It contains a list of 'Available Values:' with the following items: data\_type\_lookup\_ad, determined\_priority\_lookup\_ad, direction\_lookup\_ad, dispatch\_mode\_ad, edialogs\_definitions Prod deployment, edialogs\_definitions Test deployment, email\_address\_type\_lookup\_ad, email\_recipient\_type\_lookup\_ad, escalated\_flag\_report\_ad, fortune\_class\_lookup\_ad, gender\_lookup\_ad, generation\_lookup\_ad, generic\_priority\_lookup\_ad, and generic\_status\_justify\_lookup\_ad. The 'edialogs\_definitions Test deployment' item is selected. There are 'OK' and 'Cancel' buttons on the right.

- 4 Add a **button** widget to the form.
- 5 Double-click the **button** widget to assign extensions to events.

**a Onclick event:**

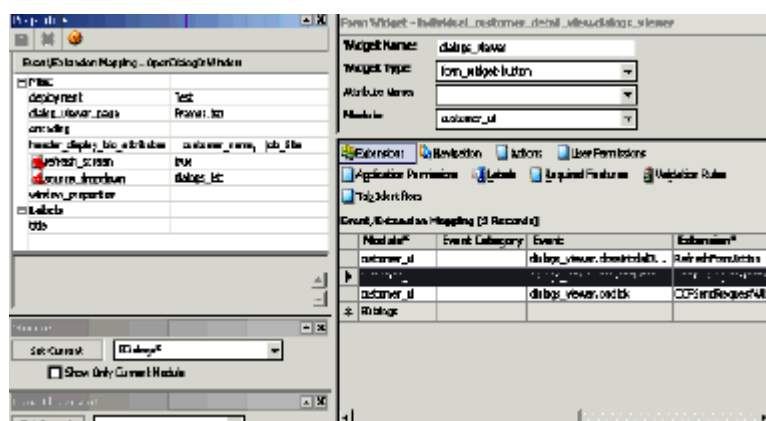


In the Properties pane, click on **sourceWidgets** and pick the **dropdown** widget that was defined previously. Save and close the popup window, then save the properties.

Vector Values (Custom Filter) [ 1 Records ]				
	Module*	Object Value	Object Val	List Order*
...				
	customer_ui	mer_detail_view.dialogs_list_dropdown	form_widget	
*	customer_ui	individual_customer_detail_view.dialogs_list_dropdown		
		individual_customer_detail_view.date_of_birth		
		individual_customer_detail_view.date_entered		
		individual_customer_detail_view.customer_category_lkp		
		individual_customer_detail_view.salutation_lkp		
		individual_customer_detail_view.strategic_account_flag		
		individual_customer_detail_view.owner_agent		
		individual_customer_detail_view.territory		
		individual_customer_detail_view Toolbar.new_button		
		individual_customer_detail_view Toolbar.individual_customer_detail_view Toolbar		
		individual_customer_detail_view Toolbar.actions_menu		
		individual_customer_detail_view Toolbar.save_button		

**b CcfSendData event:**

The extension is **OpenDialogInWindow**



This extension has properties that you have to set appropriately

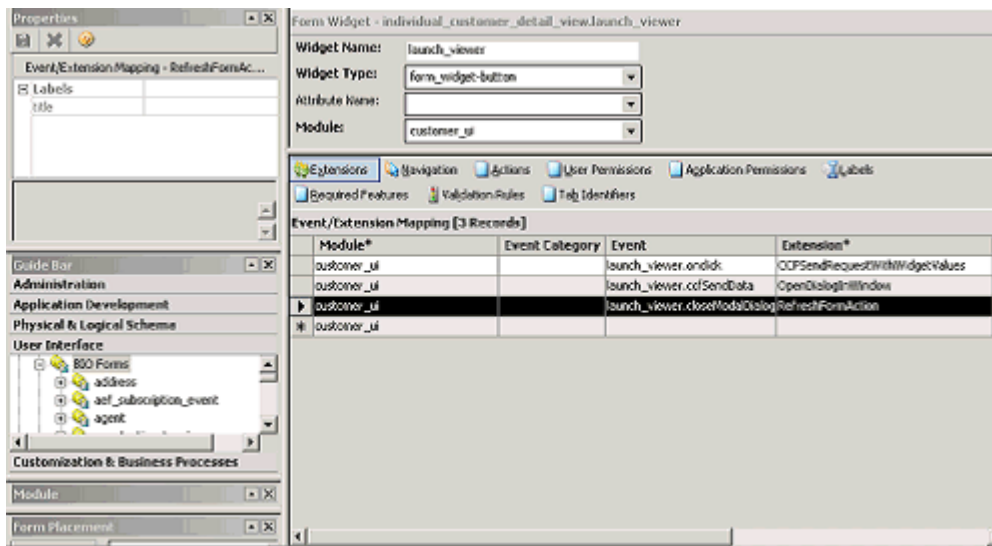
Property Name	Description
deployment	Value should be Test or Production . If none provided, it is derived from the global user preference ref-code-Dialog-Depl-Domain
dialog_viewer_page	The default JSP page invoked is InvokeDialog.jsp if none is entered here. Relative paths can be entered, but not absolute paths.
encoding	Specifies the encoding of dialog names (if non-ASCII) so the viewer can decode them properly
refresh_screen	If set to true, it enables automatic refresh of application screen at the conclusion of the dialog. It however requires additional configuration on the widget—refer the section on Refresh for more details.
source_dropdown	This is the name of the dropdown widget that is the source of the dialog name(s).
window_properties	<p>Properties that can be set on the Dialog Viewer window. For example, status=yes, resizable=yes, toolbar=no, location=no.</p> <p>Status specifies a priority or transient message in the window's status bar. Setting resizable=yes allows you to resize the window. Toolbar represents the browser window's toolbar. If you set toolbar = yes, a standard browser toolbar with <b>Back</b> and <b>Forward</b> buttons is created. Location contains information on the current URL. If you set location=yes, a URL entry field is created. For more information on properties, see Netscape documentation.</p>

- 6 After defining this extension, save all work.
- 7 Place the defined widgets on the **individual\_customer\_detail\_view** form.
- 8 To view the dialog, pick a dialog from the dropdown on the **individual\_customer\_detail** form in the application, and then click the button. This launches a separate browser window with the dialog display.

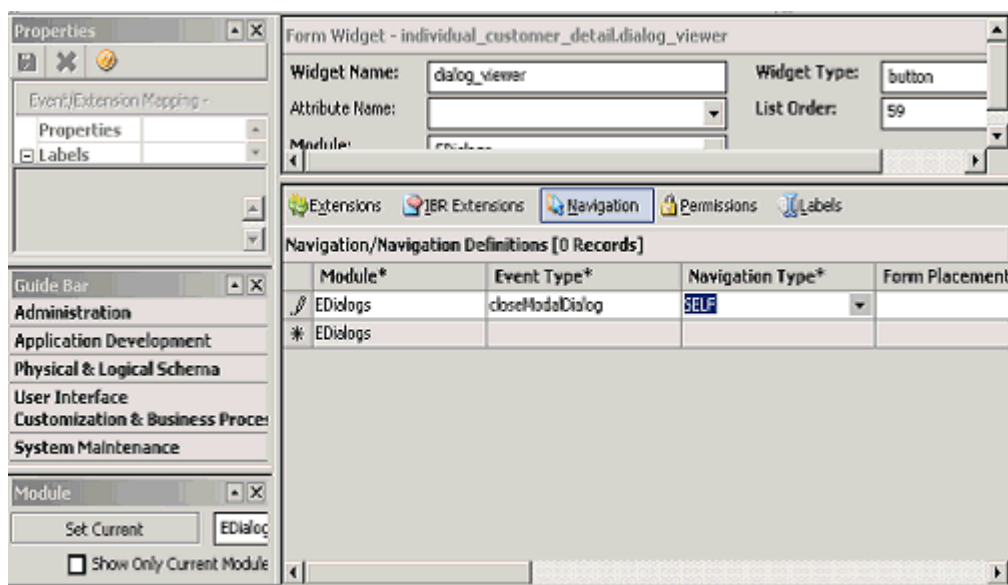
## Refresh

To add the capability to refresh the application screen upon successful completion of the dialog, perform the following steps before the form's JSP is made:

- 1 Add the **RefreshFormAction** extension to the **button** widget for the **closeModalDialog** event.



- 2 Add **navigation** to **SELF** for the **closeModalDialog** event using the **Navigation** tab.



3 Place the widgets (if not already done) on the form.

## OpenDefaultDialogInWindow

The **OpenDefaultDialogInWindow** extension is used to view a specific dialog chosen at form-design time, in a separate window. The procedure is same as **OpenDialogInWindow**, except for the following requirements:

- **Dropdown** widget is not required—only a button to invoke the dialog.
- **OnClick** event property **sourceWidgets** should be set to any widget on the same form.
- **CcfSendData** event should be handled by the **OpenDefaultDialogInWindow** extension.

Set the Property dialog to the default dialog name for extension **OpenDefaultDialogInWindow**.

## Launching Dialogs from a Sub Form (on an Iframe)

Use the following steps to launch a Dialog from a sub form off a detail view.

- 1 Define a detail form that will be used as a container to display the dialog. In Studio, navigate to User Interface > BIO Forms and create a new form. Give the form a name, set the form type to be Normal, and the BIO Class to be the same as the BIO Class on the form under which your dialog will appear as a sub form. In the following example, the Bio class is set to e\_lead to launch the Dialog from the lead detail screen as a sub form. After Studio creates the widgets and the toolbar, select 'detail view' for perspective. When you see “The combination of fields selected is not unique (BIO Class, Perspective, Special Form Type)” message, ignore the message by clicking OK. After the wizard finishes running, your final screen should appear as follows..



lead_ui	lead_dialogs	NORMAL	e_lead	0
* EDM				0

Widgets

Form Placement

Extensions

Navigation

Action

Text

UI Handler

Labels

User Permissions

Application Permissions

Validation Rules

Required Features

**Form Widget/Form Slots [50 Records]**

Module*	Widget Type*	Attribute Name	Widget Name*
lead_ui	form_widget-text	address	address
lead_ui	form_widget-text	alignment_id	alignment_id
lead_ui	form_widget-text	assigned_by_user_id	assigned_by_user_id
lead_ui	form_widget-text	audit	audit
lead_ui	form_widget-text	churn_score	churn_score
lead_ui	form_widget-text	cid	cid
lead_ui	form_widget-text	city	city
lead_ui	form_widget-link	contact_full_name	contact_full_name
lead_ui	form_widget-text	credit_score	credit_score
lead_ui	form_widget-text	credit_score	credit_score

- To make the form ready to display the dialog, remove all unnecessary widgets from the form. This example assumes that the only thing you want to display on this form is the Dialog along with the drop-down that allows you to select a dialog from the list. The only widgets that should be on the form are a Drop- down widget for selecting the Dialogs and an IFrame to display the Dialogs.

Form Name:

lead\_dialogs\_view

BIO Class:

e\_lead

Form Type:

NORMAL

Attributes:

Module:

EDM

Widgets

Form Placement

Extensions

Navigation

Action

Text

UI Handler

Labels

User Permissions

Application Permissions

Validation Rules

Required Features

**Form Widget/Form Slots [3 Records]**

Module*	Widget Type*	Attribute Name	Widget Name*
EDM	form_slot-toolbar		lead_dialogs_view Toolbar
EDM	form_widget-dropdown		select_dialog
EDM	form_widget-iframe		view_dialog
* EDM			

- 3 Set the toolbar as required and create a new Perspective for your new form (**User Interface > Maintenance > Perspective** ).

**Perspectives (Custom Filter) [1 Records]**

Module*	Name*	Attributes
...		
lead_dialogs	leads_dialogs_perspective	
* EDM		

- 4 Define the UI Handler for the form. Set the correct BIO Class and the Perspective that you created in Step 3.

Form Name: lead\_dialogs\_view

BIO Class: e\_lead

Form Type: NORMAL

Attributes:

Module: EDM

Widgets Form Placement Extensions Navigation Action Text UI Handler Labels

User Permissions Application Permissions Validation Rules Required Features

**UI Handlers [1 Records]**

Module*	BIO Class	Perspective	Special Form Type
EDM	e_lead	leads_dialogs_perspective	

- 5 Finally, reference this form in the Sales\_TabGroup\_Shell form. You have to define a new tab that will hold this new form. On a blank row, create the required metadata with the following values.
- Name: Use the convention from previous entries and set up a name.
- BIO Path: Will be the BIO used in your form.
- Form Placement: Sales\_TabGroup\_Shell\_"your bio"\_Bio\_Sales\_Detail\_List\_Shell
- Tab Identifier: This will be the position you want the tab to appear. Be careful when you define this because there will most likely be other sub forms that are already set up in different slots. Manually select the correct tab number.
- Content : This will be the perspective you set up in step 3.
- Title: On the properties section for this row, give your-sub form an appropriate title.

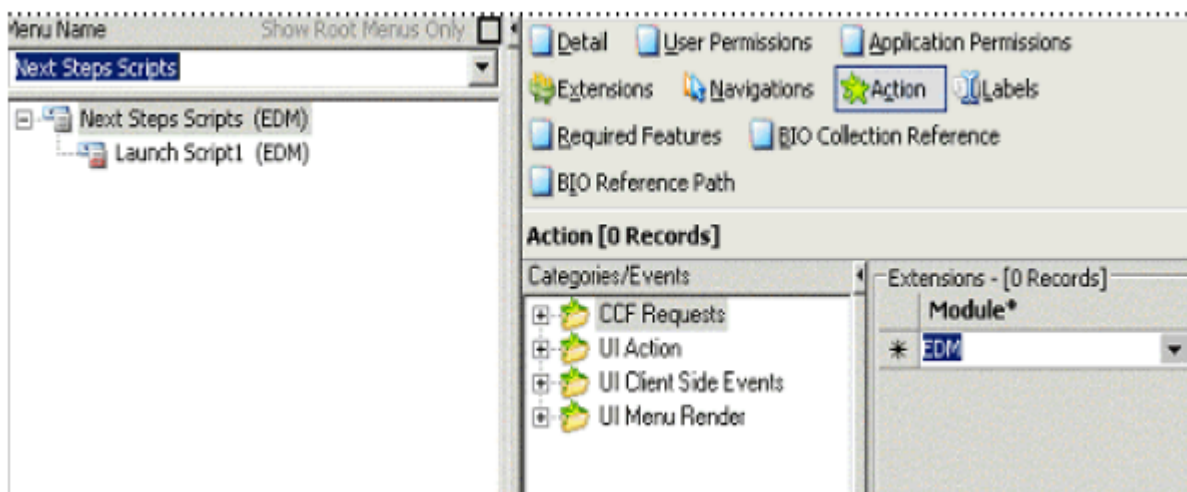




## Launching Dialogs from a Toolbar (launching Default Dialog on a Separate Window)

To launch a Dialog from a toolbar, do the following:

- 1 Create a new Menu. This menu will be used to set up the Dialogs that will be launched. Add as many menu items as needed. Each menu item is used to launch a specific dialog. In this example, one dialog is set up. You can add as many as you need. After you set up your menu items, your menu should appear as follows:



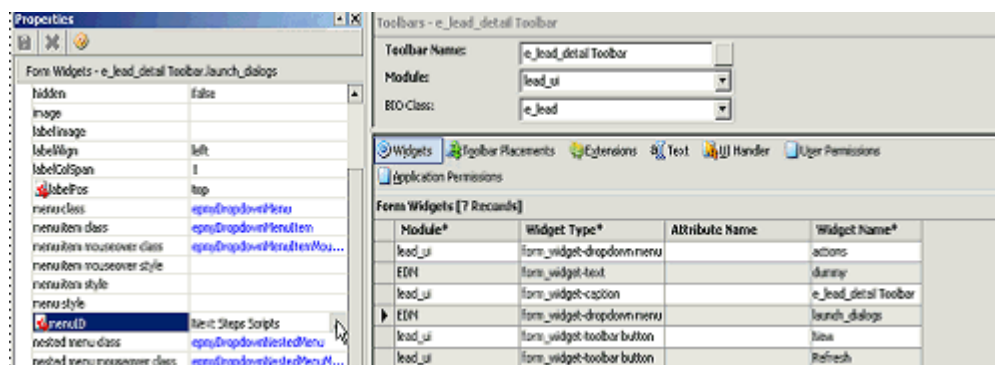
- 2 Highlight the first menu item and click on the **Action** tab. Set up the CCF Request and the UI Client Side Action events. Use the following settings:

CCFRequest- CCFSendData - Extensions- Select **OpenDefaultDialogInWindow** from the list of available extensions. Set the Deployment, Dialog, and optionally the dialog\_viewer\_page.

UI Client Side events - onclick -Extensions - Select **CCFSendRequestWithWidgetValues** from the available list.

Detail - Menu Attributes - Check the **Root Menu** checkbox.

- 3 Open the toolbar from step 1. For the drop-down menu, set the menu id to be the menu you created in the previous steps.



- 4 If everything is deployed correctly, you should see your drop-down widget on the toolbar and from there you should be able to launch your dialog.

## IBR and EDialogs

In EDialogs, like in Workflow, you can use IBR to control which transitions are taken. However, the decision is based on responses to questions that are asked at runtime. Thus, if two agents answer the same question differently, the next question they are asked might be different. The EDialogs developer also has access to variables when writing an IBR Expression.

Possible EDialogs IBR applications include

- If the user answered “a” to question 1 and “a” to question 5, then skip question 6.
- If the RealTime service returns a high score for this user, only show the last three question screens.

## Best Practices

See "Dialogs" on page 42

This chapter describes the Infor Open Architecture Replication service.

## Overview

The Replication Service is used to replicate data between two sets of BIO types. It is typically used to replicate one BIO from an externalized data source using EAI and another BIO that is mapped directly to a SQL data source. The Replication Service is a service framework service with its own set of APIs that can be invoked using via the Service Locator. Typically, the Replication Service is invoked as a scheduled job in the Admin Console. The Replication Service supports two modes of replication: Initial Replication and Batch Replication.

Each scheduled job requires the following configuration settings depending on the job type:

Setting	Function
Job Name	A name used to identify the job.
Type of Job	Either Initial Replication or Batch Replication
Bio Transformation Map	The Bio Transformation Map to use to replicate between the two BIO types.
Source Query	Query expression string to be used on the source BIO type. Only instances resulting from this query will be participating in replication.
Target Query (batch only)	Query expression string to be used on the target BIO type. Only instances resulting from this query will be participating in replication.
Direction (batch only)	Set to “BiDirectional” to indicate that replication will be performed from source-to-target, and from target-to-source.  Set to “UniDirectional” to indicate that replication will be performed from source-to-target only.

Setting	Function
Timeout	A value in seconds that should be set to the maximum possible time this job can run before it is considered to have failed.

## Mapping BIO types

The Replication Service uses the Bio Transformation Service to map schema differences between the two BIOs. The transformation map must be pre-defined in Studio before the Replication Service can perform the replication. A map defines a “source” BIO type and a “target” BIO type. In practice, a source BIO type will refer to any BIO that has been externalized and a target BIO will be one that is SQL based.

Related BIO types can also be mapped and will participate in the transformation. See Bio Transformation Service documentation on how to map related BIOs.

## Initial Replication

Initial Replication is used to read all the data from the externalized BIO data source that conforms to the source query criteria and insert the transformed SQL data source BIOs. Initial Replication can only be run once and must be run on a SQL data source that initially is empty for that BIO type. No BIOs will be changed/modified on the source data source.

The screenshot shows a web-based configuration interface for the Replication Service. It consists of two main sections, each with a title bar and a help icon. The first section, titled 'Create/Edit Job', contains two fields: 'Job Name' with the value 'Initial Job' and 'Type of Job' with a dropdown menu showing 'InitialReplication'. The second section, titled 'Initial Replication', contains three fields: 'Bio Transformation Map' with a dropdown menu showing 'individual\_customer\_mapping', 'Source Query' with a dropdown menu showing 'individual\_customers', and 'Timeout' with a text input field containing the value '20'.

## Batch Replication

Batch Replication is used to incrementally replicate between two sets of BIOs after Initial Replication has been run. Batch Replication cannot run unless Initial Replication completes for a scheduled job.

Batch Replication supports bi-directional replication of inserts or updates from one set of BIOs to the other set of BIOs. It has been optimized to support the “source BIO” being one of an external Data Source and a “target BIO” to be of a SQL Data Source. Unlike an Initial Replication job, which can take hours to complete, Batch Replication jobs are designed to be completed in minutes. It achieves this by only querying for inserts/updates from the source BIO data source since the last time Replication Service was run successfully and will only replicate changes to the target BIO data source. Then, in the reverse direction, it queries for inserts/updates from the target BIO data source since the last successful Replication and applies those changes back to the source BIO data source. See process flow diagram for more details.

For Batch Replication support, each BIO instance must define a persisted attribute with name “date\_last\_synched\_or\_updt” and of type “date” that will hold the last change date of that instance. It is up to the application to maintain this field whenever the instance of that BIO or related BIO changes to update this attribute with the current date/time from the server. The Replication Service does not support replicating deleted BIOs at this time. Any deletions, whether physical or logical, will need to be made on the source and target BIOs manually.



The screenshot shows a web form titled "Create/Edit Job". It has a "Job Name" field with the value "Batch Job" and a "Type of Job" dropdown menu with "BatchReplication" selected. Below this is a section titled "Batch Replication" which contains a "Bio Transformation Map" dropdown with "individual\_customer\_mapping" selected, a "Source Query" dropdown with "contacts" selected, a "Timeout" field with the value "10", a "Direction" dropdown with "Bidirectional" selected, and a "Target Query" dropdown with "individual\_customers" selected. The form has a clean, modern design with light blue accents and red asterisks indicating required fields.

## Deletions

The Replication Service does not support replicating deleted BIOs at this time. Any deletions, whether physical or logical, will need to be made on the source and target BIOs manually.

## Mapping Keys

The Replication Service will need to map the primary key of a source and target BIO such that it can determine the existence of the entity on the other system and be able to fetch it on demand. Key mapping requires that the target BIO (SQL Data Source) have an alternate key field to hold the source BIO’s primary key. The alternate key field is then applied to the Bio Transformation Service map. The Replication Service will be responsible for populating values to this field.

## Replication State BIO

The following BIO definition is used internally by the Replication Service. There will be one and only one record in this table for each Replication Job defined in the job scheduler.

BIO name: *e\_replication\_state*

Attribute Name	Data Type	Description
e_replication_state_id	Binary(16)	Primary key
bio_xform_map_id	Binary(16)	ID of the BIO transformation map ( <i>UNIQUE</i> )
lock_time	DATE	Time of record locking (indicating that either Batch or Initial replication job is in progress)
last_sync_date	DATE	Last time when the databases were queried for modifications. Can be NULL if the Initial job is still in progress.

## Transaction Handling

Both of the extreme approaches: replicating all objects in the same transaction, or using a separate transaction for every object, may not be always appropriate. Performing an initial replication would be especially problematic in both cases, because of a large number of objects involved in the process.

To achieve better reliability and optimization, the replication service will replicate objects in small groups, using one transaction for a group of 20-50 objects, depending on the total number of objects selected for the replication. If a group transaction fails to commit (with a soft error), the replication service will immediately try to replicate this group of objects again, using separate transaction for each object this time, and trying to isolate and report the error(s). The size of the initial replication transaction is configured in Studio as a property of the Replication Service called "InitialLoadBatchSize". It has a default value of 64.

## Error Handling

The Replication Service recognizes two general classes of errors:

**Soft Error** – an error indicating that the replication of the current object cannot be completed, but the reason is not severe enough to cancel the entire process (if this is a batch or initial replication). Therefore, the error will be logged, but the execution will continue.

**Hard Error** – an error, indicating that a serious failure occurred, and the system is not stable enough to continue the process. This may be an intermittent failure, or something that will require the restart of the server. In any case, the batch/initial replication will be aborted, and the error will be logged.

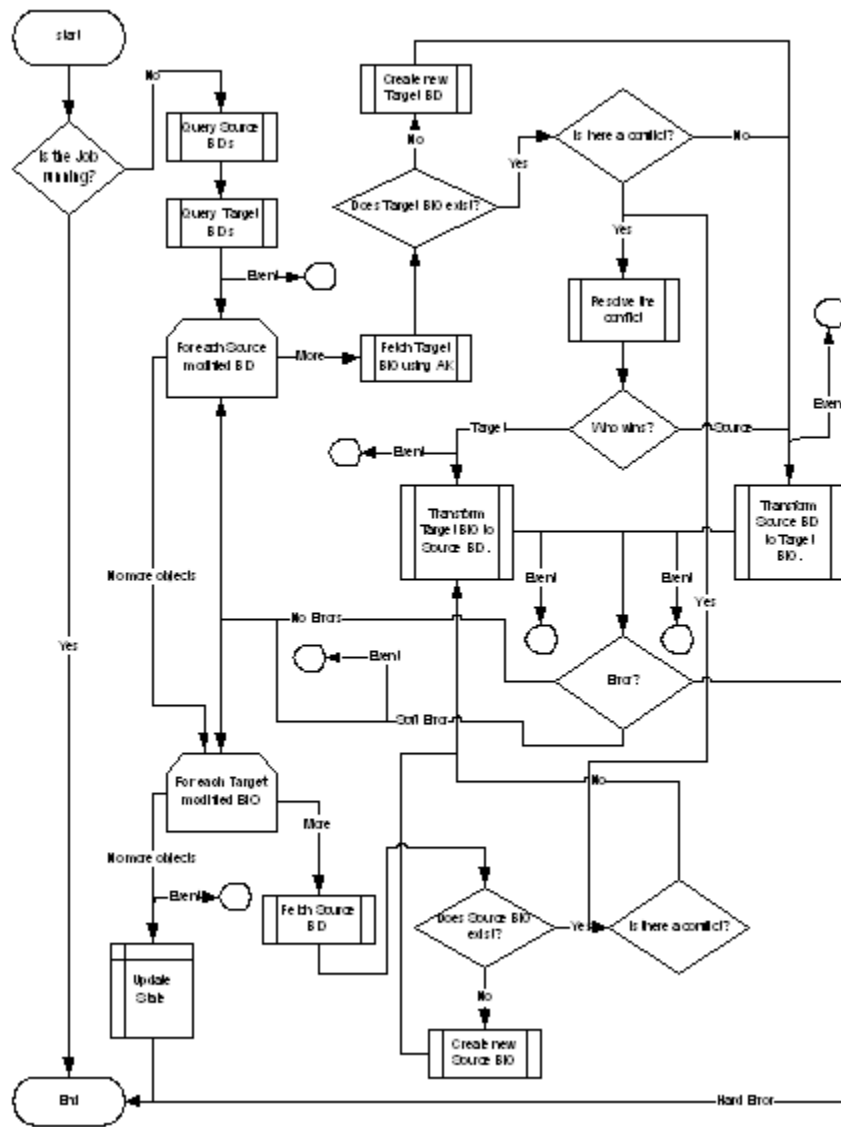
The following table classifies the most common exceptions:

### Error Messages

Exception	Severity	Comment
I/O exceptions, Socket exceptions	HARD	Connectivity error usually means a serious problem
Null Pointer Exception	HARD	Most likely, a serious error that should be urgently fixed
Optimistic lock failures	SOFT	Concurrent access to one record should not prevent the replication service to work with other records
EAI exceptions	SOFT	A data specific error most likely on the source BIO.
Unknown (Default)	SOFT	If the service cannot recognize the exception, let's assume that it is a soft error, and try to continue.

The error message will contain as much information as possible about the exception and the particular replication service activity that was the cause of this error. All errors are logged to the `epny.log4j` log file on the server which was running the job. You cannot access these logs from the admin console, however, you can determine if a replication job has succeeded or failed from the history form of each job.

## Replication State Process Flow Diagram



## Replication Events

The event category Replication Events contains the following events:

**EXTERNAL\_TIME\_REQUEST\_EVENT** – will be fired just before the filtering queries are executed. It is a responsibility of an extension to calculate the clock skew, and store it in the output parameter. The following parameters are passed with this event as an array:

[0]	String	ID of the Bio Transformation map
-----	--------	----------------------------------



[1]	ClockSkew	Output parameter. Provides the placeholder for the clock skew calculated by extensions.
-----	-----------	---

The `ClockSkew` class represents the time difference between the source and target data source systems in milliseconds. Positive value of clock skew means that Source BIO data source time is ahead of the CRB server time, and vice versa.

The class implements the following methods:

```
public void setClockSkew (long clockSkew);
```

```
public long getClockSkew ();
```

**BEFORE\_REPLICATION\_EVENT** – will be fired after the filtering queries have been executed, but before the replication process is started. The following parameters are passed with this event as an array:

[0]	String	ID of the Bio Transformation map
[1]	String	Direction of the transformation

**AFTER\_REPLICATION\_EVENT** – will be fired after the replication process has been completed. The following parameters are passed with this event as an array:

[0]	String	ID of the Bio Transformation map
[1]	Integer	Duration of process (in milliseconds)
[2]	List of BioRefs	Objects modified on the source side during the replication.
[3]	List of BioRefs	Objects modified on the target side during the replication.

**ON\_REPLICATION\_ERROR\_EVENT** – will be fired if an error occurs during the replication and prevented the replicated object from saving. The following parameters are passed with this event as an array:

[0]	String	ID of the Bio Transformation map
[1]	EpiException	Exception class, containing the error
[2]	Bio	Source BIO
[3]	Bio	Target BIO

**ON\_CONFLICT\_RESOLUTION\_EVENT** – will be fired when a conflict should be resolved. The extension should return `RET_SOURCE` if the source object wins or `RET_TARGET` if the target object wins. The following parameters are passed with this event as an array:

[0]	String	ID of the Bio Transformation map
-----	--------	----------------------------------

[1]	Bio	Source BIO
[2]	Bio	Replicated Target BIO

The `ReplicationExtensionBase` class will provide basic implementation of event handlers to facilitate easier creation of CRB extensions. The base class will define the following methods to be overloaded by the extension classes (each method matches an event with its parameters).

- `int beforeReplication (String btMapId, String direction);`
- `int externalTimeRequest (String btMapId, ClockSkew result);`
- `int afterReplication (String btMapId, Integer duration, List sourceRefs, List targetRefs);`
- `int onReplicationError (String btMapId, EpiException exception, Bio sourceObj, Bio targetObj);`
- `int onConflictResolutionEvent (String btMapId, Bio sourceObj, Bio targetObj);`

This chapter covers how Access Control Lists are implemented in Infor Epiphany Sales and offers implementation guidelines and best practices.

## Overview

Permissions control what operations a user can perform within Infor applications. You can apply permissions on BIOs, BIO attributes, forms, form widgets, toolbars, menus, menu-items and query expressions. On all of these objects, you can apply permissions at the application level and at the user level.

Permissions can be applied at the application level where the application user has access/does not have access to objects based on the application that the user is logged into. The list of applications and their respective URLs can be found in Infor Studio (**Guide Bar > Application Development > Applications** ). You can set application Permissions from the **Application Permissions** tab on all of the above mentioned objects.

Permissions can be applied at a per-user level where the application user has access/does not have access to objects based on the user profile the user belongs to. User profiles are mapped to user roles which in turn are mapped to the SSO user role. You can set user permissions from the **User Permissions** tab on all of the above mentioned objects.

Permissions are additive, that is, if a user belongs to any of the SSO roles which map to a user role which in turn maps to a user profile that has access to the object, then the user has access to the object irrespective of the fact that all the other profiles mapped to his/her role do not have access to the object.

In addition to application and user permissions, BIOs can have additional permissions applied on a per instance basis.

## BIO Row Permissions and Access Control Lists

BIO row level permissions control who has permission to perform what operation on a specific instance of a BIO. This gives you the capability to define granular permission control at every instance of a particular BIO type.

The Access Control List (ACL) feature leverages the row level permissions infrastructure to define which user has what permission on a given instance of a BIO. To enforce ACL in the application, you have to define BIO row permissions using ACL definitions. A given user profile or an application may be exempted from access control checks.

Explicit ACL also provides the ability to define inheritance of ACL to other related BIO instances. For example, you can define that everyone who has permission to view/edit a customer can also view/edit all the related information, such as address and telephone, without having to explicitly create ACL entries to track these permissions.

User permissions (Static permissions) are applied on a per BIO basis, so they apply to all the instances of the BIO, as opposed to Row level permissions which are applied on a per BIO instance basis. Using the user permissions feature, you can implement the functionality where, only users with profile of "Cleared Security" can view financial records of a customer (assuming financial records are represented as a BIO in the system). User permissions do not provide any infrastructure to define explicit permission granting mechanism to specific users of the system on specific instances of BIO rows. For example, if only user "Bob Smith" can see details of all customers belonging to a certain territory, BIO row permissions is the mechanism to implement this logic.

The above use cases are very common when you implement Infor Epiphany Sales. To facilitate implementation of explicit permission granting model to instances of a BIO, the Infor architecture provides Access Control Lists. This list is represented in the database as a table (e\_acl\_entry) and some of its salient attributes are

- BIO type this permission is for
- BIO instance ID (primary Key) of the BIO on which the permission is defined
- Agent ID of the user to whom this permission has been granted
- Type of permission that has been granted

With this infrastructure available, you can grant explicit permission to users on a specific BIO instance. You can do this by defining query expressions that can then be applied as BIO row permissions to implement ACLs. BIO row permissions can be applied for the READ, UPDATE, DELETE and SYNC operations.

### Types of Permissions in Infor Epiphany Sales

This section describes implementation of ACLs in the Infor Epiphany Sales application. The Infor Epiphany Sales application implements READ and SYNC permissions. These permissions are defined as BIO row permissions.

Permissions can be defined as named query expressions. You can create the named expression in Infor Studio (**Guide Bar > Physical and Logical Schema > Maintenance > Query Expressions** ). A named query expression is a re-usable query expression. Changes to the named expression will be immediately reflected on all BIO row permission filters where the named expression is used.

Infor Epiphany Sales uses the following query expressions to implement ACL permissions:

Important	These query expressions should never be modified in the field.
CheckSyncACL	Checks for the sync permission (include ACLs from the user's downline).
SyncVisibleACL	Checks for the read/update/delete permission (include ACLs from user's downline).
CheckSyncACL_noDownline	Same as CheckSyncACL except it does not include downline's ACL.
CheckVisibleACL_noDownline	Same as CheckVisibleACL except it does not include downline's ACL.

Query expressions can have a list of exemptions based on user profiles (for example, administrators and Infor Epiphany Service users are exempt from all ACLs out-of-the-box).

**Note:** All of the Sync permissions are used for Mobile Sync which is deprecated as of 7.0.4 FP1. The documentation is left here for informational purposes only, should you need to remove these during upgrading.

## Permission Inheritance

In addition to providing BIO Row Permissions, the Infor architecture also provides a mechanism to define permission inheritance from a BIO to its related BIOs. By using this feature you can implement functionality such as

- Anyone who has access to see details of a customer has access to all the opportunities and leads associated with the customer.
- Anyone who can see details of a customer can see all the notes and attachments for that customer without having to explicitly list every user note and every attachment for this customer in an access control list.

In the above example, both opportunities and leads have the Customer BIO as a parent. However they can be accessed directly from the Navigation menu, compared to user notes and attachments which always show up as a sub-form on the customer forms.

Two types of inheritance features are provided. The next section explains what they do and when to use them.

### Sync and View Permission Inheritance

There are a number of BIOs that need to be accessed directly without the context of the parent BIO. If the BIO you defined requires access to that BIO without the context of the parent, but still needs to inherit permissions from the parent BIO, you have to set the "Inherit Sync and View Permissions" flag on the rel-mapped BIO attribute. For example, the e\_opportunity BIO is accessed directly from the Navigation Bar, so the e\_opportunities RELATION mapped BioCollection attribute on the Customer BIO has the "Inherit Sync and View Permissions" checked.

When you use this inheritance, every time you access the BIO, the BIO layer automatically applies the permission check, where it looks for explicit permissions defined on BIO instances from which it inherits the "Inherit Sync and View Permission".

Do not use this property if your BIO is not accessed without the context of a parent BIO because this can introduce performance overhead.

### Sync Permission Inheritance

When the BIO you defined is always accessed in the context of its parent BIO (for example, notes, attachments, and so on), then use the "Inherit Sync Permission" setting. For example, the user\_notes BIO is always accessed as a sub-form of customer or some other parent, so the user\_notes RELATION mapped BioCollection attribute on the Customer BIO has the "Inherit Sync Permissions" checked.

When you set the property to the "Inherit Sync Permission" setting, the BIO layer uses this setting only when synchronizing the data to the mobile database.

**Note:** All of the Sync permissions are used for Mobile Sync which is deprecated as of 7.0.4 FP1. The documentation is present here for informational purposes only, should you need to remove these during upgrading.

## Objects Permissions in Out-of-the-box Infor Epiphany Sales

### Direct ACL Permissions

The following BIOs have ACLs defined out-of-the-box (**Infor Studio Guide Bar > Physical & Logical Schema > Maintenance > Query Expressions**). Select **CheckSyncACL** and click the **BIO Row Permissions** tab.

### Customers (Customer BIO)

Any user (and the user's upline) who is in the team for a customer has permissions to view/edit the details of the customer. Anyone who can view/edit the customer can also view/edit all the details of the customer, such as addresses, telephone numbers, e-mail addresses, and notes. Organization and Individual BIOs are subclasses of customer and automatically inherit ACLs from customer - their superclass. A team is associated with every instance of a BIO exposed through the **Team** tab on the Individual/Organization forms within the application. For example, assume that User1 and User2 can be on the team of Customer1 and User1 can belong to the team of Customer2 who may have User3 and User4 as team members.

### Lead (e\_lead BIO)

Any user (and the user's upline) who is in the team for a lead has permissions to view/edit the details of the lead. Anyone who can view/edit the details of a lead can also view/edit other details of the lead, such as products, contacts (related to the lead), and notes. In addition, the Lead BIO(mapped as e\_leads Bio Collection on the Customer BIO) inherits Sync and View Permission from the Customer.

## Opportunity (e\_opportunity BIO)

Any user (and the user's upline) who is in the team for an Opportunity has permissions to see the details of the Opportunity. Anyone who can view/edit details of an opportunity can also view/edit other details of an opportunity such as sales process, products, tasks, contacts, notes, and so on. In addition, the Opportunity BIO (mapped as e\_opportunities BIO Collection on the Customer BIO) inherits Sync and View Permission from the Customer.

## Interaction (Interaction BIO)

By default the creator of an interaction has permissions to view/edit the details of the interaction. The necessary ACLs for the creator are inserted by deploying the `ACLManagerExtension` on the Interaction BIO. Anyone who can view/edit the details of an interaction can also view/edit other details of the interaction, such as audit, tasks, and notes. In addition, the Interaction BIO (mapped as interactions BIO Collection on the Customer BIO) inherits Sync and View Permission from the Customer.

## Quota (e\_quota Bio)

By default the creator of a quota has permissions to view/edit the details of the quota. The necessary ACLs for the creator are inserted by deploying the `ACLManagerExtension` on the e\_quota BIO.

## Forecast (e\_forecast BIO)

By default the creator of a forecast has permissions to view/edit the details of the forecast. The necessary ACLs for the creator are inserted by deploying the `ACLManagerExtension` on the e\_forecast BIO.

## Appointment (e\_cal\_series BIO)

Anyone who is in the invitee list of an appointment can view/edit the details of the appointment.

## Contacts (individual BIO)

If a contact is associated with an Organization then anyone one who can view/edit the organization can view/edit all the contacts for this organization.

If the contact is not associated with any organization, then only the creator of the contact can view/edit the details of this contact.

## Inherited ACL Permissions

The following BIOs 'Inherit Sync Permissions' from their parent(s):

address, audit, e\_appointment\_invite, e\_opp\_competitor\_org, e\_opp\_partner\_org, e\_organization\_hier, e\_prod\_competitor\_org, e\_project\_team, e\_rel\_attachment, e\_rel\_knowledge\_doc, e\_related\_individual, e\_related\_product, email\_address, organization\_indiv, organization\_rel, row\_level\_audit, subprocess\_map, telephone, user\_note, workflow\_activity, workflow\_event, workflow\_task, workflow\_transition, workflow\_variable

The Quote, Lead, Opportunity, Interaction BIOs 'Inherit Sync and View Permissions' from their parent(s).

To get the most updated list for current customized metadata, in Infor Studio, go to **Tools > Filter > AdvancedFilter** and select **BioAttributesThatInheritSync** or **BioAttributesThatInheritSyncAndView**.

To see a list of all the BIOs that have at least one BIO row permission defined, in Infor Studio, go to **Tools > Filter > AdvancedFilter** and select **BiosWithAtleastOneRowPermissionDefined**.

## Upline and Downline

In Infor Epiphany Sales, there is a feature to track who directly/indirectly reports to a user. This feature allows customers to model the reporting hierarchy of their organization. The OA architecture provides BIO query macros to simplify implementing granting permission to the manager of the user.

The user's manager, the manager's manager (and so on) in the organization hierarchy constitute the upline for a user.

All the direct reports, the people who report to direct reports (and so on) in the organization hierarchy constitute the downline for a user.

Users on the upline have permissions to view/edit the data that the user has permissions to, without requiring explicit definition for each of the users above the user in the Organization hierarchy. The current implementation allows a user to have only one manager.

## Defining Exemptions

The OA architecture supports exemption of one or more user profiles and application profiles from ACL checks. Exemption means that a user with these profiles who logs into the application is exempt from ACL checks and can view/edit all data. To define exemptions, go to **Infor Studio > Physical & Logical Schema > Maintenance > Query Expressions**, select a Query Expression and mark the appropriate **Exempt** flag on the **User** or **Application Permissions** tabs or both.

The Administrator User profile is exempt from ACL checks. The Self-Service application and Service application profiles are also exempt from ACL checks.

## ACL Enforcement

**Note:** This section describes the ACL Enforcement for Mobile client. Mobile Sync is deprecated as of 7.0.4 FP1. The documentation is present here for informational purposes only.

The BIO layer enforces ACLs on access to data, while the Slicer enforces ACLs when creating the slice, and the Sync Service enforces ACLs when creating the synchronization package.

Granting new ACLs to a mobile user creates mini-slices on the next synchronization. For example, a user (with an existing slice) is added to a project team of an organization. On the next synchronization down, the organization and all BIOs that inherit from it such as address are added to the synchronization package.

Removal of ACLs does not delete data from mobile client, only access to that data is affected. A change in the downline requires a re-slice for the manager to get access to the latest information.



## Sales Team Implementation

The Teams module is implemented as a polymorphic BIO called the e\_project\_team. This BIO relates users to other BIO instances including Customer, Lead and Opportunity. Its ACL entry is generated using System extensions. Every entry in e\_project\_team generates rows in the ACLEntries table using system extensions on the e\_project\_team BIO.

Sales teams enable a collaborative model between sales representatives. Sales representatives can manually add team members to their sales team.

ACLs enable you to control who gets to see what. Typically, sales managers up line can see information for all their down line representatives.

The Teams module appears in the subform for the following Infor Epiphany Sales modules:

- Accounts (Organization and Individual)
- Leads
- Opportunity

## Implementing Access Control Lists

This section outlines the high level steps you have to follow to implement Access Control Lists using BIO row permissions. Use these high level steps to implement ACLs within the application.

**Note:** Some steps below refers to mobile client which is deprecated as of 7.0.4 FP1. The documentation is present here for informational purposes only.

## Implementing Access Control Lists

- 1 Identify the feature that needs to be enabled on a mobile client. Also identify all the objects and the related information that needs to be made available on the mobile client (typically a user's laptop)..
- 2 Identify how the rows of data will be limited for download on to a mobile client. This implies explicit ACL definitions on the BIO and all ACL inheritance.
- 3 Set up recordsets to be synced onto the mobile client.
- 4 Review all the extensions on the BIOs that are synced to see if they reference another BIO that may not be available on the mobile client.
- 5 Review all the extensions on the BIOs that are synced to see if any of these extensions need to be played at sync time. In Infor Studio, go to **Physical and Logical Schema > BIO Related Info > Main BIOs**, select **customer**, and then click the **Extensions** tab. Double-Click on any of the extensions. On the lower pane of the window, you will see a property called **Replay Limitation** with the following possible values: **At All Non-mobile, At All Servers, At Master Only, At Named Server, At Parent of Mobile** . You can also specify the **Direction - Upward Sync, Downward Sync, Both**.

- 6** If any of these recordsets have reference\_code then plan to change the setting for the reference\_code generator to use the prefixes.
- 7** Define an ACL query expression if none of the out-of-the-box query expressions work for the BIODs. Define an exemption for the query expression for the user/application profile that needs to be exempt from permissions.
- 8** Define ACLs on the BIOD(s).
- 9** Define appropriate ACL inheritance from this BIOD to all the related BIODs sharing both the parent and child relationship.
- 10** Modify the forms that will be made available on to the mobile client if they reference another ACL enabled BIOD so that users can see the data from that BIOD even if the user does not have permission to actually see all the details.  
  
For example, instead of having a customer sub-form on the request detail form(with the focus set to request.customer BIOD), create individual form widgets which map indirect mapped attributes such customer name, contact name, and so on.
- 11** Review all the user interface extensions to see if they reference any of the BIODs that the user may not have access to on the mobile client or may not have permission to review due to new ACL definitions.
- 12** Modify the definition of the out-of-the-box Infor Epiphany Sales application to include the new modules into it. (This assumes that only the metadata related to the Sales application gets downloaded onto the mobile client).
- 13** Test out the connected application to make sure the feature performs as expected, including all customizations.
- 14** Create a disconnected user and create a slice and test out the application and syncing of data.
- 15** An example on how to implement ACLs for the e\_opportunity recordset/BIOD follows.

The e\_opportunity BIOD represents Opportunities in Infor Epiphany Sales. Everyone who is explicitly given permission to view/update/delete an opportunity is stored in the e\_project\_team BIOD.

The requirement is to secure the e\_opportunity BIOD so that only the following people should have permissions to view/update/delete an instance of the e\_opportunity BIOD.

- The user who is explicitly granted permissions (as represented in e\_project\_team BIOD)
- The managers (upline) of the users who have explicit permissions to see an instance of e\_opportunity BIOD.
- Anyone who has permission to see the Customer BIOD this opportunity is associated with.
- The managers (upline) of the users who have explicit permission to see the customer to whom this opportunity is associated with.
- Anyone who has permissions to see the instance of the e\_opportunity BIOD should also be able to see all the information about this opportunity (information such as products, notes, attachments, and so on.)

Detailed instructions on the above follow. These instructions assume that you are familiar with OA architecture concepts, Infor Epiphany Sales functionality, and using Infor Studio.

## Implementing Access Control Lists

### Identify the BIO to be secured

The first step is to identify which BIO needs to be secured and what kind of permission logic is required.

In this example, the e\_opportunity BIO is identified as the BIO and permissions have to be implemented on this BIO.

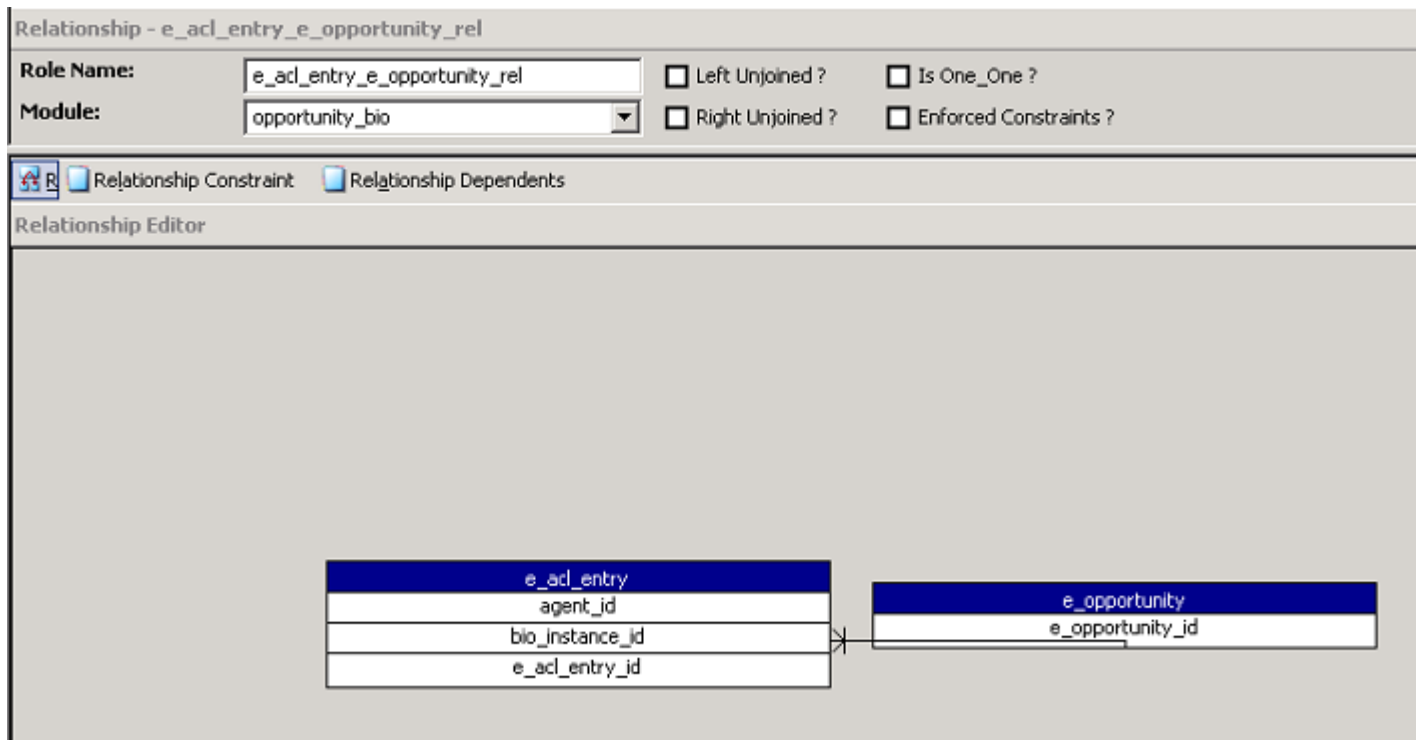
Define the permissioning requirements clearly before proceeding with the implementation. Once the requirements are defined, the rest of the process becomes easier to follow.

### Create Recordset Relationship

Define a relationship between your BIO and e\_acl\_entry.

In Infor Studio, go to **Physical and Logical Schema > Recordsets and Relationships > Relationships** and create a relationship with e\_acl\_entry on the left and your BIO on the right. Use e\_acl\_entry.bio\_instance\_id ' PK on your BIO to define the relationship

In this example, the relationship is e\_acl\_entry\_e\_opportunity\_rel on (e\_acl\_entry.bio\_instance\_id' e\_opportunity.e\_opportunity\_id)



### Create BIO Attribute in your BIO

Define a RELATION mapped BIO attribute of type BIOCLECTION of e\_acl\_entry BIOs, in your BIO so that it can later be used when generating ACL entries.

In Infor Studio, go to **Physical and Logical Schema > BIO Related Info > BIOs** . Go to your BIO and create a BIO attribute with the following values:

Name: <name of the attribute> Use ACLEntries for out-of-the-box BIOs

Type: BIOCOLLECTION

Mapping Type: RELATION

Click Save. Navigate to this attribute and go to **Mapping** tab and select the relationship created above.

Make sure **Cascade Delete** property is set properly.

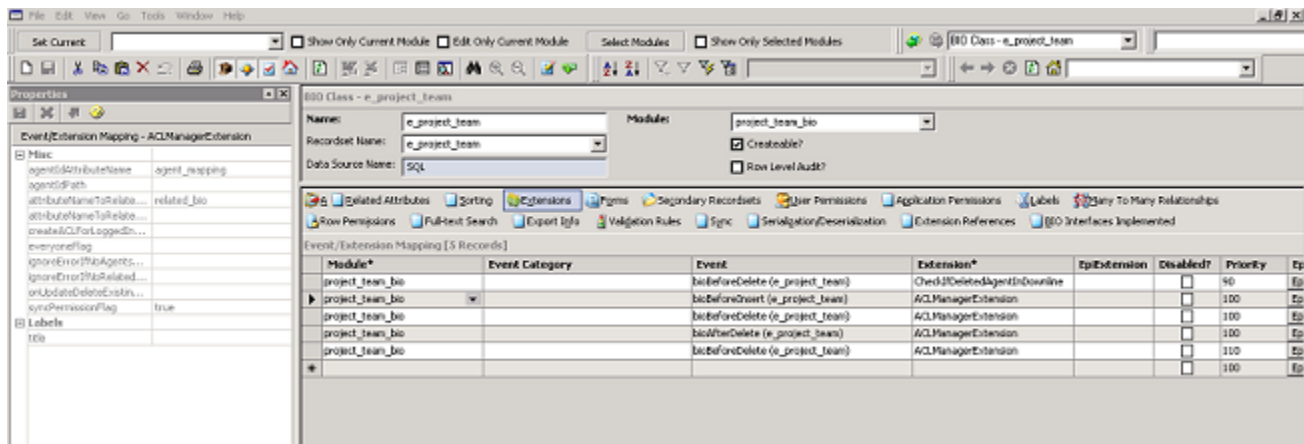
The screenshot shows the 'BIO Attribute - e\_opportunity.ACLEntries' configuration window. The 'Mapping' tab is active, displaying the relationship 'e\_ad\_entry\_e\_opportunity\_rel (e\_opportunity-e\_ad\_entry)' and the module 'opportunity\_bio'. The 'Cascade Delete' checkbox is checked. Other tabs visible include 'Row Permissions', 'User Permissions', 'Labels', 'Related Localized Attribute', 'Validation Rules', 'BIO Attribute Dependencies', and 'Extension References'.

## Generate ACL Entries for Explicit Permissions

For the BIO on which you have to define permissions, make sure that entries are getting written into the e\_acl\_entries BIO. Use the generic core extension called ACLManagerExtension to attach to the beforeInsert and beforeUpdate events of your BIO.

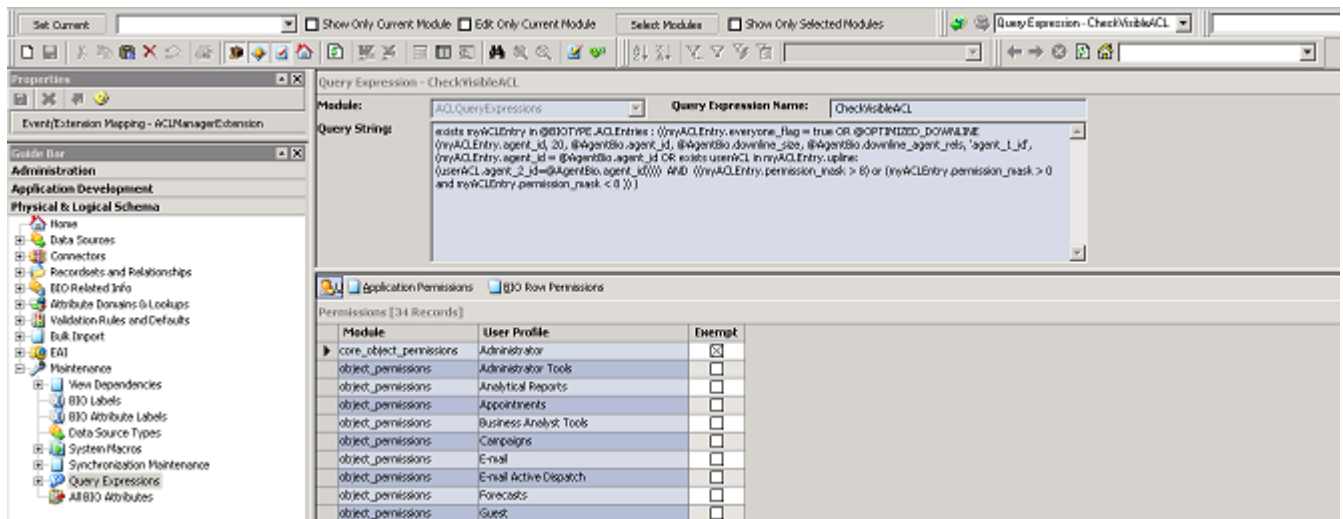
In Infor Studio go to **Physical and Logical Schema > BIO related Info > BIOs** , and the BIO under consideration. Click the **Extensions** tab and associate the ACLManagerExtension extension.

In this example, as everyone who has permissions to see e\_opportunity BIO instances are represented in e\_project\_team BIO, you have to attach the ACLManagerExtension extension to the e\_project\_team BIO.



In this example, the `e_project_team` BIO keeps track of permissions for more than just opportunities (that is, it is a polymorphic BIO that relates agents to various other BIOs, including Customer, Lead and Opportunity). For the parameter **attributeNameToRelatedBioInstance** specify a value of **related\_bio**. This indicates to the extension that this entry in `e_project_team` represents permission on the BIO that is represented in the attribute "related\_bio".

Define Query Expression. Out-of-the-box, a default **CheckVisibleACL** expression is available.



Define exemptions

## Using Access Control Lists

Query Expression - CheckVisibleACL

Module: ACLQueryExpressions Query Expression Name: CheckVisibleACL

Query String:

```
exists myACLEntry in @BIOType.ACLEntries : ((myACLEntry.everyone_flag = true OR @OPTIMIZED_DOWNLINE  
(myACLEntry.agent_id, 20, @AgentBio.agent_id, @AgentBio.downline_size, @AgentBio.downline_agent_rels, 'agent_1_id',  
(myACLEntry.agent_id = @AgentBio.agent_id OR exists userACL in myACLEntry.upline:  
(userACL.agent_2_id=@AgentBio.agent_id)))) AND ((myACLEntry.permission_mask > 8) or (myACLEntry.permission_mask > 0  
and myACLEntry.permission_mask < 8)))
```

Application Permissions BIO Row Permissions

Permissions [34 Records]

Module	User Profile	Exempt
core_object_permissions	Administrator	<input checked="" type="checkbox"/>
object_permissions	Administrator Tools	<input type="checkbox"/>
object_permissions	Analytical Reports	<input type="checkbox"/>
object_permissions	Appointments	<input type="checkbox"/>
object_permissions	Business Analyst Tools	<input type="checkbox"/>
object_permissions	Campaigns	<input type="checkbox"/>
object_permissions	E-mail	<input type="checkbox"/>

Attach BIO Row Permission.

BIO Class - e\_opportunity

Name: e\_opportunity Module: opportunity\_bio

Recordset Name: e\_opportunity

Data Source Name: SQL

☒ Createable?

☒ Row Level Audit?

Related Attributes Sorting Extensions Forms Secondary Recordsets User Permissions Application Permissions Labels Many To Many Relationships

Row Permissions Full-text Search Export Info Validation Rules Sync Serialization/Deserialization Extension References BIO Interfaces Implemented

Row Permissions [2 Records]

Module	Permissions*	Query Expression Na	Query Expression String
opportunity_bio	READ	CheckVisibleACL	exists myACLEntry in @BIOType.ACLEntries : ((myACLEntry.everyone_flag = true OR @OPTIMIZED_DOWNLINE(myACLEntry.agent_id, 20, @AgentBio.agent_id, @AgentBio.downline_size, @AgentBio.downline_agent_rels, 'agent_1_id', (myACLEntry.agent_id = @AgentBio.agent_id OR exists userACL in myACLEntry.upline: (userACL.agent_2_id=@AgentBio.agent_id)))) AND ((myACLEntry.permission_mask > 8) or (myACLEntry.permission_mask > 0 and myACLEntry.permission_mask < 8)))
opportunity_bio	SYNC	CheckSyncACL	exists myACLEntry in @BIOType.ACLEntries : ((myACLEntry.everyone_flag = true OR @OPTIMIZED_DOWNLINE(myACLEntry.agent_id, 20, @AgentBio.agent_id, @AgentBio.downline_size, @AgentBio.downline_agent_rels, 'agent_1_id', (myACLEntry.agent_id = @AgentBio.agent_id OR exists userACL in myACLEntry.upline: (userACL.agent_2_id=@AgentBio.agent_id)))) AND ((myACLEntry.permission_mask > 8) or (myACLEntry.permission_mask > 0 and myACLEntry.permission_mask < 8)))

Define ACL inheritance from this BIO's parent BIO.

BIO Class - customer

Name: customer Module: customer\_base

Recordset Name: customer

Data Source Name: SQL

☒ Createable?

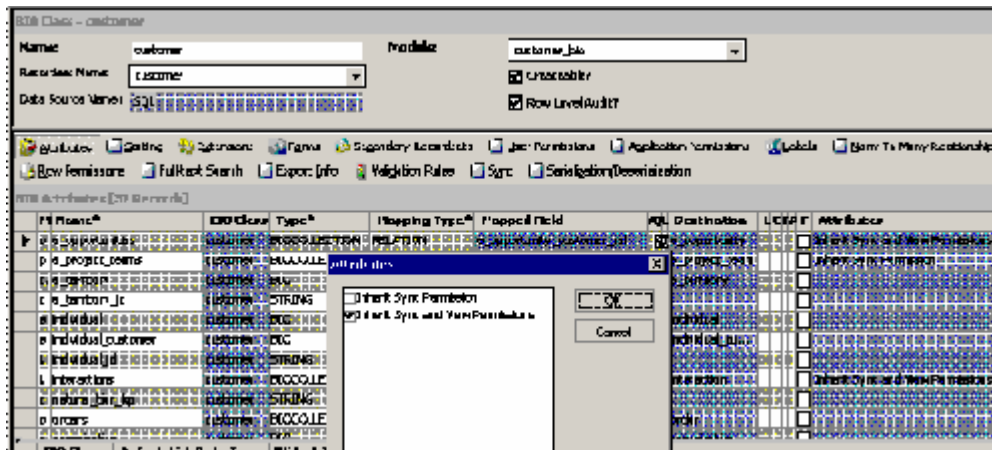
☒ Row Level Audit?

Related Attributes Sorting Extensions Forms Secondary Recordsets User Permissions Application Permissions Labels Many To Many Relationships

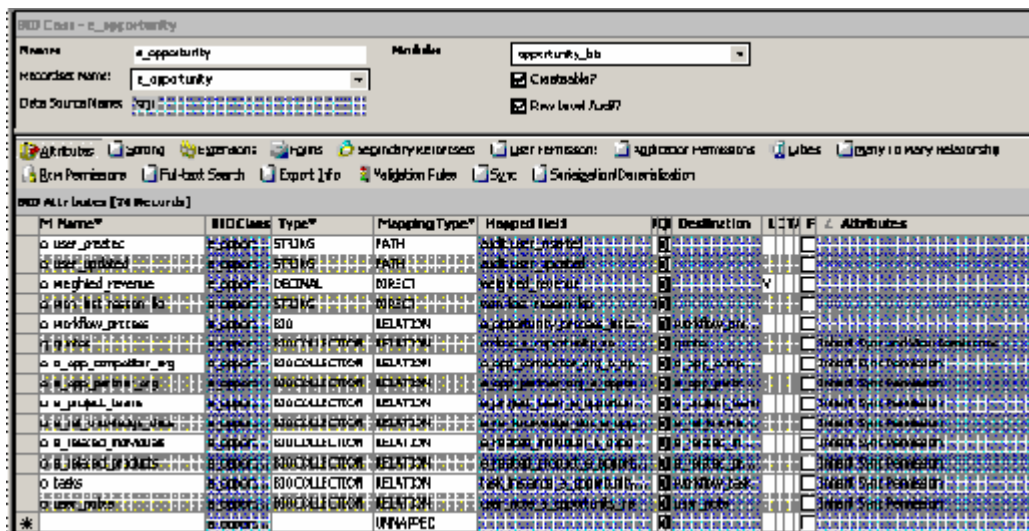
Row Permissions Full-text Search Export Info Validation Rules Sync Serialization/Deserialization Extension References BIO Interfaces Implemented

BIO Attributes [50 Records]

Module*	Name*	BIO Class*	Type*	Mapping Type*	Mapped Field	Destination BI
project_team_bio	ACLEntries	customer	BIOCOLLECTION	RELATION	e_acl_entry_customer_rel	e_acl_entry
customer_base	actual_bier_bip	customer	STRING	DIRECT	actual_bier_bip	
interaction_bio	all_interactions	customer	BIOCOLLECTION	QUERY	interaction.customer = @t...	interaction



Define Inheritance for child BIOs of this BIO.



## User Interface Related Changes Needed

If the BIO you have defined needs to display a value from another related BIO that has explicit ACL defined, then you can create an INDIRECT mapped attribute which points to a single attribute on the secondary BIO. The INDIRECT mapped attributes are mapped at the recordset level and are exempt from the BIO row permissions, thus providing an access mechanism to attributes from other BIOs.

In this example, the e\_opportunity BIO has an attribute customer\_name that comes from related Customer BIO. In the out-of-the-box application, the application is defined in such that a user may be granted permission to see an opportunity, and the same user may not have permission to see the details of Customer BIO. To be able to display only the customer name (not the other details), create an INDIRECT mapped attribute mapped to customer.customer\_name on the e\_opportunity BIO. If you have such a use case, follow the steps below.



---

[illegible][illegible]



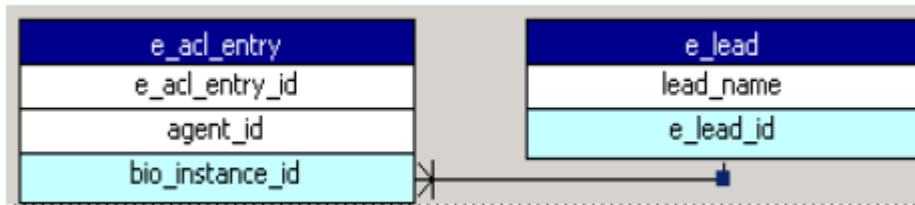
## Integrating Project Teams

Project Team (e\_project\_team BIO) is used to store agents related to any BIO which has ACLs on them, such as Customer (both Organization and Individual), Lead, and Opportunity. These BIOs use the e\_project\_team BIO in the ProjectTeam tab, that is, the Project Team tab for those forms displays the e\_project\_team\_list\_view form. You can add agents to the customer/lead/opportunity by clicking on the **'Add Existing'** button in the toolbar of that e\_project\_team\_list\_view form. This displays the list of users and when you select one or more individuals, they are then saved as e\_project\_team records. The 'Add Existing' button uses the UI extension for many-to-many BioCollections.

The following steps describe how to setup e\_project\_team to use this new functionality.

These steps show e\_lead as an example. Substitute your new module in the following steps.

- 1 e\_project\_team\_e\_lead\_rel (joining e\_project\_team.related\_id to e\_lead.e\_lead\_id).
- 2 If it does not exist, create a relationship between e\_acl\_entry.bio\_instance\_id and e\_lead.e\_lead\_id as shown below.



- 3 On the BIO Interface e\_project\_team\_interface, create an entry under BIO Interface Implementation for the new module similar to e\_lead.
- 4 On the e\_project\_team BIO, create a relation mapped BIO attribute of type BIO, for the BIO that is related to many types of BIOs - in this case an attribute named 'e\_lead' of type BIO whose destination BIO is e\_lead.
- 5 On the e\_lead BIO, create a M:M mapped Bio Collection of the 'many' BIO. For example, in the case of the e\_project\_team scenario, on the lead BIO, setup a M:M mapped Bio Collection attribute 'project\_team\_members' (that is, destination BIO = agent\_user).
- 6 On the e\_project\_team BIO, double click on the RelationIFC mapped attribute named related\_bio. A screen similar to the following will appear.

**Bio Attribute - e\_project\_teamrelated\_bio - Row Permissions**

Module:  Name:

BIO Class:  Mapping Type:

Type:  Destination BIO:

Mapped Field:  Attribute Domain:

Localized Data Type:

Time zone:

☐ Field Level Audit? ☒ Updatable?

☒ Queryable?

Row Permissions [0 Records]

Module	Permissions*	Query Expression Na	Query Expression String
* project...			

7 Click on the Mapping binoculars. The following screen appears.

**Relation Interface Mapping - Relation Interface Definition**

Class Type Attribute:  BIO Attributes:

Module:

Relation Interface Definition [1 Records]

Module*	Interface Implementor Class	Relationship	Is Left to Right	Cascade Delete
Lead	<input type="text" value="e_lead"/>	e_project_team_e_lead_rel(e_project_team-e_lead)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
* project_team			<input type="checkbox"/>	<input type="checkbox"/>

On the **Relation Interface Definition**, Create an entry by selecting the entry, e\_lead, created in step 3 (the relation mapped BIO attribute) as the Interface Implementor Class.

- 8 On the e\_project\_team BIO, created a relation mapped BIO attribute of type BIO, for the BIO that is related to many types of BIOs, in this case, an attribute named 'e\_lead' of type BIO whose destination BIO is e\_lead.
- 9 In the case of e\_project\_team, although the m:m BioCollection of project\_team\_members is used to create new e\_project\_team records for a lead, the related agents (project\_team) are displayed using a form based on e\_project\_team, that is, e\_project\_team\_list\_view rather than a form based on agent\_user. This enables the e\_project\_team to have an optional role column that the user can edit after adding that individual to the lead. This is the reason the e\_lead BIO has a relation mapped Bio Collection attribute 'project\_team' setup. The e\_lead detail forms have a tabgroup displaying the e\_project\_teams form, that is, displaying the project\_team related to the lead.

- 10 On the e\_lead BIO, create a RELATION mapped attribute with name ACLEntries of Type "BIOCOLLECTION". The mapping field should be the relationship created in step 3 e\_acl\_entry\_e\_lead\_rel and the destination BIO should be e\_acl\_entry.
- 11 On the e\_lead BIO, click **Row Permissions** and add the appropriate row permissions for READ (CheckVisibleACL) and SYNC(CheckSyncACL).
- 12 Create entries on the tabgroup shell of shared\_sales to create a new tabgroup to display the Project Team List form.
- 13 When creating a new lead, the creating user is added to the project team by default. To do this, on the e\_lead BIO, call a BIO extension called **AddToProjectTeamBioExtension** for the BIO BeforeInsert event. This extension has an optional attribute called attributenameforagentuser BioCollection, which has to be set to the name of your agent\_user Bio Collection if you have not named your Bio Collection on your e\_lead BIO as project\_team\_members (You would have created this in step 9).

Module*	Event Category	Event	Extension*	Disabled?	Priority
lead_bio		e_lead_bioBeforeUpdate	ParentCumCanViolatedProductBioExtension	<input type="checkbox"/>	100
Territory_bio		e_territory_bioBeforeUpdate	ParentCumCanViolatedProductBioExtension	<input type="checkbox"/>	100

Event/Extension Mapping - AddToProjectTeamBioExtension	
<b>Misc</b>	
attributenameforagentuserbioc...	
<b>Labels</b>	
title	

## ACLManagerExtension

The `ACLManagerExtension` is a system extension that helps to create ACLs. This extension accepts the following parameters:

Name	Description
syncPermissionFlag	By default, the extension creates ACL for read, write, and delete. This flag results in setting the sync permission as well.
everyoneFlag	If enabled, the ACL entry is created for everyone and not for a particular agent.

Name	Description
<code>attributeNameToRelatedBioInstance</code>	The name of the BIO attribute on the modified BIO that holds the related item BIO. Set it to null if this is the BIO being ACLed.
<code>agentIdPath</code>	Optional BIO attribute name to a Bio Collection containing the actual agent_user BIO.
<code>agentIdAttributeName</code>	The Bio attribute on this BIO mapped to agent_user. Set to null if using everyoneFlag.
<code>onUpdateDeleteExistingACLs</code>	Deletes all existing ACLs on updates.
<code>attributeNameToRelatedBioRef</code>	The name of the BIO attribute on the attached BIO that holds a Bio ref to the related item. Set to null if this is the BIO with ACL.
<code>ignoreErrorIfNoAgentsDefined</code>	If set to true and everyoneFlag is set to false, then the extension will not return RET_CANCEL when agentIdAttributeName points to no agents.
<code>createACLForLoggedInUser</code>	By enabling this, an ACL will also be created for the logged-in user. (This is only used for insert/update events).
<code>ignoreErrorIfNoRelatedBioDefined</code>	If set to true, then the extension does not return RET_CANCEL and throws an exception if the related BIO is not found.

Attach the `ACLManagerExtension` to all BIOs that implement ACLs and do not have project teams associated.

The BIOs that have the project team tab displayed and the appropriate project team extensions attached do not need the `ACLManagerExtension` attached to the BIO. The `AddToProjectTeam` extension attached to the Project Team(`e_project_team`) Bio creates the necessary ACL entries. The `attributeNameToRelatedBioInstance` parameter for the `ACLManager` extension on the `e_project_team` BIO is specified as "related\_bio". This indicates to the extension that this entry in `e_project_team` represents permission on the BIO that is represented in the attribute "related\_bio".

In case of the Customer BIO, although the project team tab is displayed and the `AddToProjectTeam` extension associated, you additionally attach the `ACLManagerExtension` to the BIO and pass in "individual" as the value for the "Related Bio" parameter that the extension optionally expects. The Individual BIO is a subclass of customer(organization is also a subclass of customer) and also shares the underlying recordset with Contact. For this reason, when creating an Individual (individual\_customer BIO), you have to create one entry each for the individual as well as the Customer BIO.

In addition, the `AddToProjectTeam` Bio extension automatically adds the logged in user to the Project Team. It also adds the Template Team of the sales person and territory combination to the Project Team.

## Current Limitations

The `CheckSyncACL` is not customizable and multiple Data Sources cannot be supported in disconnected mode.



This chapter describes how you can use Infor Studio to translate application strings.

## Translating Application String Steps

### 1 Install Infor Studio.

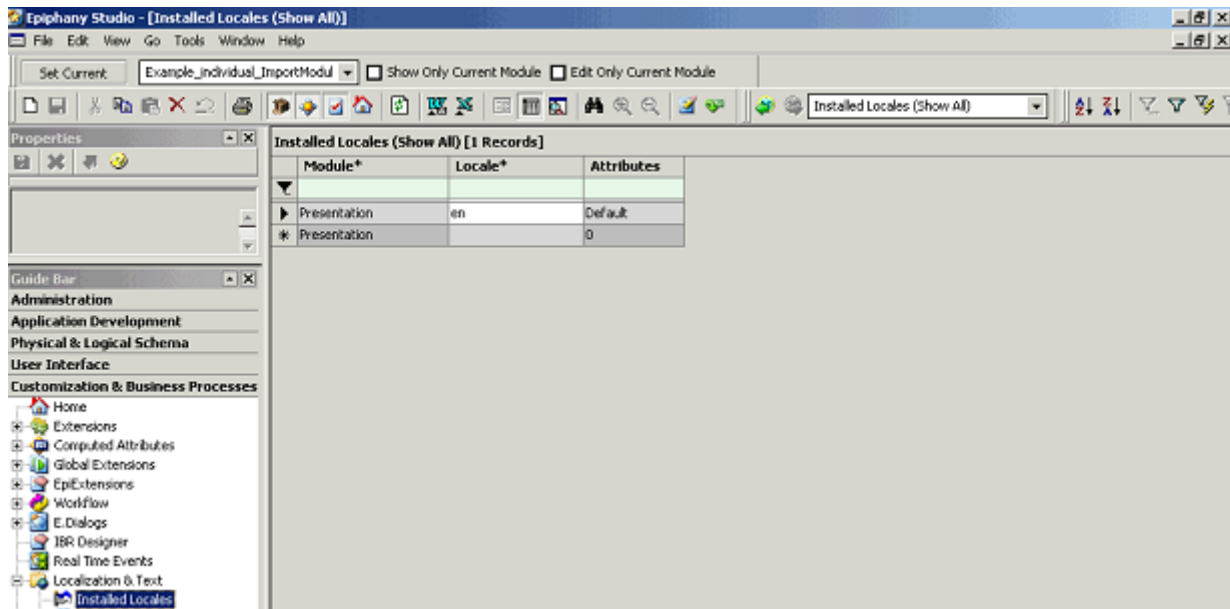
Initially, the 'Translations' folder in the Studio installation path (C:\Program Files\Infor\Tools\Studio\) contains only the 'en' (for English) folder. If the application TUM file for the destination locale is available, create a folder for the destination locale and place the application TUM file in that folder. The application TUM file name will have the following pattern: 'Service\_App\_Meta\_<locale\_id>\_Tum.mdb.' For example, a Japanese application TUM file will have 'Service\_App\_Meta\_ja\_Tum.mdb' name and the folder will be C:\Program Files\Infor\Tools\Studio\Translations\ja. By placing the application TUM file in the ja folder, the TUMS process in Studio uses the string translations that have been defined in the application TUM file. For strings that do not have their translation in the application TUM file, they will be displayed in their original language, assuming no pseudo translation is defined.

### 2 Set up Infor Studio to login as an internal user by executing the following SQL Statements:

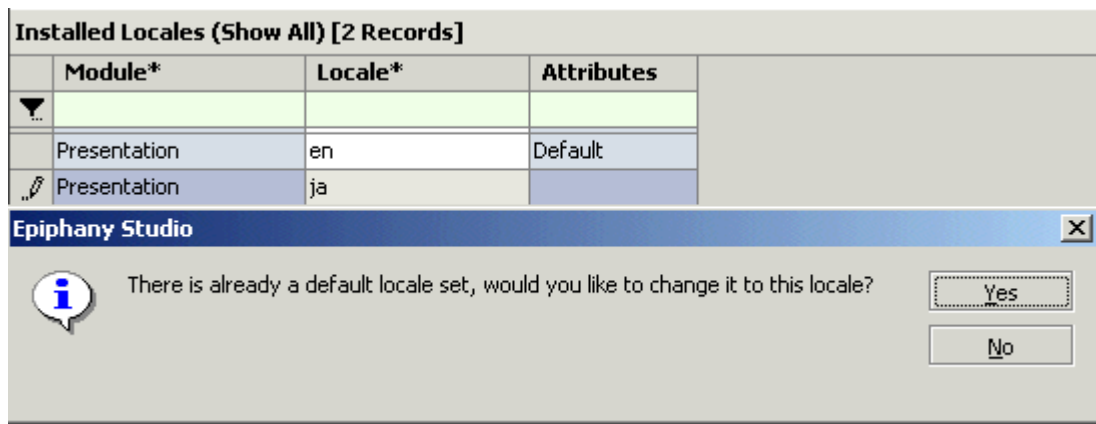
```
/* for external users (ships this way out-of-the-box) */
delete from tsy_users
delete from tsy_users_groups
delete from tsy_groups where group_id = 'epny'

/* for epny internal */
delete from tsy_users
delete from tsy_users_groups
delete from tsy_groups where group_id = 'epny'
INSERT INTO
    tsy_groups(group_id,app_module_id,group_name,group_description)
VALUES('
    epny','STUDIO','EPNY','Epiphany Engineering')
```

- 3 Open Infor Studio after making sure that it points to the correct database.
- 4 From the *Guide Bar*, navigate to **Installed Locales (Customization & Business Processes > Localization & Text > Installed Locales)**.



- 5 Add the Locale or Locales that you want Infor Studio to perform the translation to.

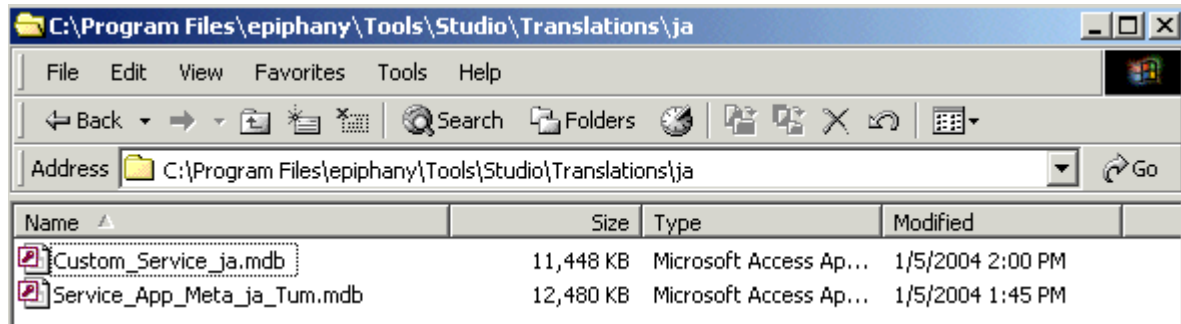


- 6 Click **No** to the dialog box.
- 7 Right-click on any row in the **Installed Locale** and select the **Execute TUMS...** menu option.  
You can execute two different types of TUMS (a Windows based translation utility) depending on user permission. If you have internal user permission, the **Execute TUMS** menu option appears. This option is used by Infor engineering. If you do not have internal user permission, the **Execute Custom TUMS** menu option appears. Custom TUMS is for people outside Infor engineering, such as Professional Services people and customers. To enable internal user permission, before using Infor Studio, run the SQL statement described in Step 2.
- 8 When you select the **Execute TUMS...** option, a dialog box displays the following message: *"The TUMS process will create translations for the following locales: ko to ja from locale en ."*



- 9 Click **OK** to proceed with the translation. If the translation is successful, a dialog box notifies you that the TUMS process is completed successfully

For the Japanese locale, a new file called Service\_app\_meta\_ja\_Tum.mdb is created under C:\Program Files\Infor\Tools\Studio\Translations\ja



The Service\_app\_meta\_ja\_Tum.mdb file is created only if you are logged in as an internal user. If you are logged in as an external user, the Custom\_service\_<locale\_id>.mdb is created.

- 10 Verify that the strings are translated (or pseudo translated depending on whether the pseudo translated flag is set). Pseudo translation appends/reverses the strings so that the newly generated strings are easily identified. The pseudo translated strings appear as follows:

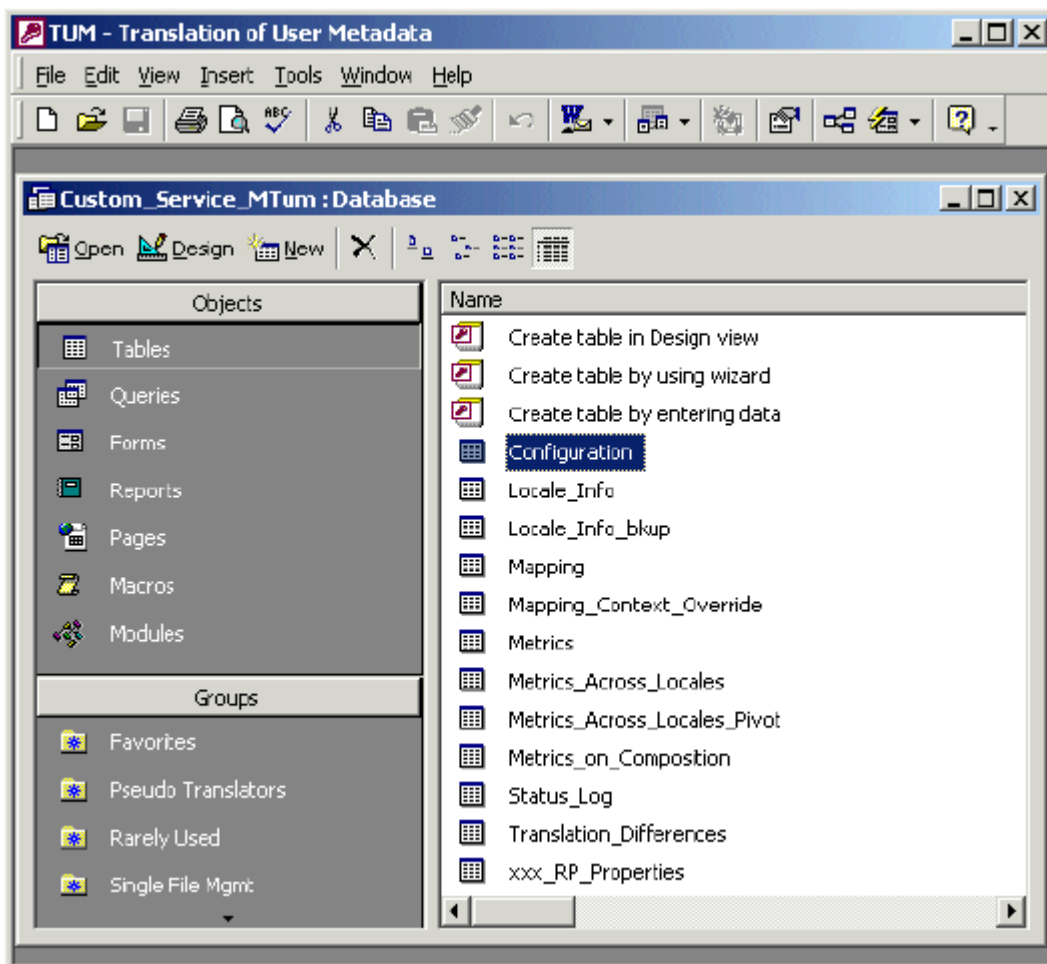
User Interface	System	ja	ERROR_PAGE_BODY_...	Back Trace
Customization & Business Processes	System	en	ERROR_PAGE_BODY_...	Stack Trace
Home	System	ja	ERROR_PAGE_MSG	Stack Trace
Extensions	System	en	ERROR_PAGE_MSG	Stack Trace
EpiExtensions	System	ja	ERROR_PAGE_MSG	Error Message
Workflow	System	en	ERROR_PAGE_SUB_T...	Error Message
C.Dialogs	System	ja	ERROR_PAGE_SUB_T...	Server Error
IEP Editor	System	en	ERROR_PAGE_TITLE	Server Error
Real Time Events	System	ja	ERROR_PAGE_TITLE	Server Error
Localization & Text	System	en	HomeInteractionName	Home
Installed Locales	System	ja	HomeInteractionName	Home
Labels	product_bio	en	INVALID_PRODUCT	Invalid Product
Text Messages	product_bio	ja	INVALID_PRODUCT	Invalid Product
Global Message	order_bio	en	Order	Order
Client Side Javascript	order_bio	ja	Order	Order
LOG_Messages	product_bio	en	PARENT_PRODUCT	Parent Product
TXT_Messages	product_bio	ja	PARENT_PRODUCT	Parent Product
EXP_Messages	shared_bio	en	PortalProductTitle	E.phiphany Web Self-Service
EpiExtension User Defined Messages	shared_bio	ja	PortalProductTitle	E.phiphany Web Self-Service
User Defined Text				
Maintenance				

The boxes that are displayed in the above screen shot are Japanese characters. The characters are displayed as boxes because the machine that runs Infor Studio does not support display of Japanese characters.

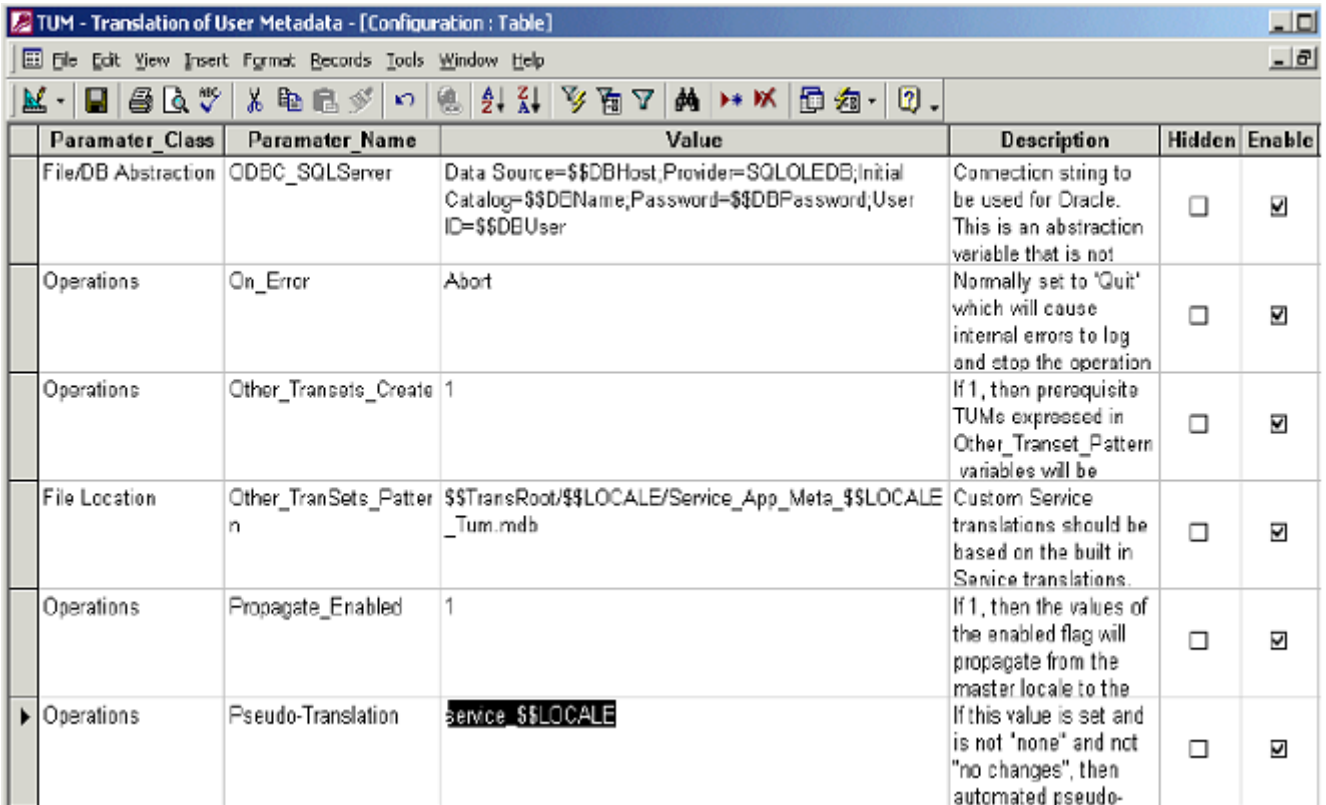
The fully translated strings, in addition to the pseudo translation set appear as follows.

Real Time Events	System	ja	ERROR_PAGE_TITLE	Server Error
Localization & Text	System	en	ERROR_PAGE_TITLE	Server Error
Installed Locales	System	ja	HomeInteractionName	Home
Labels	System	en	HomeInteractionName	Home
Text Messages	product_bio	en	INVALID_PRODUCT	Invalid Product
Global Message	product_bio	ja	INVALID_PRODUCT	Invalid Product
Client Side Javascript	order_bio	en	Order	Order
LOG_Messages	order_bio	ja	Order	Order
TXT_Messages	product_bio	en	PARENT_PRODUCT	Parent Product
EXP_Messages	product_bio	ja	PARENT_PRODUCT	Parent Product
EpiExtension User Defined Messages	shared_bio	en	PortalProductTitle	E.phiphany Web Self-Service
User Defined Text	shared_bio	ja	PortalProductTitle	E.phiphany Web Self-Service
Maintenance				

- 11 To set up pseudo translation, open the Custom\_Service\_MTum.mdb database located in C:\Program Files\Infor\Tools\Studio\Translations.



- 12 Open the **Configuration** table and set the **Pseudo-Translation** Paramater\_Name to service\_\$\$LOCALE.



Paramater_Class	Paramater_Name	Value	Description	Hidden	Enable
File/DB Abstraction	ODBC_SQLServer	Data Source=\$DBHost;Provider=SQLOLEDB;Initial Catalog=\$DDEName;Password=\$DBPassword;User ID=\$DEUser	Connection string to be used for Oracle. This is an abstraction variable that is not	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Operations	On_Error	Abort	Normally set to 'Quit' which will cause internal errors to log and stop the operation	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Operations	Other_Transets_Create	1	If 1, then prerequisite TUMs expressed in Other_Transet_Pattern variables will be	<input type="checkbox"/>	<input checked="" type="checkbox"/>
File Location	Other_Transets_Pattern	\$\$TransRoot/\$\$LOCALE/Service_App_Meta_\$\$LOCALE_Tum.mdb	Custom Service translations should be based on the built in Service translations.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Operations	Propagate_Enabled	1	If 1, then the values of the enabled flag will propagate from the master locale to the	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Operations	Pseudo-Translation	Service \$\$LOCALE	If this value is set and is not 'none' and not 'no changes', then automated pseudo-	<input type="checkbox"/>	<input checked="" type="checkbox"/>

When you install TUMSInfor, you also install a TUMS Access database.

After successfully installing TUMS, in Infor Studio, you will see boxes next to the text string for 'jp' locale (for example). This is expected behavior. The boxes are displayed because your machine does not support displaying the characters. You need to have the locale installed to be able to see these characters.

Certain strings are not translated on the application user interface and this occurs due to a number of reasons. For words that are not specified out-of-the-box, you have to manually translate the text into Japanese. There is no one specific place to translate the text for its use throughout the Infor application. There are a few places.

You have to implement translations on the widget in the form that the widgets are used in. For example, you have to go to the form where the drop-down widget that holds the 'All' text and carry out the translation at that level. You have to do this because "All" is hard coded in SQL-based attribute domains, Crystal Reports, and Java extensions.

Once you set up the translations in TUMS and run TUMS, you should be able to get the translations. If you don't use TUMS, you can manually change the text in Infor Studio (**Guide Bar > Customization & Business Processes > Localization and Text > Text Messages**). Setting up the translation using Infor Studio is optional.

You also have to set up the translations in the TUMS file. Search for the English version of the word in the Mapping table and make sure that you change the 'Look In' field in the Search facility to 'Mapping:Table'. Translate the word and set the Translated field in the Mapping table to enabled (change the 0 to a 1). The **Enabled** column also indicates that the text will be translated.

Within the out-of-the-box TUMS database, all exceptions and log message translations are stored in one place. All labels, user defined text and other text messages are stored in another place. For

example, with Japanese translations, all exceptions and log message translations are stored in the Mapped table in the Service\_LogExp\_Meta\_ja\_Tum.mdb database. All labels, user defined text and other text messages are stored in the Mapped table in the Service\_App\_Meta\_ja\_Tum.mdb database. If you set the translation strings here, then you do not have to set the translation in Infor Studio from the **Localization and Text** areas.

Once the words are translated in all these areas, do a **reloadmeta** from the command prompt. Also, make sure that you drop Internet Explorer cache to see your changes.

## Overview

The Attachments module allows users to upload and link a document from the local file system for the Infor Epiphany Service application modules. Users use this module to create a new attachment, change the attributes of the attachment (notes and requested fields), and delete an existing attachment. Note that editing the uploaded content of an uploaded attachment is not allowed. If you want to change the content, you have to edit it externally and re-attach it.

The attachment module is common for both Infor Epiphany Sales and Service Customer Service.

## Integrating Attachments

This procedure provides an example of how a request can be integrated with attachments. This procedure assumes that you are familiar with Infor Studio functionality, especially with setting up many-to-many relationships between BIOs.

- 1 In Infor Studio, navigate to Relationships (**Physical and Logical schema > Recordsets and Relationships > Relationships**).
- 2 Create a Many-to-Many relationship with e\_rel\_attachment BIO as follows:
  - a Create relationship to e\_rel\_attachment BIO.
  - b Make sure that e\_rel\_attachment is on the left.
  - c Drag the related\_id to your bio\_id (product\_id)
  - d Make sure that the many fork is on the related\_id in e\_rel\_attachment.

## Integrating a Request with Attachments

Relationships (Custom Filter) [9 Records]

Module*	Role Name*	Left Unjoined ?	Right Unjoin	Is One_On	Enforced C
	%e_rel_atta%	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
lists_bio	e_rel_attachment_e_list_rel	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
opportunity_bio	e_rel_attachment_e_opportunity_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
customer_bio	e_rel_attachment_individual_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
order_quote_bio	e_rel_attachment_orders_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
customer_bio	e_rel_attachment_organization_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
product_bio	e_rel_attachment_product_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
request_bio	e_rel_attachment_request_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Relationship Editor

Relationship Constraint Relationship Dependents

```
graph TD
    request[request  
request_id]
    e_rel_attachment[e_rel_attachment  
e_attachment_id  
related_string_id  
related_id  
e_rel_attachment_id]
    request --> e_rel_attachment
```

3 Click the **Relationship Constraint** tab on this new relationship.

- Set the **Data Field** to **related\_tbl\_name**.
- Set the **Constraint Value** to the name of your BIO (in this example, the Request BIO). You do this so that the **related\_tbl\_name** is auto populated on the polymorphic table (e\_rel\_attachments).

Relationships (Custom Filter) [9 Records]

Module*	Role Name*	Left Unjoined ?	Right Unjoin	Is One_On	Enforced C
	%e_rel_atta%	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
lists_bio	e_rel_attachment_e_list_rel	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
opportunity_bio	e_rel_attachment_e_opportunity_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
customer_bio	e_rel_attachment_individual_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
order_quote_bio	e_rel_attachment_orders_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
customer_bio	e_rel_attachment_organization_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
product_bio	e_rel_attachment_product_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
request_bio	e_rel_attachment_request_rel	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

☒ Relationship Constraint
 ☐ Relationship Dependents

Relationship Constraints [1 Records]

Module*	Data Field*	Constraint Value*
request_bio	related_tbl_name	request
* request_bio		

- 4 Go to the BIO Interfaces section, and select the interface called **attachment\_bio\_interface** . Add your BIO to the **Bio Class**.

BIO Interfaces (Custom Filter) [1 Records]

Module*	BIO Class*	Class Type Attribute
	attachment_bio_interface	
attachment_hr	attachment_hr_interface	
*		

☒ BIO Interface Implementation

BIO Interface Implementation [1 Records]

Module*	BIO Class*	Attribute Type Value
request_bio	request	request
*		

- 5 Create a relation mapped BIOCLECTION attribute on your Request BIO called "e\_rel\_attachments" using the relationship you created in Step 1.

## Integrating a Request with Attachments

BIO Class (Custom Filter) [4 Records]						
Module*	Name*	Data Source Name	Recordset Name	Createable?	Row Level Audit?	Attributes
	request			<input type="checkbox"/>	<input type="checkbox"/>	
request_bio	request	SQL	request	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
request_bio	request_interaction	SQL	request_interaction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
order_bio	request_order	SQL	request_order	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
request_bio	request_rel	SQL	request_rel	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*				<input checked="" type="checkbox"/>	<input type="checkbox"/>	0

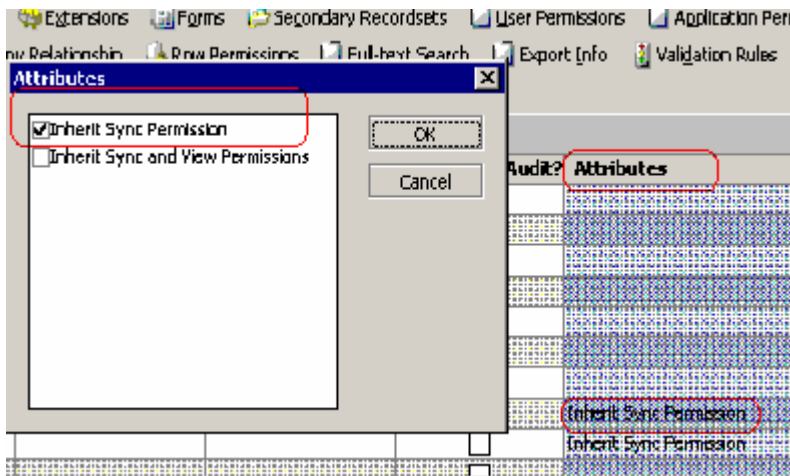
  

Related Attributes	Sorting	Extensions	Forms	Secondary Recordsets	User Permissions	Application Permissions	Labels	Many To Many Relationships
Row Permissions	Full-text Search	Export Info	Validation Rules	Sync	Serialization/Deserialization	Extension References	BIO Interfaces Implemented	

BIO Attributes [66 Records]								
Name*	BIO Class*	Type*	Mapping Type*	Mapped Field	Destination BIO	Attribute Domain	Queryable?	Updateable?
date_updated	request	DATE	PATH	audit.date_updated			<input checked="" type="checkbox"/>	<input type="checkbox"/>
description	request	STRING	DIRECT	description			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
determined_priority_lkp	request	STRING	DIRECT	determined_priority_lkp		request_determined_priority_lo...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
e_prod_instance_request	request	BIOCOLLECTION	RELATION	e_prod_instance_request...	e_prod_instance_request		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
e_rel_attachments	request	BIOCOLLECTION	RELATION	e_rel_attachment_request...	e_rel_attachment		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
e_rel_knowledge_docs	request	BIOCOLLECTION	RELATION	e_rel_knowledge_doc_req...	e_rel_knowledge_doc		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
e_related_attachments	request	BIOCOLLECTION	MANY_TO_MANY		attachment		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
e_related_documents	request	BIOCOLLECTION	MANY_TO_MANY		knowledge_document		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

- 6 In the **Attributes** column, check the checkbox next to **Inherit Sync Permission**.



- 7 Create another attribute called "e\_related\_attachments" of type biocollection on your Product BIO. Set the mapping type to be many-to-many. Make the Attachment BIO the destination BIO.



BIO Class (Custom Filter) [4 Records]

Module*	Name*	Data Source Name	Recordset Name	Createable?	Row Level Audit?
request	request	SQL	request	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
request_bio	request	SQL	request	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
request_bio	request_revision	SQL	request_revision	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
request_bio	request_id	SQL	request_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Validation Rules

BIO Attributes [11 Records]

Name*	Type*	Mapping Type*	Mapped Field	Destination BIO	Queryable?	Updatable?	BIO Class*
date_created	DATE	PATH	date_created		<input checked="" type="checkbox"/>	<input type="checkbox"/>	request
description	STRING	DIRECT	description		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
determined_order_id	STRING	DIRECT	determined_order_id		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_revision_id	INTEGER	RELATION	request_revision_id	request_revision_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_id	INTEGER	RELATION	request_id	request_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_revision_id	INTEGER	RELATION	request_revision_id	request_revision_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_id	INTEGER	RELATION	request_id	request_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_revision_id	INTEGER	RELATION	request_revision_id	request_revision_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_id	INTEGER	RELATION	request_id	request_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_revision_id	INTEGER	RELATION	request_revision_id	request_revision_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_id	INTEGER	RELATION	request_id	request_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request
request_revision_id	INTEGER	RELATION	request_revision_id	request_revision_id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	request

- 8 Click on the **Mapped Field** and verify that your screen looks like below, and then click **OK**.

**Many to Many Relation Mapping**

Relationship Role:

Attributes:  ...

Module:

OK Cancel

- 9 Go to the `e_rel_attachment` BIO and create the relationship between your Request BIO and `e_rel_attachment` BIO.

BIO Class (Custom Filter) [1 Records]

Module*	Name*	Data Source Name	Recordset Name	Createable?	Row Level Audit?	Attributes
e_rel_attachment	e_rel_attachment	SQL	e_rel_attachment	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0

Related Attributes

Sorting

Extensions

Forms

Secondary Recordsets

User Permissions

Application Permissions

Labels

Many To Many Relationships

Row Permissions

Full-text Search

Export Info

Validation Rules

Sync

Serialization/Deserialization

Extension References

BIO Interfaces Implemented

BIO Attributes [19 Records]

Module*	Name*	BIO Class*	Type*	Mapping Type*	Mapped Field	Destination BIO	Attribute Domain
product_bio	product	e_rel_attachment	BIO	RELATION	e_rel_attachment_product...	product	
attachment...	related_bio	e_rel_attachment	BIO	RELATIONIFC	individual_customer,organi...	attachment_bio_interface	
attachment...	related_id	e_rel_attachment	STRING	DIRECT	related_id		
attachment...	related_int_id	e_rel_attachment	INT	DIRECT	related_int_id		
attachment...	related_string_id	e_rel_attachment	STRING	DIRECT	related_string_id		
attachment...	related_tbl_name	e_rel_attachment	STRING	DIRECT	related_tbl_name		
request_bio	request	e_rel_attachment	BIO	RELATION	e_rel_attachment_request...	request	
attachment...	revision_number	e_rel_attachment	INT	DIRECT	revision_number		
*		e_rel_attachment	UNMAPPED				

- 10 On the e\_rel\_attachment BIO, double click on the "related\_bio" attribute with the **RelationIFC** mapping type.

BIO Class (Custom Filter) [1 Records]						
Module*	Name*	Data Source Name	Recordset Name	Createable?	Row Level Audit?	Attributes
	e_rel_attachment			<input type="checkbox"/>	<input type="checkbox"/>	
attachment_bio	e_rel_attachment	SQL	e_rel_attachment	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0
*				<input checked="" type="checkbox"/>	<input type="checkbox"/>	0

Module*	Name*	BIO Class*	Type*	Mapping Type*	Mapped Field	Destination BIO
product_bio	product	e_rel_attachment	BIO	RELATION	e_rel_attachment_product_rel	product
attachment_bio	related_bio	e_rel_attachment	BIO	RELATIONIFC	individual_customer,organization...	attachment_bio_interface
attachment...	related_id	e_rel_attachment	STRING	DIRECT	related_id	
attachment...	related_int_id	e_rel_attachment	INT	DIRECT	related_int_id	
attachment...	related_string_id	e_rel_attachment	STRING	DIRECT	related_string_id	
attachment...	related_tbl_name	e_rel_attachment	STRING	DIRECT	related_tbl_name	
request_bio	request	e_rel_attachment	BIO	RELATION	e_rel_attachment_request_rel	request
attachment...	revision_number	e_rel_attachment	INT	DIRECT	revision_number	
*		e_rel_attachment		UNMAPPED		

- 11 Select the Mapped Field icon to go to the **Relation Interface Definition** tab. Add a new definition with your class in the **Interface Implementor Class** column, and select your BIO. Note that you have to define the BIO interface in Step 2 for your BIO to appear.

Relation Interface Definition [9 Records]				
Module*	Interface Implementor Class*	Relationship	Is Left to Right ?	Cascade Delete?
request_bio	request	e_rel_attachment_request_rel...	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Your Many to Many should be set up now.

- 12 Create the Attachments tab group in the Sales\_TabGroup\_Shell. Specify a tab identifier and remember the tab identifier you choose because you will use it later. Use the appropriate **generic\_tab\_identifier** with the right **tab identifier list order**.

Module*	Form Name*	Form Type*	Menu Name	Menu Order
request	request	LINKEDTABLE		
attachment	attachment	LINKEDTABLE		
request	request	LINKEDTABLE		

Module*	Table Type*	Attribute Name	Attribute Name*
request	request		request
attachment	attachment		attachment

Form*	Form Path	Form Placement	Table Identifier	Default	Module*	Perspective
request	request	request	request		request	
attachment	attachment	attachment	attachment		attachment	
request	request	request	request		request	
attachment	attachment	attachment	attachment		attachment	
request	request	request	request		request	
attachment	attachment	attachment	attachment		attachment	
request	request	request	request		request	
attachment	attachment	attachment	attachment		attachment	
request	request	request	request		request	
attachment	attachment	attachment	attachment		attachment	
request	request	request	request		request	
attachment	attachment	attachment	attachment		attachment	

The next screen shot shows the tab definition for the product implementation. The picture has been split into two to be legible.

Name*	BIO Path	Form Placement	Tab Identifier
Message_TabGroup_Slot_Requests_View		Sales_TabGroup_Shell_Message_Bio_Both	generic_tab_identifier_004
Request_TabGroup_Slot_Audit_Trails	requestaudittrail_level_audits	Sales_TabGroup_Shell_Request_Bio_Both	generic_tab_identifier_005
Request_TabGroup_Slot_Blank_Tab		Sales_TabGroup_Shell_Request_Bio_Sales_...	
Request_TabGroup_Slot_Detail_Tab		Sales_TabGroup_Shell_Request_Bio_Sales_L...	
Request_TabGroup_Slot_Interactions	requestinteractions	Sales_TabGroup_Shell_Request_Bio_Both	
Request_TabGroup_Slot_Notes	requestuser_notes	Sales_TabGroup_Shell_Request_Bio_Both	
Request_TabGroup_Slot_Orders	requestorders	Sales_TabGroup_Shell_Request_Bio_Both	
Request_TabGroup_Slot_Product_Instances	requestproduct_instances	Sales_TabGroup_Shell_Request_Bio_Both	
Request_TabGroup_Slot_Requests_Tab	requestrequests	Sales_TabGroup_Shell_Request_Bio_Both	
Request_TabGroup_Slot_Tasks	requesttasks	Sales_TabGroup_Shell_Request_Bio_Both	
*			

Content	Module*	Perspective	Special Form Type	List Order*
request_ui	request_ui		blank	
request_ui	request_ui	Sales_Toggle_Shell		11
request_ui	request_ui		blank	1
request_ui	request_ui	detail view		1
request_ui	request_ui	Sales_Toggle_Shell		9
request_ui	request_ui	Sales_Toggle_Shell		8
request_ui	request_ui	Sales_Toggle_Shell		4
request_ui	request_ui	Sales_Toggle_Shell		3
request_ui	request_ui	Sales_Toggle_Shell		2
request_ui	request_ui	Sales_Toggle_Shell		7

- 13 Add the following extension to the Save button on the toolbar that will be saving new attachments. "request\_detail\_view\_main Toolbar" and "")
- "verifycontentattachment" (should run first, before the saveaction) -requires the name of the tabidentifier specified in the "Tab\_Group\_Slot" form slot in the "Sales\_TabGroup\_shell" form.

Properties

Event/Extension Mapping - VerifyContent/Attachment

Properties

tab\_identifier generic\_tab\_identifier\_001

Labels

title

Form Widget - request\_detail\_view\_main Toolbar.save

Widget Name: save

Widget Type: form\_widget-toolbar button

Attribute Name:

Module: request\_ui

Actions Extensions EgExtensions Navigation Permissions

Event/Extension Mapping [4 Records]

Event	Extension*	Disabled?	Priority	Ext
save.clickEvent	VerifyContentAttachment	<input type="checkbox"/>	30	
save.clickEvent	SetNavigationProperty	<input type="checkbox"/>	100	
15540	SaveAction	<input type="checkbox"/>	100	
save.clickEvent	SetScreenTabContext	<input type="checkbox"/>	120	
*		<input type="checkbox"/>	100	

**14** Start the application server.

This chapter describes tools and procedures for customizing application online help systems for Infor Epiphany Sales, Infor Epiphany Service, and Infor Epiphany Self-Service users. Customization includes adding, omitting, or modifying help topic HTM files, as well as the project XML files that govern the Contents, Index, and Full-Text Search features. In a Ming.le integrated environment, the Documentation web part can also be customized. For more information, refer to the S&S Installation Guide).

## Overview

The Infor Epiphany Sales, Infor Epiphany Service, and Infor Epiphany Self-Service applications include context-sensitive online help. When you click the question mark icon on the toolbar within an application list or detail view, the help opens to a topic appropriate for that list or view. The context-sensitive topics often include links to other related procedural and conceptual help.

The application help systems are context-sensitive at the form level and are fully customizable. The out-of-the-box help matches the out-of-the-box application, with explicitly defined correspondence between a form and its help topic.

The help was developed using RoboHelp x5, and the output is in WebHelp format. WebHelp provides server-based help, integrating contents, indexing and full-text searching into one hypertext system using a combination of HTML, XML, DHTML, and JavaScript. Style within the help system is maintained by cascading style sheets. Content is presented within a three-pane frameset.

Infor Epiphany Sales, Infor Epiphany Service, and Infor Epiphany Self-Service application help may be customized using RoboHelp x5, a text editor, or an HTML editor.

**Note:** Earlier versions of RoboHelp HTML will not open the help project file correctly.

## Customizing Help Roadmap

Your customization roadmap should include:

- How to handle issues related to HTML authoring, including how to select a tool (to use RoboHelp or not; what to do if a different authoring tool is desired)
- How to set up a project from the source provided
- How to back up and protect source for your customization work
- How to map the help file to the form using Infor Studio in order to create context-sensitive topics
- What to do when help topics need to be added, modified, or omitted
- How to omit out-of-the-box help topics

## Selecting an Authoring Tool

This project has been authored with RoboHelp x5. Like most WYSIWYG HTML authoring tools, RoboHelp leaves artifacts such as paragraph styles within the BODY and the HEAD. Subsequent customization using RoboHelp is easier but not required, since the HTML is generic and the XML data files are rigorously defined.

If you are considering customizing with a different tool, you will need to modify the HTML files while leaving headers, footers, and CSS style tags intact. The data structure that generates the Contents and Index entries is stored in separate data files in the \whxdata folder. These files must be updated manually if you are not using RoboHelp. Instructions are included in this chapter.

**Note:** Tools that insert many artifacts into HTML files, such as Microsoft FrontPage or Macromedia Dreamweaver, may create conflicts within existing metadata. Infor recommends using either RoboHelp or a non-intrusive text/HTML editor.

If you are using a tool other than RoboHelp, avoid altering the following RoboHelp-specific information:

- Any <SCRIPT> tags
- The style sheet <LINK> tag
- <META> declarations
- <TABLE> tag for the frameset page

Additionally, RoboHelp uses such attributes as the “class” attribute of the paragraph <P> tag frequently. In some cases, you may need to mimic these usages to successfully format the topic.

## Locating Source Files

Source files for project customization and translation, including RoboHelp project files, are available from Infor Global download site at <https://infor.subscribenet.com>.

The installed application online help is located at <install\_dir>\shared\webroot\app\help\<app\_name>\<locale>. The default locale is en. For example, on a default Windows installation of Infor Epiphany Service, the complete path for application online help is:

```
C:\infor\shared\webroot\app\help\service\en
```

## Setting Up a Project

To get started, unzip Core\_7000\_docs\_olh\_src.zip. This archive contains the following application specific archives:

- SalesService\_7000\_olh\_src.zip
- Self-Service\_7000\_olh\_src.zip

Unzip the application-specific archive that you want to customize (with folder names intact). These archives create the following folder structure:

- epny\_help  
This is the root working folder for help customization
- Sales, Service, or Self-Service  
Depending on the application-specific help you're unarchiving.

The following folders are sub-folders of the application level folders:

- !SkinSubFolder!  
The help skin file.
- !SSL!  
This is the folder where the RoboHelp-generated WebHelp output files are stored.

An archived set of custom banners is also included that is called banners.zip. The contents of the banners.zip file is discussed in more detail in "RoboHelp Files Customized by Infor" on page 242

## Setting Up a RoboHelp Project

The application-specific RoboHelp x5 WebHelp project includes the project file (.xpj) and JavaScript (.JS), XML, and HTML files for the frameset and its banner. The data structure files are included in the \whdata, \whgdata, and \whxdata subfolders. (Because Infor applications support only later versions of Internet Explorer and Netscape Navigator, only the files in the \whxdata subfolder are actually required.)

- 1 Unzip the application-specific archive with folders intact to a folder on a local drive. This creates the \epny\_help\EpiphanyHelp folder for Customer Service or \epny\_help\Self-Service for Web Self-Service. .
- 2 Double-click the project file (.xpj) to start RoboHelp.
- 3 In RoboHelp, go to **File > Generate** Primary Layout to generate the project in your own test area before proceeding.  
Customer Service uses two layouts. Service WebHelp is the primary layout; Sales WebHelp is the other.

## Setting Up a Non-RoboHelp Project

- 1 Back up the installed help files or copy them to a working folder. The installed help is located at `<install_dir>/shared/webroot/app/help/<app_name>/<locale>`. The default locale is en. For example, on a default Windows installation of Infor Epiphany Service, the complete path for application online help is:

`C:\Infor\shared\webroot\app\help\service\en.`

- 2 Use the default.htm file in to view the contents, index, and search tabs in order to locate individual HTM and XML files.

To check your changes, you will keep this file open and refresh frequently to view changes, clicking nodes to view specific files you edit, add, or omit.

## Context-Sensitive Help Conventions

Each context-sensitive topic provides a brief overview of the form object that is currently active and how that relates to other forms or modules. Procedures of four steps or fewer may be provided in the current topic, but longer procedures require separate HTM file topics. Links to separate procedure topics and other related topics are provided in the context-sensitive topic.

Context-sensitivity is controlled within application-specific help based on the user's locale at runtime. The default.htm file for the locale is called, and the form's Help File property is passed to it as a bookmark with .HTM appended. For example, if the user's locale is English and the form is customer\_list\_view, the following URL is constructed from the web root folder:

`web/app/help/<app_name>/en/default.htm#customer_list_view.htm`

If a new form with a help button in its toolbar is included as part of your implementation, you must map a help file to it. Generally, the help file name is the same as the form name with .HTM appended. The help topic title tag and the top heading within the file should be the same as the label in the user interface if possible.

## Form Help Files

Within Infor Studio, each form includes a **Help File** property. This property is usually the same as the name of the form, except that spaces are not permitted in the help file property. Underscores replace any spaces. To locate the **Help Files** list in Infor Studio, click **User Interface > Maintenance > Form Help Files**. The **Help Files** list appears on the right. Once the **Help File** property is defined, you must add a help file called helptopic.htm to the project.

All forms with a help icon must have a corresponding help topic or a File Not Found (HTTP 404) error is produced. The help file property for forms that have no help icon is **no\_help\_topic**. The no\_help\_topic property calls the default.htm file. This provides a fail safe in case a help file is omitted or the no\_help\_topic property is improperly set for a form.



# Help Customization

Help customization procedures vary according to whether you are modifying existing topics, adding new topics, or deleting existing help topics. They also vary according to the tool you choose.

## Modifying a Help Topic

When you modify an existing help topic, you change, add, or delete text or graphics with the HTM file. This section includes a procedure for modifying help topics using RoboHelp and one for using a different tool.

### Modifying a Topic Using RoboHelp

- 1 Open the project file (.xpj) in RoboHelp..
- 2 Locate the file you want to modify.
- 3 Change, add, or delete text or graphics as desired.
- 4 Save your work.
- 5 Generate the project.
- 6 Copy the modified files in the \WebHelp and \WebHelp\art subdirectory to <install\_dir>/service/web/app/help/<app\_name>/<locale>, replacing existing files.
- 7 If you changed contents or index entries, replace the \WebHelp\whxdata folder and files in the <install\_dir>/service/web/app/help/<app\_name>/<locale> with new ones from your project.

### Modifying a Topic Using a Different Tool

If you need to change contents or index entries, follow the instructions provided in either "Adding a Topic Using a Different Tool" on page 241 or "Deleting a Topic Using a Different Tool" on page 238.

- 1 If you know the file name of the topic you need to modify, open the file in your editor.  
If not, open the default.htm file in a browser, locate the topic from the Contents, and then open the file.
- 2 Locate the <TITLE> tag and modify the title, if necessary. This tag appears in users' help searches.
- 3 Locate the <H1> tag for the help topic title, and edit the text as you would any HTML document.
- 4 Make sure the modifications appear correctly by doing the following:
  - a Open the default.htm file in a browser, and find the modified file, or
  - b Open a browser and pass default.htm#<filename>.htm to it.
- 5 Copy the modified files to <install\_dir>/service/web/app/help/<app\_name>/<locale>, replacing existing files.

## Deleting a Help Topic

When implementers omit an out-of-the-box form during customization, the help must also be updated to omit the corresponding help topic and any links to it. This includes any references in the contents. You may also need to revise procedural topics if they included a reference to the form.

### Deleting a Topic Using RoboHelp

- 1 Open the project file (.xproj) in RoboHelp.
- 2 In the Project tab, right-click the file you want to remove, and select Delete from the shortcut menu. A message box may open with the following prompt:  
The file, <file path and name>, will be deleted from disk, causing references to that file to no longer work. Would you also like to remove those references in other files?
- 3 Click Yes if prompted. Any links and contents entries for the deleted file are removed.
- 4 Generate the project.
- 5 Delete the file from <install\_dir>/service/web/app/help/<app\_name>/<locale>.
- 6 Replace the \WebHelp\whxdata folder and files in the <install\_dir>/service/web/app/help/<app\_name>/<locale> with new ones from your project.

### Deleting a Topic Using a Different Tool

- 1 Identify the form that has been removed.
- 2 Remove the corresponding help file from the project folder.
- 3 To locate links to the deleted file, do a full-text search for within all HTM files for the file name. Modify those files accordingly.
- 4 Remove references to the deleted file in the help contents as follows:
  - a In the \whxdata subdirectory, search all of the whtdata<#>.xml files (where <#> represents an integer equal to or greater than 0) for the name of the file that you deleted.
  - b In each whtdata<#>.xml file where you find the file name, delete the item line that contains the file name. The following is an example of an item line from the whtdata0.xml file:  

```
<item name="Using Online Help" url="using_online_help.htm" />
```
- 5 Remove references to the deleted file in the help index as follows:
  - a In the \whxdata subdirectory, search all of the whidata<#>.xml files (where <#> represents an integer equal to or greater than 0) for the name of the file that you deleted.
  - b In each whidata<#>.xml file where you find the file name, delete the topic line that contains the file name. For example, if you deleted the topic file organization\_competitor\_detail\_view.htm, you

would delete the entire line shown in bold in the following snippet from the whwidata0.xml file for the Sales Force Automation application.

```
<key name="competitor" >
  <topic name="Adding a Competitor to an Opportunity"
    url="adding_a_competitor_to_an_opportunity.htm" />
  <topic name="Competitor"
    url="organization_competitor_detail_view.htm" />
</key>
```

- c Examine the entire key to determine if the whole thing should be removed. For example, in the previous snippet, you are removing the help for the competitor detail view. Is that because your implementation does not track competitors? If so, you would want to remove the entire key.
- 6 Remove references to the deleted file in the full text search as follows:
  - a In the \whxdata subdirectory, search all of the whftdata<#>.xml files (where <#> represents an integer equal to or greater than 0) for the name of the file that you deleted.
  - b In each whftdata<#>.xml file where you find the file name, delete each topic line that contains the file name.
- 7 Test the help in your browser.
- 8 Delete the file from <install\_dir>/service/web/app/help/<app\_name>/<locale>.
- 9 Copy updated whtdata<#>.xml, whidata<#>.xml, and whftdata<#>.xml files to <install\_dir>/service/web/app/help/<app\_name>/<locale>/whxdata, replacing existing files.

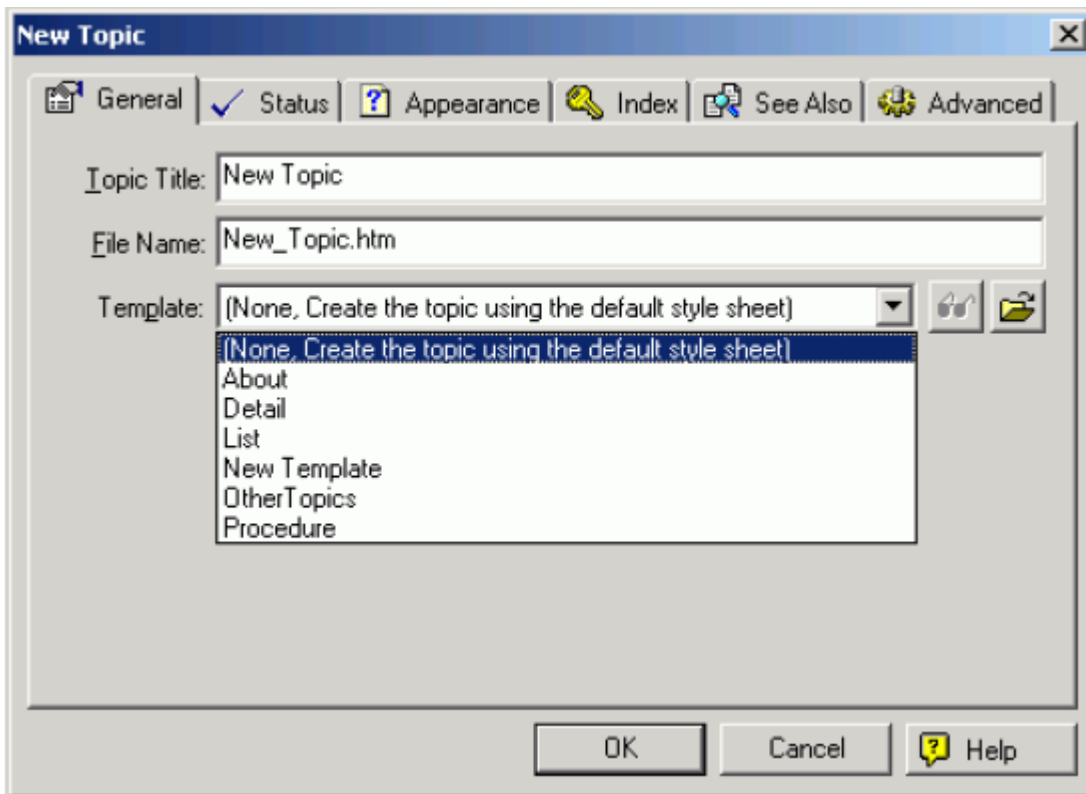
## Adding a Help Topic

When implementers add a form that includes a help button during customization, you may add help for it. If you do not add help, be sure that the help file for the new form is set to `no_help_topic` in order to avoid file-not-found errors. See "Form Help Files" on page 236.

If you choose to add a context-sensitive topic, and perhaps other procedural topics, you may also add them to the contents, index, and full-text search, regardless of which tool you use. The procedures to do so are included in this section. To add links to your new topics, follow the instructions in "Modifying a Help Topic" on page 237.

## Adding a Topic Using RoboHelp

- 1 If you plan to use image files in your new topic, copy them to the <project> /Art folder so that they are available while you create the new topic.
- 2 Open the project file (.xprj).
- 3 Press **Ctrl+T** to open the New Topic tabbed dialog box.



- 4 Enter a **Topic Title** . This becomes the <TITLE> tag for the document, so it should be a user friendly name. Often this is the same title used in the first heading of the topic.  
The **File Name** is automatically created.
- 5 If this is a context-sensitive topic, edit the **File Name** to match the form help file name plus the htm extension.  
**Note:** If you are running your application server or Web server on UNIX or Linux, be sure the file name is in lower case to avoid potential issues with case-sensitivity.
- 6 Select an appropriate **Template** from the drop-down list.
  - Use CommonElements for topics that are common to both Customer Service. The Customer Service help use the same source files and RoboHelp's conditional build tags to generate specialized output as needed.
- 7 On the **Index** tab, add keywords for the topic, and click **OK** to open the new topic in the **WYSIWYG** editor.
- 8 Generate the project.
- 9 Copy the new files in the \WebHelp and \WebHelp\art subdirectory to <install\_dir>/service/web/app/help/<app\_name>/<locale>.
- 10 Replace the \WebHelp\whxdata folder and files in the <install\_dir>/service/web/app/help/<app\_name>/<locale> with new ones from your project.

## Adding a Topic Using a Different Tool

- 1 Open the default.htm file in a browser window, and keep the browser open as you work to view your work. .
- 2 Open a help topic file that is similar to your planned new topic in your editor.
- 3 Save the file with a new name, using the form help file name plus the .htm for context-sensitive topics.

If the file is not context-sensitive, give it a name that makes it easy to identify the contents. Use only lower-case alphanumeric characters and underscores in the file name.

- 4 Edit the <TITLE> tag to give your new topic a user-friendly name. This appears in full-text search results.
- 5 Locate the <H1> tag, and replace its contents with the topic title the users will see.
- 6 Edit the other text in the file as needed.

Avoid altering the following RoboHelp-specific tags in the file:

- Any <SCRIPT> tags
- The style sheet <LINK> tag
- <META> declarations
- <TABLE> tag for the frameset page

- 7 Save your new file.

- 8 To add the new topic to the contents, do the following:

- In the \whxdata subdirectory, examine the whtdata<#>.xml files (where <#> represents an integer equal to or greater than 0) to determine where your new topic belongs.
- In the whtdata<#>.xml file where you want to add the topic, add a new item line for your topic as follows:

```
<item name="New Topic" url="new_topic.htm" />
```

The value of the `item name` is how your topic will be listed in the contents. The value of the `url` is the topic's file name.

- 9 To add references to the new file in the help index, do the following:

- a Open the \whxdata\whidx.xml file to identify the range of keyword in the whidata<#>.xml files. Below is a sample of the contents of this file:

```
<?xml version='1.0' encoding='windows-1252' ?>
<index>
  <chunkinfo url="whidata0.xml" first="About"
    last="Microsoft Outlook" num="123" />
  <chunkinfo url="whidata1.xml" first="MIME"
    last="Uploading" num="116" />
  <chunkinfo url="whidata2.xml" first="User"
    last="Workflow Designer" num="16" />
</index>
```

- b** In the `whidata<#>.xml` files, follow the syntax to add your new topic to an existing key name or to add a new key name referencing your new topic. The following is a syntax sample:

```
<key name="offer" >  
    <topic name="About Offers" url="about_offers.htm" />  
</key>
```

The value of `key name` is the index entry, the value of `topic name` is how your topic will appear in the index, and the value of `url` is the new topic's file name.

- 10** Test the help in your browser.
- 11** Add the file from `<install_dir>/service/web/app/help/<app_name>/<locale>`.
- 12** Copy the updated `whdata<#>.xml` and `whidata<#>.xml` files to `<install_dir>/service/web/app/help/<app_name>/<locale>/whxdata`, replacing existing files.

## RoboHelp Files Customized by Infor

The following RoboHelp auto-generated files have been customized by Infor so that build information is captured within the help banner. Any time you generate the help project using RoboHelp, it will attempt to overwrite these files. These files are included in your application specific archive in the file `banners.zip`. Checking in any of these files to your source will result in a regression that breaks this structure:

- `pixel.gif`  
The About button image file.
- `whskin_banner.htm`  
The banner window layout file.
- `whskin_tbars.htm`  
The help window toolbar includes a button to click to get build information.
- `wht_abgw.jpg`  
The first banner image file.
- `wht_abtw.jpg`  
The second banner image file.
- `whtbar.js`  
The JavaScript that provides rollover text for the About button.

This chapter covers how to handle *time zone* in Infor Epiphany Service.

Customer Service can handle date and datetime values in different time zones based on user needs. You can use time zone settings to modify the time zone in which the date is displayed on and processed from the user interface, and the time zone in which the date data is stored in the database.

## UI Display

For UI display, you can set the time zone at the widget level, at the BIO attribute level, and at the user preference level.

## Widget Timezone

Some widgets such as the date-picker and datetime-picker have a timezone property. When this timezone property is set, the date value that is displayed is converted to the time zone specified and dates that are input are assumed to be in this time zone. This setting is applicable for the widget across all users.

The default value for the widget timezone setting is blank. If it is blank, the value of the property from the BIO attribute is used.

You can use Infor Studio to set the timezone property on a widget (**Guide Bar > User Interface > Bio Forms > Select a form > Select the widget > Properties > Timezone** ).

## Datetime Widget Timezone Attribute Domain

In addition to the timezone property, the datetime widget also has a timezone attribute domain property. This property is designed to work in conjunction with BIO attributes whose Localized Data Type is set to those described in section "Variable Timezone (VARIABLE\_TIMEZONE)".

The timezone attribute domain property allows the user to specify a time zone for entering a datetime value. This setting is applicable for the widget across all users.

The default value for widget timezone attribute domain setting is blank. If it is blank, the option to choose a time zone is not provided.

To set the timezone attribute domain on a widget go to **Infor Studio > Guide Bar > User Interface > BIO Forms > Select a form > Select the widget > Properties > timezone attribute domain**.

**Note:** Due to some limitations induced by the Infor SoHo (UX 3.0) implementation, datetime picker widgets are rendered as date only widgets on list view search fields (filter row and more fields). In this case, the timezone dropdown will be hidden.

## BIO Attribute

You can set the timezone value on a BIO attribute of type date. The timezone can be set statically at design time or dynamically at runtime. Both these methods are described below.

### Timezone

When the timezone parameter of a BIO attribute is set, that date attribute is converted to the time zone specified before being displayed to the user, and inputs from the user are assumed to be in this time zone. This timezone setting is applicable for the attribute across all users. If the recordset field has a time zone of GMT, and you set the BIO attribute time zone to GMT, no time zone conversions will occur.

The default value for BIO attribute timezone setting is blank. Blank means convert to the user's time zone.

You can use Infor Studio to set the timezone parameter of a BIO attribute. Do the following:

- 1 Navigate to a BIO attribute of type date (**Guide Bar > Physical & Logical Schema > BIO Related Info > BIOs > Select a BIO > Select an attribute of type Date** ).
- 2 Double click on the attribute row to open the **BIO Attribute** form.
- 3 For **Localized Data Type**, select TIMEZONE.
- 4 For **Timezone**, select the time zone that you want from the drop-down.

## User Preference

The timezone setting in user preference is applicable to all date attributes displayed across the application for that user. User preference timezone is optional and can be set to blank in which case the date-time is displayed in the browser time.

Out-of-the-box, user preference defaults to Pacific Time.

To set the Timezone user preference, log into the Customer Service application and navigate to the **User Preferences** screen. (**Navigation > User Preferences > Regional & General > Timezone** ).





**Caution:** There is a hierarchy of settings on processing dates through the UI. Widget timezone overrides BIO Attribute timezones. BIO Attribute timezones override User Preference and User Preference overrides the workstation timezone setting as transmitted by the browser. These conversions, configuration, and hierarchy also apply to dates entered by the user (on the way in) and sent to the BIO layer.

## Database Storage

For database storage, you can use the date field timezone setting.

### Data Field Timezone

You can set the timezone value on a date data field. When this value is set to a time zone value, the date is converted and stored in the time zone specified. If you have a system where all the datetime values are stored in Pacific Time and you set the field's timezone to GMT-8 Pacific Time, the BIO service interprets (reads and writes) the dates as Pacific Time.

The default value for the timezone setting is blank. Blank implies GMT, therefore the values are stored and interpreted as GMT.

You can set the time zone on a field level from Infor Studio (**Physical & Logical Schema > Recordsets & Relationships > Recordsets > Pick a recordset > Pick the field of type > Date > Time zone**).

Change the timezone value only before initial use of a production system. If you change the time zone on a production system with existing data, you have to manually update the database and convert the dates in the existing records to be in the new timezone.

The data-field timezone setting described above for writing to a database applies equally to reading a date from any system (including a read-only system), and also applies to writing/reading from EAI data sources.

## Out-of-the-Box Behavior

Every date and datetime value goes through two conversions, one at the data layer on its way in and out of the database, and one at the UI layer for display and processing of user data.

Out-of-the-box, the timezone setting for all date attributes is such that the date displayed on the UI is based on the timezone set at user preference and the date stored in the database is always in GMT. Widget timezones, variable timezones, BIO attribute timezones and data field timezones are not usually set.

Irrespective of the display and the database date timezone, all the Infor Epiphany Service internal processing is performed in GMT. Therefore, if you get a date BIO attribute in an extension, the value is always in GMT.

Note that if you change any timezone parameters at the recordset level, you should also make sure that any scheduled jobs that run queries on dates are updated appropriately.

## Use Cases

### *Use case 1*

User needs to see interactions in the date and timezone in which the user creates them.

Set *User Preference Timezone* value to user's time zone. The date value entered on the user interface will be in the user's time zone and will be converted and stored as GMT in the database. For display, the database GMT date value will be converted to the user's time zone by the application.

### *Use case 2*

A user in San Francisco enters a record at 9 p.m Pacific Time and another user enters a record at 8 p.m Central Time. The manager needs to know which record was created first in the system regardless of the time zone in which the record was entered.

Set *User Preference Timezone* value to manager's time zone. Date is stored and processed in GMT regardless of the time zone used for data entry. Therefore, sort will be correctly ordered by creation time.

### *Use case 3*

Enter and display birthday (no conversion for different users)

Set widget timezone or BIO attribute timezone to match the timezone setting for the corresponding recordset field. If you leave the recordset timezone as blank, set the widget timezone or BIO attribute timezone to GMT.

### *Use case 4*

Display date coming from a legacy system in a specific time zone.

- 1 If the date in the legacy system is not GMT, sets the data field's timezone to be the same as the date's timezone from the legacy system.
- 2 Set the widget timezone or BIO attribute timezone to the required display timezone.

### *Use Case 5*

Need to run a query/ reporting system on the databases using combined data from Infor Epiphany Sales, Infor Epiphany Service, and Infor Epiphany Self-Service and a legacy system where date is not in GMT.

Set the data field's timezone to be the same as the legacy system so that data is converted to GMT before being processed.

### *Use Case 6*

Need to show one e\_appointment BIO in Pacific Standard Time (PST) and another one in Greenwich Mean Time (GMT).

In the e\_appointment BIO, create a BIO attribute of type String to store the time zone, for example, biotimezone. For each date in the BIO that should use the timezone, set the *Localized Data Type* to VARIABLE\_TIMEZONE and the *Related Localized Attribute* to biotimezone. Create a widget on the appropriate form that allows the user to specify the time zone value for the biotimezone attribute. Or use some other method to set the biotimezone attribute to a timezone name when a user creates a BIO instance.

#### *Use Case 7*

Set a date value for UI display using an extension, say as a label on a screen.

Irrespective of the timezone values set, the date in the extension will be in GMT as all processing of dates within the application is in GMT. You will have to convert the date to the correct timezone in your extension before setting the value. Similar processing is required if a screen date has to be stored in the database as a string and a field level timezone value is set to anything other than GMT.

#### *Use Case 8*

For timezone-less dates like birthdays, set the display timezone for the logical date (defined on the BIO attribute) to match the timezone storage format for the physical date (defined on the corresponding recordset field). By doing this, you will get no conversions when displaying the date. For the out-of-the-box operational database, the default storage format is GMT, therefore, set the attribute to GMT.

#### *Use Case 9*

The recordset lives in an external data source that does not store the dates in GMT.

For example, all date and times in a customer's system may be stored in GMT-8 (Pacific Time since the client servers are located in Seattle). For this case, you set the Timezone in the recordset to GMT-8 and the display format in the BIO to GMT-8. You still get the same timezone-less behavior and the Infor core can read and write these date fields without conflicting with the online system that owns the data source.



This chapter describes *internationalization* (I18N) issues related to Infor Epiphany Service and covers user locale, date and time, numbers, and currencies. The Infor Epiphany Service application provides flexibility for customization and also provides defaults so that most of the time you can work with the defaults and modify the defaults only when required.

## User Locale

User locale is used to display all the text a user sees. The user locale is determined using the following order of precedence:

- Default locale
- Browser locale
- Locale user preference

## Default Locale

The default locale is the installed locale with the attribute value set to Default.

## Browser Locale

The server uses all the locales specified by the browser in the header and takes the first one that is an installed locale or has a parent that is an installed locale in Infor Studio.

## Locale User Preference

Users with administrative rights can set the locale they prefer in the Infor Epiphany Service application.

## Date and Time

Four parameters are involved in how a date BIO attribute is rendered in the user interface. These are Format, Locale, Time zone, and Daylight saving.

The formatted string is in the specified format and locale, and the time is shown in the specified time zone adjusted for daylight saving (if the date is in daylight saving).

When you use the date picker widget (not the datetime picker widget), the time zone and daylight saving is ignored and the date is displayed and set as if it was noon GMT of that day. The next section describes how the four parameters are determined.

## Format

The format is determined by the format property of the form widget. This is set in Infor Studio on the widget properties.

For form widgets linked to DATE BIO attributes, the format property is one of the following:

- xxxDate yyyTime
- Custom
- xxxDate
- xxxTime

where xxx and yyy are either Full, Default, Long, Medium or Short.

If xxx or yyy is Full, Long, Medium or Short, the date or time part of the format is what Java specifies for that locale.

If xxx or yyy is Default, it will result in either Full, Long, Medium or Short. This is determined using the following order of precedence:

- Default formats
- User preference formats

If it is Custom, the datetime format label of the form widget determines the format. The value is set in Infor Studio on the widget properties under Labels/Datetime format, and it should follow the `java.text.SimpleDateFormat` specification.

Typical use would be to use “DefaultDate” or “DefaultDate DefaultTime” so that all dates would appear formatted the same way, and the user can override them in user preferences.

The default value for the format parameter is NULL which in this case means “DefaultDate DefaultTime” except if used with the date picker for which it is “Default date” because the time is not displayed.

## Default Formats

The default date format and the default time format are text messages called DEFAULT\_DATE\_FORMAT and DEFAULT\_TIME\_FORMAT and they can be different to match specific country preferences. These text messages must contain the English word Full, Long, Medium or Short (not the translation of Medium or other in other languages).

The default values are set in Infor Studio under **Customization and Business Processes > localization and Text > Text Messages > .**

## User Preference Formats

Users can pick their preferred date and time format in the User Preference screen in Infor Epiphany Service by entering Full, Long, Medium or Short. The user preferences are called “date format” and “time format.”

## Locale

### User Locale

See the section on "User Locale" on page 249.

## Time zone

Time zone is determined using the following order of precedence:

- Local time zone (the browser time zone, not the server time zone)
- Time zone user preference
- localized\_data\_type BIO attribute column
- Time zone widget parameter

Typical usage is to not set time zone user preference, localized\_data\_type BIO attribute column, and time zone widget parameter. This results in dates showing in local time zone and the user can override this by changing user preferences (time zone user preference).

The time zone table contains all time zones plus a row for “UserTimeZone.” If in localized\_data\_type BIO attribute column or time zone widget parameter, the time zone is determined to be “UserTimeZone,” the time zone is determined by local time zone or time zone user preference.

## Local Time Zone

This is the time zone of the browser. The offset is with GMT. So if this determines the time zone, the times will appear with time zone named GMT+/-H:mm instead of a string such as Pacific Standard Time (PST). For example, if the browser time zone was PST, it shows GMT-8:00 in winter and GMT-7:00 in summer during daylight saving.

## Time Zone User Preference

One of the user preferences is “time zone.” It is a string which is a time zone name.

## localized\_data\_type BIO Attribute Column

If the localized\_data\_type is TIMEZONE, the timezone\_id BIO attribute column determines the time zone.

If the localized\_data\_type is VARIABLE\_TIMEZONE, the polymorphic\_bio\_attribute table gives you a link to another BIO attribute of the same BIO that contains a time zone name STRING similar to the ones defined in timezone\_name in metadata. For performance reasons, there is no support for GUIDs as foreign key to another table. In Infor Studio, if you double-click on a BIO attribute, you see “Related Localized Attribute.” This allows having different instances of the same BIO in different time zones, for example, an Appointment BIO could be in PST and another one in GMT. At runtime, the time zone is determined by the value of the BIO attribute. If the value is null, an exception is thrown.

## Time zone Widget Parameter

Widgets have a parameter named **timezone**. It is a drop-down of all time zones in E.piphany Studio. The default value is NULL. This value is customizable at run time.

## Daylight Saving

If the date to display is in daylight saving for the specified time zone, the date is adjusted to daylight saving. For example, if the date is December 1st, 9 p.m. GMT and the time zone is PST, the formatted date will be December 1st, 1 p.m. PST. If the date is July 1st, 9 p.m. GMT and the time zone is PST, the formatted date will be July 1st, 2 p.m. PDT.

## Numbers

For Decimal, Float, and Int BIO attributes, two parameters are required in formatting the numbers. The parameters are format and locale.

The UI displays numbers in a format determined by the locale and format parameter. The user can modify the number and, as long as the user respects the decimal point, the server can unformat the



number (that is, the user does not need to enter grouping (thousand) separators. The user can add spaces at the beginning or end and can use - or () for negative etc).

For decimal, float or int BIO attribute that are not currencies, the `localized_data_type` BIO attribute column should be NORMAL or NULL.

## Format

The format is determined by two form widget properties: format property and precision property.

### Format Property

The format property is set in Infor Studio on the widget properties and it can be Normal or NoGrouping. The default value is NULL which means Normal. NoGrouping, percentage, and percentage NoGrouping means no thousand separator.

Typical use would be to leave the format property to its default value. If you prefer without the thousand separator, change the default value of this parameter to NoGrouping.

### Precision Property

The precision property is an integer specifying the number of decimals. All numbers have exactly a specific number of digits, even if the trailing digits are zeros.

If precision property is not specified by default, it will be the default precision of that locale. For most locales, it is from 0 to 3 digits.

If the BIO attribute is an INT, the property is ignored and no decimals are shown. The decimal symbol display depends on the locale.

## Locale

The locale determines what symbols are used and the formatting rules on top of the rules described above. For more information, See "User Locale" on page 249. One of the user preferences is "number currency locale." Users can specify in their user preferences a different locale and currency formatting. Other wise, it is provided by the user locale.

## Currencies

Three parameters are involved in how currency attributes are formatted. The parameters in order of precedence from higher to lower are:

- Format

- User's currency locale
- Global default currency locale

The UI displays numbers in a format determined by these parameters. The user can modify the amount and, as long as the user respects the decimal point, you can unformat this amount (that is, the user does not need to enter grouping (thousand) separators, can add spaces at the beginning or end, can use - or () for negative etc).

For decimal, float or int BIO attribute that are currencies, the `localized_data_type` BIO attribute column must be `CURRENCY` or `VARIABLE_CURRENCY`.

Infor discourages using a float for currencies.

The default user locale is used for determining the decimal symbol and the grouping symbol. The actual currency being used determines everything else (currency symbol, position of the currency symbol, default number of digits, negative number formatting etc).

There is no support for scale factors to display amounts in thousands or millions. You have to implement this as a computed attribute.

## Format

The format is determined by two form widget properties: Format property and Precision property.

### Format Property

The format property can be

- Normal
- NoGrouping (no thousand separator)
- NoCurrency (no currency symbol)
- NoGrouping NoCurrency

The default value is `NULL`, which means Normal. Leave this property to its default value and change the default value according to customer requirements.

### Precision Property

If Precision property is not specified by default, it will be the default precision of that currency. For most locales, it is two digits (with trailing zeros if needed).

Put 0 to have digits after the decimal. The symbol used to display the decimal point depends on the user's default currency locale.

If the BIO attribute is an INT, this property is ignored and no digits after the decimal point are shown. Again, the symbol displayed for the decimal point depends on the user's default currency locale.

## Currency Locale

The default currency locale determines what symbols are used and the formatting rules on top of the rules described above, except the decimal symbol and the grouping symbol.

For example, if the user's default currency locale is en\_US, a US dollar amount would be \$1,234.56 and a Euro amount would be 1,234.56E. If the user's default currency locale is fr\_FR\_EURO, a US dollar amount would be \$1 234,56 and a Euro amount would be 1 234,56E. (Note: E is the Euro sign.)

European countries have migrated to the Euro but fr\_FR, de\_AT, and so on, in JDK 1.3.1 do not use the EURO.

Locale information is required for currency formatting (for example, for formatting a EURO currency). The currency lookup attribute domain in business data has the locale\_name as code\_string. The currency lookup appears as follows:

Label	Code String
United States Dollar	en_US
Canadian Dollar	en_CA
Euro	fr_FR_EURO (change this to another EURO locale to format in that locale)

Canadian dollars are formatted the English way and not the French way and Euros are formatted the French way. Customization can change this lookup to tailor it to a customer's preference.

The currency locale is determined using the following order of precedence:

- Default currency locale
- Localized\_data\_type BIO attribute column

Typical usage is to not set localized\_data\_type BIO attribute column. This results in currencies showing up in the default currency; except for BIOs which can have amounts in different currencies (see below).

## Default Currency Locale

The default currency locale is a global setting called defaultCurrencyLocale. This determines the defaultCurrency Locale of each user. A user can modify his or her own defaultCurrencyLocale under User Preferences.

### localized\_data\_type BIO Attribute Column

For an attribute to be displayed as a currency, it must have the localized\_data\_type currency. If the localized\_data\_type is CURRENCY, the currency locale column determines the currency locale.

If the currency\_locale\_id is NULL, the default currency locale defined above is used.

If the localized\_data\_type of an attribute is VARIABLE\_CURRENCY, another BIO attribute of type STRING in the same BIO will contain a locale name.

In Studio, on the attribute form for the variable\_currency attribute, the attribute containing the locale needs to be specified on the **Related Localized Attribute** form. This allows having different instances of the same BIO in different currency locales, for example an Order BIO could be in US dollars and another one in Euro. At runtime, the currency locale is determined by the value of that BIO attribute. If the value is null, an exception is thrown.

## User Locale

The user locale determines what decimal and grouping symbols are used.

## Overview

User satisfaction is a key aspect to a successful business application. The user interface has been completely redesigned to bring a better and richer user experience. This is achieved by making conscious design changes to comply with the Infor SoHo Visual Style guidelines (also referred to as UX 3.0 in this document).

The Infor SoHo (UX 3.0) Visual Style guidelines is the authoritative resource for UI design and interaction standard of Infor business applications.

An HTML5-based control library (inforControlsCombined.js) and CSS style sheet (inforControlsCombined.css) are included as part of this guidelines to achieve the unified Infor SoHo (UX 3.0) Visual Style Guide look and behavior.

Part of the design changes include integrating the current widgets library on the client side with the Infor SoHo (UX 3.0) HTML5-based control library and modifying the existing CSS style sheets to conform to the look and feel defined by the Infor SoHo (UX 3.0) Visual Style guidelines.

Some of the notable UI changes are listed below:

- More user friendly UI widgets.
- Change of the color theme of the application.
- Support for 'hover' effect over Text/Icon Toolbar button and Dropdown Menu.
- Configurable display options for Toolbar button and Dropdown Menu: Image Only, Label Only or Image and Label.
- Support for HTML5 'placeholder' attribute has been decluttered and the application top banner has been simplified.
- Repositioned and reoriented the application navigation menu.
- Use of message dialogs instead of 'pop-up' windows

This chapter describes the Studio related configurations that enable some of these features.

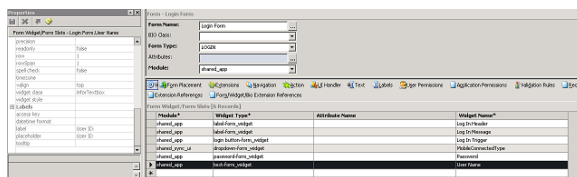
# Enabling Infor SoHo (UX 3.0) Features

## 'Placeholder' for Input Fields

HTML5 extends the input element with a new attribute named 'placeholder'. The placeholder attribute specifies a short hint that describes the expected value of an input field (for example, a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value. The short hint should clear when the cursor enters the field.

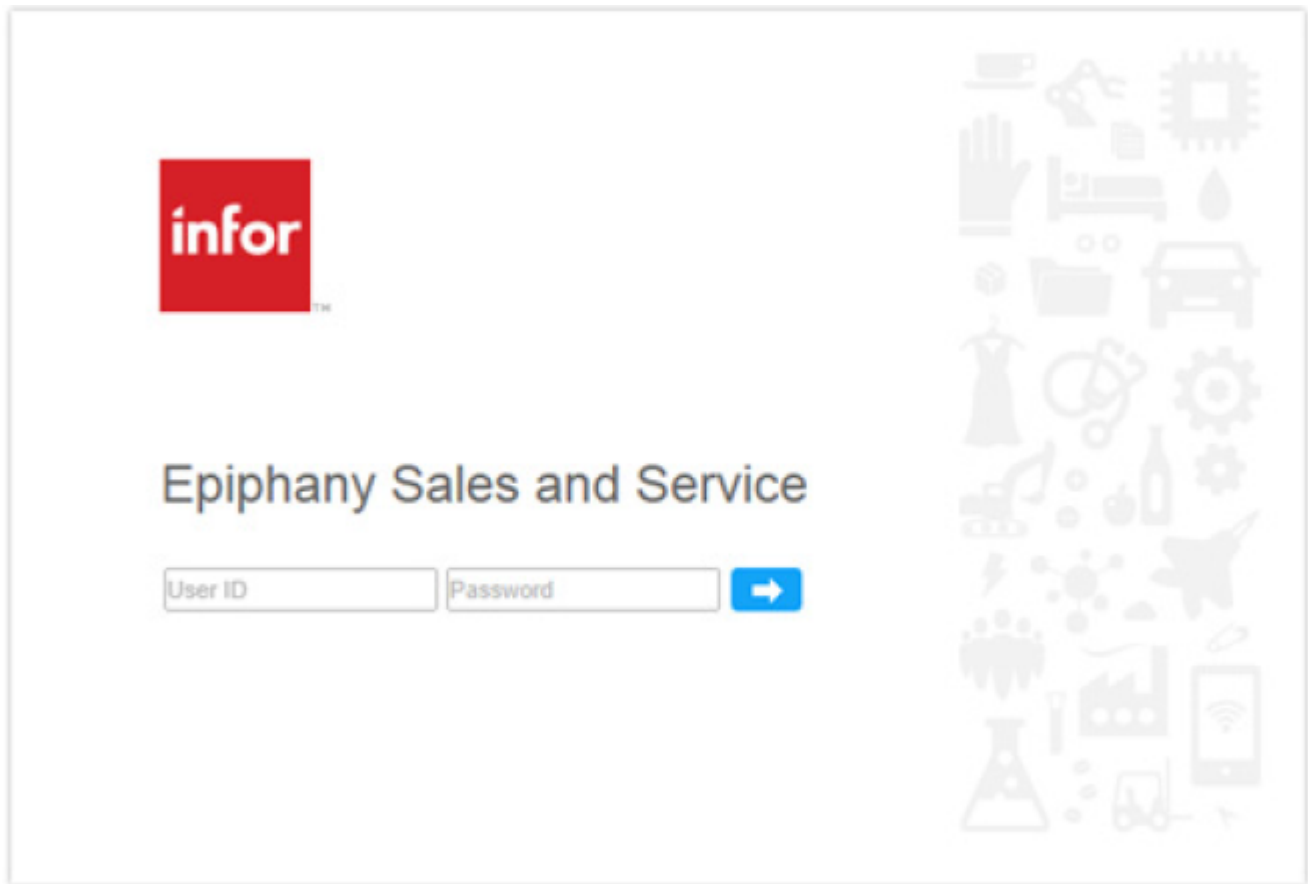
The text widget has been extended with a new label type named 'placeholder' to allow defining a placeholder attribute value for the text widget. You should see this new label type in the Properties section of any text widget in Studio.

For example, the 'User Name' and 'Password' widgets on the Login form have placeholder labels 'User ID' and 'Password' respectively.



**Figure 3: placeholder' Configuration in Studio**

With the placeholder configured for these widgets, you will see the short hints appearing in the input fields for User Name and Password on the Login form:



**Figure 4: 'placeholder' Display on Sign-in Screen**

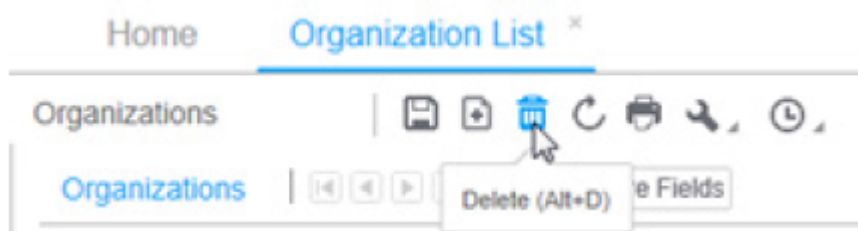
**Note:**

- 1 Support for 'placeholder' configuration for other widget types will be considered in future releases.
- 2 The 'placeholder' attribute is supported in Internet Explorer 10, Firefox, Chrome, Safari and Opera. The 'placeholder' attribute of the <input> tag is not supported in Internet Explorer 9 and earlier versions.

## Hover Image for Toolbar Button and Dropdown Menu

You may now associate a hover image to a toolbar button and dropdown menu. The hover image is the image that is displayed when you mouse- over a toolbar button or a dropdown menu that shows an image.

There is no need to configure anything in Studio for the hover image. All that is needed is to name the hover image appropriately and add it to the images folder of the application. You should name your hover image by appending '\_hover' to the file name of the image associated with the toolbar button and dropdown menu. For example, if the image property value for the toolbar button is 'images/icon\_delete.gif', the corresponding hover image should be named as 'icon\_delete\_hover.gif'.



**Figure 5: Hover over 'Delete' Toolbar Button**

## Configurable Display Options for Toolbar Button and Dropdown Menu

The toolbar button widget and dropdown menu have been extended with a new widget property named 'show' to allow setting the display option. The options are: 'Image only', 'Label only' or 'Image and Label'. This setting applies only if both image and label properties are configured for the toolbar button and dropdown menu.

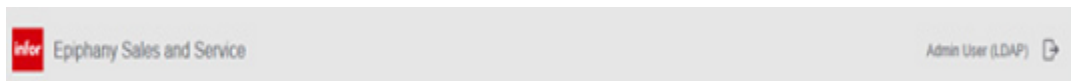
The default 'show' preference and 'widget class' for the toolbar button and dropdown menu is 'Image Only' and 'inforIconButton', respectively. If you decide to change the 'show' preference to 'Label Only' or 'Image and Label', you will need to reassign the 'widget class' to 'inforTextButton' which allows the widget to be larger. Otherwise, the toolbar button or dropdown menu may appear truncated as shown in the screenshot below (for the 'New' button).



**Figure 6: Truncated Toolbar Button (New)**

## Application Top Banner

The application top banner has been simplified and decluttered. The new top banner now only includes the logo, application name, user name and a Sign Out button (as per Infor SoHo (UX 3.0) guidelines).



**Figure 7: Application Top Banner**

The application top banner is modelled in Studio using the special form 'Top Banner' found in **User Interface > Forms > Other Forms**. Consequently, the content of the TopSlot of the Root Screen in the Login Form Navigation associated with a successful login has also been remapped to reference this form.

If you wish to change the logo or the application name in the application top banner, or even customize it very differently from the one shipped out-of-the-box, this is the form to apply your customization.



**Note:** The application top banner is hidden by design in a workspace integrated environment.

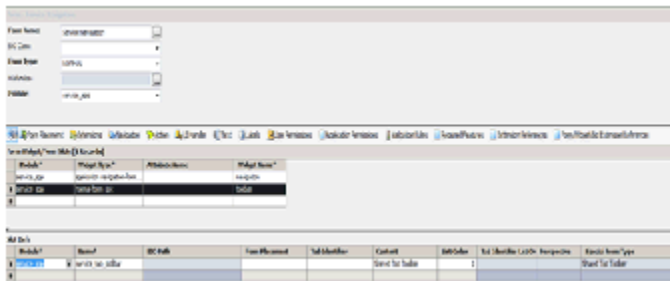
## Application Top Toolbar

As part of decluttering the application top banner, the toolbar that appear in the top banner in previous release has been repositioned to the right of the application navigation bar:



**Figure 8: Application Top Toolbar**

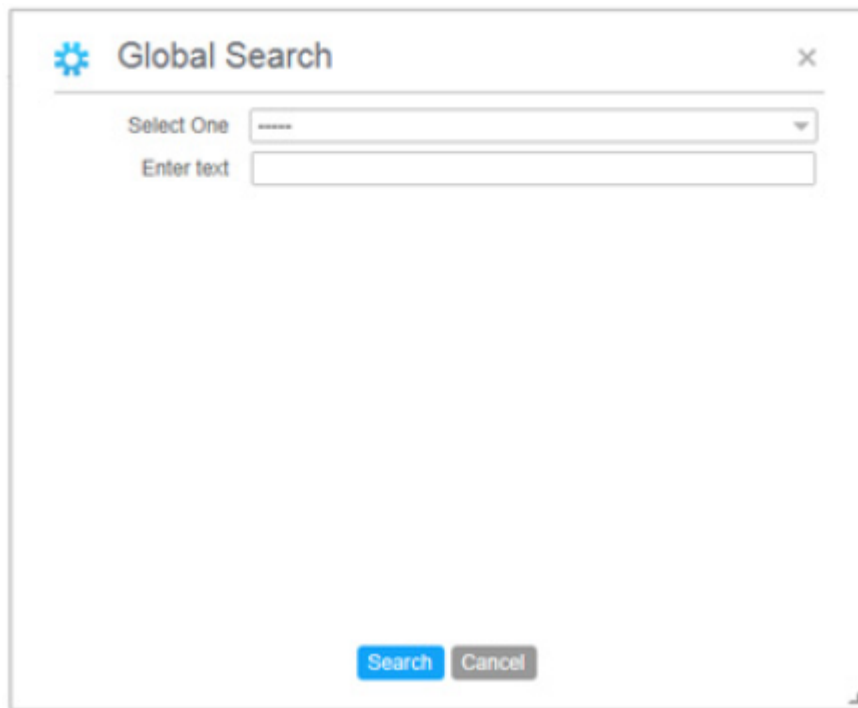
You may find the form slot containing this toolbar in the 'Service Navigation' and 'Sales Navigation' form.



**Figure 9: Application Top Toolbar in Service Navigation Form**

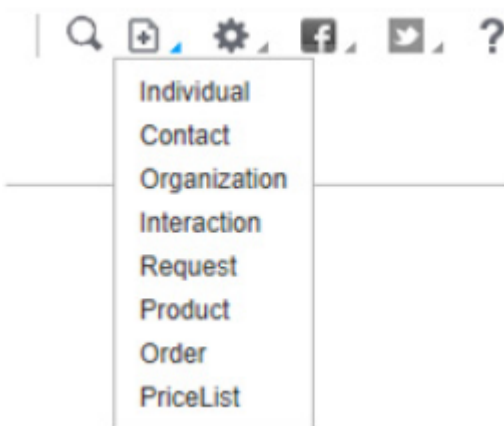
Out-of-the-box, the application top toolbar has been extended with a 'quick search' button and a 'quick add' dropdown menu.

Clicking on the 'quick search' button shows the 'Global Search' message dialog that allows you to perform quick searches on selected BIOs similar to the 'Find' section seen in the left navigation section in previous releases.



**Figure 10: Global Search message Dialog**

The 'quick add' dropdown menu allows quick access to create new items without having to navigate to the item list view and click on the 'New' toolbar button.



**Figure 11: Quick Add dropdown Menu**

If you wish to further customize the application top toolbar, you may navigate to **Studio > User Interface > Toolbars > Shared Top Toolbar**.

## Application Navigation Menu

In order to comply with the Infor SoHo (UX 3.0) Visual Style guidelines, the application navigation menu has been repositioned from the left collapsible section to the top of the main application area right below the application top banner, according to the schematic of the Root screen shown below.



**Figure 12: Schematic of the Root Screen**

This repositioning requires assigning a different 'screen' JSP page to the Root Screen of the application. The Root Screen is now assigned to the 'main.jsp' that defines the page structure corresponding to the diagram shown above.

Screen List (Custom Filter) [1 Records]			
	Module*	Screen Name*	Resource*
▼		Root	
	System	Root Screen	/main.jsp
*			

**Figure 13: Root Screen Configuration in Studio**

In addition to the above change, the navigation menu has been reoriented horizontally using the new widget type 'application navigation'. Two new forms containing the 'application navigation' widget type were added: Service Navigation (for Service application) and Sales Navigation (for Sales application).

Consequently, the content of the NavSlot of the Root Screen in the Login Form Navigation associated with a successful login has also been remapped to reference these forms.

The screenshot below shows the navigation definition for a successful log in to a service application. A similar change was made for other log in successful events.

Form - Login Form

**Form Name:** Login Form ...

**BIO Class:** ...

**Form Type:** LOGIN

**Attributes:** ...

**Module:** shared\_app

Form Placement
 Extensions
 **Navigation**
 Action
 UI Handler
 Text
 Labels
 User Permissions
 Application Permissions
 Validation Rules

Extension References
 Form/Widget/Bio Extension References

Navigation/Navigation Definitions [16 Records]

Module*	Event Type*	Navigation Type*	Form Placement	Navigation Name*	List Order
shared_app	loginUnsuccessfulEvent	SELF		Login UnSuccessful Navig...	0
shared_app	loginSuccessfulEvent	SHELL		Login Successful Navigation	1
home_screen_ui	loginSuccessfulEvent	SHELL		Service	2
Sync_UI	loginSuccessfulEvent	SHELL		SyncResultsDefault	3
Sync_UI	loginSuccessfulEvent	SHELL		SyncOptions	4
home_screen_sales_ui	loginSuccessfulEvent	SHELL		Sales	5
Sync_UI	loginSuccessfulEvent	SHELL		SyncResultsNavigate	6
shared_app	loginSuccessfulEvent	SHELL		sales blank home	7
shared_app	loginSuccessfulEvent	SHELL		service blank home	8
selfregister	loginSuccessfulEvent	MODAL		prompt	9
selfregister	loginSuccessfulEvent	MODAL		prompt	10

Navigation Def

Module*	Container	BIO Source*	BIO Path	Content
home_screen_ui	NavSlot	None		service navigation
home_screen_ui	AgentNotesSlot	None		agentslot_bottom
home_screen_ui	TopSlot	None		Top Banner
home_screen_ui	MainSlot	None		

**Figure 14: Navigation Definition of Login Form**

Similar to the 'navigation menu' used in the left collapsible section, the application navigation widget type requires that you assign a menu that defines the navigation hierarchy of the application as part of the widget properties.

Out-of-the-box, the 'menuID' widget property of the application navigation widget type in Service Navigation and Sales Navigation is assigned to the 'Infor Epiphany Service' and 'Infor Epiphany Sales' menu respectively. These menus define the navigation hierarchies of the Service and Sales application. You should also notice that the navigation hierarchies had been intentionally flattened to make it easier for end-user to navigate among the different screens within the application with fewer clicks. For example, you may now access 'Organizations' with one click instead of two clicks since 'Organizations' is now a top level menu item in the flattened navigation hierarchy.

If you have a navigation hierarchy different from the one shipped out-of-the-box, you need to make sure to bind the menus associated with your customized navigation hierarchies to the 'application navigation' widget type in 'Service Navigation' and 'Sales Navigation'.

The navigation menu associated with this widget is laid out horizontally across the main application page. Besides the 'menuID' widget property, the application navigation widget type has three additional configurable widget properties similar to the 'navigation menu' widget used in the left collapsible section.

The properties are described below.

Property	Description
Initial number of levels	The number of initial levels down the hierarchy tree to be retrieved.

Property	Description
subsequent number of levels	This value determines how many levels of data-driven menus are retrieved at once. Imagine you have 100 data-driven elements, each having 100 children. If the level is 1, the server will make 1 database call and display 100 elements. If the level is 2, the server will make 101 database calls and display 10,100 elements. If the level is 3, the server will make 10,201 database calls and display 1,020,100 elements. If the level is 4, the server will make 1,030,301 database calls and display 103,030,100 elements. The default value is 1. Increase this value only if the number of children at each level is a small number.
hide root menu	If set to true, the root menu is hidden and its children are rendered as level 0 menus.

The navigation menu represented by the application navigation widget type looks and behaves very differently from that represented by the navigation menu widget type. Besides being oriented differently from the navigation menu widget type (horizontal vs. vertical), it has an overflow feature where an overflow control appears when the available screen real estate is not large enough to display all of the primary menu items. This overflow control has multiple functions: users can perform a simple select to shift the menu bar to display the required item; or they can also drag and drop to personalize the order of the menu items in the menu bar.

## Message Dialog

Most of the pop-up navigation, alerts and warning messages have been replaced by the Infor SoHo (UX 3.0) message dialog control.

Message dialogs are pop-up windows (not second browser windows) that appear over the current content area. From the end-user perspective, this change is not intrusive, and you should not need to tweak any meta-data to see this change.

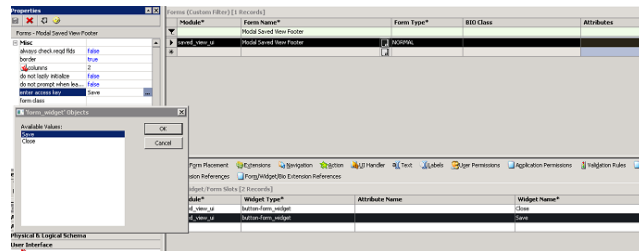
In order to maintain backward compatibility where pop-up content rendering is done in 'Quirks' mode, the 'MODAL' navigation definition has been extended with a new property named 'Use Popup Window'. This flag indicates if a second browser window should be used for displaying the popup content. The default value is 'false'.

Out-of-the-box, two modal navigations still use a second browser window for displaying the pop-up content. These are the popup that shows the expression editor when configuring the trigger expression for the alert event, and subscription. The trigger expression editor uses the 'xtcf' plugin which only works with 'Quirks' mode. The expression editor will not render if the 'Use Popup Window' property is set to 'false' for these two modal navigations.

## 'Default' Button

'Default' button refers to button widget that should be activated when the Enter key is pressed on the form.

You may assign a button widget on a form as the 'default' button by setting the 'enter access key' property of the form. The 'enter access key' allows you to pick a form widget in the form.

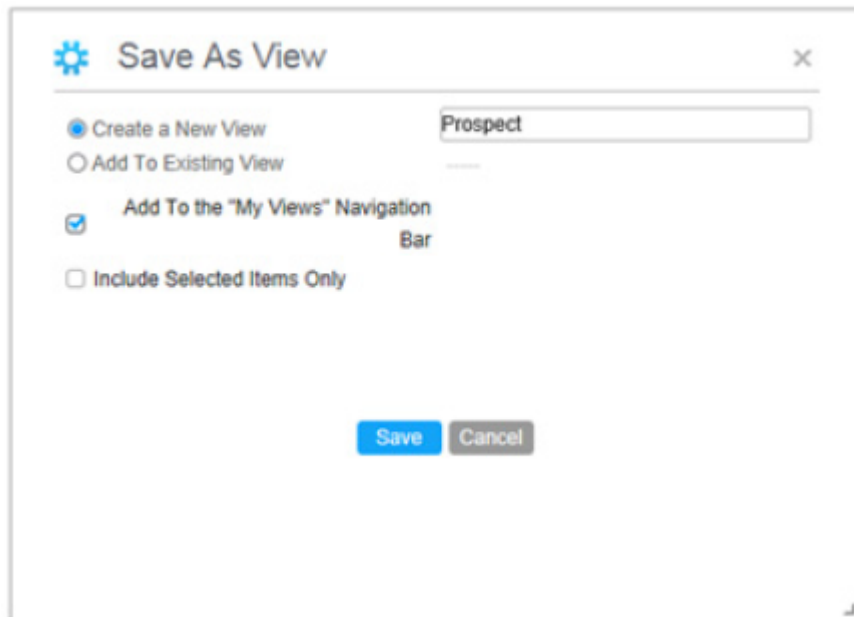


**Figure 15: 'enter access key' Form Property in Studio**

The button widget designated as the 'enter access key' widget on the form is automatically styled using an Azure color instead of the graphite.

Out-of-the-box, 'default' button has been assigned for most of the forms bound to the PopupFooterSlot of a modal navigation, where applicable.

For example, the 'Save' button on the 'Modal Saved View Footer' form is assigned as the 'enter access key'. When performing a 'Save As View', the 'Save' button appears with an Azure color on the 'Save As View' message dialog.



**Figure 16: 'Default' Button on 'Save As View'**

## Widgets and Styling

Restyling and retrofitting the existing widgets to make them look and feel like the Infor SoHo (UX 3.0) controls is one of the key requirements for compliance with the Infor SoHo (UX 3.0) Visual Style guideline. This was done by:

- 1 Adding new CSS classes and modifying the properties of the core CSS classes in the pre-existing CSS style sheets were already shipped out-of-the-box.
- 2 Changing the default 'widget class' property in Studio for some of the widgets to refer directly to the CSS classes defined in the Infor SoHo (UX 3.0) controls style sheet (inforControlsCombined.css).

The look of a widget can be customized in Studio by setting the 'widget class' property of the widget. A 'widget class' is a CSS class that refers to a named style defined in the .css file. Most of the widget types supported in Studio have default widget property values assigned. To effect the change globally to transform Epiphany widgets to look like Infor SoHo (UX 3.0) controls, some of these default widget property values were changed for the widget types. The table below summarizes the changes that have been made.

Widget Type	Widget Parameter	Deprecated	New Default
button	widget class	epnyButton	inforFormButton
checkbox	widget class	epnyCheckbox	inforCheckbox
checkbox	option class	epnyCheckboxOption	inforCheckboxLabel
chooser	widget class	epnyChooser	inforLookupField infor- Textbox
chooser	image src	images/icon_chooser_small.gif	none *The default image is now represented as a base64-encoded string of ASCII characters within the background-image property of the css selector 'inforSearch-Button' defined in inforControlsCombined.css.  If you wish to change the default image for all chooser widget, you may do so by overwriting the styles in custom.css.
combobox	widget class	epnyCombobox	inforDropDownList infor- Textbox
combobox	option class	epnyComboboxOption	none * The CSS class 'inforMenuOptions' in inforControlsCombined.css

Widget Type	Widget Parameter	Deprecated	New Default
			defines the default look of the options. You may change this default by overwriting the styles in custom.css.
combobox	option mouseover class	epnyComboboxOption-Mouseover	none *The hover psuedo class associated with the CSS class 'inforMenuOptions' in inforControlsCombined.css defines the default mouseover look of the options. You may change this default by overwriting the styles in custom.css.
date picker	widget class	epnyDatePicker	inforDateField infor-Textbox
datetime picker	widget class	epnyDatetimePicker	inforDateField infor-Textbox
dropdown	widget class	epnyDropDown	inforDropDownList infor-Textbox
dropdown	option class	eDOpt	none * The CSS class 'inforMenuOptions' in inforControlsCombined.css defines the default look of the options. You may change this default by overwriting the styles in custom.css.
dropdown menu	menu class	epnyDropDownMenu	inforMenuButton
image button	widget class	epnyImageButton	inforTextButton
lookup	widget class	epnyLookup	inforDropDownList infor-Textbox
lookup	option class	epnyLookupOption	none * The CSS class 'inforMenuOptions' in inforControlsCombined.css defines the default look of the options.



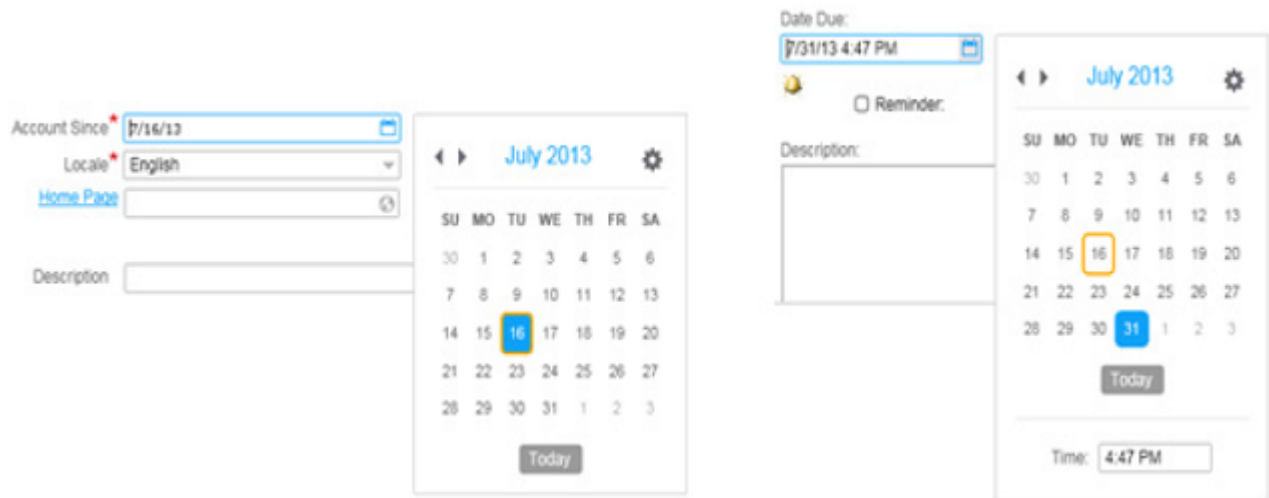
Widget Type	Widget Parameter	Deprecated	New Default
			You may change this default by overwriting the styles in custom.css.
lookup	option mouseover class	epnyLookupOption-Mouseover	none The hover psuedo class associated with the CSS class 'inforMenuOptions' in inforControlsCombined.css defines the default mouseover look of the options. You may change this default by overwriting the styles in custom.css.
popup	widget class	epnyPopup	inforLookupField infor-Textbox
popup	image src	images/icon_find.gif	none The default image is now represented as a base64-encoded string of ASCII characters within the background-image property of the css selector 'inforLookup-Button' defined in inforControlsCombined.css. If you wish to change the default image for all popup widget, you may do so by overwriting the styles in custom.css.
radio	widget class	epnyRadio	inforRadioButton
text	widget class	epnyText	inforTextbox
textarea	widget class	epnyTextarea	inforTextArea
toolbar button	widget class	epnyToolbarButton	inforIconButton

The core CSS classes are defined in master.css and framework.css shipped out-of-the-box (Note: the CSS classes defined in framework.css should not be overridden). Customizing the default look of a widget type should not be done by directly modifying the core CSS classes, as changes to the core CSS classes can result in upgrade issues. Instead, existing CSS classes that are listed in 'master.css' or 'inforControlsCombined.css' can be overwritten by copying them into 'custom.css' and then modifying them there. Any new CSS classes should also be added to 'custom.css'. It is recommended that you

store 'custom.css' in a source control system and reapply (restore into the file-system ../service/web/app) after an upgrade.

## Date and Datetime Picker

The Date and Datetime Picker widgets are two of the few widgets implemented following Infor SoHo (UX 3.0) HTML5 base control.



**Figure 17: Date and Datetime Picker Widget**

For Datetime picker widgets that have a timezone attribute domain defined in Studio, the timezone dropdown will be rendered right beside the Datetime text box. The text box and the timezone dropdown together constitute the widget.



**Figure 18: Datetime Picker Widget with Timezone**

For a Datetime picker widget without a timezone attribute domain defined, the applied timezone is displayed as text beside the text box.



**Figure 19: Datetime Picker Widget without Timezone**

In addition, with the Infor SoHo (UX 3.0) implementation of Date and Datetime picker, the properties 'manual entry in search forms' (for a Date and Datetime picker widgets) and 'manual entry in detail forms' (for a Date picker widget) are not relevant anymore and will not be available.

Date pickers will always have the option both to manually type in the date string and also to use the widget to pick a date. Datetime pickers, on the other hand, will continue to support only the use of the widget and not manual entry as in the older versions.

DateTime pickers on a list view search field (Filter rows or More fields) will be rendered as a date-only widget on the screen. Manual entry will be supported in this case, and search operators will continue to work.



---

# Index

## A

access control lists [199](#)  
ACLManagerExtension [215](#)  
alerts [36](#), [43](#)  
analyzing system performance [107](#)  
application design issues [99](#)  
application implementation/customization [33](#)

## B

best practices [33](#), [36–37](#), [39](#), [41–43](#), [45](#)  
    alerts [36](#), [43](#)  
    application implementation/customization [33](#)  
    bulk import [45](#)  
    configuration [37](#)  
    data migration [37](#)  
    databases [39](#)  
    deployment [41](#)  
    Dialogs [42](#)  
    Knowledge Management System migration [43](#)  
    other [43](#)  
    upgrade [45](#)  
BIO externalization [139](#)  
BIO mapping [155](#)  
Bio transformation [128](#)  
build process [81](#)  
build source components [89](#)  
build types [82](#)  
    full builds [82](#)  
    incremental builds [82](#)  
bulk import [45](#)  
business schema definition [43](#)  
business\_schema\_def.properties [43](#)

## C

calling context sensitive [236](#)  
config.xml [37](#)  
configuration [37](#)  
configuring system for performance optimization [106](#)  
context-sensitivity [236](#)  
creating a BIO [31](#)  
creating a data source [27](#)  
creating a recordset [28](#)  
creating an import map (CSV file data source) [126](#)  
creating extension with Eclipse [62](#)  
creating import map (database data source) [136](#)  
customization [237](#)  
customization checklist [20](#)  
customizing banners [242](#)

## D

data migration [37](#)  
database import [135](#)  
databases [39](#)  
datasource [25](#)  
DATE type fields [43](#)  
debug.memory.leaks [33](#)  
debugging bulk import [137](#)  
defining a form and integrating it with sales detail shell [31](#)  
defining a relationship between recordsets [29](#)  
delta modules [21](#)  
Delta modules and EDialogs [22](#)  
deployment [41](#)  
deployment issues [100](#)  
deployment\_config.xml [37](#), [51](#)  
detail template [113](#)  
development builds [82](#)  
Dialogs [42](#), [175](#), [181](#), [185](#)  
    launching from Service application [175](#)  
    OpenDefaultDialogInWindow extension [185](#)  
    OpenDialogInFrame extension [175](#)  
    OpenDialogInWindow extension [181](#)  
Dialogs service [175](#)  
document configuration [102](#)  
downloading [234](#)

## E

extension development best practices [100](#)

## F

form [26](#)  
full builds [82](#)

## H

help [236–238](#), [242](#)  
    calling context sensitive [236](#)  
    context-sensitivity [236](#)  
    customization [237](#)  
    customizing banners [242](#)  
    project files [237](#)  
    removing topic [238](#)  
    Robohelp banners [242](#)  
help authoring [234–236](#)  
    Robohelp [235–236](#)  
    tools [234](#)  
help customization [233](#)  
    roadmap [233](#)  
Help File property [236](#)  
Help Files list [236](#)

- help property [236](#)
  - no\_help\_topic [236](#)
- help source [234](#)
  - downloading [234](#)
- help topic [236](#)
  - title tag [236](#)
  - top heading [236](#)
- high level steps [17](#)
- HTML 2002 [233–234](#)
- HTML widget [33](#)

## I

- identifying areas for optimization [103](#)
- implementation and customization [17](#), [20–21](#), [25](#)
  - customization checklist [20](#)
  - delta modules [21](#)
  - high level steps [17](#)
  - using Infor Studio [25](#)
- import map [126](#)
- import process [126](#)
- incremental builds [82](#)
- integrating a form to sales detail and list shell [27](#)

## K

- KMS migration [43](#)
- Knowledge Management System migration [43](#)

## L

- launching from Service application [175](#)
- list template [113](#)

## M

- Macros [41](#)
  - white-space in [41](#)
- making modifications to improve performance [104](#)
- max results count [33](#)
- memory leaks [33](#)
- metadata.memory.profiler [33](#)

## N

- no\_help\_topic [236](#)

## O

- OpenDefaultDialogInWindow extension [185](#)
- OpenDialogInFrame extension [175](#)
- OpenDialogInWindow extension [181](#)
- other [43](#)

## P

- path traversal map [170](#)
- performance [99–108](#)
  - analyzing system performance [107](#)
  - application design issues [99](#)
  - configuring system for performance optimization [106](#)
  - deployment issues [100](#)
  - document configuration [102](#)
  - extension development best practices [100](#)
  - identifying areas for optimization [103](#)
  - making modifications to improve performance [104](#)
  - process of improving application performance [101](#)
  - quick checks for problems [105](#)
  - sample task list [104](#)
  - system optimizations undertaken in implementations [108](#)
  - tuning in deployments [105](#)
  - using lazy loaded option [108](#)
  - verifying application settings and behavior [101](#)
- process of improving application performance [101](#)
- project files [237](#)

## Q

- quick checks for problems [105](#)

## R

- recordset [26](#)
- recordset relationships [26](#)
- removing topic [238](#)
- roadmap [233](#)
- Robohelp [233–236](#)
  - HTML 2002 [233–234](#)
  - XML [233](#)
- Robohelp banners [242](#)

## S

- sample build process [91](#)
- sample task list [104](#)
- sample version control structure [89](#)
- services [175](#)
  - Dialogs service [175](#)
- SQL Repl [41](#)
- SSO related error messages [37](#)
- system modules [33](#)
- system optimizations undertaken in implementations [108](#)

## T

- tabgroup [114](#)
- tabgroup template [114](#)
- templates [113–114](#)
  - detail template [113](#)
  - list template [113](#)

- templates (*continued*)
  - tabgroup template [114](#)
  - toggle template [114](#)
- title tag [236](#)
- toggle template [114](#)
- tools [234](#)
- top heading [236](#)
- tuning in deployments [105](#)
- typical customization [27–29](#), [31](#)
  - creating a BIO [31](#)
  - creating a data source [27](#)
  - creating a recordset [28](#)
  - defining a form and integrating it with sales detail shell [31](#)
  - defining a relationship between recordsets [29](#)

## U

- upgrade [45](#)

- UseSQLNVarCharLikeQuery [39](#)
  - using Eclipse [53](#)
  - using Infor Studio [25](#)
  - using lazy loaded option [108](#)

## V

- verifying application settings and behavior [101](#)

## W

- White space in macros [41](#)
- white-space in [41](#)

## X

- XML [233](#)

