



Microsoft Dynamics™ GP  
**eConnect Programmer's Guide**  
Release 10.0

**Copyright**

Copyright © 2007 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation. Notwithstanding the foregoing, the licensee of the software with which this document was provided may make a reasonable number of copies of this document solely for internal use.

**Trademarks**

Microsoft, Microsoft Dynamics, Visual Basic, Visual Studio, BizTalk Server, SQL server, Windows, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation or its affiliates in the United States and/or other countries.

The names of actual companies and products mentioned herein may be trademarks or registered marks - in the United States and/or other countries - of their respective owners.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

**Intellectual property**

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

**Warranty disclaimer**

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

**Limitation of liability**

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Microsoft Corporation. Microsoft Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual. Neither Microsoft Corporation nor anyone else who has been involved in the creation, production or delivery of this documentation shall be liable for any indirect, incidental, special, exemplary or consequential damages, including but not limited to any loss of anticipated profit or benefits, resulting from the use of this documentation or sample code.

**License agreement**

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

**Publication date**

April 2007 -- Last updated April 6, 2007

# Contents

|  |               |
|--|---------------|
| <b>Introduction .....</b>                                  | <b>2</b>      |
| What's in this manual.....                                 | 2             |
| Symbols and conventions .....                              | 3             |
| Product support .....                                      | 3             |
| <br><b>Part 1: eConnect Overview .....</b>                 | <br><b>6</b>  |
| <b>Chapter 1: Overview .....</b>                           | <b>7</b>      |
| What is eConnect?.....                                     | 7             |
| What eConnect can do .....                                 | 7             |
| eConnect Example .....                                     | 8             |
| Getting started .....                                      | 10            |
| <b>Chapter 2: Architecture .....</b>                       | <b>13</b>     |
| Architecture diagram .....                                 | 13            |
| Business objects .....                                     | 14            |
| eConnect APIs .....  | 16            |
| BizTalk .....  | 17            |
| Transaction Requester .....                                | 18            |
| <br><b>Part 2: eConnect Schema and XML Documents .....</b> | <br><b>22</b> |
| <b>Chapter 3: eConnect Schema .....</b>                    | <b>23</b>     |
| eConnect schema overview .....                             | 23            |
| Installing eConnect schema .....                           | 23            |
| Using eConnect schema .....                                | 23            |
| eConnect schema reference .....                            | 24            |
| <b>Chapter 4: eConnect XML Documents .....</b>             | <b>25</b>     |
| eConnect XML document structure .....                      | 25            |
| Creating eConnect XML documents .....                      | 27            |
| Sample eConnect XML documents .....                        | 28            |
| Using eConnect to update existing data.....                | 28            |
| Automating document number assignment.....                 | 28            |
| Special characters in eConnect XML documents .....         | 30            |
| <b>Chapter 5: XML Document Examples .....</b>              | <b>31</b>     |
| Create a customer.....                                     | 31            |
| Delete a customer address .....                            | 32            |
| Retrieve a single customer .....                           | 32            |
| Assign a document number .....                             | 33            |
| <br><b>Part 3: .NET Development .....</b>                  | <br><b>38</b> |
| <b>Chapter 6: .NET Development Overview .....</b>          | <b>39</b>     |
| eConnect and .NET.....                                     | 39            |

|  |           |
|--|-----------|
| Adding a reference.....                                    | 39        |
| Including the namespace .....                              | 40        |
| <b>Chapter 7: eConnect Assembly .....</b>                  | <b>41</b> |
| Microsoft.Dynamics.GP.eConnect assembly.....               | 41        |
| eConnectMethods class.....                                 | 41        |
| EnumTypes class .....                                      | 44        |
| eConnectException class.....                               | 44        |
| <b>Chapter 8: Serialization Assembly .....</b>             | <b>47</b> |
| Microsoft.Dynamics.GP.eConnect.Serialization assembly..... | 47        |
| Serialization classes .....                                | 47        |
| eConnect serialization example .....                       | 48        |
| Using serialization flags.....                             | 53        |
| eConnectOut serialization example .....                    | 54        |
| <b>Chapter 9: Miscellaneous Routines Assembly .....</b>    | <b>59</b> |
| Microsoft.Dynamics.GP.eConnect.MiscRoutines assembly.....  | 59        |
| GetNextDocNumbers class .....                              | 59        |
| DocumentRollback class .....                               | 62        |
| RollBackDocument class.....                                | 65        |
| GetSopNumber class .....                                   | 66        |
| PricingMethods class.....                                  | 68        |
| <b>Part 4: MSMQ Development.....</b>                       | <b>72</b> |
| <b>Chapter 10: MSMQ .....</b>                              | <b>73</b> |
| Microsoft Message Queue overview .....                     | 73        |
| Windows Services used with MSMQ.....                       | 73        |
| eConnect MSMQ Control .....                                | 73        |
| <b>Chapter 11: Incoming Service .....</b>                  | <b>75</b> |
| Creating an eConnect XML document .....                    | 75        |
| Creating an MSMQ message.....                              | 75        |
| Incoming Service example.....                              | 76        |
| <b>Chapter 12: Outgoing Service .....</b>                  | <b>79</b> |
| Publishing the eConnect XML documents.....                 | 79        |
| Retrieving the MSMQ message.....                           | 79        |
| Outgoing Service Example .....                             | 80        |
| <b>Part 5: Business Logic .....</b>                        | <b>84</b> |
| <b>Chapter 13: Business Logic Overview.....</b>            | <b>85</b> |
| Business logic.....  | 85        |
| Extending business logic.....                              | 85        |
| Calling the business objects.....                          | 85        |
| <b>Chapter 14: Custom XML Nodes.....</b>                   | <b>87</b> |
| Adding an XML node.....                                    | 87        |

|  |            |
|--|------------|
| Creating a SQL stored procedure .....                          | 88         |
| <b>Chapter 15: Business Logic Extensions .....</b>             | <b>91</b>  |
| Modifying business logic .....                                 | 91         |
| Using Pre and Post stored procedures .....                     | 91         |
| <b>Part 6: Transaction Requester .....</b>                     | <b>96</b>  |
| <b>Chapter 16: Using the Transaction Requester .....</b>       | <b>97</b>  |
| The Transaction Requester Service .....                        | 97         |
| Requester documents types .....                                | 97         |
| Requester document tables .....                                | 98         |
| Using the RequesterTrx element .....                           | 100        |
| <b>Chapter 17: Customizing the Transaction Requester .....</b> | <b>103</b> |
| Create a custom Transaction Requester Service .....            | 103        |
| Using the <taRequesterTrxDisabler> XML node .....              | 109        |
| <b>Part 7: eConnect Samples .....</b>                          | <b>112</b> |
| <b>Chapter 18: Create a Customer .....</b>                     | <b>113</b> |
| Overview .....   | 113        |
| Running the sample application .....                           | 113        |
| How the sample application works .....                         | 114        |
| How eConnect was used .....                                    | 114        |
| <b>Chapter 19: Create a Sales Order .....</b>                  | <b>115</b> |
| Overview .....   | 115        |
| Running the sample application .....                           | 115        |
| How the sample application works .....                         | 116        |
| How eConnect was used .....                                    | 116        |
| <b>Chapter 20: XML Document Manager .....</b>                  | <b>117</b> |
| Overview .....   | 117        |
| Running the sample application .....                           | 117        |
| How the sample application works .....                         | 118        |
| How eConnect was used .....                                    | 119        |
| <b>Chapter 21: Get a Document Number .....</b>                 | <b>121</b> |
| Overview .....   | 121        |
| Running the sample applications .....                          | 121        |
| How the sample applications work .....                         | 122        |
| How eConnect was used .....                                    | 122        |
| <b>Chapter 22: Retrieve Data .....</b>                         | <b>125</b> |
| Overview .....   | 125        |
| Running the sample application .....                           | 125        |
| How the sample application works .....                         | 126        |
| How eConnect was used .....                                    | 126        |
| <b>Chapter 23: MSMQ Document Sender .....</b>                  | <b>127</b> |

C O N T E N T S

Overview ..... 127

Running the sample application..... 127

How the sample application works ..... 128

How eConnect was used ..... 128

**Glossary ..... 129**

**Index ..... 131**



# Introduction

Welcome to eConnect for Microsoft Dynamics™ GP. eConnect provides files, tools, and services that allow applications to integrate with Microsoft Dynamics GP. This documentation explains how to use eConnect to develop application integration solutions. Before you begin installing and using eConnect, take a few moments to review the information presented here.

## What's in this manual

The Microsoft Dynamics GP eConnect Programmer's Guide is designed to give you an in-depth understanding of how to work with eConnect. Information is divided into the following parts:

- [Part 1, eConnect Overview](#), provides an introduction to eConnect, its components, and the application programming interfaces (APIs) it provides.
- [Part 2, eConnect Schema and XML Documents](#), discusses how eConnect uses XML documents to describe Microsoft Dynamics GP documents and operations. Review this portion of the documentation to learn how to construct an eConnect XML document.
- [Part 3, .NET Development](#), discusses how you can use eConnect's .NET assemblies to submit or request XML documents.
- [Part 4, MSMQ Development](#), describes how eConnect uses MSMQ to transport XML documents to and from integrating applications.
- [Part 5, Business Logic](#), explains how you can supplement or modify the business rules eConnect uses to process documents.
- [Part 6, Transaction Requester](#), describes the available options for retrieving XML documents that represent documents or transactions in Microsoft Dynamics GP.
- [Part 7, eConnect Samples](#), describes the sample applications that are included with an eConnect SDK installation.

To learn about installing or maintaining eConnect for Microsoft Dynamics GP, refer to the eConnect Installation and Administration Guide.



For additional information about eConnect XML documents, use the reference sections in the eConnect help documentation. The eConnect install places the help document in the directory:

c:\Program Files\Common Files\Microsoft Shared\eConnect 10\eConnect Help.



## Symbols and conventions

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

| Symbol  | Description  |
|---|--|
|  | The light bulb symbol indicates helpful tips, shortcuts, and suggestions.  |
|  | Warnings indicate situations you should be aware of when completing tasks.   |
| <i>Margin notes summarize important information.</i>                              | Margin notes call attention to critical information and direct you to other areas of the documentation where a topic is explained. |
| Convention  | Description  |
| <b>Part 2, XML Documents</b>  | Bold type indicates a part name.   |
| Chapter 1, "Overview"   | Quotation marks indicate a chapter name.   |
| <i>Getting started</i>  | Italicized type indicates a section name.  |
| <code>using System.IO;</code>   | This font is used to indicate script examples.   |
| Microsoft Message Queuing (MSMQ)  | Acronyms are spelled out the first time they're used.  |
| TAB or ALT+M  | Small capital letters indicate a key or a key sequence.  |

## Product support

Microsoft Dynamics GP technical support can be accessed online or by telephone. Go to [www.microsoft.com/Dynamics](http://www.microsoft.com/Dynamics) and click the CustomerSource or PartnerSource link, or call 888-477-7877 (in the US and Canada) or 701-281-0555.





# Part 1: eConnect Overview

This portion of the documentation provides an introduction to eConnect. Review the following to learn what eConnect can do and understand the components it uses to support your application development efforts. The list that follows contains the topics that are discussed:

- [Chapter 1, “Overview,”](#) introduces eConnect and how you can use eConnect to integrate Microsoft Dynamics GP data and functionality into your applications.
- [Chapter 2, “Architecture,”](#) describes the components and application programming interfaces (APIs) that eConnect provides. Use this information to understand how eConnect works and to determine which API best supports your development environment and tools.

# Chapter 1: Overview

Microsoft Dynamics GP eConnect allows you to integrate your business applications with Microsoft Dynamics GP. The following topics introduce Microsoft Dynamics GP eConnect:

- [What is eConnect?](#)
- [What eConnect can do](#)
- [eConnect Example](#)
- [Getting started](#)

## What is eConnect?

eConnect is a collection of tools, components, and interfaces that allow applications to programmatically interact with Microsoft Dynamics GP. The key eConnect components and interfaces include:

- A .NET managed code assembly
- A Microsoft BizTalk® Application Integration Component (AIC)
- Microsoft Message Queuing (MSMQ) services

These eConnect interfaces allow external applications like web storefronts, web services, point-of-sale systems, or legacy applications to integrate with Microsoft Dynamics GP. The external applications can perform actions like creating, updating, retrieving, and deleting back office documents and transactions. While eConnect supplies a large number of documents, not every Microsoft Dynamics GP feature is available through eConnect.



*Throughout the documentation, the terms back office and front office are used. The term back office refers to the financial management system, in this case, Microsoft Dynamics GP. The term front office refers to customer relationship management systems, data warehouses, web sites, or other applications that communicate with the back office.*

eConnect allows you to leverage the existing transaction-based business logic of Microsoft Dynamics GP. This allows you to focus your time and energy on creating or enhancing custom applications for the front office.

## What eConnect can do

eConnect allows you to enhance your applications as follows:

### 1. Add real-time access to Dynamics GP data.

eConnect provides real-time access to back office data. It offers a way to add up-to-date back office information to existing front office applications like web storefronts or service applications.

### 2. Share financial management data across applications.

eConnect allows multiple applications to share financial management data. The eConnect interfaces can support a number of independent applications. Changes to financial data in Dynamics GP are simultaneously available to all applications with an eConnect connection to that company in Dynamics GP.

Application integrations using eConnect include the following benefits:

**1. Reduce development time.**

eConnect has a large number of integration points for Microsoft Dynamics GP. Software developers can quickly add back office integration to an application. This reduces cost by simplifying the development effort while providing fast access to Microsoft Dynamics GP data. eConnect also reduces development time when the business logic contained in the back office is reused by new custom applications.

An eConnect integration also reduces costs by reducing data re-entry. An automated eConnect integration between Microsoft Dynamics GP and a new or existing online storefront, web service, or other data source eliminates the time and cost of manually copying data.

**2. Reuse existing development tools.**

eConnect allows software developers to select their tool of choice when working with eConnect. Developers can use Microsoft .NET, Microsoft SQL Server stored procedures, BizTalk, or MSMQ.

**3. Leverage industry-standard technologies.**

eConnect includes components for MSMQ and BizTalk Server, which are industry standard tools that support integration between applications.

eConnect also uses XML documents to move data into and out of Microsoft Dynamics GP. The XML documents are a text-based representation of back office data. An XML schema specifies the data that is included in each type of XML document. This allows eConnect to provide back office integration to any application capable of creating or consuming these XML documents.

## eConnect Example

To help you understand how eConnect benefits your development effort, the following example presents a business problem and its solution using eConnect and Microsoft Dynamics GP.

### Introduction

A theater business owns dozens of dinner theaters scattered throughout the United States. The company differentiates itself from its competitors by delivering high-quality service to its customers. To build upon this advantage, the company wants to allow customers to reserve specific theater seating while online.

The company wants a web-based system that customers use to reserve seats. In addition, the company wants to provide customers the ability to view the previous functions they attended. The web portal should also provide customers access to other valuable information and services.

### Requirements

To provide the expected services, the solution must address the following requirements:

- Use Windows Live ID for security for the online reservation system.
- Allow customers to reserve one or more specific seats at a theater (for example, a single customer reserves 10 seats in a row for his or her family members).

- Create a sales invoice in the back office when the reservation is submitted.
- Allow a sales invoice to be cancelled.
- Record a Microsoft Dynamics GP deposit for the reservation fee when the reservation is submitted.
- Give customers the ability to request their dinner of choice from a specified group of vendors.
- Allow customers to request specific items in the theater by using a handheld device that is situated at each table.
- For non-reservation customers, allow a theater card to be swiped at arrival. The card automatically creates a sales invoice in the back office.
- Create a payables transaction in the back office when food and beverage items are ordered. Submit a sales order to the vendor.
- At the end of the theater presentation, generate a receipt for each customer.
- Create payroll transactions for employee tips.
- At the end of the theater presentation, submit a check to each vendor. Each check includes all customer transactions for that vendor.

## Solution

To meet these requirements, a web-based solution is proposed. The web application uses BizTalk and eConnect to integrate with Microsoft Dynamics GP. The combination of eConnect and BizTalk allow the web application to perform the following tasks:

- Use eConnect to create the sales invoices in Microsoft Dynamics GP. The web application creates an eConnect XML document and sends it to a BizTalk queue. eConnect receives the XML document from BizTalk and uses the XML document to create the sales invoice.
- Use eConnect to cancel an existing sales invoice. The web application creates an XML document and sends it to a BizTalk queue. eConnect receives the XML document from BizTalk and uses the XML document to void the specified sales invoice.
- Use eConnect to create payables transactions representing the customer's food and beverage orders. The web application creates an XML document and sends it to a BizTalk queue. eConnect receives the XML document from BizTalk and uses the XML document to create the payables transactions. The web application could also submit a receivables transaction to the vendor through the BizTalk server if the vendor is also using eConnect or an accounting system that supports a similar type of document exchange.
- Use eConnect to process check submissions. The web application creates an XML document and sends it to a BizTalk queue. eConnect receives the XML document from BizTalk and completes the custom check submission. This step includes the following customizations:

- Create a new stored procedure to handle the creation of checks for all vendor transactions.
- Create a cash receipt. After creating the vendor check, create an XML document using Microsoft Dynamics GP data that details the check's contents. Perform a transform of the XML document to create a cash receipt for the vendor. This transaction occurs after making a payment through an online credit card processing system.
- Submit the cash receipts to the vendors. If the vendors also use BizTalk server, eConnect, and Microsoft Dynamics GP, develop a process to electronically submit the cash receipts.
- Use eConnect to retrieve a specified invoice. The web application uses the XML document that is returned by eConnect to generate a printout for the customer.
- Use eConnect to update payroll to reflect employee tips. The web application creates a XML document and sends it to a BizTalk queue. eConnect receives the XML document from the BizTalk queue and updates Microsoft Dynamics GP to reflect tip amounts for each employee.

## Summary

The example shows how eConnect simplifies the development of the web solution. eConnect's schema-based XML documents allows the application to easily incorporate back office functionality using existing development tools.

The example also shows how reusing the business logic and the transaction processing abilities of Microsoft Dynamics GP simplify development. The web application can submit the document and rely upon Microsoft Dynamics GP to successfully complete the transaction.

The example uses eConnect as part of a web-based solution. A web-based solution simplifies the deployment of features that use eConnect integrations. An update of the web application or web service makes your new or updated features immediately available to all users.

## Getting started

To use eConnect in a development project, complete the following:

1. *Review the eConnect architecture.*  
Review [Chapter 2, "Architecture,"](#) to familiarize yourself with eConnect's components. eConnect supports several application programming interfaces (APIs) that you can use to integrate with Microsoft Dynamics GP. If you understand how eConnect's underlying components work together, you can quickly identify the eConnect API that meets the needs of your integration project.
2. *Discuss the installation process.*  
Before starting a new project, discuss the eConnect installation procedure with your system administrator. You need to ensure the eConnect business objects are installed on the Microsoft Dynamics GP server. You also need to identify any unique configuration settings that occurred during installation. You should evaluate how configuration settings impact each eConnect API.



3. *Learn about eConnect XML documents.*

eConnect uses XML documents to describe Microsoft Dynamics GP documents and transactions. Refer to [Part 2, eConnect Schema and XML Documents](#), to learn how eConnect XML documents are structured. Refer to the Schema Reference and an XML Node Reference of the eConnect help documentation to learn about specific eConnect schemas, nodes, and elements.

4. *Select the API for your project.*

Once you select the eConnect API you intend to use, review the portion of the Programmer's Guide that discusses that API. For example, if you want to use eConnect with a .NET development project, review [Part 3, .NET Development](#) to learn how to add and use eConnect in your project.



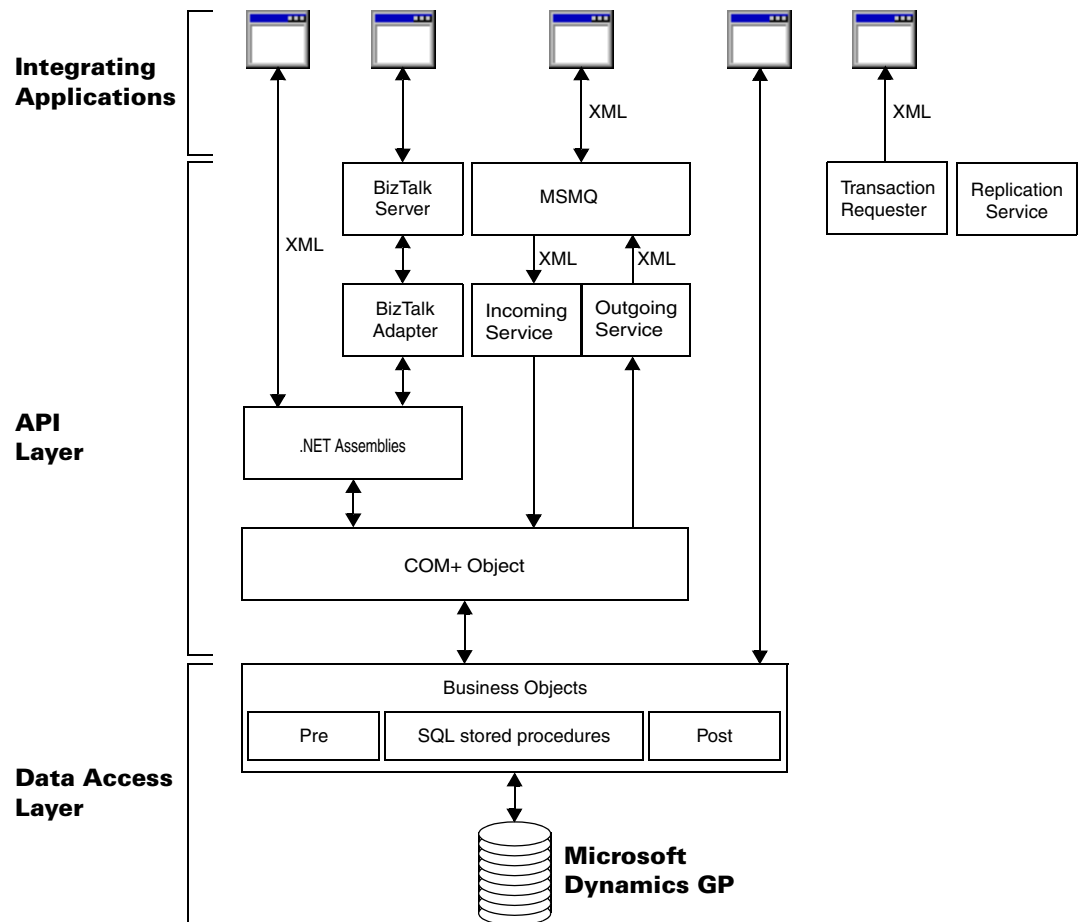
## Chapter 2: Architecture

When using Microsoft Dynamics GP eConnect, it is helpful to understand its architecture. Architectural information is divided into the following sections:

- [\*Architecture diagram\*](#)
- [\*Business objects\*](#)
- [\*eConnect APIs\*](#)
- [\*BizTalk\*](#)
- [\*Transaction Requester\*](#)

### Architecture diagram

eConnect installs a collection of components that work together to provide programmatic access to Microsoft Dynamics GP data. The following diagram illustrates the basic components:



*Refer to the eConnect Installation and Administration Guide for additional information about installing and configuring eConnect.*

The diagram illustrates eConnect's two key layers and the components that make up those layers. The two layers are as follows:

- The data access layer contains the eConnect business objects. The business objects are a collection of SQL stored procedures installed on the Microsoft Dynamics GP server. eConnect uses the business objects to perform all retrieve, create, update, and delete operations.
- The application programming interface (API) layer contains a collection of files and components that allow you to use the business objects. You must install the API layers on the same computer as your integrating application. The solutions you develop should use the API that best meets your integration needs.

The diagram shows that an integrating application can bypass the API layer and use the eConnect business objects directly.

The Transaction Requester is an interface that helps the API layer's Outgoing Service publish specified Microsoft Dynamics GP documents as eConnect XML documents. The Transaction Requester identifies the transactions the Outgoing Service needs to publish.

The Replication Service allows you to replicate transactions that occur in Microsoft Dynamics GP to another database.



*The Replication Service is a special-purpose service that performs a specific task. It does not provide an API. You cannot programmatically customize the Replication Service. For information about installing and configuring the Replication Service, see the eConnect Installation and Administration Guide.*

## Business objects

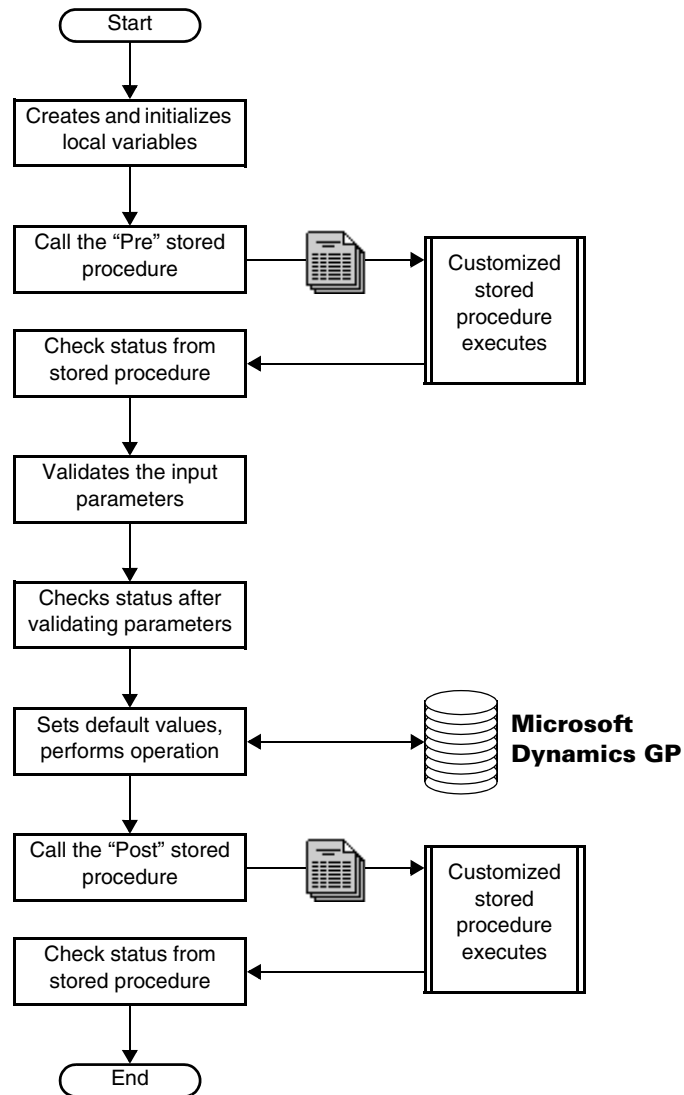
The most basic eConnect components are known as business objects. The eConnect business objects are a collection of SQL stored procedures. The eConnect install adds these stored procedures to the system database (DYNAMICS) and each specified company database.

The stored procedures contain the business logic used by eConnect. In addition, the stored procedures validate the data and supply default values. Any eConnect action that queries, creates, updates, or deletes data from Microsoft Dynamics GP is completed using one or more of these stored procedures.

The eConnect business objects include Microsoft Dynamics GP documents and transactions that are commonly used in application integration. While eConnect supplies a large number of documents, not every Microsoft Dynamics GP feature is available through eConnect.

You cannot modify eConnect stored procedures. However, eConnect provides an alternative that allows you to customize its business logic. Each stored procedure includes specially named pre and post stored procedures. You customize eConnect's business logic by adding SQL queries and commands to these pre and post procedures. The pre stored procedure runs your custom code immediately before the eConnect stored procedure, while the post stored procedure runs immediately after the eConnect stored procedure.

The following diagram shows the typical sequence of events that occur within a business object:



Notice how the business object checks the status reported by each step of the operation. If it detects that an error occurred, the stored procedure halts operation and returns an error message to the caller.

For example, assume you want to modify the business logic for the eConnect stored procedure named `taSopHdrIvcInsert`. You complete this modification by adding custom SQL code to the stored procedure named `taSopHdrIvcInsertPost`. Your custom code will run immediately after every execution of the `taSopHdrIvcInsert` procedure. To run custom code prior to the execution of the `taSopHdrIvcInsert` procedure, place the custom SQL code in the stored procedure named `taSopHdrIvcInsertPre`.

Once eConnect installs its business objects, the stored procedures are available on the server and can be utilized by your application. However, a direct call to an eConnect stored procedure requires you to:

- Create a connection to the database server.
- Implement security restrictions to prevent unauthorized use of your database connection.

- Implement transaction management to commit or rollback changes.
- Identify and handle error conditions.
- Update your application whenever changes are made to the parameters for the stored procedure.

To avoid the extra work of direct calls to the stored procedures, use one of the APIs that eConnect supplies. These APIs provide a simpler approach to using the eConnect business objects.

Refer to [Chapter 13, “Business Logic Overview,”](#) for additional information about extending the business objects and calling the eConnect stored procedures.

## eConnect APIs

eConnect provides a collection of APIs that allow you to use the business objects. There are APIs for Microsoft .NET, and Microsoft Message Queuing (MSMQ). The variety of eConnect APIs allows you to use the interface that best fits your integration project and the available development tools.

To support its API, eConnect supplies a COM+ component that manages interaction with the eConnect business objects. The COM+ object installs in Component Services on your eConnect client computer. Refer to the eConnect Installation and Administration Guide for information about installing and configuring the COM+ object.

To use the eConnect API, your application must create or read eConnect XML documents. Refer to [Chapter 4, “eConnect XML Documents,”](#) for additional information about creating eConnect XML documents.

The eConnect install includes files containing the XML schema for all its documents. A schema is an XML file (with typical extension .xsd) that describes the syntax and semantics of XML documents using a standard XML syntax. An XML schema specifies the content constraints and the vocabulary that compliant documents must accommodate.

You can use these files to perform validation. When eConnect validates a document, it uses the schema to ensure the document contains the expected information. It rejects documents that do not comply with the schema specifications. The schema files can also serve as a reference. Since the files describe each type of eConnect document, you can use them to research questions about the schemas, nodes, and elements a document may contain.

The following APIs use XML documents and the COM+ component:

### Microsoft .NET

When you install the eConnect COM+ object, the installer places three .NET assemblies on your computer. The installer also registers these assemblies in the global assembly cache.

You can add these assemblies to a Visual Studio project by adding a reference to each assembly file. Once you include the .NET assemblies in your project, you gain access to the eConnect COM+ object. This allows your application to parse eConnect XML documents, create a connection to the Microsoft Dynamics GP server and call the eConnect business objects. Your eConnect enabled solution can then use XML documents to create, delete, update, or retrieve Microsoft Dynamics GP data.

Refer to [Chapter 7, “eConnect Assembly,”](#) for information about creating solutions using the eConnect .NET assemblies.

## MSMQ

The MSMQ API includes two Windows services. The services are as follows:

- The Incoming Service monitors a specified queue and retrieves XML documents placed in that queue. The Incoming Service uses the COM+ object to parse the XML documents, create a connection to the Microsoft Dynamics GP server, and call the eConnect business objects. To use this API, you create an application that submits XML documents to the specified queue.
- The Outgoing Service publishes XML documents to a queue in response to specified events in Microsoft Dynamics GP. To use this API, you create applications that retrieve the XML documents from the queue and perform actions based on the XML data.

To develop solutions that use the MSMQ API, you should carefully consider the following:

- The MSMQ API is asynchronous. Due to the disconnected nature of the API, changes are not immediately reflected in Microsoft Dynamics GP or in the integrating application. In addition, your application cannot immediately determine whether a document submitted using the Incoming Service was successfully processed.
- All applications that use the MSMQ API must be able to access the specified MSMQ queues.
- The eConnect Outgoing Service relies on the eConnect Transaction Requester to create SQL triggers in the Microsoft Dynamics GP database. If you plan to use the Outgoing Service, you must use the Transaction Requester to identify the Microsoft Dynamics GP documents and events that you want the Outgoing Service to publish to a specified queue.

Refer to [Chapter 10, “MSMQ,”](#) for additional information about creating solutions using MSMQ and the Incoming and Outgoing Services.

## BizTalk

*See the eConnect Installation and Administration Guide for information about installing and configuring eConnect’s BizTalk adapter.*

eConnect provides a BizTalk application integration component (AIC) that you can install on your BizTalk server. The BizTalk adapter allows you to use BizTalk to manage interaction with eConnect business objects.

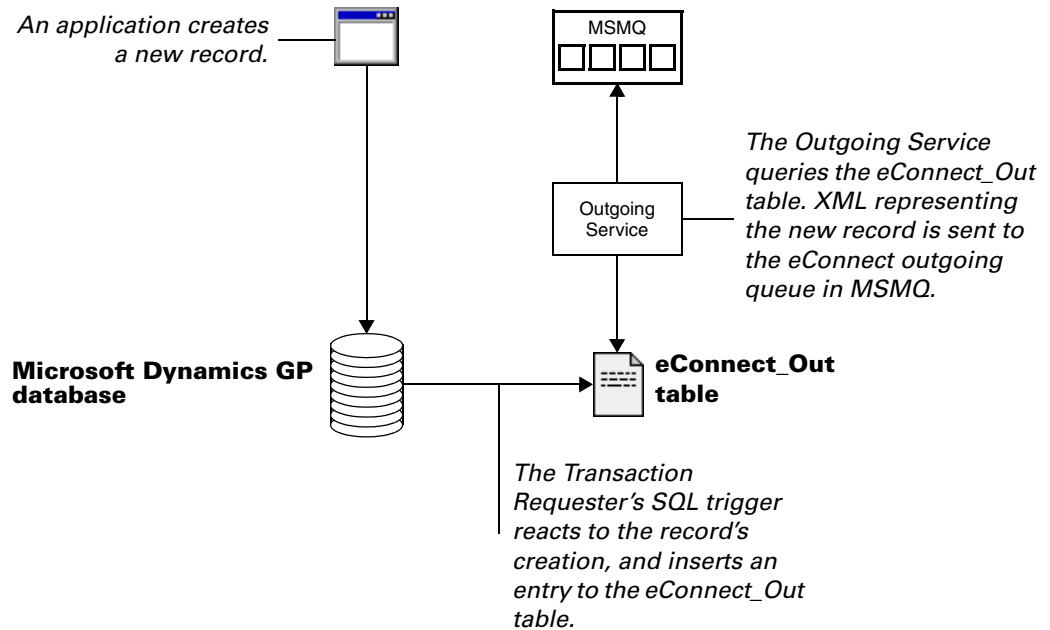
The adapter supports the use of eConnect as a part of a BizTalk orchestration or in a simple pass-through situation. A BizTalk orchestration allows applications with differing message formats to integrate, while a BizTalk pass-through simply routes messages between applications.

The choice between using a BizTalk Orchestration or a pass-through depends on the level of flexibility your solution requires. An orchestration provides the greatest flexibility. For example, you can customize the adapter by adding bindings to other applications.

Refer the BizTalk documentation for information about developing a BizTalk-based integration.

## Transaction Requester

The Transaction Requester is a collection of SQL database tables and database triggers that eConnect uses to make Dynamics GP data changes available to the Outgoing Service. The following diagram illustrates the Transaction Requester:



When you install the Transaction Requester, the installer creates three tables in each specified Microsoft Dynamics GP database:

- **eConnect\_Out** This table stores data from selected create, update, or delete operations that occur within Microsoft Dynamics GP. The data identifies the individual transactions that occurred. The Outgoing Service uses the data in this table to create an XML document that is placed in a queue.
- **eConnect\_Out\_Setup** This table contains configuration information for the Transaction Requester. To keep the Transaction Requester working, do not make changes to this table.
- **eConnectOutTemp** This table is a temporary data store.

For example, assume you want your application to be updated when a new customer is added to Microsoft Dynamics GP. To begin, you use the eConnect Requester Setup utility to specify the customer object and the SQL insert operation. The eConnect Requester Setup adds a SQL trigger to the database. When a new customer record is inserted, the SQL trigger creates a record of the event in the eConnect\_Out table.

The eConnect Outgoing Service periodically queries the eConnect\_Out table. The service uses the record in the table to create an XML document that describes the new customer document.

The Outgoing Service then places the XML document in a message queue where it can be retrieved and used by your application.



To configure the eConnect Transaction Requester, use the eConnect Requester Setup utility. The eConnect Requester Setup utility allows you to specify Dynamics GP objects and operations you want to export to another application. The utility then adds SQL triggers to Dynamics GP that populate the eConnect\_Out table for the specified objects and operations. For a detailed explanation of how to use the eConnect Requester Setup utility, see the eConnect Installation and Administration Guide.

Refer to [Chapter 17, “Customizing the Transaction Requester,”](#) for information about using and customizing the Transaction Requester Service.





# Part 2: eConnect Schema and XML Documents

To use eConnect, you must be able to create or consume eConnect XML documents. This portion of the documentation explains the XML schemas that govern how eConnect XML documents are assembled. The list that follows contains the information you need to understand eConnect's XML schema and documents:

- [Chapter 3, "eConnect Schema,"](#) introduces eConnect XML schema. The schema define how to supply data using eConnect XML documents. The schema also allow you to validate the documents you submit to ensure they can be processed by eConnect.
- [Chapter 4, "eConnect XML Documents,"](#) introduces eConnect XML documents. Review this information to understand how you use these XML documents to describe Microsoft Dynamics GP documents and operations.
- [Chapter 5, "XML Document Examples,"](#) provides examples of eConnect XML document. The examples demonstrates how XML components fit together to create an actual eConnect XML document.

## Chapter 3: eConnect Schema

To integrate your application with Microsoft Dynamics GP, eConnect requires you to submit XML documents that describe Microsoft Dynamics GP documents and transactions. To ensure the documents can be consistently processed, eConnect supplies a collection of XML schema that define the XML documents eConnect accepts. Information about the schemas include the following:

- [\*eConnect schema overview\*](#)
- [\*Installing eConnect schema\*](#)
- [\*Using eConnect schema\*](#)
- [\*eConnect schema reference\*](#)

### eConnect schema overview

eConnect uses XML schema to define what an eConnect XML document contains. A schema is an XML file (with typical extension .xsd) that describes the syntax and semantics of XML documents using a standard XML syntax. An XML schema specifies the content constraints and the vocabulary that compliant documents must accommodate.

eConnect transports the XML documents as messages between your application and eConnect.

### Installing eConnect schema

When you include the schemas component of the eConnect install, the installer places schema files in a schemas folder on your computer. The following schema resources are available:

- The install places the .xsd schema files in the directory c:\Program Files\Common Files\Microsoft Shared\eConnect 10\XML Schema\Incoming XSD Individual Schemas. The files in the directory contain the schema for each eConnect XML document.
- The install places a file named eConnect.xsd that contains the schema definition for all eConnect XML documents. The install typically places this file in the directory c:\Program Files\Common Files\Microsoft Shared\eConnect 10\XML Schema\Incoming XSD Schemas.

### Using eConnect schema

To use the eConnect application programming interfaces (APIs), your application must be able to create XML documents or read XML documents based on these schema. If you submit a document that does not comply with its schema definition, it will be rejected and an error will be logged in the eConnect event log.

The schema files also allow you to perform validation of the documents you create. The eConnect API allow you to specify the schema file for the document.

- Use the eConnect.xsd file when your application needs to validate all types of XML documents.
- Use the individual document XSD files to perform validation for a specific eConnect XML document.

The schema files contain the definition of each eConnect XML document, transaction type schema, and XML node. If you have questions about the schema XML nodes, and elements for a specified eConnect document, the schema files are the definitive source of the information you need.

## **eConnect schema reference**

The eConnect online help documentation contains two reference sections that describes the eConnect transaction type schemas and the XML nodes. These references help you identify the nodes, elements, and values you can use in an eConnect XML document.

## Chapter 4: eConnect XML Documents

The eConnect APIs require your application to create or read eConnect XML documents. The XML documents contain the information that describes a Microsoft Dynamics GP transaction or document.

Information about the eConnect XML documents includes the following:

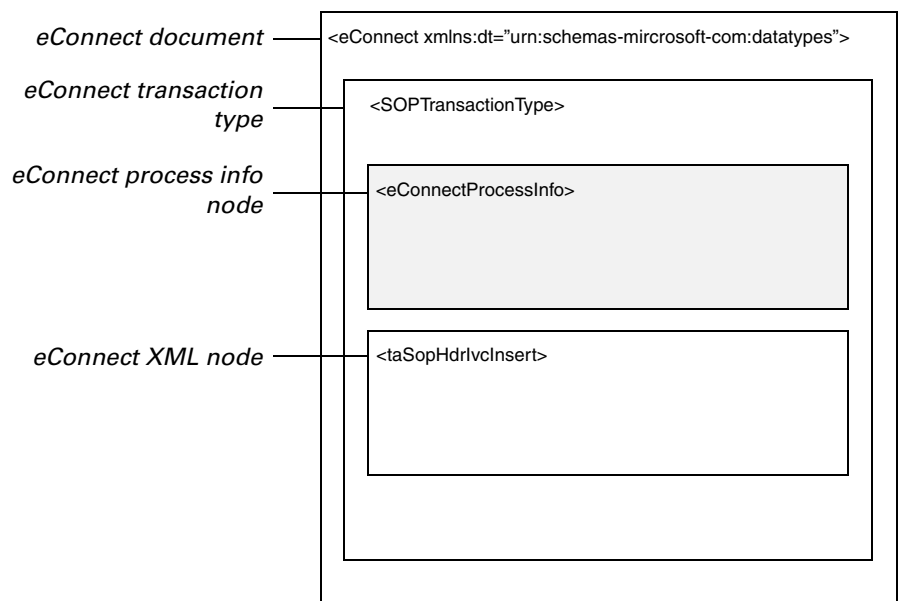
- [\*eConnect XML document structure\*](#)
- [\*Creating eConnect XML documents\*](#)
- [\*Sample eConnect XML documents\*](#)
- [\*Using eConnect to update existing data\*](#)
- [\*Automating document number assignment\*](#)
- [\*Special characters in eConnect XML documents\*](#)

### eConnect XML document structure

The eConnect schemas define eConnect XML documents as hierarchical structures of schema types and XML nodes. The hierarchy establishes parent/child relationship between each level of the document. The component layers of an eConnect XML document include the following:

- eConnect document
- eConnect transaction type
- eConnect XML node

The following diagram illustrates a simple eConnect XML document:



Notice how this eConnect document implements the parent/child relationships specified by the eConnect schema. The eConnect document is the parent to a single eConnect transaction type schema. The transaction type is the parent to the `<eConnectProcessInfo>` node and a single eConnect XML node. The XML nodes

contain a set of elements which can be populated with data values. All the XML documents you use with eConnect will follow this basic pattern.

## eConnect document

At its most basic, an eConnect XML document is a text-based data structure that contains a single <eConnectType> node. This is the document's parent node. The child nodes of the <eConnectType> contain XML that describe individual transactions in Microsoft Dynamics GP.

A document's <eConnectType> node defines the scope of the SQL transaction that eConnect uses when processing the document. If any child transaction of the <eConnectType> node fails, the subsequent rollback removes all transactions included with that document.

When you construct an XML document, you should include information related to a single Microsoft Dynamics GP operation. This ensures all the component pieces of the operation are consistently applied or rolled back.

If you encounter a situation that requires using unrelated transaction types within a document, evaluate whether a SQL rollback will cause problems for your application or your Microsoft Dynamics GP data. When you include multiple types inside a single document, ensure each child transaction uses the correct transaction type XML tag.

## eConnect Transaction Type

The <eConnectType> parent node can have one or more than one child nodes. The eConnect schema requires that each child node of the <eConnectType> node to be an eConnect transaction type. For example, in a document that creates a new customer, add the <RMCustomerMasterType> transaction type node to the document.

An eConnect transaction type is a XML entity that describes a Microsoft Dynamics GP document and operation. The transaction type is the parent node to one or more XML nodes.

Some transaction types contain XML nodes that may include one or more than one of a specified XML node. The schema identifies these nodes by appending “\_Items” to the node name.

## The <eConnectProcessInfo> node

The eConnect schema specifies that the first XML node of each eConnect transaction type schema must be an <eConnectProcessInfo> node. You can use the elements of the <eConnectProcessInfo> node to change how specific transaction types are processed. The following example uses the <eConnectProcessInfo> node to override the default eConnect connection string:

```
<eConnectProcessInfo>
  <ConnectionString> Provider=SQLOLEDB.1;
    Integrated Security= SSPI;
    PersistSecurity Info= False;
    Initial Catalog= TWO;
    Data Source= machinename
  </ConnectionString>
</eConnectProcessInfo>
```



The order in which the nodes, or elements, are processed can vary depending on the API you use. If you use the COM library (eConnect9.dll), custom third-party transaction types are always processed before the eConnect transaction types. To process third-party elements after the core eConnect elements, set the <eConnectProcessRunFirst> element to True. The following XML sample demonstrates how to populate this element:

```
<eConnectProcessInfo >
    <eConnectProcsRunFirst>TRUE</eConnectProcsRunFirst>
</eConnectProcessInfo>
```

## eConnect XML nodes

A transaction type is the parent to one or more XML nodes. An XML node is the parent to one or more XML elements. The elements contain data values. The eConnect schema defines the elements contained by each XML node.

eConnect uses the data values of the elements to perform operations on Microsoft Dynamics GP data.

The eConnect schema defines the following properties for each XML node:

- The element name. You must always supply an XML tag that names the element you are populating.
- Required fields. You must always supply a value for all required fields. If you do not populate a required field, the transaction will produce an error.
- The data type. This information tells you what the expected type is for each element. For example, an element that contains text will have a string data type.
- Data constraint. An element may restrict the value or values that it can accept. You must supply a value that satisfies the element's data constraints. If you supply a value that does not meet the data constraint, an error occurs and the transaction fails.

## Creating eConnect XML documents

Since XML is a text-based representation of data, there are many tools that can produce eConnect XML documents. You can use any text tools to produce the document as long as the XML structure and nodes comply with the eConnect schema for the document.



*For .NET development, use the .NET serialization assembly (Microsoft.Dynamics.GP.eConnect.Serialization.dll) to create XML documents. The serialization assembly helps you programmatically produce eConnect XML documents. The assembly automatically orders the XML node elements to comply with the eConnect schema.*

If you manually create XML documents, take care to ensure the schema transaction types and XML nodes are in the order specified by the Schema Reference, the XML Node Reference, or the schema file.



*If you use the .NET assembly (Microsoft.Dynamics.GP.eConnect.dll) to submit your XML document, the .NET assembly will not re-order the XML document to comply with eConnect schema requirements. The assembly submits the XML document with the XML node elements in the order you provide.*

## Sample eConnect XML documents

To assist you with developing and testing eConnect solutions, the eConnect install includes the following:

- The schema components install includes a collection of files that contain sample eConnect XML documents that contain test data. The installer typically places these files in the directory `c:\Program Files\Common Files\Microsoft Shared\eConnect 10\XML Sample Documents\Incoming`. Use these files to test your solution or as a model for your XML documents.
- The schema components install includes a collection of files that contain empty eConnect XML documents. The installer typically places these files in the directory `c:\Program Files\Common Files\Microsoft Shared\eConnect 10\XML Sample Documents\Incoming empty samples`. The files contain XML that lists the transaction type schemas, XML nodes and elements for most eConnect XML document types. Use these files as models for your documents or use the XML they contain to help you create new eConnect XML documents.

## Using eConnect to update existing data

Many eConnect XML documents allow you to update existing Microsoft Dynamics GP data documents. To perform an update, your eConnect XML document must include XML nodes that provide update functionality.

XML nodes with update functionality represent eConnect business objects that can determine whether the node identifies an existing Microsoft Dynamics GP data document. If the document exists, the business object updates that document. If an existing document is not found, the business object creates a new Microsoft Dynamics GP data document.

When the eConnect business object updates an existing Microsoft Dynamics GP document, it uses one of the following techniques:

- The business object completes a document exchange. A document exchange replaces all existing data with the values supplied by the XML node elements. If the XML node leaves an element empty, the business object replaces the previous value in Microsoft Dynamics GP with the eConnect default value. Document exchange requires your XML node to include values for all the elements and not just the elements that are being updated.
- The business object completes field level updates. Field level updates allow your XML node to include only the elements that have new values. If the XML node excludes an element, the existing value in Microsoft Dynamics GP remains unchanged.

## Automating document number assignment

eConnect XML documents that create Microsoft Dynamics GP documents require you to supply a number that uniquely identifies the Microsoft Dynamics GP document being created. To simplify the numbering of new documents, several types of eConnect XML documents can automatically retrieve and assign a document number from Microsoft Dynamics GP.

To have eConnect supply the document number, your XML document must contain a schema that supports automatic numbering. The XML nodes in the schema must include the XML element that specifies the document number, but the element's value must remain empty.

The following XML example shows an XML node from a GL transactions schema that uses eConnect to assign the GL journal entry number. Notice how the value of the <JRNENTRY> element of the <taGLTransactionHeaderInsert> XML node is empty.

```
<taGLTransactionHeaderInsert>
  <BACHNUMB>TEST14</BACHNUMB>
  <JRNENTRY></JRNENTRY>
  <REFERENCE>General Transaction</REFERENCE>
  <TRXDATE>2007-01-21</TRXDATE>
  <RVRSNGDT>1900-01-01</RVRSNGDT>
  <TRXTYPE>0</TRXTYPE>
  <SQNCLINE>16384</SQNCLINE>
</taGLTransactionHeaderInsert>
```

The empty <JRNENTRY> element prompts eConnect to query Microsoft Dynamics GP for the next available GL journal entry number. The query returns a number that eConnect uses to populate the empty <JRNENTRY> element.



*SOP documents do not require you to include an empty element. For new SOP documents, eConnect automatically assigns the next SOP document number if the <SOPNUMBE> element is empty or the SOP transaction schema's XML node excludes the <SOPNUMBE> element.*

The following table displays the eConnect documents and schemas that support the automatic assignment of document numbers:

| Document Type             | Schema name              | XML element           |
|---------------------------|--------------------------|-----------------------|
| General Ledger            | GL transactions          | <JRNENTRY></JRNENTRY> |
| Inventory                 | IV inventory transaction | <IVDOCNBR></IVDOCNBR> |
| Inventory                 | IV inventory transfer    | <IVDOCNBR></IVDOCNBR> |
| Purchase Order Processing | POP receivings           | <POPRCTNM></POPRCTNM> |
| Purchase Order Processing | POP transaction          | <PONUMBER></PONUMBER> |
| Purchasing                | PM transaction           | <DOCNUMBR></DOCNUMBR> |
| Receivables               | RM transaction           | <DOCNUMBR></DOCNUMBR> |
| Sales Order Processing    | SOP transaction          | <SOPNUMBE></SOPNUMBE> |

If your eConnect XML document populates the XML elements that identify the document, eConnect always uses your value when creating the Microsoft Dynamics GP document.



*To automatically assign a number to a SOP document, eConnect uses the values of the <SOPTYPE> and <DOCID> elements of the <taSopHdrIvcInsert> XML node to determine the type of SOP document number to retrieve.*

The following XML example shows an eConnect XML document that uses automatic document numbering. Notice how the <taGLTransactionLineInsert> and <taGLTransactionHeaderInsert> XML nodes include an empty <JRNENTRY> element.

```

<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <GLTransactionType>
    <taGLTransactionLineInsert_Items>
      <taGLTransactionLineInsert>
        <BACHNUMB>TEST14</BACHNUMB>
        <JRNENTRY></JRNENTRY>
        <SQNCLINE>16384</SQNCLINE>
        <ACTINDX>0</ACTINDX>
        <CRDTAMNT>15.00</CRDTAMNT>
        <DEBITAMT>0.00</DEBITAMT>
        <ACTNUMST>000-2300-00</ACTNUMST>
      </taGLTransactionLineInsert>
      <taGLTransactionLineInsert>
        <BACHNUMB>TEST14</BACHNUMB>
        <JRNENTRY></JRNENTRY>
        <SQNCLINE>32768</SQNCLINE>
        <ACTINDX>0</ACTINDX>
        <CRDTAMNT>0.00</CRDTAMNT>
        <DEBITAMT>15.00</DEBITAMT>
        <ACTNUMST>000-2310-00</ACTNUMST>
      </taGLTransactionLineInsert>
    </taGLTransactionLineInsert_Items>
    <taGLTransactionHeaderInsert>
      <BACHNUMB>TEST14</BACHNUMB>
      <JRNENTRY></JRNENTRY>
      <REFERENCE>General Transaction</REFERENCE>
      <TRXDATE>2007-01-21</TRXDATE>
      <RVRSNGDT>1900-01-01</RVRSNGDT>
      <TRXTYPE>0</TRXTYPE>
      <SQNCLINE>16384</SQNCLINE>
    </taGLTransactionHeaderInsert>
  </GLTransactionType>
</eConnect>

```

## Special characters in eConnect XML documents

If your XML data contains one or more special characters, you must add a CDATA format tag to your data element. The following table lists the special characters that require the use of a CDATA tag.

| Special character | Special meaning | Entity encoding |
|-------------------|-----------------|-----------------|
| <                 | Begins a tag    | &lt;            |
| >                 | Ends a tag      | &gt;            |
| "                 | Quotation mark  | &quot;          |
| '                 | Apostrophe      | &apos;          |
| &                 | Ampersand       | &amp;           |

The MSXML parser requires a CDATA format tag when you use one of these characters. The following example demonstrates the use of a CDATA format tag:

```

<VENDNAME>
  <![CDATA[ Consolidated Telephone & Telegraph]]>
</VENDNAME>

```

## Chapter 5: XML Document Examples

The portion of the documentation contains XML examples that demonstrate the eConnect XML document structure. The examples include the following documents:

- [\*Create a customer\*](#)
- [\*Delete a customer address\*](#)
- [\*Retrieve a single customer\*](#)
- [\*Assign a document number\*](#)

### Create a customer

This example demonstrates an eConnect XML document that creates a new customer in Microsoft Dynamics GP. Note the following characteristics of the document:

- The eConnect document contains a single <RMCustomerMasterType> transaction type schema. The transaction type schema contains the XML nodes that represent the new customer.
- To ensure the new customer record is available to third-party eConnect applications, the value of the <eConnectProcsRunFirst> element of the <eConnectProcessInfo> XML node is set to TRUE.
- The <RMCustomerMasterType> transaction type schema uses a <taUpdateCreateCustomerRcd> XML node to describe the customer.
- The elements of the <taUpdateCreateCustomerRcd> XML node are populated with the data for the new customer.

The XML from the document is as follows:

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <RMCustomerMasterType>
    <eConnectProcessInfo>
      <eConnectProcsRunFirst>TRUE</eConnectProcsRunFirst>
    </eConnectProcessInfo>
    <taUpdateCreateCustomerRcd>
      <CUSTNMBR>JEFF0002</CUSTNMBR>
      <CUSTNAME>JL Lawn Care Service</CUSTNAME>
      <STMTNAME>JL Lawn Care Service</STMTNAME>
      <SHRTNAME>JL Lawn Care</SHRTNAME>
      <ADRSCODE>PRIMARY</ADRSCODE>
      <ADDRESS1>123 Main Street</ADDRESS1>
      <CITY>Valley City</CITY>
      <STATE>ND</STATE>
      <ZIPCODE>58072</ZIPCODE>
      <COUNTRY>USA</COUNTRY>
      <PHNUMBR1>55532336790000</PHNUMBR1>
      <PHNUMBR2>55551161817181</PHNUMBR2>
      <FAX>55584881000000</FAX>
      <UPSZONE>red</UPSZONE>
      <SHIPMTHD>PICKUP</SHIPMTHD>
      <TAXSCHID>USALLEXMPT-0</TAXSCHID>
    </taUpdateCreateCustomerRcd>
  </RMCustomerMasterType>
</eConnect>
```

```

        <PRBTADCD>PRIMARY</PRBTADCD>
        <PRSTADCD>PRIMARY</PRSTADCD>
        <STADDRCD>PRIMARY</STADDRCD>
        <SLPRSNID>GREG E.</SLPRSNID>
        <SALSTERR>TERRITORY 6</SALSTERR>
        <COMMENT1>comment1</COMMENT1>
        <COMMENT2>comment2</COMMENT2>
        <PYMTRMID>Net 30</PYMTRMID>
        <CHEKBKID>PAYROLL</CHEKBKID>
        <KPCALHST>0</KPCALHST>
        <RMCSHACTNUMST>000-1100-00</RMCSHACTNUMST>
        <UseCustomerClass>0</UseCustomerClass>
        <UpdateIfExists>1</UpdateIfExists>
    </taUpdateCreateCustomerRcd>
</RMCustomerMasterType>
</eConnect>

```

## Delete a customer address

This example demonstrates an eConnect XML document that deletes a customer address from Microsoft Dynamics GP. Note the following characteristics of the document:

- The eConnect document contains a single <RMDeleteCustomerAddressType> transaction type schema. The transaction type schema contains the XML node that specifies the customer address to remove.
- The <RMDeleteCustomerAddress> transaction type schema uses a <taDeleteCustomerAddress> XML node to describe the customer.
- The required elements of the <taDeleteCustomerAddress> XML node are populated with the data that identifies the customer address to delete.

The XML from the document is as follows:

```

<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <RMDeleteCustomerAddressType>
        <taDeleteCustomerAddress>
            <CUSTNMBR>AARONFIT0001</CUSTNMBR>
            <ADRSCODE>WAREHOUSE</ADRSCODE>
        </taDeleteCustomerAddress>
    </RMDeleteCustomerAddress>
</eConnect>

```

## Retrieve a single customer

This example demonstrates an eConnect XML document that retrieves a customer record from Microsoft Dynamics GP. Note the following characteristics of the document:

- The eConnect document contains a single <RQeConnectOutType> transaction type schema. The transaction type schema contains the XML nodes that describe the customer record to retrieve.
- The <eConnectProcessInfo> XML node's <Outgoing> element indicates this is a data request. The <MessageID> element contains a descriptor.

- The <RQeConnectOutType> transaction type schema uses a <eConnectOut> XML node to describe the customer.
- The <OUTPUTTYPE> element instructs eConnect to return the complete customer record. The <INDEX1TO> and <INDEX1FROM> elements identify the customer. The values of the <FORLOAD>, <FORLIST>, and <ACTION> elements instruct eConnect to return the document directly and not create a record in the eConnect\_Out table.

The XML from the document is as follows:

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <RQeConnectOutType>
    <eConnectProcessInfo>
      <Outgoing>TRUE</Outgoing>
      <MessageID>Customer</MessageID>
    </eConnectProcessInfo>
    <eConnectOut>
      <DOCTYPE>Customer</DOCTYPE>
      <OUTPUTTYPE>2</OUTPUTTYPE>
      <INDEX1TO>ADAMPARK0001</INDEX1TO>
      <INDEX1FROM>ADAMPARK0001</INDEX1FROM>
      <FORLOAD>0</FORLOAD>
      <FORLIST>1</FORLIST>
      <ACTION>0</ACTION>
      <ROWCOUNT>0</ROWCOUNT>
      <REMOVE>0</REMOVE>
    </eConnectOut>
  </RQeConnectOutType>
</eConnect>
```

## Assign a document number

This example uses an eConnect XML document to create a Microsoft Dynamics GP sales order. The content of the document prompts eConnect to query Microsoft Dynamics GP for the next sales order number. eConnect uses the result of the query to populate the document's <SOPNUMBE> elements. Note the following characteristics of the document:

- The <taSopLineIvcInsert> and <taSopHdrIvcInsert> XML nodes include the required <SOPNUMBE> element but do not provide a value for the element.
- The <taSopLineIvcInsert> and <taSopHdrIvcInsert> XML nodes populate the <SOPTYPE> element with the value 2 to indicate the document is a sales order.
- The <taSopHdrIvcInsert> XML node's <DOCID> element contains the value STDORD.
- eConnect uses the values of the <SOPTYPE> and <DOCID> elements to determine the type of Microsoft Dynamics GP document number to request.

The XML from the document is as follows:

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <SOPTransactionType>
    <taSopLineIvcInsert_Items>
```

```

<taSopLineIvcInsert>
  <SOPTYPE>2</SOPTYPE>
  <SOPNUMBE></SOPNUMBE>
  <CUSTNMBR>ALTONMAN0001</CUSTNMBR>
  <DOCDATE>2007-03-03</DOCDATE>
  <LOCNCODE>WAREHOUSE</LOCNCODE>
  <ITEMNMBR>ACCS-CRD-12WH</ITEMNMBR>
  <UNITPRCE>9.95</UNITPRCE>
  <XTNDPRCE>19.90</XTNDPRCE>
  <QUANTITY>2</QUANTITY>
  <MRKDNAMT>0</MRKDNAMT>
  <COMMTID>TEST</COMMTID>
  <COMMENT_1>cmt1</COMMENT_1>
  <COMMENT_2>cmt2</COMMENT_2>
  <COMMENT_3>cmt3</COMMENT_3>
  <COMMENT_4>cmt4</COMMENT_4>
  <ITEMDESC>yes</ITEMDESC>
  <TAXAMNT>0</TAXAMNT>
  <QTYONHND>0</QTYONHND>
  <QTYRTRND>0</QTYRTRND>
  <QTYINUSE>0</QTYINUSE>
  <QTYINSVC>0</QTYINSVC>
  <QTYDMGED>0</QTYDMGED>
  <NONINVEN>0</NONINVEN>
  <LNITMSEQ>0</LNITMSEQ>
  <DROPSHIP>0</DROPSHIP>
  <QTYTBAOR>0</QTYTBAOR>
  <DOCID>STDORD</DOCID>
  <SALSTERR>TERRITORY 2</SALSTERR>
  <SLPRSNID>GREG E.</SLPRSNID>
</taSopLineIvcInsert>
</taSopLineIvcInsert_Items>
<taSopHdrIvcInsert>
  <SOPTYPE>2</SOPTYPE>
  <DOCID>STDORD</DOCID>
  <SOPNUMBE></SOPNUMBE>
  <ORIGNUMB>0</ORIGNUMB>
  <ORIGTYPE>0</ORIGTYPE>
  <TAXSCHID>USASTCITY-6*</TAXSCHID>
  <FRTSCHID>USASTCITY-6*</FRTSCHID>
  <MSCSCHID>USASTCITY-6*</MSCSCHID>
  <SHIPMTHD>UPS GROUND</SHIPMTHD>
  <TAXAMNT>0</TAXAMNT>
  <LOCNCODE>WAREHOUSE</LOCNCODE>
  <DOCDATE>2007-03-03</DOCDATE>
  <FREIGHT>3.00</FREIGHT>
  <MISCAMNT>2.00</MISCAMNT>
  <TRDISAMT>0</TRDISAMT>
  <DISTKNAM>0</DISTKNAM>
  <MRKDNAMT>0</MRKDNAMT>
  <CUSTNMBR>ALTONMAN0001</CUSTNMBR>
  <CUSTNAME>Alton Manufacturing</CUSTNAME>
  <CSTPONBR>4859</CSTPONBR>
  <ShipToName>SERVICE</ShipToName>
  <ADDRESS1>P.O. Box 3333</ADDRESS1>
  <CNTCPRSN>person1</CNTCPRSN>

```



```

<FAXNUMBR>55553200810000</FAXNUMBR>
<CITY>Detroit</CITY>
<STATE>MI</STATE>
<ZIPCODE>48233-3343</ZIPCODE>
<COUNTRY>USA</COUNTRY>
<PHNUMBR1>55553289890000</PHNUMBR1>
<PHNUMBR3>55553200810000</PHNUMBR3>
<SUBTOTAL>19.90</SUBTOTAL>
<DOCAMNT>24.90</DOCAMNT>
<PYMTRCVD>0</PYMTRCVD>
<SALSTERR>TERRITORY 2</SALSTERR>
<SLPRSNID>GREG E.</SLPRSNID>
<USER2ENT>sa</USER2ENT>
<BACHNUMB>TEST</BACHNUMB>
<PRBTADCD>PRIMARY</PRBTADCD>
<PRSTADCD>SERVICE</PRSTADCD>
<FRTTXAMT>0</FRTTXAMT>
<MSCTXAMT>0</MSCTXAMT>
<ORDRDATE>2007-03-03</ORDRDATE>
<MSTRNUMB>0</MSTRNUMB>
<NONINVEN>0</NONINVEN>
<PYMTRMID>2% 10/Net 30</PYMTRMID>
<USINGHEADERLEVELTAXES>0</USINGHEADERLEVELTAXES>
<CREATECOMM>0</CREATECOMM>
<CREATETAXES>1</CREATETAXES>
<DEFTAXSCHDS>0</DEFTAXSCHDS>
<FREIGHTBLE>1</FREIGHTBLE>
<MISCTBLE>1</MISCTBLE>
</taSopHdrIvcInsert>
</SOPTransactionType>
</eConnect>

```





## Part 3: .NET Development

This portion of the documentation discusses eConnect support for .NET development. The eConnect install includes .NET assemblies you can use to integrate Microsoft Dynamics GP data and functionality with your Microsoft .NET solutions. The list that follows describes how to use the eConnect .NET assemblies:

- [Chapter 6, “.NET Development Overview,”](#) introduces the eConnect assemblies and namespaces. Review this information to learn how you can add eConnect to your .NET development project.
- [Chapter 7, “eConnect Assembly,”](#) provides information about the Microsoft.Dynamics.GPeConnect assembly and namespace. This assembly contains a class you can use to send and request eConnect XML documents.
- [Chapter 8, “Serialization Assembly,”](#) provides information about the Microsoft.Dynamics.GPeConnect.Serialization assembly and namespace. You use the classes in the assembly when you need to construct an eConnect XML document.
- [Chapter 9, “Miscellaneous Routines Assembly,”](#) provides information about the Microsoft.Dynamics.GPeConnect.MiscRoutines assembly and namespace. You use the classes in this assembly to retrieve certain types of information from Microsoft Dynamics GP.

## Chapter 6: .NET Development Overview

The following is an introduction to using eConnect with .NET. These topics are discussed:

- [\*eConnect and .NET\*](#)
- [\*Adding a reference\*](#)
- [\*Including the namespace\*](#)

### eConnect and .NET

A .NET assembly is the fundamental building block of all .NET applications. An assembly includes the types and resources need to produce a logical unit of functionality. An assembly is usually stored as a .dll or .exe file.

The install of the eConnect .NET components includes the following assemblies:

- Microsoft.Dynamics.GP.eConnect.dll
- Microsoft.Dynamics.GP.eConnect.MiscRoutines.dll
- Microsoft.Dynamics.GP.eConnect.Serialization.dll

The eConnect installer typically places these files in the directory c:\Program Files\Common Files\Microsoft Shared\eConnect 10\Objects\Dot Net.

These assemblies allow you to add eConnect functionality to a .NET solution. To include eConnect in a .NET development project, you must add a reference to the eConnect assemblies to your development project.



*To use the eConnect assemblies, you must have Microsoft .NET Framework 2.0, or Microsoft Visual Studio 2005 or later installed on your computer.*

### Adding a reference

To use the objects in the assembly, create a reference to that assembly. The reference allows you to access the properties, methods, and events of defined objects and apply them in your programming. Use Microsoft Visual Studio to complete the following procedure:

**1. Open the Add Reference dialog window.**

From the Project menu, select Add Reference. The Add Reference dialog window opens and displays the .NET tab.

**2. Click Browse.**

Click the Browse button to open the Select Component dialog window.

**3. Navigate to the assembly file.**

Use the navigation options in the Select Component dialog to view the contents of the c:\Program Files\Common Files\Microsoft Shared\eConnect 10\Objects\Dot Net. Select the assembly and click Open.

**4. Close the Add Reference dialog window.**

The Selected Components text box should contain the name of the assembly selected. Click OK. The assembly is added to the list of references in your project.

## Including the namespace

Each eConnect assembly defines a namespace for the classes that it contains. A .NET namespace is a second organizational method that groups type names in an effort to reduce the chance of a name collision. The eConnect namespaces match the assembly that contains them. The namespaces are as follows:

- Microsoft.Dynamics.GP.eConnect
- Microsoft.Dynamics.GP.eConnect.MiscRoutines
- Microsoft.Dynamics.GP.eConnect.Serialization

You use these namespaces to specify the type of the eConnect object when you instantiate it. The following Visual Basic .NET example demonstrates the use of an eConnect namespace:

```
'Use GetSopNumber from the Microsoft.Dynamics.GP.eConnect.MiscRoutines
'namespace
Dim SopNumber As New Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetSopNumber
```

In this example, Microsoft.Dynamics.GP.eConnect.MiscRoutines specifies the namespace. GetSopNumber is the class that is being instantiated.

To simplify your code, use the **Imports** statement in Visual Basic or **using** statement in C#. These statements allow you to use only the portion of the namespace necessary to supply a unique reference.

The following **Imports** statement adds the Microsoft.Dynamics.GP.eConnect.MiscRoutines namespace to the .vb file in the project:

```
Imports Microsoft.Dynamics.GP.eConnect.MiscRoutines
```

The following C# example shows the **using** statement:

```
using Microsoft.Dynamics.GP.eConnect.MiscRoutines;
```

After you add these statements, you supply only the eConnect class name. The following Visual Basic .NET example shows how to import the Microsoft.Dynamics.GP.eConnect.MiscRoutines namespace and create a new GetSopNumber object.

```
Imports Microsoft.Dynamics.GP.eConnect.MiscRoutines

'Use GetSopNumber from the Microsoft.Dynamics.GP.eConnect.
'MiscRoutines namespace
Dim SopNumber As New GetSopNumber
```

## Chapter 7: eConnect Assembly

The Microsoft.Dynamics.GP.eConnect assembly allows you to send and request eConnect XML documents. The following topics describe how to use the eConnect assembly and what it can do for your applications

- [\*Microsoft.Dynamics.GP.eConnect assembly\*](#)
- [\*eConnectMethods class\*](#)
- [\*EnumTypes class\*](#)
- [\*eConnectException class\*](#)

### Microsoft.Dynamics.GP.eConnect assembly

The assembly contains the Microsoft.Dynamics.GP.eConnect namespace. This namespace contains three classes:

**eConnectMethods** The eConnectMethods class allows you to send and receive eConnect XML documents. You will use this class to perform most operations.

**EnumTypes** The EnumTypes class contains enumerations you use as parameters for some methods of the eConnectMethods class.

**eConnectException** The eConnectException class allows you to catch eConnect- specific errors.



*To use the classes in the Microsoft.Dynamics.GP.eConnect namespace, you must include a reference to the Microsoft.Dynamics.GP.eConnect.MiscRoutines assembly in your project.*

### eConnectMethods class

The eConnectMethods class inherits from **System.EnterpriseService.ServicedComponent** class. This parent class enables eConnectMethods to use the services supplied by the eConnect COM+ object. Refer to the .NET Framework documentation for information about the **System.EnterpriseService.ServicedComponent** class.

To use the eConnectMethods class you must first instantiate an object. The following Visual Basic .NET code sample creates an eConnectMethods object:

```
'Instantiate an eConnectMethods object  
Dim eConnectObject As New eConnectMethods
```

The eConnectMethods classes provides two methods that you use to send or request data:

## eConnect\_EntryPoint method

The eConnect\_EntryPoint method allows you to submit an XML document. The method has five parameters

| Parameter        | Data type   | Description   |
|------------------|---|---|
| ConnectionString | string  | Specifies your data server and database.  |
| ConnectionType   | Microsoft.Dynamics.GP.eConnect.EnumTypes.ConnectionStringType | Use the ConnectionStringType enumeration member that specifies the type of your data server. ConnectionStringType includes the following members:<br>SqlClient<br>OleDb |
| sXML             | string  | An eConnect XML document.   |
| ValidationType   | Microsoft.Dynamics.GP.eConnect.EnumTypes.SchemaValidationType | Use a SchemaValidationType enumeration member to specify the type of data validation to perform. SchemaValidationType includes the following members:<br>None<br>XSD    |
| eConnectSchema   | string  | Optional.If you set the ValidationType parameter to use XSD validation, you must specify the filepath to the .xsd file that contains the schema definition.             |



*For detailed information about eConnect connection strings, see the eConnect Installation chapter of the eConnect Installation and Administration Guide.*

The method returns a boolean value that indicates whether the XML document was successfully submitted. A return value of True indicates the operation was successfully completed.

The following Visual Basic .NET example uses the **eConnect\_EntryPoint** method to submit an eConnect XML document:

```
Dim ConnectionString As String
Dim eConnectResult As Boolean
Dim eConnectObject As New eConnectMethods
Dim xmlDoc As XmlDocument

'Set the connection string
'This connection string uses integrated security to connect to the
'TWO database on the local computer
ConnectionString = "DataSource=127.0.0.1;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"

'Load the contents of the textbox into the xmlDoc object
xmlDoc.LoadXml(XmlDoc_TextBox.Text)

'Instantiate an eConnectMethods object
Dim eConnectObject As New eConnectMethods

'If eConnectResult is TRUE, the XML document was successfully submitted
eConnectResult = eConnectObject.eConnect_EntryPoint(ConnectionString, _
    EnumTypes.ConnectionStringType.SqlClient, _
    xmlDoc.OuterXml, _
    EnumTypes.SchemaValidationType.None)
```



Notice that the `ConnectionType` parameter uses the `EnumTypes.ConnectionStringType.SqlClient` enumeration to specify a SQL client connection. The `ValidationType` parameter uses the `EnumTypes.SchemaValidationType.None` enumeration to specify that the XML document is not validated. If you set the `ValidationType` parameter to `None`, the `eConnect_EntryPoint` method does not require the `eConnectSchema` parameter. The parameter defaults to an empty string.

## eConnect\_Requester method

The `eConnect_Requester` method allows you to retrieve data from Microsoft Dynamics GP. The method has three parameters:

| Parameter                     | Data type   | Description   |
|-------------------------------|---|---|
| <code>ConnectionString</code> | string  | Specifies your data server and database.  |
| <code>ConnectionType</code>   | Microsoft.Dynamics.GP.eConnect.EnumTypes.ConnectionStringType | Use the <code>ConnectionStringType</code> enumeration member that specifies the type of your data server. <code>ConnectionStringType</code> includes the following members:<br>SqlClient<br>OleDb |
| <code>sXML</code>             | string  | An eConnect XML document.   |

The method returns a string. The string is an XML document that represents the requested data.

The following Visual Basic .NET example uses the `eConnect_Requester` method to retrieve and display an XML document:

```
Dim ConnectionString As String
Dim eConnectObject As New eConnectMethods
Dim xmlDoc As XmlDocument

'Set the connection string
'This connection string uses integrated security to connect to the
'TWO database on the local computer
ConnectionString = "DataSource=127.0.0.1; Integrated Security=SSPI;" _
    & "Persist Security Info=False; Initial Catalog=TWO;"

'Instantiate an eConnectMethods object
Dim eConnectObject As New eConnectMethods

'Display the XML document in a textbox
ReturnData_TextBox.Text=eConnectObject.eConnect_Requester(ConnectionString, _
    EnumTypes.ConnectionStringType.SqlClient, _
    xmlDoc.OuterXml)
```

In this example, the `sXML` parameter is a specific type of XML document used to request data. Refer to [Chapter 5, “XML Document Examples.”](#) for an example of a requester document.

## EnumTypes class

The EnumTypes class defines two enumerations. The enumerations are as follows:

### SchemaValidationType enumeration

You use this enumeration in the eConnect\_EntryPoint method to specify the type of schema validation. If you use a value other than None, you must populate the eConnect\_EntryPoint method's eConnectSchema parameter. The following table displays the enumeration members.

| Member name | Value | Description                     |
|-------------|-------|---------------------------------|
| None        | 0     | No validation is performed      |
| XSD         | 1     | Use an .xsd file for validation |

### ConnectionStringType enumeration

You use this enumeration to specify the type of data server. The following table displays the enumeration members.

| Member name | Value | Description   |
|-------------|-------|---|
| SqlClient   | 0     | The connection string is for Microsoft SQL Server                             |
| OleDb       | 1     | The connection string is for a database server that supports ODBC connections |

## eConnectException class

The eConnectException class allows you to catch and handle eConnect-specific errors. If an error occurs during a call to either the **eConnect\_EntryPoint** or **eConnect\_Requester** methods, eConnect throws an eConnectException object. You can use the properties of the eConnectException object to identify the type of error and its accompanying error message.

The eConnectException class inherits from the System.ApplicationException class. eConnectException uses the properties of the parent class. Refer to the .NET Framework documentation for information about the System.ApplicationException class.

You should include code that can catch and handle eConnect exceptions. The most common exception handling technique is to use a Try/Catch block. The Catch statement allows you to specify the type of exception you want to handle. When you catch an eConnectException, you can attempt to correct the error, immediately report the error to the user, or record the error in a log.

The following segment of Visual Basic .NET code illustrates the handling of an eConnect exception:

```
Dim ConnectionString As String
Dim eConnectResult As Boolean
Dim eConnectObject As New eConnectMethods
Dim xmlDoc As XmlDocument

'Set the connection string
'This connection string uses integrated security to connect to the
'TWO database on the local computer
ConnectionString = "DataSource=127.0.0.1;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"
```

```

'Load the contents of the textbox into the xmlDoc object
xmlDoc.LoadXml(XmlDoc_TextBox.Text)

Try
    'Instantiate an eConnectMethods object
    Dim eConnectObject As New eConnectMethods

    'If eConnectResult is TRUE, the XML document was successfully submitted
    eConnectResult = eConnectObject.eConnect_EntryPoint(ConnectionString, _
        EnumTypes.ConnectionStringType.SqlClient, _
        xmlDoc.OuterXml, _
        EnumTypes.SchemaValidationType.None)

    'If an eConnect error occurs, display the error message
    Catch eConnectError as eConnectException
        ReturnData_TextBox.Text = eConnectError.Message
    'If an unexpected error occurs, display the error message
    Catch ex As Exception
        ReturnData_TextBox.Text = ex.Message
    End Try

```

Notice how the first Catch statement handles an eConnectException and the second handles all other exception types. In this example, the application informs the user of the error by displaying the exception object's message property in a text box.



## Chapter 8: Serialization Assembly

The Microsoft.Dynamics.GP.eConnect.Serialization assembly provides additional classes you can use to create eConnect XML documents. The following items are discussed:

- [\*Microsoft.Dynamics.GP.eConnect.Serialization assembly\*](#)
- [\*Serialization classes\*](#)
- [\*eConnect serialization example\*](#)
- [\*Using serialization flags\*](#)
- [\*eConnectOut serialization example\*](#)

### Microsoft.Dynamics.GP.eConnect.Serialization assembly

The assembly contains the Microsoft.Dynamics.GP.eConnect.Serialization namespace. The namespace includes classes that represent each eConnect transaction type schema and XML node.

To view the list of transaction type schemas and XML nodes, refer to the Schema Reference and the XML Node References in the eConnect help. You may also use the Visual Studio Object Browser to view the individual schema and XML node classes.

### Serialization classes

The serialization classes allow you to instantiate objects that represent eConnect transaction types and XML nodes. The following Visual Basic .NET sample instantiates a <taSopHdrIvcInsert> XML node and populates it with values:

```
Dim salesHdr As New taSopHdrIvcInsert

With salesHdr
    .SOPTYPE = 3
    .SOPNUMBE = "INV2001"
    .DOCID = "STDINV"
    .BACHNUMB = "eConnect"
    .TAXSCHID = "USASTCITY-6*"
    .FRTSCHID = "USASTCITY-6*"
    .MSCSCHID = "USASTCITY-6*"
    .LOCNCODE = "WAREHOUSE"
    .DOCDATE = DateString 'Today
    .CUSTNMBR = "CONTOSOL0001"
    .CUSTNAME = "Contoso, Ltd"
    .ShipToName = "WAREHOUSE"
    .ADDRESS1 = "2345 Main St."
    .CNTCPRSN = "Joe Healy"
    .FAXNUMBR = "13125550150"
    .CITY = "Aurora"
    .STATE = "IL"
    .ZIPCODE = "65700"
    .COUNTRY = "USA"
    .SUBTOTAL = 53.8
    .DOCAMNT = 53.8
    .USINGHEADERLEVELTAXES = 0
    .PYMTRMID = "Net 30"
End With
```

You can combine this with other objects to create an eConnect transactions type. The following Visual Basic .NET example creates a new SOPTransactionType object and populates the taSopHdrIvcInsert property with the object created in the previous example:

```
Dim salesOrder As New SOPTransactionType

salesOrder.taSopHdrIvcInsert = salesHdr
```

Once the transaction type is populated, you can add it to the eConnectType object. The following Visual Basic .NET sample instantiates an eConnectType object and populates its SOPTransactionType property with the object from the previous example:

```
Dim eConnect As New eConnectType

eConnect.SOPTransactionType = salesOrder
```

The eConnect object now represents a complete eConnect XML document. With a document object created, you can perform the following tasks:

- You can use the eConnect object as the XML document parameter of the eConnect.eConnect\_EntryPoint method.
- You can also use the classes with .NET serialization to write the XML document to a file. The file can be used with Microsoft message queuing (MSMQ) or stored to a disk.

## eConnect serialization example

The following Visual Basic .NET sample demonstrates a solution that uses an eConnect sales invoice object. It serializes the object as XML to a file, and then creates an XML document object from the file. It then uses the XML document to create a new sales invoice in Microsoft Dynamics GP.

For more information about using serialization with .NET, refer to the .NET Framework SDK.

As you review the sample code, note the following actions:

- The Main subroutine begins by calling the SerializeSalesOrderObject subroutine.
- The SerializeSalesOrderObject subroutine creates a hierarchy of objects that correlate to an eConnect XML document. The eConnect object is populated with a SOPTransactionType object. The SOPTransactionType contains two taSopLineIvcInsert XML nodes and a taSopHdrIvcInsert XML node. The data in the properties of these objects describes a new sales order for Microsoft Dynamics GP.
- The SerializeSalesOrderObject subroutine uses an XMLSerializer to create an XML representation of the eConnect object. A FileStream object writes the serialized XML to a file named SalesOrder.xml

- The Main subroutine opens the SalesOrder.xml file and loads the serialized XML into an XmlDocument object.
- A string is created from the XML in the XmlDocument object.
- A string is created that contains the connection parameters for the Microsoft Dynamics GP data server



*The connection string must specify a server where eConnect business objects are installed.*

- The Main subroutine instantiates an eConnectMethods object. The object's eConnect\_EntryPoint method is called. The connection string and XML string are passed as parameters. eConnect attempts to create the document defined in the XML.
- Notice how Try/Catch blocks are used to handle both eConnect and system exceptions.

The following Visual Basic .NET code performs the prescribed actions:

```
Imports System
Imports System.Xml
Imports System.Xml.Serialization
Imports System.IO
Imports System.Text
Imports Microsoft.Dynamics.GP.eConnect
Imports Microsoft.Dynamics.GP.eConnect.Serialization

Public Class CreateInvoice
    Public Shared Sub Main()

        Dim salesInvoice As New CreateInvoice
        Dim salesOrderDocument As String
        Dim sConnectionString As String
        Dim eConCall As New eConnectMethods

        Try
            'Call the SerializeSalesOrderObject subroutine and specify
            'a file name
            salesInvoice.SerializeSalesOrderObject("SalesOrder.xml")

            'Create an XML document object and load it with the XML from the
            'file that the SerializeSalesOrder subroutine created
            Dim xmldoc As New Xml.XmlDocument
            xmldoc.Load("SalesOrder.xml")

            'Convert the XML to a string
            salesOrderDocument = xmldoc.OuterXml

            'Create a connection string to the Microsoft Dynamics GP server
            'Integrated Security is required (Integrated security=SSPI)
            sConnectionString = "data source=127.0.0.1;" _
                & "initial catalog=Two;integrated security=SSPI;" _
                & "persist security info=False; packet size=4096"
```

```

        'Create the invoice in Microsoft Dynamics GP
        eConCall.eConnect_EntryPoint(sConnectionString, _
            EnumTypes.ConnectionStringType.SqlClient, _
            salesOrderDocument, _
            EnumTypes.SchemaValidationType.None)

    Catch exp As eConnectException
        Console.WriteLine(exp.ToString)
    Catch ex As System.Exception
        Console.WriteLine(ex.ToString)
    Finally
        eConCall.Dispose()
    End Try

End Sub

'This subroutine creates an eConnect invoice XML document and
'writes the XML to a file
Public Sub SerializeSalesOrderObject(ByVal filename As String)

    Dim salesOrder As New SOPTransactionType
    Dim salesLine As New taSopLineIvcInsert_ItemsTaSopLineIvcInsert
    Dim salesLine2 As New taSopLineIvcInsert_ItemsTaSopLineIvcInsert
    Dim salesHdr As New taSopHdrIvcInsert
    Dim LineItems(1) As taSopLineIvcInsert_ItemsTaSopLineIvcInsert

    Try

        'Populate the elements of the first invoice line
        With salesLine
            .Address1 = "2345 Main St."
            .CUSTNMBR = "CONTOSOL0001"
            .SOPNUMBE = "INV2001"
            .CITY = "Aurora"
            .SOPTYPE = 3
            .DOCID = "STDINV"
            .QUANTITY = 2
            .ITEMNMBR = "ACCS-CRD-12Wh"
            .ITEMDESC = "Phone Cord - 12' White"
            .UNITPRCE = 10.95
            .XTNDPRCE = 21.9
            .LOCNCODE = "WAREHOUSE"
            .DOCDATE = DateString 'Today
        End With

        'Add the invoice line to the array
        LineItems(0) = salesLine

        'Populate the elements of the second invoice line
        With salesLine2
            .Address1 = "2345 Main St."
            .CUSTNMBR = "CONTOSOL0001"
            .SOPNUMBE = "INV2001"
            .CITY = "Aurora"
            .SOPTYPE = 3
            .DOCID = "STDINV"
        End With
    End Try
End Sub

```



```

        .QUANTITY = 2
        .ITEMNMBR = "ACCS-CRD-25BK"
        .ITEMDESC = "Phone Cord - 25' Black"
        .UNITPRCE = 15.95
        .XTNDPRCE = 31.9
        .LOCNCODE = "WAREHOUSE"
        .DOCDATE = DateString 'Today
End With

'Add the invoice line to the array
LineItems(1) = salesLine2

'Use the array of invoice lines to populate the transaction types
'array of line items
ReDim Preserve salesOrder.taSopLineIvcInsert_Items(1)
salesOrder.taSopLineIvcInsert_Items = LineItems

'Populate the elements of the taSopHdrIvcInsert XML node
With salesHdr
    .SOPTYPE = 3
    .SOPNUMBE = "INV2001"
    .DOCID = "STDINV"
    .BACHNUMB = "eConnect"
    .TAXSCHID = "USASTCITY-6*"
    .FRTSCHID = "USASTCITY-6*"
    .MSCSCHID = "USASTCITY-6*"
    .LOCNCODE = "WAREHOUSE"
    .DOCDATE = DateString 'Today
    .CUSTNMBR = "CONTOSOL0001"
    .CUSTNAME = "Contoso, Ltd."
    .ShipToName = "WAREHOUSE"
    .ADDRESS1 = "2345 Main St."
    .CNTCPRSN = "Joe Healy"
    .FAXNUMBR = "13125550150"
    .CITY = "Aurora"
    .STATE = "IL"
    .ZIPCODE = "65700"
    .COUNTRY = "USA"
    .SUBTOTAL = 53.8
    .DOCAMNT = 53.8
    .USINGHEADERLEVELTAXES = 0
    .PYMTRMID = "Net 30"
End With

'Add the header node to the transaction type object
salesOrder.taSopHdrIvcInsert = salesHdr

'Create an eConnect document object and populate it with
'the transaction type object
Dim eConnect As New eConnectType
ReDim Preserve eConnect.SOPTransactionType(0)
eConnect.SOPTransactionType(0) = salesOrder

'Create a file on the hard disk
Dim fs As New FileStream(filename, FileMode.Create)
Dim writer As New XmlTextWriter(fs, New UTF8Encoding)

```

```

        'Serialize using the XmlTextWriter to the file
        Dim serializer As New XmlSerializer(GetType (eConnectType))
        serializer.Serialize(writer, eConnect)
        writer.Close()

    Catch ex As System.Exception
        Console.WriteLine(ex.ToString)
    End Try

End Sub

End Class

```

If you open the file created by this sample, it contains the following XML:

```

<?xml version="1.0" encoding="utf-8"?>
<eConnect xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <SOPTransactionType>
    <taSopLineIvcInsert_Items>
      <taSopLineIvcInsert>
        <SOPTYPE>3</SOPTYPE>
        <SOPNUMBE>INV2001</SOPNUMBE>
        <CUSTNMBR>CONTOSOL0001</CUSTNMBR>
        <DOCDATE>05-07-2004</DOCDATE>
        <LOCNCODE>WAREHOUSE</LOCNCODE>
        <ITEMNMBR>ACCS-CRD-12WH</ITEMNMBR>
        <UNITPRCE>10.95</UNITPRCE>
        <XTNDPRCE>21.9</XTNDPRCE>
        <QUANTITY>2</QUANTITY>
        <ITEMDESC>Phone Cord - 12' White</ITEMDESC>
        <DOCID>STDINV</DOCID>
        <ADDRESS1>2345 Main St.</ADDRESS1>
        <CITY>Aurora</CITY>
      </taSopLineIvcInsert>
      <taSopLineIvcInsert>
        <SOPTYPE>3</SOPTYPE>
        <SOPNUMBE>INV2001</SOPNUMBE>
        <CUSTNMBR>CONTOSOL0001</CUSTNMBR>
        <DOCDATE>05-07-2004</DOCDATE>
        <LOCNCODE>WAREHOUSE</LOCNCODE>
        <ITEMNMBR>ACCS-CRD-25BK</ITEMNMBR>
        <UNITPRCE>15.95</UNITPRCE>
        <XTNDPRCE>31.9</XTNDPRCE>
        <QUANTITY>2</QUANTITY>
        <ITEMDESC>Phone Cord - 25' Black</ITEMDESC>
        <DOCID>STDINV</DOCID>
        <ADDRESS1>2345 Main St.</ADDRESS1>
        <CITY>Aurora</CITY>
      </taSopLineIvcInsert>
    </taSopLineIvcInsert_Items>
    <taSopHdrIvcInsert>
      <SOPTYPE>3</SOPTYPE>
      <DOCID>STDINV</DOCID>
      <SOPNUMBE>INV2001</SOPNUMBE>
      <TAXSCHID>USASTCITY-6*</TAXSCHID>
    </taSopHdrIvcInsert>
  </SOPTransactionType>
</eConnect>

```

```

<FRTSCHID>USASTCITY-6*</FRTSCHID>
<MSCSCHID>USASTCITY-6*</MSCSCHID>
<LOCNCODE>WAREHOUSE</LOCNCODE>
<DOCDATE>05-07-2004</DOCDATE>
<CUSTNMBR>CONTOSOL0001</CUSTNMBR>
<CUSTNAME>Contoso, Ltd.</CUSTNAME>
<ShipToName>WAREHOUSE</ShipToName>
<ADDRESS1>2345 Main St.</ADDRESS1>
<CNTCPRSN>Joe Healy</CNTCPRSN>
<FAXNUMBR>13125550150</FAXNUMBR>
<CITY>Aurora</CITY>
<STATE>IL</STATE>
<ZIPCODE>65700</ZIPCODE>
<COUNTRY>USA</COUNTRY>
<SUBTOTAL>53.8</SUBTOTAL>
<DOCAMNT>53.8</DOCAMNT>
<BACHNUMB>eConnect</BACHNUMB >
<PYMTRMID>Net 30</PYMTRMID>
</taSopHdrIvcInsert>
</SOPTransactionType>
</eConnect>

```

## Using serialization flags

The eConnect serialization assembly includes classes with properties that use serialization flags. A serialization flag specifies whether to use or discard the value assigned to that property when creating an eConnect XML document.

The XML Node Reference identifies the elements within each XML node where the eConnect serialization assembly includes a serialization flag.



*The serialization flags in the eConnect serialization assembly always append the word “Specified” to the element name.*

When you assign a value to a property that includes a serialization flag, you must also set the serialization flag’s value to True.

The following Visual Basic .NET example, demonstrates the use of the `HOLDSpecified` serialization flag for the `HOLD` property of the `taUpdateCreateCustomerRcd` class:

```

public sub SerializeCustomerObject(ByVal filename As String)
    Try
        'Create the eConnect XML document objects
        Dim customer As New taUpdateCreateCustomerRcd
        Dim customertype As New RMCustomerMasterType
        Dim eConnect As New eConnectType

        With customer
            'Assign a customer number
            .CUSTNMBR = "Customer001"

            'Place the customer on Hold.
            'The HOLD element includes a serialization flag.
            'Set the serialization flag HOLDSpecified to True.
            .HOLD = 1
        End With
    End Try
End sub

```

```

        .HOLDSpecified = True

        'Populate additional customer data
        .CUSTNAME = "Customer 1"
        .ADDRESS1 = "20 Main St"
        .ADRSCODE = "Primary"
        .CITY = "NewCity"
        .CHEKBKID = "FIRST NATIONAL"
        .ZIPCODE = "53022"
    End With

    'Add the XML node object to the customer schema object
    customertype.taUpdateCreateCustomerRcd = customer

    'Add the schema object to the eConnect document object
    Dim mySMCustomerMaster(0) As RMCustomerMasterType
    mySMCustomerMaster(0) = customertype
    eConnect.RMCustomerMasterType = mySMCustomerMaster

    'Create a file on the hard disk and an object to write the file
    Dim fs As New FileStream(filename, FileMode.Create)
    Dim writer As New XmlTextWriter(fs, New UTF8Encoding)

    'Serialize the eConnect XML document object to the file.
    Dim serializer As New XmlSerializer(eConnect.GetType())
    serializer.Serialize(writer, eConnect)
    writer.Close()

Catch ex As System.Exception
    Console.WriteLine(ex.ToString())
End Try
End Sub

```

## eConnectOut serialization example

To request data using eConnect, you must create an XML document that describes the data you want to retrieve. The <eConnectOut> XML node gives you the ability to query Microsoft Dynamics GP using eConnect's XML-based syntax.

The following Visual Basic .NET example uses the eConnectOut class from the Microsoft.Dynamics.GP.eConnect.Serialization namespace to request the complete customer document for a specified customer record.

As you review the example, note the following actions:

- The request begins with the creation of an eConnectOut object. The values in the object's elements describe the single customer document to retrieve.
- The value of the OUTPUTTYPE element instructs eConnect to return the customer master document.
- The values in the INDEX1FROM and INDEX1TO elements specify the customer ID.
- The values in the FORLIST and FORLOAD elements instruct eConnect to return the XML document directly to the caller.

- The example populates the eConnectOut property of the <RQeConnectOutType> transaction type object with the eConnectOut object.
- To complete the eConnect XML document, the RQeConnectOutType object populates the eConnect document's RQeConnectOutType property.
- The XMLSerializer object converts the XML document object to a string.
- The next step creates an eConnectMethods object and calls its eConnect\_Requester method. Notice how the string representation of the eConnect XML document is the method's third parameter. Refer to [Chapter 7, "eConnect Assembly"](#) for additional information about the eConnectMethods class and the eConnect\_Requester method.
- The result of the eConnect\_Requester method is written to a file named "Customer.xml".

The following Visual Basic .NET code performs the prescribed actions:

```
Imports System
Imports System.Xml
Imports System.Xml.Serialization
Imports System.IO
Imports System.EnterpriseServices
Imports System.Text
Imports Microsoft.Dynamics.GP.eConnect
Imports Microsoft.Dynamics.GP.eConnect.Serialization

Module Module1
    Sub Main()
        Dim serializer As New XmlSerializer(GetType(eConnectType))
        Dim eConnect As New eConnectType()
        Dim eConnectouttype As New RQeConnectOutType()
        Dim eConnectOut As New eConnectOut()
        Dim entrypoint As New eConnectMethods()
        Dim requesterDoc As String
        Dim sConnectionString As String

        Try
            'Populate the eConnectOut object to specify the data request
            With eConnectOut
                .DOCTYPE = "Customer"
                .OUTPUTTYPE = 1
                .INDEX1FROM = "AARONFIT0001"
                .INDEX1TO = "AARONFIT0001"
                .FORLIST = 1
            End With

            'Create an XML document for the request
            eConnectouttype.eConnectOut = eConnectOut
            ReDim Preserve eConnect.RQeConnectOutType(0)
            eConnect.RQeConnectOutType(0) = eConnectouttype

            'Serialize the eConnect object to a memory stream
            Dim memStream As New MemoryStream()
            serializer.Serialize(memStream, eConnect)
```

```

memStream.Position = 0

'Use the memory stream to load an Xml document
Dim xmlDoc As New XmlDocument()
xmlDoc.Load(memStream)
memStream.Close()

'Create a connection string to Microsoft Dynamics GP
sConnectionString = "data source=MyServer;" _
    & "initial catalog=TWO; integrated security=SSPI;" _
    & "persist security info=False; packet size=4096"

'Request the data from the server
requesterDoc = entrypoint.eConnect_Requester(sConnectionString, _
    EnumTypes.ConnectionStringType.SqlClient, xmlDoc.OuterXml)

'Write the customer XML to a file
Dim fs As New FileStream("Customer.xml", FileMode.Create)
Dim writer As New StreamWriter(fs, New UTF8Encoding)
writer.Write(requesterDoc)
writer.Close()

Catch exp As eConnectException
    Debug.Write(exp.ToString)
Catch ex As System.Exception
    Debug.Write(ex.Message & vbCrLf & ex.StackTrace)
End Try
End Sub
End Module

```

When the application runs, the `eConnect_Requester` method returns a string that is assigned to the `requesterDoc` variable. The `requesterDoc` string contains the following XML:

```

<root>
  <eConnect ACTION="0" Requester_DOCTYPE="Customer" DBNAME="TWO"
    TABLENAME="RM00101" DATE1="1900-01-01T00:00:00" CUSTNMBR="AARONFIT0001">
    <Customer>
      <CUSTNMBR>AARONFIT0001</CUSTNMBR>
      <ADDRESS1>One Microsoft Way</ADDRESS1>
      <ADDRESS2 />
      <ADDRESS3 />
      <ADRSCODE>PRIMARY</ADRSCODE>
      <CITY>Redmond</CITY>
      <CNTCPRSN>Bob Fitz</CNTCPRSN>
      <COUNTRY>USA</COUNTRY>
      <CPRCSTNM />
      <CURNCYID>Z-US$</CURNCYID>
      <CUSTCLAS>USA-ILMO-T1</CUSTCLAS>
      <CUSTDISC>0</CUSTDISC>
      <CUSTNAME>Aaron Fitz Electrical</CUSTNAME>
      <PHONE1>42555501010000</PHONE1>
      <PHONE2>00000000000000</PHONE2>
      <PHONE3 />
      <FAX>31255501010000</FAX>
      <PYMTRMID>Net 30</PYMTRMID>
    </Customer>
  </eConnect>
</root>

```

```

<SALSTERR>TERRITORY 1</SALSTERR>
<SHIPMTHD>LOCAL DELIVERY</SHIPMTHD>
<SLPRSNID>PAUL W.</SLPRSNID>
<STATE>WA</STATE>
<TAXSCHID>USASTCITY-6*</TAXSCHID>
<TXRGNNUM />
<UPSZONE />
<ZIP>98052-6399</ZIP>
<STMTNAME>Aaron Fitz Electrical</STMTNAME>
<SHRTNAME>Aaron Fitz Elec</SHRTNAME>
<PRBTADCD>PRIMARY</PRBTADCD>
<PRSTADCD>WAREHOUSE</PRSTADCD>
<STADDRCD>PRIMARY</STADDRCD>
<CHEKBKID>UPTOWN TRUST</CHEKBKID>
<CRLMTTYP>2</CRLMTTYP>
<CRLMTAMT>35000.00000</CRLMTAMT>
<CRLMTPER>0</CRLMTPER>
<CRLMTPAM>0.00000</CRLMTPAM>
<RATEPID />
<PRCLEVEL />
<MINPYTYP>0</MINPYTYP>
<MINPYDLR>0.00000</MINPYDLR>
<MINPYPCT>0</MINPYPCT>
<FNCHATYP>1</FNCHATYP>
<FNCHPCNT>150</FNCHPCNT>
<FINCHDLR>0.00000</FINCHDLR>
<MXWOFTYP>2</MXWOFTYP>
<MXWROFAM>25.00000</MXWROFAM>
<COMMENT1 />
<COMMENT2 />
<USERDEF1>Retail</USERDEF1>
<USERDEF2 />
<TAXEXMT1 />
<TAXEXMT2 />
<BALNCTYP>0</BALNCTYP>
<STMTCYCL>5</STMTCYCL>
<BANKNAME />
<BNKBRNCH />
<FRSTINDT>1900-01-01T00:00:00</FRSTINDT>
<INACTIVE>0</INACTIVE>
<HOLD>0</HOLD>
<CRCARDID />
<CCRDNUM />
<CCRDXPDT>1900-01-01T00:00:00</CCRDXPDT>
<KPDSTHST>1</KPDSTHST>
<KPCALHST>1</KPCALHST>
<KPERHIST>1</KPERHIST>
<KPTRXHST>1</KPTRXHST>
<CREATDDT>1970-01-01T00:00:00</CREATDDT>
<MODIFDT>2004-01-30T00:00:00</MODIFDT>
<Revalue_Customer>1</Revalue_Customer>
<Post_Results_To>0</Post_Results_To>
<FINCHID />
<GOVCRPID />
<GOVINDID />
<DISGRPER>0</DISGRPER>

```

```
<DUEGRPER>0</DUEGRPER>
<DOCFMTID />
<Send_Email_Statements>0</Send_Email_Statements>
<GPSFOINTEGRATIONID />
<INTEGRATIONSOURCE>0</INTEGRATIONSOURCE>
<INTEGRATIONID />
</Customer>
</eConnect>
</root>
```



## Chapter 9: Miscellaneous Routines Assembly

The Microsoft.Dynamics.GP.eConnect.MiscRoutines assembly provides additional classes you can use to get information you need to create XML documents. The following items are discussed:

- [\*Microsoft.Dynamics.GP.eConnect.MiscRoutines assembly\*](#)
- [\*GetNextDocNumbers class\*](#)
- [\*DocumentRollback class\*](#)
- [\*RollBackDocument class\*](#)
- [\*GetSopNumber class\*](#)
- [\*PricingMethods class\*](#)

### Microsoft.Dynamics.GP.eConnect.MiscRoutines assembly

The assembly contains the Microsoft.Dynamics.GP.eConnect.MiscRoutines namespace. This namespace includes three classes:

**GetNextDocNumbers** The GetNextDocNumbers class allow you to get next available number for several types of Microsoft Dynamics GP documents.

**GetSopNumber** The GetSopNumber class allows you to retrieve a SOP number for a sales document. This class also allows you to return a SOP number that was retrieved but not used.

**PricingMethods** The PricingMethods class allows you to create customer specific pricing.

### GetNextDocNumbers class

The GetNextDocNumbers class inherits from **System.EnterpriseService.ServicedComponent** class. This parent class enables GetNextDocNumbers to use the services supplied by the eConnect COM+ object. Refer to the .NET Framework documentation for information about the **System.EnterpriseService.ServicedComponent** class.

The GetNextDocNumbers class allows you to retrieve the next valid document number for several Microsoft Dynamics GP document types.

To use the GetNextDocNumbers class you must first instantiate an object. The following Visual Basic .NET example creates a GetNextDocNumbers object:

```
'Instantiate a GetNextDocNumbers object  
Dim NextDocNumberObject As New GetNextDocNumbers
```

The GetNextDocNumbers class has seven methods

### GetNextGLJournalEntryNumber method

This method will retrieve the next general ledger journal entry document number from Microsoft Dynamics GP.

The method has two parameters:

| Parameter          | Data type  | Description   |
|--------------------|--|---|
| IncrementDecrement | Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetNextDocNumbers.IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next GL journal entry number. Do not use the enumeration member Decrement in this version of eConnect. |
| ConnectionString   | string   | Specifies the SQL server and database.  |

This method returns a string that contains the requested general ledger journal entry document number.

## GetNextIVNumber method

This method will retrieve the next document number for the specified type of Microsoft Dynamics GP inventory document.

The method has three parameters:

| Parameter          | Data type  | Description  |
|--------------------|--|--|
| IncrementDecrement | Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetNextDocNumbers.IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next IV document number. Do not use the enumeration member Decrement in this version of eConnect.           |
| DocType            | Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetNextDocNumbers.IVDocType          | Use an IVDocType enumeration member to specify the type of IV document number to return. IVDocType includes the following members:<br>IVAdjustment<br>IVVariance<br>IVTransfer |
| ConnectionString   | string   | Specifies the SQL server and database.   |

This method returns a string that contains the inventory document number.

## GetNextPMPaymentNumber method

This method will retrieve the next payables management document number from Microsoft Dynamics GP.

The method has two parameters:

| Parameter          | Data type  | Description  |
|--------------------|--|--|
| IncrementDecrement | Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetNextDocNumbers.IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next PM payment document number. Do not use the enumeration member Decrement in this version of eConnect. |
| ConnectionString   | string   | Specifies the SQL server and database.   |

This method returns a string that contains the requested payables management payment document number.

## GetNextPONumber method

This method will retrieve the next purchase order document number from Microsoft Dynamics GP.

The method has two parameters:

| Parameter          | Data type  | Description  |
|--------------------|--|--|
| IncrementDecrement | Microsoft.Dynamics.GP.<br>eConnect.MiscRoutines.<br>GetNextDocNumbers.<br>IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next PO document number. Do not use the enumeration member Decrement in this version of eConnect. |
| ConnectionString   | string   | Specifies the SQL server and database.   |

This method returns a string that contains the requested purchase order document number

## GetNextPOPReceiptNumber method

This method will retrieve the next purchase order processing receipt document number from Microsoft Dynamics GP.

The method has two parameters:

| Parameter          | Data type  | Description   |
|--------------------|--|---|
| IncrementDecrement | Microsoft.Dynamics.GP.<br>eConnect.MiscRoutines.<br>GetNextDocNumbers.<br>IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next POP receipt document number. Do not use the enumeration member Decrement in this version of eConnect. |
| ConnectionString   | string   | Specifies the SQL server and database.  |

This method returns a string that contains the requested purchase order processing receipt document number.

## GetNextRMNumber method

This method will retrieve the next document number for the specified type of Microsoft Dynamics GP receivables management document.

The method has three parameters:

| Parameter          | Data type  | Description   |
|--------------------|--|---|
| IncrementDecrement | Microsoft.Dynamics.GP.<br>eConnect.MiscRoutines.<br>GetNextDocNumbers.<br>IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next RM document number. Do not use the enumeration member Decrement in this version of eConnect.  |
| DocType            | Microsoft.Dynamics.GP.<br>eConnect.MiscRoutines.<br>GetNextDocNumbers.<br>RMPaymentType      | Use an RMPaymentType enumeration member to specify the type of RM document number to return. RMPaymentType includes the following members:<br>RMInvoices<br>RMScheduledPayments<br>RMDebitMemos<br>RMFinanceCharges<br>RMServiceRepairs<br>RMWarranty<br>RMCreditMemo<br>RMReturn<br>RMPayments |
| ConnectionString   | string   | Specifies the SQL server and database.  |

This method returns a string that contains the requested receivables management document number.

## GetNextSOPNumber method

This method will retrieve the next document number for the specified type of Microsoft Dynamics GP sales order processing document.

The method has four parameters:

| Parameter          | Data type  | Description  |
|--------------------|--|--|
| IncrementDecrement | Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetNextDocNumbers.IncrementDecrement | Use the IncrementDecrement enumeration member Increment to get the next SOP document number. Use Decrement to return the DocIDKey parameter as an unused document number.                          |
| DocIDKey           | string   | Specifies the document ID  |
| DocType            | Microsoft.Dynamics.GP.eConnect.MiscRoutines.GetNextDocNumbers.SopType            | Use a SopType enumeration member to specify the type of SOP document number to return. SopType includes the following members:<br>SOPQuote<br>SOPOrder<br>SOPInvoice<br>SOPReturn<br>SOPBackOrder. |
| ConnectionString   | string   | Specifies the SQL server and database.   |

This method returns a string that contains the requested SOP document number.

## RollBackDocumentList method

This method allows you to restore a Microsoft Dynamics GP document number that was retrieved using one of the previous methods. The RollBackDocumentList method requires an arraylist of **RollBackDocument** objects that specify the document numbers. The arraylist allows you to restore a single document number or several document numbers.

The method has two parameters:

| Parameter            | Data type | Description   |
|----------------------|-----------|---|
| RollBackDocumentList | ArrayList | A collection of one or more RollBackDocument objects. Each RollBackDocument object specifies a Microsoft Dynamics GP document number. |
| BackOfficeConnString | string    | Specifies the SQL server and database.  |

The method does not return a value.

## DocumentRollback class

The **DocumentRollback** class allows you to create a collection that stores Microsoft Dynamics GP document numbers.

Use a **DocumentRollback** object to store the document numbers you retrieve for an eConnect transaction. If an error occurs during your eConnect transaction, the **DocumentRollback** class allows you to produce an arraylist that specifies all the document numbers in your transaction. The **RollBackDocumentList** method of the

**GetNextDocNumbers** class uses this arraylist to restore the specified Microsoft Dynamics GP document numbers.

The DocumentRollback class has four methods:

### **Add(transactionType, documentNumber) method**

This method adds a Microsoft Dynamics GP document number to the **DocumentRollback** collection. You must specify the transaction type associated with each document number.

The method has two parameters:

| Parameter       | Data type   | Description  |
|-----------------|---|--|
| transactionType | Microsoft.Dynamics.GP.eConnect.MiscRoutines.TransactionType | Use a TransactionType enumeration member to specify the transaction type associated with the document number. TransactionType includes the following members:<br>GL<br>IVTrans<br>IVTransfer<br>PM<br>POP<br>POPReceipt<br>RM<br>SOP |
| documentNumber  | string  | A string that specifies a Microsoft Dynamics GP document number.   |

The method does not return a value.

### **Add(transactionType, documentType, documentNumber) method**

This method adds a Microsoft Dynamics GP document number to the **DocumentRollback** collection. You must specify the transaction type and document type associated with each document number.

The method has three parameters:

| Parameter       | Data type   | Description  |
|-----------------|---|--|
| transactionType | Microsoft.Dynamics.GP.eConnect.MiscRoutines.TransactionType | Use a TransactionType enumeration member to specify the transaction type associated with the document number. TransactionType includes the following members:<br>GL<br>IVTrans<br>IVTransfer<br>PM<br>POP<br>POPReceipt<br>RM<br>SOP |
| documentType    | Int16   | An integer that specifies the document type associated with the Microsoft Dynamics GP document number.   |
| documentNumber  | string  | A string that specifies a Microsoft Dynamics GP document number.   |

The method does not return a value.

### **Add(transactionType, documentType, documentNumber, documentID) method**

This method adds a Microsoft Dynamics GP document number to the **DocumentRollback** collection. You must specify the transaction type, document type, document number, and document ID.

The method has three parameters:

| Parameter       | Data type   | Description  |
|-----------------|---|--|
| transactionType | Microsoft.Dynamics.GP.eConnect.MiscRoutines.TransactionType | Use a TransactionType enumeration member to specify the transaction type associated with the document number. TransactionType includes the following members:<br>GL<br>IVTrans<br>IVTransfer<br>PM<br>POP<br>POPReceipt<br>RM<br>SOP |
| documentType    | Int16   | An integer that specifies the document type associated with the Microsoft Dynamics GP document number.   |
| documentNumber  | string  | A string that specifies a Microsoft Dynamics GP document number.   |
| documentID      | string  | A string that specifies the document ID associated with the document number.   |

The method does not return a value.

### **CollectionContainsDocuments method**

This method returns a boolean value that specifies whether the collection of document numbers is empty.

This method has no parameters.

A return value of **true** indicates one or more document numbers have been added to the collection. A return value of **false** indicates no document numbers have been added to the collection.

### **RollBackDocuments method**

This method returns an arraylist of **RollBackDocument** objects.

This method has no parameters.

**RollBackDocuments** returns an arraylist that contains the collection of document numbers that were added to this **DocumentRollback** object. To restore these Microsoft Dynamics GP document numbers, use this arraylist as a parameter for the **RollBackDocumentList** method of the **GetNextDocNumbers** class.

## RollBackDocument class

The **RollBackDocument** class allows you to create an object that represents a single Microsoft Dynamics GP document number. Use a **RollBackDocument** object whenever your code needs to retain Microsoft Dynamics GP document number information.

The parameters of the **RollBackDocument** constructors allow you to create a **RollBackDocument** object that uniquely identifies a Microsoft Dynamics GP document number.

The RollBackDocument class has two methods:

### **RollBackDocument(transactionType, documentType, documentNumber) method**

This **RollBackDocument** constructor instantiates a **RollBackDocument** object. The constructor uses three parameters to identify a specific Microsoft Dynamics GP document number.

The **RollBackDocument** constructor requires the following parameters:

| Parameter       | Data type   | Description  |
|-----------------|---|--|
| transactionType | Microsoft.Dynamics.GP.eConnect.MiscRoutines.TransactionType | Use a TransactionType enumeration member to specify the transaction type associated with the document number. TransactionType includes the following members:<br>GL<br>IVTrans<br>IVTransfer<br>PM<br>POP<br>POPReceipt<br>RM<br>SOP |
| documentType    | Int16   | Specifies the document type associated with the document number. This parameter accepts a null value.  |
| documentNumber  | string  | Specifies a Microsoft Dynamics GP document number.   |

This method does not have a return value.

The **RollBackDocument** class provides three properties. These properties allow you to view the data members of an existing **RollBackDocument** object. The **RollBackDocument** class includes the following properties:

### **RollBackDocument(transactionType, documentType, documentID, documentNumber) method**

This **RollBackDocument** constructor instantiates a **RollBackDocument** object. The constructor uses four parameters to identify a specific Microsoft Dynamics GP document number.

The **RollBackDocument** constructor requires the following parameters:

| Parameter       | Data type   | Description  |
|-----------------|---|--|
| transactionType | Microsoft.Dynamics.GP.<br>eConnect.MiscRoutines.<br>TransactionType | Use a TransactionType enumeration member to specify the transaction type associated with the document number. TransactionType includes the following members:<br>GL<br>IVTrans<br>IVTransfer<br>PM<br>POP<br>POPReceipt<br>RM<br>SOP |
| documentType    | Int16   | Specifies the document type associated with the document number. This parameter accepts a null value.  |
| documentID      | string  | A string that specifies the document ID associated with the document number.   |
| documentNumber  | string  | Specifies a Microsoft Dynamics GP document number.   |

This method does not have a return value.

The **RollBackDocument** class provides three properties. These properties allow you to view the data members of an existing **RollBackDocument** object. The **RollBackDocument** class includes the following properties:

### DocumentTransactionType property

Returns the **TransactionType** enumeration value of the current **RollBackDocument** object.

### DocumentNumber property

Returns a string that specifies the Microsoft Dynamics GP document number value of the current **RollBackDocument** object.

### DocumentType property

Returns a 16-bit signed integer value that specifies the document type associated with the current **RollBackDocument** object. This property may contain a null value.

## GetSopNumber class

The **GetSopNumber** class inherits from **System.EnterpriseService.ServicedComponent** class. This parent class enables **GetSopNumber** to use the services supplied by the eConnect COM+ object. Refer to the .NET Framework documentation for information about the **System.EnterpriseService.ServicedComponent** class.

The **GetSopNumber** class allows you to retrieve the next valid SOP number from Microsoft Dynamics GP. You can use the SOP number to create new sales documents. The class allows you to return numbers that you do not use.

To use the **GetSopNumber** class you must first instantiate an object. The following Visual Basic .NET example creates a **GetSopNumber** object:



```
'Instantiate a GetSopNumber object
Dim SopNumberObject As New GetSopNumber
```

The GetSopNumber class contains two methods:

## GetNextSopNumber method

This method will retrieve the next sales document number from Microsoft Dynamics GP.

The method has three parameters:

| Parameter            | Data type | Description   |
|----------------------|-----------|---|
| SOPTypeKey           | integer   | Specifies the type of the document:<br>1 = Quote<br>2 = Order<br>3 = Invoice<br>4 = Return<br>5 = Back order<br>6 = Fulfillment order |
| DocIDKey             | string    | Specifies the document ID   |
| BackOfficeConnString | string    | Specifies the SQL server and database   |

This method returns a string that contains the requested SOP number.

The following Visual Basic .NET example uses the GetNextSopNumber method to retrieve a sales invoice number from Microsoft Dynamics GP:

```
Dim ConnectionString As String
Dim SopNumber As String

'Set the connection string
'This connection string uses integrated security to connect to the
'TWO database on the local computer
ConnectionString = "Data Source=127.0.0.1;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"

'Instantiate a GetSopNumber object
Dim SopNumberObject As New GetSopNumber

'Get the next SOP number
SopNumber = SopNumberObject.GetNextSopNumber(3, "STDINV", ConnectionString)
```

## RollBackSopNumber method

Use this method to return unused sales order numbers. This makes the numbers available for use by Microsoft Dynamics GP.

The method has four parameters:

| Parameter    | Data type | Description   |
|--------------|-----------|---|
| SOPNumberKey | string    | Specifies the number to return to Microsoft Dynamics GP |

| Parameter            | Data type | Description   |
|----------------------|-----------|---|
| SOPTypeKey           | string    | Specifies the type of the document:<br>1 = Quote<br>2 = Order<br>3 = Invoice<br>4 = Return<br>5 = Back order<br>6 = Fulfillment order |
| DocIDKey             | string    | Specifies the document ID   |
| BackOfficeConnString | string    | Specifies the SQL server and database   |

This method returns a boolean value that indicates whether the rollback request was successfully completed. A value of **True** indicates the rollback was successful.

The following segment of Visual Basic .NET code uses the RollBackSopNumber method to make an unused SOP number available in Microsoft Dynamics GP:

```
Dim ConnectionString As String
Dim SopNumber As String

'Set the connection string
'This connection string uses integrated security to connect to the
'TWO database on the local computer
ConnectionString = "Data Source=127.0.0.1;Integrated Security=SSPI;" _
    & "Persist Security Info=False;InitRoial Catalog=TWO;"

'Instantiate a GetSopNumber object
Dim SopNumberObject As New GetSopNumber

'Get the next sales invoice number
SopNumber = SopNumberObject.GetNextSopNumber(3, "STDINV", ConnectionString)

'Return the sales invoice number to Microsoft Dynamics GP
SopNumber = SopNumberObject.RollBackSopNumber(SopNumber, 3, "STDINV", _
    ConnectionString)
```

## PricingMethods class

The PricingMethods class inherits from **System.EnterpriseService.ServicedComponent** class. This parent class enables PricingMethods to use the services supplied by the eConnect COM+ object. Refer to the .NET Framework documentation for information about the **System.EnterpriseService.ServicedComponent** class.

The PricingMethods class retrieves customer specific pricing information.

To use the PricingMethods class you must first instantiate an object. The following Visual Basic .NET example creates a PricingMethods object:

```
'Instantiate a PricingMethods object
Dim PricingMethodsObject As New PricingMethods
```

The PricingMethods class contains the following methods:

## GetCustomerSpecificItemPriceForSellingUOFM method

This method gets customer-specific pricing for an item's default selling unit of measure.

The method has seven parameters:

| Parameter            | Data type | Description                                |
|----------------------|-----------|--|
| ItemList             | string    | An array of strings that specify the items |
| CustomerKey          | string    | Specifies the customer                     |
| CurrencyKey          | string    | Specifies the currency to use              |
| QuantityKey          | string    | Specifies the quantity to use              |
| UOFMKey              | string    | Specifies the unit of measure to use       |
| PRCLEVELKey          | string    | Specifies the price level to use           |
| BackOfficeConnString | string    | Specifies the SQL server and database      |

This method returns a string that specifies the price for the specified quantity breaks.

## GetItemPriceForAllPriceLevelsAndAllUnitsOf Measure method

This method calculates a price level that complies with Microsoft Dynamics GP rules. It uses the Currency key to calculate the price of an item for each unit of measure defined for that item.

The method has four parameters:

| Parameter            | Data type | Description                                |
|----------------------|-----------|--|
| ItemList             | string    | An array of strings that specify the items |
| CustomerKey          | string    | Specifies the customer                     |
| CurrencyKey          | string    | Specifies the currency to use              |
| BackOfficeConnString | string    | Specifies the SQL server and database      |

This method returns a string.

## GetItemPricePerPriceLevelAndAllUnitsOf Measure method

This method will return item prices per price level with all units of measure prices. Also the XML document returned is ordered by the base unit of measure. The first item price node is the price for the base unit of measure.

The method has four parameters:

| Parameter            | Data type | Description                                |
|----------------------|-----------|--|
| ItemList             | string    | An array of strings that specify the items |
| CustomerKey          | string    | Specifies the customer                     |
| CurrencyKey          | string    | Specifies the currency to use              |
| BackOfficeConnString | string    | Specifies the SQL server and database      |

This method returns a string that represents the return item prices per price level with all units of measure prices.





## Part 4: MSMQ Development

This portion of the documentation discusses how to use Microsoft Message Queuing (MSMQ) with eConnect's Incoming and Outgoing services. The services allow you to submit and retrieve XML documents. The following information is discussed:

- [Chapter 10, "MSMQ,"](#) explains how MSMQ and the Incoming and Outgoing Services work together.
- [Chapter 11, "Incoming Service,"](#) discusses how to use the Incoming Service to integrate your application's data into Microsoft Dynamics GP.
- [Chapter 12, "Outgoing Service,"](#) discusses how to use the Outgoing Service to retrieve XML documents that represent transactions or documents in Microsoft Dynamics GP.

## Chapter 10: MSMQ

eConnect provides an interface built upon the Microsoft Message Queue (MSMQ) infrastructure. You can use the interface to transport XML documents between your application and Microsoft Dynamics GP. This portion of the document provides an introduction to using the MSMQ interface and discusses the following:

- [\*Microsoft Message Queue overview\*](#)
- [\*Windows Services used with MSMQ\*](#)
- [\*eConnect MSMQ Control\*](#)

### Microsoft Message Queue overview

Message queuing is a message infrastructure and development platform for creating distributed, loosely-coupled messaging applications. Message queuing provides guaranteed message delivery, efficient routing, security, transaction support, and priority-based messaging. The eConnect MSMQ interface leverages the abilities of MSMQ to handle messages between applications.

A queue is a logical container that MSMQ uses to store messages. Applications can send messages to queues where they are stored until a receiving application retrieves the message from the queue.

Applications typically create queues, locate existing queues, send messages to queues, and read messages in queues. To perform an operation on a queue, an application must first reference the queue.

### Windows Services used with MSMQ

The eConnect installation includes two Windows Services that are used with the MSMQ interface.

- The Incoming Service periodically monitors a specified queue. When it finds messages in the queue, it takes the message, validates the XML document the message contains, and uses the eConnect business object to perform a Microsoft Dynamics GP operation.
- The Outgoing Service publishes XML documents to a specified queue. You can configure the Microsoft Dynamics GP documents and operations that are published. Messages published to the queue can be retrieved by other applications. The application can retrieve the XML document from the message and perform actions based upon the data the document contains.

Refer to the eConnect Installation and Administration Guide for information about installing and configuring the Incoming Service and the Outgoing Service.

### eConnect MSMQ Control

The eConnect installation provides a utility you use to monitor queues and messages. The eConnect MSMQ Control allows you to open a queue, see the list of messages in the queue, and view the contents of individual messages.

Use the utility during development to ensure messages are getting delivered and that they contain the expected XML data. You can also use the control to debug messages. You can view, edit, and resend messages. For additional information

about using the eConnect MSMQ Control, refer to the Utilities chapter in the eConnect Installation and Administration Guide.



# Chapter 11: Incoming Service

The Incoming Service allows you to create applications that place eConnect XML documents into an MSMQ message and store the message in a queue. Once a message is placed in the queue, the Incoming Service is able to retrieve the message from the queue. The Service creates an XML document from the body of the message. The Incoming Service takes the data from the XML document and uses it with the business objects to perform the specified operation.

The Income Service can validate messages to ensure the XML document complies with the schema, and then call the eConnect Business Objects to perform the operation contained in the XML document.

To use the Incoming Service refer to the following sections:

- [Creating an eConnect XML document](#)
- [Creating an MSMQ message](#)
- [Incoming Service example](#)

## Creating an eConnect XML document

To use the Incoming Service, your application must be able to create eConnect XML documents. eConnect provides a .NET assembly named eConnect Serialization that simplifies creating these documents. To use eConnect serialization, you must add a reference to your project to the Microsoft.Dynamics.GP.eConnect.Serialization assembly.

Refer to [Chapter 8, "Serialization Assembly"](#) for additional information about the eConnect serialization classes.

## Creating an MSMQ message

To use MSMQ, add a reference to the System.Messaging assembly of the .NET framework. To send your XML document to the queue, complete the following steps:

### 1. Specify the MSMQ destination.

Create a message queue object and specify the eConnect incoming queue. The Incoming Service uses the private queue named **eConnect\_incoming9**. The following Visual Basic .NET code creates the object and specifies the **eConnect\_incoming9** queue on the local server:

```
Dim MyQueue As New MessageQueue(".\private$\econnect_incoming9")
```

### 2. Populate an MSMQ message.

Create a message object. The following example creates a message object, and then populates the label and message body properties:

```
Dim MyMessage As New Message
MyMessage.Label = "eConnect Test with ActiveXMessageFormatter"
MyMessage.Body = sCustomerXmlDoc
```

Notice how the string representation of the XML document is used to populate the message body.

**3. Specify the message format.**

Create a message formatter object to specify how to serialize and deserialize the message body. The following sample uses an `ActiveXMessageFormatter`:

```
Dim MyFormatter As New ActiveXMessageFormatter

MyMessage.Formatter = MyFormatter
MyFormatter.Write(MyMessage, sCustomerXmlDoc)
```

**4. Use a queue transaction.**

The `eConnect_incoming9` is a transactional queue. A transactional queue requires a `MessageQueueTransaction` object. The following Visual Basic .NET code shows how to create and use `MessageQueueTransaction` to send a message to the `eConnect_incoming9` queue:

```
Dim MyQueueTrans As New MessageQueueTransaction
MyQueueTrans.Begin()
    MyQueue.Send(MyMessage, MyQueueTrans)
MyQueueTrans.Commit()
```

**5. Close the queue connection.**

Once the message is sent, close the queue object. The following sample closes the queue and frees its resources.

```
MyQueue.Close()
```

Refer to the `System.Messaging` documentation of the .NET Framework for additional information about message queue classes and enumerations.

## Incoming Service example

This example creates a customer XML document and sends it to the queue the Incoming Service monitors. This example requires references to the `System.Messaging` and `Microsoft.Dynamics.GP.eConnect.Serialization` assemblies.

Notice how the example performs the following steps:

- Creates an `eConnect` serialization object for a customer and populates its elements with data.
- Creates an `eConnect` XML document that describes a new customer to add to Microsoft Dynamics GP.
- Serializes the `eConnect` XML document object to create a string representation of the XML.
- Specifies the MSMQ queue that will receive the message.
- Places the string representation of the XML document in an MSMQ message object.
- Sends the message to the specified queue.

```
Private Sub CreateCustomerSendtoMSMQ()
    Try
        'XML document components
        Dim eConnect As New eConnectType
```

```

Dim CustomerType As New SMCustomerMasterType
Dim MyCustomer As New taUpdateCreateCustomerRcd

'Serialization objects
Dim serializer As New XmlSerializer(GetType(eConnectType))
Dim MemStream As New MemoryStream
Dim sCustomerXmlDoc As String

'Populate the MyCustomer object with data.
With MyCustomer
    .CUSTNMBR = "JOEH0001"
    .CUSTNAME = "Joe Healy"
    .ADRSCODE = "PRIMARY"
    .ADDRESS1 = "789 First Ave N"
    .CITY = "Rollag"
    .STATE = "MN"
    .ZIPCODE = "23589"
End With

'Build the XML document
CustomerType.taUpdateCreateCustomerRcd = MyCustomer
ReDim eConnect.SMCustomerMasterType(0)
eConnect.SMCustomerMasterType(0) = CustomerType

'Serialize the XML document
serializer.Serialize(MemStream, eConnect)
MemStream.Position = 0

'Use the Memory Stream to create an xml string
Dim xmlreader As New XmlTextReader(MemStream)
While xmlreader.Read
    sCustomerXmlDoc = sCustomerXmlDoc & xmlreader.ReadOuterXml & vbCr
End While

'Create the MSMQ queue and message objects
Dim MyQueue As New MessageQueue(".\private$\econnect_incoming9")
Dim MyMessage As New Message
Dim MyQueTrans As New MessageQueueTransaction
Dim MyFormatter As New ActiveXMessageFormatter

'Build the MSMQ message and send it to the queue
MyMessage.Label = "eConnect Test with ActiveXMessageFormatter"
MyMessage.Body = sCustomerXmlDoc
MyMessage.Formatter = MyFormatter
MyFormatter.Write(MyMessage, sCustomerXmlDoc)
MyQueTrans.Begin()
MyQueue.Send(MyMessage, MyQueTrans)
MyQueTrans.Commit()
MyQueue.Close()

Catch ex As System.Exception
    Debug.Write(ex.Message & vbCrLf & ex.StackTrace)
    SerializedXmlDoc.Text = ex.Message & vbCrLf & ex.StackTrace
End Try
End Sub

```



## Chapter 12: Outgoing Service

You use the Outgoing Service to publish XML documents that represent specified documents and operations in Microsoft Dynamics GP. The Outgoing Service periodically queries the eConnect\_Out tables in Microsoft Dynamics GP. It uses entries in that table to generate XML documents. The documents are placed in an XML message and sent to the default queue `./private$/econnect_outgoing9`.

Refer to the eConnect Installation and Administration Guide for information about configuring the Outgoing Service.

The following topics are discussed:

- [\*Publishing the eConnect XML documents\*](#)
- [\*Retrieving the MSMQ message\*](#)
- [\*Outgoing Service Example\*](#)

### Publishing the eConnect XML documents

To begin using the Outgoing Service, you specify the Microsoft Dynamics GP documents and operations to publish. eConnect supplies a utility named the Requester Enabler/Disabler to manage this. The utility creates SQL triggers in the Microsoft Dynamics GP database that update the eConnect\_Out table.

For information about configuring the Requester Enable/Disabler utility, refer to the Utilities chapter of the eConnect Installation and Administration Guide.

The Outgoing Service queries the eConnect\_Out tables to identify the documents to publish. The Outgoing Service creates eConnect XML documents that represent the Microsoft Dynamics GP documents to publish. The service encloses the eConnect XML document in an MSMQ message, and routes the message to the specified queue.

### Retrieving the MSMQ message

To use the Outgoing Service, your application needs to retrieve the messages from the queue. The default queue the Outgoing Services uses is `./private$/econnect_outgoing9`. To develop applications that retrieve messages, add a reference to the System.Messaging assembly of the .NET framework. The basic procedure to retrieve a message from the queue is as follows:

#### 1. Create a message queue object.

Instantiate a MessageQueue object. Use the path to the local outgoing queue `./private$/econnect_outgoing9`. Populate the queue object's Formatter property to allow the message to be deserialized. The following Visual Basic .NET example demonstrates these steps:

```
Dim myQueue As New MessageQueue("private$/econnect_outgoing9")
myQueue.Formatter = New ActiveXMessageFormatter
```

#### 2. Create a transaction object.

The `econnect_outgoing9` queue is a transactional queue. You must include a queue transaction object with your request. The following example creates the transaction object:

```
Dim myTransaction As New MessageQueueTransaction
```

### 3. Create a message object.

Instantiate an object that will receive the message retrieved from the specified queue:

```
Dim myMessage As New Message
```

### 4. Retrieve a message.

Use the object you created to retrieve a message from the queue. This example retrieves the first available message from the queue:

```
myTransaction.Begin()
myMessage = myQueue.Receive(myTransaction)
myTransaction.Commit()
```

### 5. Get the XML data from the message.

The body of the message will contain a string. The string represents the XML document that describes the Microsoft Dynamics GP operation that triggered the message. The following example retrieves the string from an MSMQ message:

```
Dim myDocument As [String] = CType(myMessage.Body, [String])
```

## Outgoing Service Example

This example retrieves an MSMQ message from the Outgoing Service queue. This example requires references to the System.Messaging and System.XML assemblies.

Notice how the example performs the following steps:

- Creates a MessageQueue object to access the Outgoing Service's message queue.
- Creates the MSMQ Formatter, MessageQueueTransaction, and Message objects.
- Retrieves the message from the queue.
- Retrieves the string from the Message object
- Loads the string into an XML document object and uses it to display the XML in a textbox. To allow access to specific XML elements and values, the example parses the string into XML. Refer to the .NET Framework documentation for information about creating XML from a string.

```
Private Sub GetMessage()
    'Create queue object to retrieve messages from the default outgoing queue
    Dim MyQueue As New MessageQueue(".\private$\econnect_outgoing9")

    'Create an MSMQ formatter and transaction objects
    MyQueue.Formatter = New ActiveXMessageFormatter
    Dim MyTransaction As New MessageQueueTransaction

    'Create a message object
    Dim MyMessage As Message
```

```

Try
    'Retrieve a message from the queue
    'This example assumes there is always a message waiting in the queue
    MyTransaction.Begin()
    MyMessage = MyQueue.Receive(MyTransaction)
    MyTransaction.Commit()

    'Retrieve the string from the message
    Dim MyDocument As [String] = CType(MyMessage.Body, [String])

    'Load the string into an XML document object
    Dim MyXml As New XmlDocument
    MyXml.LoadXml(MyDocument)

    'Display the XML from the queue message
    MessageText.Text = MyXml.InnerXml

Catch err As SystemException
    ErrorText.Text = err.InnerException.ToString()
End Try
End Sub

```







# Part 5: Business Logic

This portion of the documentation explains how to work with the eConnect business objects. The discussion includes information about using and extending the business rules contained in the business objects. The following information is discussed:

- [Chapter 13, “Business Logic Overview,”](#) provides an overview of eConnect’s business logic and how it can be used or extended by your application.
- [Chapter 14, “Custom XML Nodes,”](#) discusses how you add custom XML nodes to an eConnect XML document.
- [Chapter 15, “Business Logic Extensions,”](#) discusses how to use the Pre and Post stored procedures to modify eConnect’s business logic.

## Chapter 13: Business Logic Overview

This portion of the documentation describes options to use and extend eConnect's business logic. The following topics are discussed:

- [Business logic](#)
- [Extending business logic](#)
- [Calling the business objects](#)

### Business logic

Business logic is the collection of rules that constrain and guide the handling of business data. eConnect encapsulates its business logic in business objects. The business objects recreate Microsoft Dynamics GP's business logic for the documents and operations that eConnect supports.

The eConnect business objects implement business logic using SQL stored procedures. Any eConnect action that queries, creates, updates, or deletes data from Microsoft Dynamics GP uses one or more stored procedures. The eConnect install encrypts these stored procedures so you cannot edit the SQL instructions they contain.

While you cannot directly modify eConnect's core stored procedures, eConnect's business logic can be modified to respond to unique business problems. To adjust its business logic to a unique business problem, eConnect allows you to customize its XML documents and add custom SQL code that supplements the eConnect stored procedures.

### Extending business logic

When you develop an application that uses eConnect, you may encounter business situations that do not conform to eConnect's existing business logic. To resolve these situations, eConnect allows you to refine its existing business logic. Use the following to supplement eConnect's business logic:

**Add XML nodes to an existing schema** eConnect allows you to add custom XML nodes to its document schema. When you add an XML node to a document schema, you must also add a custom SQL stored procedure that processes your XML node's data. For more information about adding XML nodes, see [Chapter 14, "Custom XML Nodes."](#)

**Extend the business logic** Each eConnect SQL stored procedure provides a named Pre and Post procedure. To modify eConnect's business logic, add SQL code to the Pre and Post procedures that meet your unique business requirement. For more information about extending eConnect's business logic, see [Chapter 15, "Business Logic Extensions."](#)

### Calling the business objects

You may add eConnect business logic to an application by directly calling an eConnect SQL stored procedure.

eConnect encapsulates its business logic in a collection of SQL stored procedures. When you use Microsoft Dynamics GP Utilities to create a new company, GP Utilities automatically install the eConnect SQL stored procedures on your

Microsoft Dynamics GP SQL server. Since the stored procedures are available on the SQL server, you can use them to add eConnect business logic to your application.



*You should avoid direct calls to the stored procedures. To add eConnect business logic to an application, use the eConnect application programming interface (API) that supports your application's development environment.*

Refer to the SQL Server help documentation for information about calling a SQL stored procedure from an application.

If you encounter a situation that requires a direct call to an eConnect stored procedure, your application must address the following:

- Create a connection to the database server.
- Implement security restrictions to prevent unauthorized use of your database connection.
- Implement transaction management to commit or rollback changes.
- Identify and handle error conditions.
- Update your application whenever changes are made to the parameters for the stored procedure.

If you call an eConnect SQL stored procedure, you must always assess whether the procedure succeeded. The eConnect stored procedures use the `ErrorState` element to indicate whether the procedure encountered an error. If the value of `ErrorState` is 0, the procedure was successful. If the `ErrorState` value is anything other than 0, an error occurred and the transaction must be rolled back.



*You must check the value of the `ErrorState` element after each call to an eConnect stored procedure. All eConnect stored procedures reset the `ErrorState` element to zero when they start.*

# Chapter 14: Custom XML Nodes

This portion of the documentation discusses how you add custom XML nodes to eConnect XML documents. You use custom XML nodes to allow eConnect to process new types of data. The following topics are discussed:

- [\*Adding an XML node\*](#)
- [\*Creating a SQL stored procedure\*](#)

## Adding an XML node

eConnect allows you to add XML nodes to its document schema. Custom XML nodes enable you to use custom data elements within an eConnect XML document. Use custom data elements to provide additional data or to trigger custom business logic.

When eConnect processes an XML document, it maps the name of each XML node to a SQL stored procedure. The following XML creates a node named <eConnectCustomProcedure>:

```
<eConnectCustomProcedure>
  <CUSTNMBR>CONTOSOL0002</CUSTNMBR>
</eConnectCustomProcedure>
```

When eConnect processes a document containing <eConnectCustomProcedure>, it looks for a SQL stored procedure with the same name, in this case eConnectCustomProcedure. You must create the eConnectCustomProcedure stored procedure to handle your custom XML node.

To create a custom XML node you must:

- Define the data elements of your custom XML node.
- Provide each element with a unique name.
- Add the XML node to an existing eConnect transaction type.

Use the eConnect XML document's <eConnectProcessInfo> node to control the stored procedures' order of execution. To execute the eConnect core stored procedures prior to any custom stored procedures, set the <eConnectProcsRunFirst> element to TRUE. To execute custom stored procedures prior to the eConnect core stored procedures, set <eConnectProcsRunFirst> to FALSE.

The following XML document example adds the <eConnectCustomProcedure> XML node to a customer eConnect XML document. Notice how the custom node is added within the <RMCustomerMasterType> transaction type.

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <RMCustomerMasterType>
    <eConnectProcessInfo>
      <eConnectProcsRunFirst>TRUE</eConnectProcsRunFirst>
    </eConnectProcessInfo>
    <eConnectCustomProcedure>
      <CUSTNMBR>CONTOSOL0002</CUSTNMBR>
    </eConnectCustomProcedure>
    <taUpdateCreateCustomerRcd>
```

```

<CUSTNMBR>CONTOSOL0002</CUSTNMBR>
<CUSTNAME>Contoso, Ltd.</CUSTNAME>
<TAXSCHID>USALLEXMPT-0</TAXSCHID>
<SHIPMTHD>PICKUP</SHIPMTHD>
<ADDRESS1>321 Main S </ADDRESS1>
<CITY>Valley City</CITY>
<STATE>ND</STATE>
<ZIPCODE>56789</ZIPCODE>
<COUNTRY>USA</COUNTRY>
<PHNUMBR1>13215550100</PHNUMBR1>
<PHNUMBR2>13215550110</PHNUMBR2>
<FAX>13215550120</FAX>
<SALSTERR>TERRITORY 6 </SALSTERR>
<SLPRSNID>SEAN C .</SLPRSNID>
<SLPRSNFN>Sean</SLPRSNFN>
<SPRSNSLN>Chai</SPRSNSLN>
<UPSZONE>red</UPSZONE>
<CNTCPRSN>Joe Healy</CNTCPRSN>
<CHEKBKID>PAYROLL</CHEKBKID>
<PYMTRMID>Net 30 </PYMTRMID>
<COMMENT1>comment1</COMMENT1>
<COMMENT2>comment2</COMMENT2>
<USERDEF1>Retail</USERDEF1>
<PRBTADCD>PRIMARY</PRBTADCD>
<PRSTADCD>PRIMARY</PRSTADCD>
<ADRSCODE>PRIMARY</ADRSCODE>
<STADDRCD>PRIMARY</STADDRCD>
<CRCARDID>Gold Credit </CRCARDID>
<STMTNAME>Contoso, Ltd.</STMTNAME>
<SHRTNAME>Contoso, Ltd.</SHRTNAME>
<Revalue_Customer>1</Revalue_Customer>
<Post_Results_To>0</Post_Results_To>
<CRLMTAMT>90000.00</CRLMTAMT>
</taUpdateCreateCustomerRcd>
</RMCustomerMasterType>
</eConnect>

```

## Creating a SQL stored procedure

When you add a custom XML node to an eConnect XML documents, you must supply a SQL stored procedure to process that node. The previous section added the <eConnectCustomProcedure> node to an XML document. To process XML documents that contain this node, create a stored procedure with a name that exactly matches the name of the XML node. To continue the <eConnectCustomProcedure> example, the stored procedure must be named **eConnectCustomProcedure**.

The custom stored procedure's parameters must include the following:

- The stored procedure must provide an input parameter for each element of the custom XML node.
- The stored procedure's input parameters must match the order of the elements in the custom XML node.

- To comply with eConnect's error handling process, include two output parameters. Name the output parameters ErrorState and ErrString.

Refer to the SQL Server help documentation for information about creating and installing SQL stored procedures.

The following SQL example creates a stored procedure for the <eConnectCustomProcedure> XML node. Notice how the procedure is named, the way the XML node's data element maps to the input parameter, and the implementation of error handling:

```

/* Begin_Procs eConnectCustomProcedure */
if exists (select * from dbo.sysobjects where id =
    Object_id('dbo.eConnectCustomProcedure') and type = 'P')
begin
    drop proc dbo.eConnectCustomProcedure
end
go

create procedure dbo.eConnectCustomProcedure

    @I_vCUSTNMBR char(15), /* Customer Number - only required field */
    @O_iErrorState int output, /* Return value: 0 = No Errors, Any Errors > 0 */
    @oErrString varchar(255) output /* Return Error Code List */

as

declare
    @CUSTBLNC int,
    @O_oErrorState int,
    @iError int,
    @iStatus smallint, @iAddCodeErrState int

/***** Initialize locals *****/
select
    @O_iErrorState = 0,
    @oErrString = '',
    @iStatus = 0,
    @iAddCodeErrState = 0

/***** Custom Procedure edit check validation *****/
/*If the @I_vCUSTNMBR variable is '' then we need to add the error code */
/*35010 to the @oErrString output variable.*/
/*The method that eConnect uses to append all error string is the */
/*taUpdateString procedure.*/
/*Error codes can be appended to the @oErrString variable: for example you */
/*could append a 33 44 55 66 to the @oErrString variable */
/*After the error codes have been appended to the @oErrString variable. */
/*****
if ( @I_vCUSTNMBR = '' )
begin
    select @O_iErrorState = 35010 /* Customer number is empty */
    exec @iStatus = taUpdateString
        @O_iErrorState,
        @oErrString,
        @oErrString output,

```

```
        @iAddCodeErrState output
    end
    /* Do some custom business logic */
    select @CUSTBLNC = CUSTBLNC
        from RM00103 (nolock)
        where CUSTNMBR = @I_vCUSTNMBR
    /* End custom business logic */

    return (@O_iErrorState)
go

grant execute on dbo.eConnectCustomProcedure to DYNGRP
go
```



# Chapter 15: Business Logic Extensions

This portion of the documentation discusses how to customize the eConnect business logic to address unique business requirements. The following topics are discussed:

- [\*Modifying business logic\*](#)
- [\*Using Pre and Post stored procedures\*](#)

## Modifying business logic

When you use Microsoft Dynamics GP Utilities to create a company, the creation process places all the eConnect stored procedures for that company's database on your Microsoft Dynamics GP SQL server. These stored procedures contain eConnect's business logic. You cannot modify any of the eConnect core stored procedures.

When eConnect processes an XML document, it executes a SQL stored procedure for each XML node in the document. When the stored procedure executes, it executes its Pre and Post SQL stored procedures. For example, when the **taSopHdrIvcInsert** stored procedure runs, it calls the procedures named **taSopHdrIvcInsertPre** and **taSopHdrIvcInsertPost**.

- The Pre stored procedure runs prior to the core stored procedure. In the example, the **taSopHdrIvcInsertPre** executes before the **taSopHdrIvcInsert** procedure starts.
- The Post stored procedure runs immediately after the core stored procedure. In the example, the **taSopHdrIvcInsertPost** stored procedure executes after the successful conclusion of the **taSopHdrIvcInsert** procedure.

For additional information about the sequence of events in an eConnect stored procedure, refer to the [\*Business objects\*](#) on page 14.

The eConnect Pre and Post stored procedures include two output parameters you use to implement custom error handling. The error-handling parameters are as follows:

- The Pre and Post stored procedures use the `ErrorState` parameter to specify whether an error occurred. The core eConnect stored procedure checks the value of the Pre and Post stored procedures' `ErrorState` parameter. If you encounter an error during your custom processing, set `ErrorState` to a non-zero value. This will cause the eConnect stored procedure to halt and roll back the transaction.
- The Pre and Post stored procedures include an `ErrString` output parameter. If you encounter an error during your custom processing, use the `ErrString` parameter to describe the error.

## Using Pre and Post stored procedures

To modify eConnect's business logic, place custom SQL code in the Pre or Post procedures. The custom code in the Pre and Post procedures allow you to modify or extend the behavior of the core eConnect stored procedure. To customize a Pre or Post stored procedure, complete the following steps:

**1. Open the .sql file for the stored procedure.**

eConnect supplies a file for each Pre and Post stored procedure you can modify. To find a specific file, open the directory C:\Program Files\Common Files\Microsoft Shared\eConnect 10\Custom Procedures. This directory contains a subdirectory for each transaction type schema. Open the subdirectory that contains the stored procedure you want to modify.

As an example, assume you want to modify the taUpdateCreateCustomerRcdPost stored procedure. Open the C:\Program Files\Common Files\Microsoft Shared\eConnect 10\Custom Procedures\Receivables directory. Next, open the taUpdateCreateCustomerRcdPost.sql file. You may edit the file using any text editor or Microsoft SQL Server Management Studio.

**2. Add your custom SQL code.**

With the .sql file open, you can add custom SQL code to the file. The only parts of the document you should change are the Revision History and the section of the file specified for custom business logic. Your SQL code should be added between the following comments:

```
/* Create Custom Business Logic */

/* End Create Custom Business Logic */
```

To avoid errors or unexpected results, do not modify any of the other statements in the file. After adding your custom business logic, save the file.

**3. Run the .sql file in Microsoft SQL Server Management Studio.**

Open the modified file with Microsoft SQL Server Management Studio. Use the drop-down list from the toolbar to specify the Microsoft Dynamics GP database that contains the target stored procedure. Click the Execute button. The Query Messages window displays whether the stored procedure was successfully updated. If it succeeded, the stored procedure now includes your custom SQL code.

The following SQL example shows a customized taCreateTerritoryPre stored procedure. The example overrides the value in the Territory Description (SLTERDSC) parameter to reflect that the sales territory was created using eConnect:

```
/* Begin_Procs taCreateTerritoryPre */
if exists (select * from sysobjects where id =
object_id('dbo.taCreateTerritoryPre')and type = 'P')
begin
    drop procedure dbo.taCreateTerritoryPre
end
go

create procedure dbo.taCreateTerritoryPre
/*
*****
* (c) 2004 Microsoft Business Solutions, Inc.
*****
*
* PROCEDURE NAME:taCreateTerritoryPre
```

```

*
* SANSRIPT NAME:NA
*
* PARAMETERS:
*
* DESCRIPTION:taCreateSalespersonPost Integration Stored Procedure
*
* TABLES:
*
*      Table NameAccess
*      =====
*
* PROCEDURES CALLED:
*
* DATABASE:Company
*
* RETURN VALUE:
*
*      0      = Successful
*      non-0= Not successful
*
* REVISION HISTORY:
*
*      Date      Who      Comments
*      -----
*
*
*
*****
*
*****
*/
@I_vSALSTERR char(15) output,/*Territory ID <Required>*/
@I_vSLTERDSC char(30) output,/*Territory Description <Optional>*/
@I_vSLPRSNID char(15) output,/*Salesperson ID <Optional>*/
@I_vSTMGRFNM char(15) output,/*Sales Terr Managers First Name <Optional>*/
@I_vSTMGRMM char(15) output, /*Sales Terr Managers Middle Name <Optional>*/
@I_vSTMGRLLM char(20) output,/*Sales Terr Managers Last Name <Optional>*/
@I_vCOUNTRY char(60) output, /*Country <Optional>*/
@I_vCOSTTODT numeric(19,5) output,/*Cost to Date <Optional>*/
@I_vTTLCOMTD numeric(19,5) output,/*Total Commissions to Date <Optional>*/
@I_vTTLCOMLY numeric(19,5) output,/*Total Commissions Last Year <Optional>*/
@I_vNCOMSLYR numeric(19,5) output,/*Non-Comm Sales Last Year <Optional>*/
@I_vCOMSLLYR numeric(19,5) output,/*Comm Sales Last Year <Optional>*/
@I_vCSTLSTYR numeric(19,5) output,/*Cost Last Year <Optional>*/
@I_vCOMSLTDT numeric(19,5) output,/*Commissioned Sales To Date <Optional>*/
@I_vNCOMSLTD numeric(19,5) output,/*Non-Comm Sales To Date <Optional>*/
@I_vKPCALHST tinyint output,/*Keep Calendar History - 0=No 1=Yes <Optional>*/
@I_vKPERHIST tinyint output,/*Keep Period History - 0=No 1=Yes <Optional>*/
@I_vMODIFDTdatetime output,/*Modified Date <Optional>*/
@I_vCREATDDTdatetime output, /*Create Date <Optional>*/
@I_vUSRDEFND1 char(50) output,/*User Defined field-developer use only*/
@I_vUSRDEFND2 char(50) output,/*User Defined field-developer use only*/
@I_vUSRDEFND3 char(50) output,/*User Defined field-developer use only*/
@I_vUSRDEFND4 varchar(8000) output,/*User Defined field-developer use only*/
@I_vUSRDEFND5 varchar(8000) output,/*User Defined field-developer use only */
@O_iErrorStateint output,/* Return value: 0=No Errors, 1=Error Occurred*/
@oErrString varchar(255) output/* Return Error Code List*/

```

```
as

set nocount on

select @O_iErrorState = 0

/* Create Custom Business Logic */

set @I_vSLTERDSC = 'Created by eConnect'

/* End Create Custom Business Logic */

return (@O_iErrorState)
go

grant execute on dbo.taCreateTerritoryPre to DYNGRP
go

/* End_Procs taCreateTerritoryPre */
```



# Part 6: Transaction Requester

This portion of the documentation contains information about the eConnect Transaction Requester Service. You use Transaction Requester to retrieve data from Microsoft Dynamics GP. The following topics are discussed:

- [Chapter 16, “Using the Transaction Requester.”](#) discusses how to use the Transaction Requester Service. The Transaction Requester publishes eConnect XML documents to an MSMQ queue.
- [Chapter 17, “Customizing the Transaction Requester.”](#) discusses creating a custom Transaction Requester Service that retrieves data in ways that a base Transaction Requester Service cannot.

## Chapter 16: Using the Transaction Requester

The Transaction Requester is an eConnect service that publishes eConnect XML documents to an MSMQ queue. You use the Transaction Requester to retrieve information about specific Microsoft Dynamics GP documents and operations. The discussion addresses the following topics:

- [\*The Transaction Requester Service\*](#)
- [\*Requester documents types\*](#)
- [\*Requester document tables\*](#)
- [\*Using the RequesterTrx element\*](#)

### The Transaction Requester Service

eConnect allows you to specify documents and operations that cause an XML document representation of that operation to be published. There are two components to the Transaction Requester Service:

**eConnect Requester Setup** This utility allows you to specify the XML documents that are published to an MSMQ queue. Use eConnect Requester Setup to identify Microsoft Dynamics GP documents, operations, and the MSMQ queue that receives the document. Refer to the eConnect Installation and Administration Guide to learn about using the eConnect Requester Setup utility.

**Outgoing Service** The Transaction Requester employs the Outgoing Service to publish the specified XML documents to the queue. To use the Transaction Requester Service, you must configure and enable the Outgoing Service. Refer to the eConnect Installation Guide to learn about the Outgoing Service's configuration options.

Once the XML documents are published to the queue, your application can retrieve them from the queue. You can then parse the XML document and perform actions based upon the information they contain. For more information on retrieving from a queue, see [Chapter 12, "Outgoing Service."](#)

### Requester documents types

The Transaction Requester Service allows you to request an XML document related to a create, update, or delete operation for the following Microsoft Dynamics GP documents:

- Cash\_Receipt
- Customer
- Customer\_Balance
- Employee
- GL\_Accounts
- GL\_Hist\_Trans
- GL\_Open\_Trans
- GL\_Work\_Trans
- Item
- Item\_ListPrice
- Payables\_History\_Transaction
- Payables\_Posted\_Transaction
- Payables\_Transaction
- PO\_History\_Transaction

- PO\_Receiving\_Hist\_Trans
- PO\_Receiving\_Transaction
- Project\_Acct\_Contract
- Project\_Acct\_Contract\_Template
- Project\_Acct\_Cost\_Category
- Project\_Acct\_Employee\_Rate
- Project\_Acct\_EmployeeExpense
- Project\_Acct\_Equipment\_Rate
- Project\_Acct\_MiscLog
- Project\_Acct\_Position\_Rate
- Project\_Acct\_Project
- Project\_Acct\_Project\_Access
- Project\_Acct\_Project\_Template
- Project\_Acct\_Timesheet
- Purchase\_Order\_Transaction
- Receivables\_Hist\_Trans
- Receivables\_Posted\_Transaction
- Receivables\_Transaction
- Sales\_History\_Transaction
- Sales\_Transaction
- Vendor

## Requester document tables

The following table shows the Microsoft Dynamics GP tables the Transaction Requester uses to retrieve data for the specified document type:

| Document type                | Alias             | Tables used                      |
|------------------------------|-------------------|----------------------------------|
| Cash_Receipt                 | Cash_Receipt      | RM10201                          |
| Customer                     | Customer          | RM00101<br>RM00102               |
| Employee                     | Employee Address  | UPR00100<br>UPR00102             |
| GL_Hist_Trans                | GL_Hist_Trans     | GL30000                          |
| GL_Open_Trans                | GL_Open_Trans     | GL20000                          |
| Item                         | Item              | IV00101<br>IV00102               |
| Item_ListPrice               | Item_ListPrice    | IV00101<br>IV00105               |
| Payables_History_Transaction | PM_Hist_Trans     | PM30200<br>PM30700               |
| Payables_Posted_Transaction  | PM_Posted_Trans   | PM2000<br>PM10500                |
| Payables_Transaction         | PM_Trans          | PM1000<br>PM10500                |
| PO_History_Transaction       | PO_Hist_Trans     | POP30100<br>POP30110<br>POP10150 |
| PO_Receiving_Hist_Trans      | PO_Receiving_Hist | POP30300<br>POP30310<br>POP10500 |
| PO_Receiving_Transaction     | PO_Receiving      | POP10300<br>POP10310<br>POP10500 |



| Document type                  | Alias   | Tables used  |
|--------------------------------|---|--|
| Project_Acct_Contract          | PA_Contract<br>Cont_Bill_Cycle<br>PA_Project  | PA01101<br>PA02401<br>PA01201  |
| Project_Acct_Contract_Template | PA_Contract_Temp<br>Cont_Bill_Cycle_Temp  | PA41501<br>PA42901   |
| Project_Acct_Cost_Category     | Cost_Category   | PA01001  |
| Project_Acct_Employee_Rate     | PA_Employ_Rate<br>Line  | PA01402<br>PA01403   |
| Project_Acct_EmployeeExpense   | Project_Acct_EmployeeExpense  | PA10500<br>PA10501<br>PA10502  |
| Project_Acct_Equipment_Rate    | PA_Equip_Rate<br>Line   | PA01406<br>PA01407   |
| Project_Acct_MiscLog           | Project_Acct_MiscLog  | PA10200<br>PA10201   |
| Project_Acct_Position_Rate     | PA_Employ_Rate<br>Line  | PA01404<br>PA01405   |
| Project_Acct_Project           | PA_Project<br>Bill_Cycle<br>Budget<br>Budget_IVItems<br>Fee<br>Fee_Schedule<br>Access_List<br>Equip_List              | PA01201<br>PA61020<br>PA01301<br>PA01303<br>PA02101<br>PA05200<br>PA01408<br>PA01409 |
| Project_Acct_Project_Access    | Proj_Acct_List<br>Employee_Detail<br>Address  | PA01408<br>UPR00100<br>UPR00102  |
| Project_Acct_Project_Template  | PA_Project_Temp<br>Proj_Bill_Cycle_Temp<br>Budget<br>Budget_Items<br>Fee<br>Fee_Schedule<br>Equip_List<br>Access_List | PA41601<br>PA60020<br>PA40201<br>PA40202<br>PA60040<br>PA40203<br>PA41409<br>PA41401 |
| Project_Acct_Timesheet         | Project_Acct_Timesheet  | PA10000<br>PA10001   |
| Purchase_Order_Transaction     | PO_Trans  | POP10100<br>POP10110<br>POP10150   |
| Receivables_Hist_Trans         | RM_Hist_Trans   | RM30101<br>RM30601   |
| Receivables_Posted_Transaction | RM_Posted_Trans   | RM20101<br>RM10601   |
| Receivables_Transaction        | RM_Trans  | RM10301<br>RM10601   |
| Sales_History_Transaction      | SO_His_Trans  | SOP30200<br>SOP30300<br>SOP10105   |
| Sales_Transaction              | SO_Trans  | SOP10100<br>SOP10200<br>SOP10105   |

| Document type | Alias  | Tables used        |
|---------------|--------|--------------------|
| Vendor        | Vendor | PM00200<br>PM00300 |

## Using the RequesterTrx element

eConnect allows you to specify how some incoming transactions should be handled by a Transaction Requester Service. By default, a Transaction Requester Service causes all insertions, updates, or deletions to be reflected in the eConnect\_Out shadow table. These entries will then be processed by the Outgoing Service and an XML document placed in an MSMQ queue.

To prevent some incoming transaction from producing entries in the eConnect\_Out table, set the value of the XML node's **RequesterTrx** element to 0. Setting this element signals to the business objects that the Transaction Requester Service should not reflect this transaction in the eConnect\_Out shadow table.

To enable the **RequesterTrx** element, you must add Pre and Post procedures for each transaction targeted by the custom Transaction Requester Service. By adding Pre and Post procedure calls, you prevent the transaction from being detected by the Transaction Requester Service.

To enable the **RequesterTrx** element, the parameters of the Pre and Post procedures require the following settings:

- The DOCTYPE parameter must be set to the name of the custom Transaction Requester Service. In the prior example, the Transaction Requester Service was named Employee.
- Set the index parameters (INDEX1 - 15) to values that define the indexes used by your custom Transaction Requester Service. The Pre and Post procedures must supply an index value for each index set up for the custom Transaction Requester Service. If your custom Transaction Requester Service defines three indexes, your Pre and Post stored procedures must supply values for @I\_vINDEX1, @I\_vINDEX2, and @I\_vINDEX3 parameters. In the SQL samples that follow, INDEX1 is set to @I\_v EMPLOYID since this is the single index defined by the Employee custom Transaction Requester Service example.

The following SQL code is used to update the Pre and Post procedures for the sample Employee custom Transaction Requester Service:



*The only difference between the Pre and Post procedures is the value of the @I\_vDelete parameter.*

### Pre procedure example

```

/** Call eConnectOutVerify proc */
if (@I_vRequesterTrx = 0)
begin
    exec @iStatus = eConnectOutVerify
        @I_vDOCTYPE ='Employee',
        @I_vINDEX1=@I_v EMPLOYID ,
        @I_vINDEX2='',
        @I_vINDEX3='',
        @I_vINDEX4='',
        @I_vINDEX5='',

```

```

        @I_vINDEX6='',
        @I_vINDEX7='',
        @I_vINDEX8='',
        @I_vINDEX9='',
        @I_vINDEX10='',
        @I_vINDEX11='',
        @I_vINDEX12='',
        @I_vINDEX13='',
        @I_vINDEX14='',
        @I_vINDEX15='',
        @I_vDelete = 0,
        @O_iErrorState = @ iCustomState output
select @iError = @@error
if @iStatus = 0 and @ iError <> 0
begin
    select @iStatus = @ iError
end
if (@iStatus <> 0) or (@ iCustomState <> 0)
begin
    select @O_iErrorState = 9999 /* eConnectOutVerify proc returned an
error value */
    exec @iStatus = taUpdateString
        @O_iErrorState ,
        @oErrString ,
        @oErrString output,
        @O_oErrorState output
end
end
end

```

## Post procedure example

```

/**/ Call eConnectOutVerify proc /**/
if (@I_vRequesterTrx =0)
begin
    exec @iStatus = eConnectOutVerify
        @I_vDOCTYPE = ' Employee ',
        @I_vINDEX1=@I_v EMPLOYID ,
        @I_vINDEX2='',
        @I_vINDEX3='',
        @I_vINDEX4='',
        @I_vINDEX5='',
        @I_vINDEX6='',
        @I_vINDEX7='',
        @I_vINDEX8='',
        @I_vINDEX9='',
        @I_vINDEX10='',
        @I_vINDEX11='',
        @I_vINDEX12='',
        @I_vINDEX13='',
        @I_vINDEX14='',
        @I_vINDEX15='',
        @I_vDelete = 1,
        @O_iErrorState = @ iCustomState output
select @iError = @@error
if @iStatus = 0 and @ iError <> 0
begin
    select @iStatus = @ iError

```

```
end
if (@iStatus <> 0) or (@ iCustomState <> 0)
begin
    select @O_iErrorState = 9999 /* eConnectOutVerify proc returned an
error value */
    exec @iStatus = taUpdateString
        @O_iErrorState ,
        @oErrString ,
        @oErrString output,
        @O_oErrorState output
end
end
```

# Chapter 17: Customizing the Transaction Requester

eConnect allows you to create custom Transaction Requester Services that retrieve data from Microsoft Dynamics GP. This portion of the document describes how to create a custom Transaction Requester Service. The discussion includes the following topics:

- [Create a custom Transaction Requester Service](#)
- [Using the <taRequesterTrxDisabler> XML node](#)

## Create a custom Transaction Requester Service

The eConnect Requester Service allows you to create a custom Transaction Requester Service tailored to your specific business needs. This allows you to retrieve data in ways that the base Transaction Requester Service cannot. For example, you can create a service that retrieves data from related tables that the base Transaction Requester does not include.

To create a custom Transaction Requester Service, you must define the tables that the custom Transaction Requester Service uses. The components and relationships the custom Transaction Requester uses must be added to the eConnect\_Out\_Setup table in your Microsoft Dynamics GP database. Use a SQL query to add your custom Transaction Requester Service to the eConnect\_Out\_Setup table.

After you define the data that the custom Transaction Requester Service uses, the service works the same as the base Transaction Requester Service. The Outgoing Service will publish the XML document to MSMQ where your application can retrieve your custom data.



*You must recreate all custom requester services after an installation has completed. The install drops and recreates the eConnect\_Out\_Setup table. Any custom information in the eConnect\_Out\_Setup table is lost.*

To create a custom Transaction Requester, your query uses the elements described in the following table.

| Column name    | Data type      | Description   |
|----------------|----------------|---|
| DOCTYPE        | <b>varchar</b> | Identifies the service.   |
| INSERT_ENABLED | <b>int</b>     | Determines whether the service is enabled or disabled for an insert action. Enabled=1, Disabled=0                 |
| UPDATE_ENABLED | <b>int</b>     | Determines whether the service is enabled or disabled for an update action. Enabled=1, Disabled=0                 |
| DELETE_ENABLED | <b>int</b>     | Determines whether the service is enabled or disabled for a delete action. Enabled=1, Disabled=0                  |
| TABlename      | <b>varchar</b> | Physical name of table in SQL Server.   |
| ALIAS          | <b>varchar</b> | Alias name for DOCTYPE, which is used in the output XML document. (Try to keep alias names as short as possible.) |

| Column name | Data type | Description  |
|-------------|-----------|--|
| MAIN        | int       | Determines whether this record is associated with the primary table or a child table. MAIN=1 defines a primary table; MAIN=2 or greater defines a secondary table. Increment by one for every level.   |
| PARENTLEVEL | int       | Determines the parent for this record. When you specify a table as a secondary table (the value of MAIN is greater than 1) you need to specify its parent level. Set the parent level to 1 if it directly links to the main table. If your secondary table links to a child table of the main table, use the value in the child table's MAIN column as your table's PARENTLEVEL value. |
| ORDERBY     | int       | Defines whether this level needs to be included in the order by clause. ORDERBY=1 defines this level to be included; ORDERBY=0 defines this level to be ignored.   |
| USERDEF1-5  | varchar   | Used for any user-defined purpose.   |
| REQUIRED1   | varchar   | Defines the name of the column to verify whether data exists for it during an insert transaction. If the column specified is empty, this transaction is ignored. If the column has data in it and the service is enabled, the transaction is echoed to the shadow table (eConnect_Out). If no column is specified, all transactions are written out to the shadow table.               |
| INDEX1-15   | varchar   | Specifies the column name of the primary index that should be used for this table. You can specify 1 to 15 columns.  |
| INDEXCNT    | int       | Defines the number of columns that are specified for the index columns.  |
| TRIGGER1-15 | varchar   | Specifies column names used for creating triggers. The columns specified should link back to the columns that you specified in the INDEX1-15 columns of the main table. These columns are used to determine what data needs to be written out to the shadow table (eConnect_Out).  |
| JOINTABLE   | varchar   | Specifies the table name that needs to be used for joining.  |
| JOIN1-10    | varchar   | Specifies column names from the table specified in the TABLENAME column that are used to join to the table specified in the JOINTABLE column.  |

| Column name | Data type      | Description   |
|-------------|----------------|---|
| JOINTO1-10  | <b>varchar</b> | Specifies column names from the table specified in the JOINTABLE column that are used in conjunction with the columns listed in the JOIN1-10 columns. |
| DATA1-180   | <b>int</b>     | Defines the number of columns that are specified for the data columns.  |
| DATA1-180   | <b>varchar</b> | Specifies names of the data columns that you want to appear in the XML document. You can specify 1 to 180 columns.                                    |

To prepare a query that installs a new custom Transaction Requester Service, you must include the following:

- When you create a custom requester service, you must always define a single table as the requester's main table. You specify the main table by setting the value in the queries MAIN field equal to 1. The number of tables that data is to be pulled from defines the number of records that must be inserted into the eConnect\_Out\_Setup table.
- Take care to match the columns specified by the JOIN1-10 and JOINTO1-10 fields. For example, the column specified by JOIN1 is used to create a join to another table using the column specified by JOINTO1.



*A Requester Service cannot support SQL Server data columns that are defined as either text or binary.*

The following SQL example shows an insert statement that creates a custom Requester Service for employee transactions. This custom Transaction Requester combines data from the Microsoft Dynamics GP UPR00100 and UPR00102 tables.

```
/* Employee Document Setup */
/* this insert will create the record for the parent table - UPR00100 */
insert into eConnect_Out_Setup (
    DOCTYPE,
    MAIN,
    PARENTLEVEL,
    ORDERBY,
    INDEX1,
    INDEXCNT,
    TRIGGER1,
    TRIGGERCNT,
    TABLENAME,
    ALIAS,
    DATA1, DATA2, DATA3, DATA4, DATA5, DATA6, DATA7, DATA8, DATA9, DATA10,
```

```

        DATA11,
        DATA12,
        DATA13,
        DATA14,
        DATA15,
        DATA16,
        DATA17
    )
select
    'Employee',
    1,
    0,
    1,
    'EMPLOYID',
    1,
    'EMPLOYID',
    1,
    'UPR00100',
    'Employee',
    17,
    'EMPLCLAS',
    'INACTIVE',
    'LASTNAME',
    'FRSTNAME',
    'MIDLNAME',
    'ADRSCODE',
    'SOCSCNUM',
    'BRTHDATE',
    'GENDER',
    'ETHNORGN',
    'Calc_Min_Wage_Bal',
    'DIVISIONCODE_I',
    'DEPRTMNT',
    'JOBTITLE',
    'SUPERVISORCODE_I',
    'LOCATNID',
    'WCACFPAY'
GO
/* Employee Address Document Setup */
/* this insert will create the record for the child table */
/* (UPR00102) and link it to the parent table (UPR00100) */
insert into eConnect_Out_Setup (
    DOCTYPE,
    MAIN,
    PARENTLEVEL,
    ORDERBY,
    INDEX1,
    INDEX2,
    INDEXCNT,
    TRIGGER1,
    TRIGGERCNT,
    TABLENAME,
    ALIAS,
    JOINTABLE,
    JOIN1,
    JOINTO1,

```



```

        DATACNT,
        DATA1,
        DATA2,
        DATA3,
        DATA4,
        DATA5,
        DATA6,
        DATA7,
        DATA8,
        DATA9,
        DATA10,
        DATA11,
        DATA12,
        DATA13,
        DATA14,
        DATA15,
        DATA16
    )
select
    'Employee',
    2,
    1,
    1,
    'EMPLOYID',
    'ADRSCODE',
    2,
    'EMPLOYID',
    1,
    'UPR00102',
    'Address',
    'UPR00100',
    'EMPLOYID',
    'EMPLOYID',
    16,
    'ADDRESS1',
    'ADDRESS2',
    'ADDRESS3',
    'CITY',
    'STATE',
    'ZIPCODE',
    'COUNTY',
    'COUNTRY',
    'PHONE1',
    'PHONE2',
    'PHONE3',
    'FAX',
    'Foreign_Address',
    'Foreign_StateProvince',
    'Foreign_Postal_Code',
    'CCode'
GO

```

The sample custom Transaction Requester produces XML documents that contain the following type of information:

Master Document (OutputType=1)

```

<root>
  <eConnect_Out ACTION="0" EMPLOYID="BARR0001">
    <Employee>
      <EMPLOYID>HEALY0001</EMPLOYID>
      <EMPLCLAS>INST</EMPLCLAS>
      <INACTIVE>0</INACTIVE>
      <LASTNAME>Healy</LASTNAME>
      <FRSTNAME>Joe</FRSTNAME>
      <MIDLNAME></MIDLNAME>
      <ADRSCODE>PRIMARY</ADRSCODE>
      <SOCSCNUM>944229198</SOCSCNUM>
      <BRTHDATE>1961-10-07T00:00:00</BRTHDATE>
      <GENDER>1</GENDER>
      <ETHNORGN>1</ETHNORGN>
      <Calc_Min_Wage_Bal>0</Calc_Min_Wage_Bal>
      <DIVISIONCODE_I></DIVISIONCODE_I>
      <DEPTMNT>CONS</DEPTMNT>
      <JOBTITLE>CONS1</JOBTITLE>
      <SUPERVISORCODE_I></SUPERVISORCODE_I>
      <LOCATNID></LOCATNID>
      <WCACFPAY>0</WCACFPAY>
    </Employee>
  </eConnect_Out>
</root>

```

Complete Document (OutputType=2)

```

<root>
  <eConnect_Out ACTION="0" EMPLOYID="BARR0001">
    <Employee>
      <EMPLOYID>HEALY0001</EMPLOYID>
      <EMPLCLAS>INST</EMPLCLAS>
      <INACTIVE>0</INACTIVE>
      <LASTNAME>Healy</LASTNAME>
      <FRSTNAME>Joe</FRSTNAME>
      <MIDLNAME></MIDLNAME>
      <ADRSCODE>PRIMARY</ADRSCODE>
      <SOCSCNUM>944229198</SOCSCNUM>
      <BRTHDATE>1961-10-07T00:00:00</BRTHDATE>
      <GENDER>1</GENDER>
      <ETHNORGN>1</ETHNORGN>
      <Calc_Min_Wage_Bal>0</Calc_Min_Wage_Bal>
      <DIVISIONCODE_I></DIVISIONCODE_I>
      <DEPTMNT>CONS</DEPTMNT>
      <JOBTITLE>CONS1</JOBTITLE>
      <SUPERVISORCODE_I></SUPERVISORCODE_I>
      <LOCATNID></LOCATNID>
      <WCACFPAY>0</WCACFPAY>
    <Address>
      <EMPLOYID>HEALY0001</EMPLOYID>
      <ADRSCODE>PRIMARY</ADRSCODE>
      <ADDRESS1>4567 Main Ave</ADDRESS1>
      <ADDRESS2></ADDRESS2>
      <ADDRESS3></ADDRESS3>
      <CITY>Wauwatosa</CITY>
      <STATE>WI</STATE>
      <ZIPCODE>43210-9876 </ZIPCODE>
      <COUNTY></COUNTY>
    </Address>
  </eConnect_Out>
</root>

```

```

        <COUNTRY>USA</COUNTRY>
        <PHONE1>4145550150</PHONE1>
        <PHONE2></PHONE2>
        <PHONE3></PHONE3>
        <FAX></FAX>
        <Foreign_Address>0</Foreign_Address>
        <Foreign_StateProvince></Foreign_StateProvince>
        <Foreign_Postal_Code></Foreign_Postal_Code>
        <CCode></CCode>
    </Address>
</Employee>
</eConnect_Out>
</root>

```

## Using the <taRequesterTrxDabler> XML node

The <taRequesterTrxDabler> XML node allows your XML documents to disable core and third-party Transaction Requester Services for specific transactions. All the eConnect schemas allow you to add the <taRequesterTrxDabler> to your XML documents.

To disable a specific Transaction Requester Service, your XML document must include a <taRequesterTrxDabler> XML node that identifies the target Transaction Requester Service. Use the <taRequesterTrxDabler> XML node's DOCTYPE element to specify the name of the Transaction Requester Service. To disable multiple Transaction Requester Services, add a separate <taRequesterTrxDabler> node to identify each Transaction Requester Service.

The following XML example shows how to disable the two Transaction Requester Services named "Customer" and "Sales\_Transaction" for this SOP transaction:

```

<?xml version="1.0" encoding="utf-8" ?>
<eConnect xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
    <SOPTransactionType>
        <taRequesterTrxDabler_Items>
            <taRequesterTrxDabler>
                <DOCTYPE>Customer</DOCTYPE>
                <INDEX1>CONTOSOL0001</INDEX1>
            </taRequesterTrxDabler>
            <taRequesterTrxDabler>
                <DOCTYPE>Sales_Transaction</DOCTYPE>
                <INDEX1>INV2001</INDEX1>
                <INDEX2>3</INDEX2>
            </taRequesterTrxDabler>
        </taRequesterTrxDabler_Items>
        <taSopLineIvcInsert_Items>
            <taSopLineIvcInsert>
                <SOPTYPE>3</SOPTYPE>
                <SOPNUMBE>INV2001</SOPNUMBE>
                <CUSTNMBR>CONTOSOL0001</CUSTNMBR>
                <DOCDATE>02/02/2006</DOCDATE>
                <LOCNCODE>WAREHOUSE</LOCNCODE>
                <ITEMNMBR>ACCS-CRD-12WH</ITEMNMBR>
                <UNITPRCE>10.95</UNITPRCE>
                <XTNDPRCE>21.9</XTNDPRCE>
            </taSopLineIvcInsert>
        </taSopLineIvcInsert_Items>
    </SOPTransactionType>
</eConnect>

```

```

        <QUANTITY>2</QUANTITY>
        <ITEMDESC>Phone Cord - 12' White</ITEMDESC>
        <DOCID>STDINV</DOCID>
        <ADDRESS1>2345 Main St</ADDRESS1>
        <CITY>Aurora</CITY>
    </taSopLineIvcInsert>
    <taSopLineIvcInsert>
        <SOPTYPE>3</SOPTYPE>
        <SOPNUMBE>INV2001</SOPNUMBE>
        <CUSTNMBR>CONTOSOL0001</CUSTNMBR>
        <DOCDATE>02/02/2006</DOCDATE>
        <LOCNCODE>WAREHOUSE</LOCNCODE>
        <ITEMNMBR>ACCS-CRD-25BK</ITEMNMBR>
        <UNITPRCE>15.95</UNITPRCE>
        <XTNDPRCE>31.9</XTNDPRCE>
        <QUANTITY>2</QUANTITY>
        <ITEMDESC>Phone Cord - 25' Black</ITEMDESC>
        <DOCID>STDINV</DOCID>
        <ADDRESS1>2345 Main St</ADDRESS1>
        <CITY>Aurora</CITY>
    </taSopLineIvcInsert>
</taSopLineIvcInsert_Items>
<taSopHdrIvcInsert>
    <SOPTYPE>3</SOPTYPE>
    <DOCID>STDINV</DOCID>
    <SOPNUMBE>INV2001</SOPNUMBE>
    <TAXSCHID>USASTCITY-6*</TAXSCHID>
    <FRTSCHID>USASTCITY-6*</FRTSCHID>
    <MSCSCHID>USASTCITY-6*</MSCSCHID>
    <LOCNCODE>WAREHOUSE</LOCNCODE>
    <DOCDATE>02/02/2006</DOCDATE>
    <CUSTNMBR>CONTOSOL0001</CUSTNMBR>
    <CUSTNAME>Contoso Ltd</CUSTNAME>
    <ShipToName>WAREHOUSE</ShipToName>
    <ADDRESS1>2345 Main St</ADDRESS1>
    <CNTCPRSN>Joe Healy</CNTCPRSN>
    <FAXNUMBR>13215550150</FAXNUMBR>
    <CITY>Aurora</CITY>
    <STATE>IL</STATE>
    <ZIPCODE>60507</ZIPCODE>
    <COUNTRY>USA</COUNTRY>
    <SUBTOTAL>53.8</SUBTOTAL>
    <DOCAMNT>53.8</DOCAMNT>
    <BACHNUMB>eConnect</BACHNUMB>
    <PYMTRMID>Net 30</PYMTRMID>
</taSopHdrIvcInsert>
</SOPTransactionType>
</eConnect>

```



# Part 7: eConnect Samples

This portion of the documentation describes several sample applications that demonstrate how to use eConnect. The following samples are discussed:

- [Chapter 18, "Create a Customer,"](#) describes a sample that creates a Microsoft Dynamics GP customer.
- [Chapter 19, "Create a Sales Order,"](#) describes a sample that creates a Microsoft Dynamics GP sales order document.
- [Chapter 20, "XML Document Manager,"](#) describes a sample that uses eConnect XML documents stored in XML files to complete operations in a Microsoft Dynamics GP database.
- [Chapter 21, "Get a Document Number,"](#) describes two samples that demonstrate how to retrieve document numbers from Microsoft Dynamics GP.
- [Chapter 22, "Retrieve Data,"](#) describes a sample that retrieves customer data from Microsoft Dynamics GP.
- [Chapter 23, "MSMQ Document Sender,"](#) describes a sample application that converts an eConnect XML document to MSMQ message and places the message in a queue.

## Chapter 18: Create a Customer

This sample application demonstrate how to use the eConnect .NET assemblies to create a new Microsoft Dynamics GP customer. This console application provides a basic example of creating Microsoft Dynamics GP data with eConnect. The following topics are discussed:

- [\*Overview\*](#)
- [\*Running the sample application\*](#)
- [\*How the sample application works\*](#)
- [\*How eConnect was used\*](#)

### Overview

The sample application shows how to use eConnect serialization classes, write an eConnect XML document to a file, and create a new Microsoft Dynamics GP customer. Before you run this application, set the eConnect connection string in the source file to access the appropriate Microsoft Dynamics GP database.

This sample application is available in both C# and Visual Basic .NET. To build this application, you must have Visual Studio 2005 and the 2.0 .NET Framework installed on your computer.

### Running the sample application

To run this sample application, perform the following steps:

**1. Start Visual Studio 2005 and open the solution file for the sample application.**

The solution file for the C# version of the sample is named eConnect\_CSharp\_ConsoleApplication.sln. The solution file is in the CSHARPConsoleApplication folder inside the Samples folder.

The solution file for the Visual Basic .NET versions is named eConnect\_VB\_ConsoleApplication.sln. The solution file is in the VBDOTNETConsoleApplication folder inside the Samples folder.

**2. Open the source file.**

Open the Visual Studio Solution Explorer. Open the Class1.cs file in the C# project or Module1.vb file in the Visual Basic .NET project.

**3. Update the connection string.**

Locate the variable named sConnectionString and set the Data Source value to the name of your Microsoft Dynamics GP SQL Server. Set the Initial Catalog value to the name of the Microsoft Dynamics GP company database where you would like the new customer to be created.

**4. Choose Start Debugging from the Debug menu.**

To build the solution, choose “Start Debugging” in the Debug menu. Notice how the console window opens and closes.

**5. View the customer.xml file.**

The C# project creates the customer.xml file in the project’s \bin\debug folder. The Visual Basic .NET project creates the customer.xml file in the project’s \bin

folder. The customer.xml file contains the eConnect XML document for the new customer.

#### 6. Verify the customer is in Microsoft Dynamics GP.

Use the Microsoft Dynamics GP client to verify the customer was created. Search for the customer based on the eConnect XML document in the customer.xml file.

## How the sample application works

The sample application is a basic console application that uses classes from the eConnect .NET assemblies to create a new Microsoft Dynamics GP customer. The application creates an eConnect XML customer document using several eConnect serialization objects. When the document is complete, the application writes the document's XML to the customer.xml file.

The application validates the customer XML to ensure the eConnect XML document is complete. To perform validation, the application uses the eConnect schema information in the eConnect.xsd file.

The application uses eConnect connection string information to connect to the eConnect business objects in your Microsoft Dynamics GP database.

The application takes the eConnect XML document, accesses the eConnect business objects, and creates a new Microsoft Dynamics GP customer.

If an error occurs, the error message is displayed in the console window.

## How eConnect was used

The sample application uses classes from the Microsoft.Dynamics.GP.eConnect and Microsoft.Dynamics.GP.eConnect.Serialization assemblies.

### Microsoft.Dynamics.GP.eConnect

The application uses the **eConnectMethods** class to instantiate an **eConnectMethods** object. The application uses the object's **eConnect\_EntryPoint** method to create the customer. The **eConnect\_EntryPoint** method validates the eConnect XML document and manages access to the business objects on the server specified by the eConnect connection string.

### Microsoft.Dynamics.GP.eConnect.Serialization

The application uses several serialization classes to construct an eConnect XML document. To create an XML document, the application instantiates **eConnectType**, **RMCustomerMasterType**, and **taUpdateCreateCustomerRcd** objects.

The application first populates the **taUpdateCreateCustomerRcd** properties to specify the new customer. It uses the **taUpdateCreateCustomerRcd** object to populate the **RMCustomerMasterType** object. The **RMCustomerMasterType** then populates the **eConnectType** object. The **eConnectType** object represents a complete eConnect XML document.

The application writes the XML from the **eConnectType** object to a file. The application passes the file's XML contents to the **eConnect\_EntryPoint** method.



## Chapter 19: Create a Sales Order

The sample application uses the eConnect .NET assemblies to create a Microsoft Dynamics GP sales order document. The following topics are discussed:

- [\*Overview\*](#)
- [\*Running the sample application\*](#)
- [\*How the sample application works\*](#)
- [\*How eConnect was used\*](#)

### Overview

The sample application shows how to use eConnect serialization classes, write an eConnect XML document to a file, and create a new Microsoft Dynamics GP sales order document. Before you run this application, set the eConnect connection string in the source file to access the appropriate Microsoft Dynamics GP database.

This sample application is available in both C# and Visual Basic .NET. To build this application, you must have Visual Studio 2005 and the 2.0 .NET Framework installed on your computer.

### Running the sample application

To run this sample application, perform the following steps:

**1. Start Visual Studio 2005 and open the solution file for the sample application.**

The solution file for the C# version of the sample is named eConnectSalesOrder\_CSharp\_ConsoleApplication.sln. The solution file is in the CSHARPSalesorderConsoleApplication folder inside the Samples folder.

The solution file for the Visual Basic .NET version is named eConnect\_VB\_ConsoleApplicationSales.sln. The solution file is in the VBDOTNETSalesorderConsoleApplication folder inside the Samples folder.

**2. Open the source file.**

Open the Visual Studio Solution Explorer. Open test.cs in the C# project or Module1.vb in the Visual Basic .NET project.

**3. Update the connection string.**

Locate the variable named sConnectionString and set the Data Source value to the name of your Microsoft Dynamics GP SQL Server. Set the Initial Catalog value to the name of the Microsoft Dynamics GP company database where you would like the new sales order to be created.

**4. Choose Start Debugging from the Debug menu.**

To build the solution, choose “Start Debugging” in the Debug menu. Notice how the console window opens and closes.

**5. View the SalesOrder.xml file.**

The C# project creates the SalesOrder.xml file in the project’s \bin\debug folder. The Visual Basic .NET project creates the SalesOrder.xml file in the \bin folder. The SalesOrder.xml file contains the eConnect XML document for the new sales order.

**6. Verify the sales order is in Microsoft Dynamics GP.**

Use the Microsoft Dynamics GP client to verify the sales order was created. Search for the sales order created based on the eConnect XML document in the SalesOrder.xml file.

**How the sample application works**

The sample application is a console application that uses classes from the eConnect .NET assemblies to create a new Microsoft Dynamics GP sales order document. The application creates an eConnect XML sales order document using several eConnect serialization classes. When the eConnect XML document is complete, the application writes the document's XML to the SalesOrder.xml file.

The application uses eConnect connection string information to connect to the eConnect business objects in your Microsoft Dynamics GP database.

The application takes the eConnect XML document, accesses the eConnect business objects, and creates a new Microsoft Dynamics GP sales order document.

If an error occurs, the error message is displayed in the console window.

**How eConnect was used**

The sample application uses classes from the Microsoft.Dynamics.GP.eConnect and Microsoft.Dynamics.GP.eConnect.Serialization assemblies.

**Microsoft.Dynamics.GP.eConnect**

The application uses the **eConnectMethods** class to instantiate an **eConnectMethods** object. The application uses the object's **eConnect\_EntryPoint** method to create the sales order. The **eConnect\_EntryPoint** method needs the sales order XML document and the eConnect connection string.

**Microsoft.Dynamics.GP.eConnect.Serialization**

The application uses several serialization classes to construct an eConnect XML document. To create a sales order document, the sample application instantiates the **eConnectType**, **SOPTransactionType**, **taSopLineIvcInsert\_ItemsTaSopLineIvcInsert**, and **taSopHdrIvcInsert** classes.

The application instantiates two **taSopLineIvcInsert\_ItemsTaSopLineIvcInsert** objects. It populates each object's properties to represent a sales order line item. The application completes the sales order by instantiating a **taSopHdrIvcInsert** object and populating its properties.

To combine the line items and header object into a logical unit, the application instantiates a **SOPTransactionType** object and populates it with the **taSopLineIvcInsert\_ItemsTaSopLineIvcInsert** and **taSopHdrIvcInsert** objects. The application completes the eConnect XML document by instantiating an **eConnectType** object and populating it with **SOPTransactionType** object.

The application uses a .NET XMLSerializer to write the eConnect XML document to the SalesOrder.xml file. The application passes the XML contents of this file to the **eConnect\_EntryPoint** method.

## Chapter 20: XML Document Manager

The Document Manager sample is a .NET application that uses eConnect XML documents to create, delete, update, and retrieve Microsoft Dynamics GP data. The following topics are discussed:

- [Overview](#)
- [Running the sample application](#)
- [How the sample application works](#)
- [How eConnect was used](#)

### Overview

This sample application uses classes from the eConnect .NET assemblies to process an eConnect XML document. The sample displays the message or data from each eConnect operation. To process an eConnect XML document, you must first configure the eConnect connection string.

*This is the eConnect XML document to submit.*

*Click the Send XML button to submit the document.*

```
<?xml version="1.0" encoding="utf-8"?><root>eConnect_Out ACTION="0"
INDEX1="AARONFIT0001"><Customer><CUSTNMBR>AARONFIT0001</CUSTNMBR><ADDRESS1>11403 13th
Avenue South</ADDRESS1><ADDRESS2></ADDRESS2><ADDRESS3>
</ADDRESS3><ADRSCODE>PRIMARY</ADRSCODE><CITY>Chicago
</CITY><CNTCPRSND>Bob Fitz</CNTCPRSND><COUNTRY>USA
</COUNTRY><CPRCSNM></CPRCSNM><CPRCSNM><CURNCYID>ZUS$
</CURNCYID><CUSTCLAS>USA-ILMO-T1</CUSTCLAS><CUSTDISC>0</CUSTDISC><CUSTNAME>Aaron
Fitz Electrical</CUSTNAME><PHONE1>31243464750000
</PHONE1><PYMTRMID>Net 30</PYMTRMID><SALSTERR>TERRITORY 1
</SALSTERR><SHIPMTHD>LOCAL DELIVERY</SHIPMTHD><SLPRSNID>SEAN W.
</SLPRSNID><STATE>IL</STATE><TAXSCHID>USASTCITY-6*
</TAXSCHID><TXRGNNUM></TXRGNNUM><TXRGNNUM><UPSZONE><ZIP>60603-0776
</ZIP></Customer></eConnect_Out>eConnect_Out ACTION="0"
INDEX1="ADAMPARK0001"><Customer><CUSTNMBR>ADAMPARK0001</CUSTNMBR><ADDRESS1>Suite 63
```

This sample application is available in both C# and Visual Basic .NET. To build this application, you must have Visual Studio 2005 and the 2.0 .NET Framework installed on your computer.



*The C# and Visual Basic .NET versions perform the same tasks but the user interfaces are slightly different.*

### Running the sample application

To run this sample application, perform the following steps:

- 1. Start Visual Studio 2005 and open the solution file for the sample application.**

The solution file for the C# version of the sample is named DirectDocSenderDotNet.sln. The solution file is in the CSHARPDirectDocSender folder inside the Samples folder.

The solution file for the Visual Basic .NET version is named `XmlDocumentSender.sln`. The solution file is in the `XmlDocumentSender` folder inside the `Samples` folder.

**2. Choose Start Debugging from the Debug menu.**

To build the solution, choose “Start Debugging” in the `Debug` menu. The application starts.

**3. Update the connection string.**

Click the ellipsis (...) button next to the `Connection String` box. A dialog box opens. Enter your Microsoft Dynamics GP SQL Server name, log in name, password, and database name. Click `OK`.

**4. Select an eConnect XML document file.**

Click the ellipsis (...) button next to the `Select XML File` button. A dialog box opens. Use the dialog box to find a file that contains an eConnect XML document. Select the file and click `Open`.

**5. Review the XML document.**

The `XML Document` box displays the contents of the file you selected. You may edit the XML in the box. If you edit the XML, click the `Save XML` button to write your changes to a file.

**6. Send the XML document.**

Click the `Send XML` button. The application uses classes from the eConnect .NET assemblies to perform the operations specified by the XML document.

**7. Review the results.**

The `Return Information` box displays the result.

## How the sample application works

The sample application uses a class from the eConnect .NET assemblies to process eConnect XML documents. The application creates, deletes, updates and retrieves Microsoft Dynamics GP data using the eConnect XML document.

The application requires an eConnect connection string to specify a Microsoft Dynamics GP database. Use the connection string dialog box to specify your server name, log in name , password, and database.

The application uses eConnect XML documents to perform operations on Microsoft Dynamics GP data. The contents of the eConnect XML document determines the type of operation.

Click the `Send XML` button to have the application perform the document’s create, delete, update, or request operation. The application displays the return result of each operation in the `Return Information` box.

The application allows you to edit the contents of the `XML Document` box. To save the changes you make to the eConnect XML document, click the `Save XML` button. The application writes the contents of the `XML Document` box to a file.

If an error occurs, the application displays the error message in the `Return Information` box.

## How eConnect was used

The sample application uses a class from the Microsoft.Dynamics.GP.eConnect assemblies.

### Microsoft.Dynamics.GP.eConnect

When you click the Send Xml button, the application instantiates an **eConnectMethods** object. How the object is used depends upon the eConnect XML document.

- If the document's **eConnectProcessInfo** node contains the element **Outgoing** and the element value is **TRUE**, the document's XML is requesting Microsoft Dynamics GP data.

When the eConnect XML document requests data, the application uses the **eConnectMethods** object's **eConnect\_Requester** method. The **eConnect\_Requester** method returns an eConnect XML document string that contains the requested data.

- If the **eConnectProcessInfo** node does not contain the **Outgoing** element or it is set to **FALSE**, the document's XML creates, updates, or deletes Microsoft Dynamics GP data.

When the eConnect XML document creates, updates, or deletes Microsoft Dynamics GP data, the application uses the **eConnectMethods** object's **eConnect\_EntryPoint** method. The **eConnect\_EntryPoint** method uses the eConnect XML to perform the specified operation and returns a boolean value that specifies whether the operation was successful.



*This sample application does not validate the eConnect XML document before sending its XML element values to the business objects.*



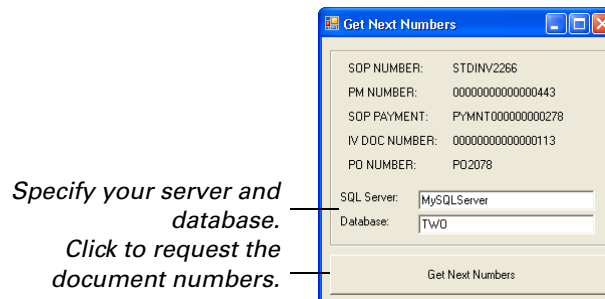
## Chapter 21: Get a Document Number

The Document Number samples include two .NET applications that retrieve the next available document numbers from Microsoft Dynamics GP. The following topics are discussed:

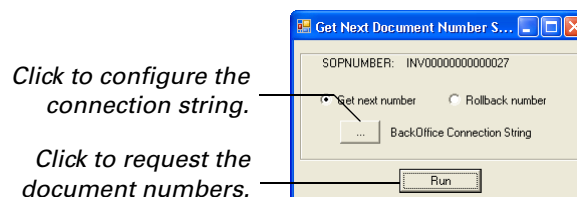
- [Overview](#)
- [Running the sample applications](#)
- [How the sample applications work](#)
- [How eConnect was used](#)

### Overview

There are two sample applications that demonstrate how to retrieve document numbers. The C# version allows you to request a document number for SOP, PM, SOP Payments, IV, and PO.



The Visual Basic .NET version demonstrates how to retrieve and return a SOP document number.



To build either sample application, you must have Visual Studio 2005 and the 2.0 .NET Framework installed on your computer.

### Running the sample applications

To run either sample application, perform the following steps:

- 1. Start Visual Studio 2005 and open the solution file for the sample application.**

The solution file for the C# sample application is named `DocumentNumberSample.sln`. The solution file is in the `CSHARPGetNextDocumentNumber` folder inside the `Samples` folder.

The solution file for the Visual Basic .NET sample application is named `NextNum_TestHarness.sln`. The solution file is in the `NextNum_TestHarness` folder inside the `Samples` folder.

**2. Choose Start Debugging from the Debug menu.**

To build either solution, choose “Start Debugging” in the Debug menu. The application starts.

**3. Update the connection string.**

To configure the connection string in the C# sample application, enter the name of your Microsoft Dynamics GP SQL server in the SQL Server box. Enter the name of your Microsoft Dynamics GP database in the Database box.

The Visual Basic .NET application uses a dialog box to manage the connection string parameters. Click the ellipsis (...) button labeled BackOffice Connection String. Use the dialog box to specify your Microsoft Dynamics GP server name, log in name, password, and database. Click OK to close the dialog window.

**4. Retrieve the next document number.**

To retrieve the next available document numbers using the C# application, click the Get Next Numbers button. The application will display the next number for each document type.

To retrieve the next available SOP document number using the Visual Basic .NET application, mark the **Get next number** option and click the Run button. The application display the next available SOP number. To return the number, mark the **Rollback number** option and click the Run button.

## How the sample applications work

Each time you click the C# application’s Get Next Numbers button, it retrieves the next document number for each document type from the company identified by the SQL Server and Database settings. The application displays the number for each type of document.

If you mark the Visual Basic .NET application’s **Get next number** option and click the Run button, the application retrieves and displays the next available SOP document number. If you subsequently mark the **Rollback number** option and click Run, the application will indicate that it has put back the specified SOP document number.

If an error occurs while running either application, the application opens a dialog box and displays the error message.

## How eConnect was used

Both sample applications use classes in the Microsoft.Dynamics.GP.eConnect.MiscRoutines assembly.

### Microsoft.Dynamics.GP.eConnect.MiscRoutines

The two applications vary in how they use the Microsoft.Dynamics.GP.eConnect.MiscRoutines classes. If you review the button click event handlers for each application you will note the following differences:

- The C# application instantiates a **GetNextDocNumbers** object and a **GetSopNumber** object. The application uses the **GetSopNumber** object’s **GetNextSopNumber** method to retrieve the next available SOP document number.



The application uses the **GetNextDocNumbers** object to retrieve numbers for the other document types it displays. To retrieve these numbers, the application uses the **GetNextPMPaymentNumber**, **GetNextRMNumber**, **GetNextIVNumber**, and **GetNextPONumber** methods.

- The Visual Basic .NET application instantiates a **GetSopNumber** object. If the **Get next number** option is marked, the application uses the **GetSopNumber** object's **GetNextSopNumber** method to retrieve the next SOP document number. If the **Rollback number** option is marked, the application uses the **GetSopNumber** object's **RollBackSopNumber** method.



## Chapter 22: Retrieve Data

The Requester Console Application uses an eConnect XML document to retrieve Microsoft Dynamics GP data. The following topics are discussed:

- [Overview](#)
- [Running the sample application](#)
- [How the sample application works](#)
- [How eConnect was used](#)

### Overview

This C# sample application uses eConnect serialization classes to create an eConnect XML request document. The application serializes the request document to a file. The application uses the XML from the file to retrieve Microsoft Dynamics GP customer data. The application displays the customer data in the console window.

Before you run this application, set the eConnect connection string in the source file to access the appropriate Microsoft Dynamics GP database.

To build this application, you must have Visual Studio 2005 and the 2.0 .NET Framework installed on your computer.



*The application's request document requires you to target a Microsoft Dynamics GP server that includes the TWO sample database.*

### Running the sample application

To run this sample application, perform the following steps:

**1. Start Visual Studio 2005 and open the solution file for the sample application.**

The solution file for this sample is named RequesterConsoleApplication.sln. The solution file is in the CSHARPRequesterConsoleApplication folder inside the Samples folder.

**2. Open the source file.**

Open the Visual Studio Solution Explorer and open the Class1.cs file.

**3. Update the connection string.**

Locate the variable named sConnectionString and set the Data Source value to the name of your Microsoft Dynamics GP SQL Server. Set the Initial Catalog value to TWO.

**4. Build the RequesterConsoleApplication.exe.**

Choose "Build RequesterConsoleApplication" in the Build menu.

**5. Open a console window.**

From the Start menu, choose All Programs >> Accessories >> Command Prompt. In the console window, open the directory where you built the RequesterConsoleApplication.exe.

**6. Run the RequesterConsoleApplication.**

Type RequesterConsoleApplication.exe and press Enter. When the application completes, the console window displays XML data for Aaron Fitz Electric.

**How the sample application works**

The RequesterConsoleApplication uses classes from the eConnect .NET assemblies to create an eConnect XML request document. The application creates the eConnect XML request document using several eConnect serialization classes. The application converts the eConnect XML request document to an XML string.

The application uses the XML string and the eConnect business objects to retrieve an XML document that contains the requested customer data. The application displays the customer data in the console window.

The application uses eConnect connection string information to connect to the eConnect business objects.

If an error occurs, the application displays the error message in the console window.

**How eConnect was used**

The sample application uses classes from the Microsoft.Dynamics.GP.eConnect and Microsoft.Dynamics.GP.eConnect.Serialization assemblies.

**Microsoft.Dynamics.GP.eConnect**

The application instantiates the **eConnectMethods** class to create an **eConnectMethods** object. The application uses the object's **eConnect\_Requester** method to retrieve the specified customer data. The **eConnect\_Requester** method requires a string that represents an eConnect XML request document and an eConnect connection string.

**Microsoft.Dynamics.GP.eConnect.Serialization**

The application uses several serialization classes to construct an eConnect XML request document. To create a request document, the sample application instantiates the **eConnectType**, **RQeConnectOutType**, and **eConnectOut** classes.

The application populates the properties of the **eConnectOut** object to specify the customer. It then populates **RQeConnectOutType** object with the **eConnectOut** object. The application completes the eConnect XML request document by populating the **eConnectType** object with the **RQeConnectOutType** object.

The application uses a .NET XmlSerializer to write the **eConnectType** object as a string. The application uses the XML string as parameter of the **eConnect\_Requester** method.

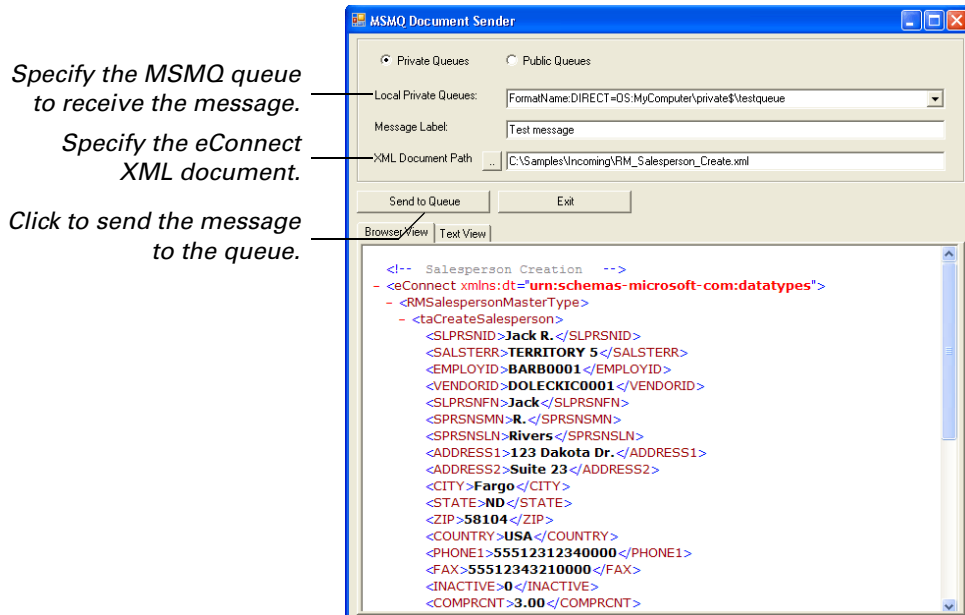
## Chapter 23: MSMQ Document Sender

The MSMQ Document Sender sample is a .NET application that converts an eConnect XML document to a Microsoft Message Queue (MSMQ) message and places that message in a specified queue. The following topics are discussed:

- [Overview](#)
- [Running the sample application](#)
- [How the sample application works](#)
- [How eConnect was used](#)

### Overview

This sample application converts an eConnect XML document to a MSMQ message and places the message in a MSMQ queue. Use this application with the eConnect Incoming Service to create, update, and delete Microsoft Dynamics GP data.



This sample application uses Visual Basic .NET. To build this application, you must have Visual Studio 2005 and the 2.0 .NET Framework installed on your computer. You must also have MSMQ installed and running on your computer.

### Running the sample application

To run this sample application, perform the following steps:

**1. Start Visual Studio 2005 and open the solution file for the sample application.**

The solution file is named QueueClientForDotNet.sln. The solution file is in the VBDOTNETQueueClient folder inside the Samples folder.

**2. Choose Start Debugging from the Debug menu.**

To build the solution, choose "Start Debugging" in the Debug menu. The application starts.

**3. Mark the MSMQ Private or Public queue option.**

You may use either Private or Public queues. The application default specifies private queues.

**4. Select the queue that will receive the message.**

The Local Private Queues drop-down list contains the MSMQ queues on your computer. Select the queue in the list you want to receive the eConnect XML message.

**5. Enter a message label.**

Use the message label to identify your message when viewing the contents of the queue.

**6. Select an eConnect XML document file.**

Click the ellipsis (...) button next to the XML Document Path box. A dialog box opens. Use the dialog box to find and open an XML file. Highlight the file you want to use and click Open.

**7. Review the XML document.**

The Browser View and Text View tabs display the contents of the XML file. You may edit the XML in the Text View tab. If you edit the XML, you will be prompted to save your changes to the file.

**8. Send the XML as a message to MSMQ.**

Click the Send to Queue button to send the eConnect XML message to the specified queue. A message box indicates the message was successfully sent to the specified queue. Click OK.

## How the sample application works

The sample application opens the specified file, reads the XML from the file, and writes the XML as a string to the Text View tab.

When you click the Send to Queue button, the application converts the XML in the Text View tab to an MSMQ message. To complete the message, the application assigns the value from the Message Label box to the message.

The application sends the message to the MSMQ queue specified in the Local Private Queue list. The application opens a dialog box that states the MSMQ message was successfully sent to the queue.

If an error occurs, the application opens a dialog box and displays the error message.

## How eConnect was used

The application uses an eConnect XML document as the basis for the MSMQ message. The sample application does not use any of the eConnect .NET assemblies.

# Glossary

## Application programming interface (API)

A set of functions or features you access to programmatically use or manipulate a software component or application.

## Back office

A financial management system. In an eConnect environment, this refers to Microsoft Dynamics GP.

## BizTalk adapter

A preconfigured BizTalk Application Integration Component (AIC) that allows BizTalk server to use eConnect.

## BizTalk server

A Microsoft platform that manages the exchange of data between applications.

## Business document

A well-formed XML document containing business data. This data may represent a sales order or other business information.

## COM+ object

A programming structure encapsulating data and functionality. In eConnect, this structure calls the SQL stored procedures (business objects).

## Connection string

A text representation of the initialization properties needed to connect to a data store.

## DCOM

A wire protocol that enables software components to communicate directly over a network.

## eConnect

A collection of tools, components, and APIs that provide programmatic integration with Microsoft Dynamics GP.

## eConnect XML document

A text document that describes Microsoft Dynamics GP data. The eConnect XML schema specifies the content and structure of data in the document.

## Extensible Stylesheet Language (XSL)

A high-level data manipulation language. XSL is used to manipulate XML documents.

## Front office

An application that communicates with the back office. Examples include customer relationship management systems, data warehouses, and web sites.

## Incoming Service

A Microsoft Windows service that monitors a queue for new eConnect XML documents. Valid documents are used to create, update, or delete records in Microsoft Dynamics GP.

## Microsoft message queuing (MSMQ)

A message infrastructure and development platform for creating distributed, loosely-coupled messaging applications.

## Middleware

Software that mediates between an application program and a network. It manages the interaction among disparate applications across the heterogeneous computing platforms.

## Outgoing Service

A Microsoft Windows service that publishes eConnect XML documents to a specified queue. The XML documents represent documents that were created, updated, or deleted in Microsoft Dynamics GP.

## Post stored procedure

A customized SQL stored procedure that runs immediately after an eConnect stored procedure.

## Pre stored procedure

A customized SQL stored procedure that runs immediately before an eConnect stored procedure.

## Replication Service

A Microsoft Windows service that copies selected data changes from Microsoft Dynamics GP to a specified target database.

## Schema

An XML file (with typical extension .XSD) that describes the syntax and semantics of XML documents using a standard XML syntax. An XML schema specifies the content constraints and vocabulary that compliant documents must accommodate.

## Serialization Flag

A boolean property that specifies whether to use or discard the value assigned to a property of an eConnect serialization class.

## Services

Microsoft Windows services are long-running applications that perform some system function. They typically do not display any user interface. eConnect uses several services for moving eConnect XML documents in and out of various message queues.

## Stored procedure

A group of Transact-SQL statements compiled into a single execution plan. The business logic for eConnect is contained in stored procedures.

## Transaction Requester

The Transaction Requester publishes eConnect XML documents to a queue. The XML documents represent Microsoft Dynamics GP documents.

## Trigger

A special class of SQL stored procedure that executes automatically when an update, insert, or delete statement is issued for a table or view.

## XML

A text-based format that uses markup tags (words surrounded by '<' and '>') to describe how a document is structured and the data it contains.





# Index

## A

- adapter, BizTalk 17
- Add methods 63-64
- AIC, *see* adapter
- API, *see* application programming interface
- application programming interface
  - architecture 16
  - defined 129
  - eConnect 16
- applications, *see* samples
- architecture
  - BizTalk 17
  - business objects 14
  - chapter 13-19
  - diagram 13
  - eConnect APIs 16
  - Transaction Requester 18
- assemblies
  - files 39
  - Microsoft.Dynaics.GP.eConnect.Misc Routines 59
  - Microsoft.Dynamics.GP.eConnect 41
  - Microsoft.Dynamics.GP.eConnect.Serialization 47
  - namespaces 40
  - references 39

## B

- back office, defined 129
- BizTalk
  - architecture 17
  - development 17
  - eConnect adapter 17
  - orchestration 17
- BizTalk adapter, defined 129
- BizTalk server, defined 129
- business document
  - see also* document, XML document
  - defined 129
- business logic
  - calling a stored procedure 85
  - custom stored procedure 88
  - custom XML nodes 87
  - customization options 85
  - customizing pre and post stored procedures 91
  - described 85
  - example 89
  - modifying 85
  - part 84-94
- Business Logic Extensions, chapter 91-94
- Business Logic Overview, chapter 85-86
- business objects
  - see also* stored procedures
  - architecture 14
  - business logic 85
  - described 14

- business objects (*continued*)
  - diagram 14
  - installed 15
  - SQL stored procedures 14
  - using 15

## C

- CDATA, using 30
- CollectionContainsDocuments method 64
- COM+ object, defined 129
- connection string, defined 129
- conventions, in documentation 3
- Create a Customer
  - chapter 113-114
  - sample application 113
- Create a Sales Order
  - chapter 115-116
  - sample application 115
- Custom XML Nodes, chapter 87-90
- Customizing the Transaction Requester, chapter 103-110

## D

- DCOM, defined 129
- document
  - automated numbering 28
  - create 27
  - create a customer 31
  - delete a customer address 32
  - described 26
  - retrieve customer data 32
  - rollbacks 26
  - sample files 28
  - serialization assembly 27
  - special characters 30
  - structure 25
  - structure diagram 25
  - updating 28
- documentation, symbols and conventions 3
- DocumentNumber property 66
- DocumentRollback class 62
  - Add methods 63-64
  - CollectionContainsDocuments method 64
  - methods 63-64
  - RollBackDocuments method 64
- DocumentTransactionType property 66
- DocumentType property 66

## E

- eConnect
  - add a .NET reference 39
  - API layer 14
  - APIs 16-17
  - architecture diagram 13
  - assemblies 39
  - benefits 7
  - BizTalk 17
  - business objects 14
  - COM+ component 16
  - data access layer 14

- eConnect (*continued*)
  - defined 129
  - described 7
  - example 8
  - getting started 10
  - MSMQ, described 17
  - .NET support, described 16
  - schema 16, 23
  - schema files 23
  - schema validation 23
  - serialization 47
  - stored procedures 14
  - support 3
  - transaction type, described 26
  - uses 7
  - using .NET namespaces 40
  - XML document
    - described 26
    - examples 31, 32
    - structure 25
- eConnect Assembly, chapter 41-45
- eConnect MSMQ Control 73
- eConnect Overview, part 6-19
- eConnect Requester Setup 97
  - see also* Requester Enabler/Disabler
  - SQL triggers 18
- eConnect Samples, part 112-128
- eConnect Schema, chapter 23-24
- eConnect Schema and XML Documents, part 22-35
- eConnect XML documents, chapter 25-30
- eConnect XML nodes, described 27
- eConnect\_Out
  - described 18
  - Outgoing Service 79
  - RequesterTrx 100
- eConnect\_Out\_Setup
  - described 18
  - install 103
  - Transaction Requester Service 103
- eConnectException class 44
- eConnectMethods class 41
  - eConnect\_EntryPoint method 42
  - eConnect\_Requester method 43
  - methods 41-43
- eConnectOut, described 54
- eConnectOutTemp, described 18
- eConnectProcessInfo, described 26
- EnumTypes class 44
  - ConnectionStringType enumeration 44
  - enumerations 44
  - SchemaValidationType enumeration 44
- ErrorState
  - post procedures 91
  - pre procedures 91
- ErrString
  - post procedures 91
  - pre procedures 91
- exception handling 44

extensible stylesheet language, *see* XSL

## F

front office, defined 129

## G

Get a Document Number  
chapter 121-123  
sample application 121

GetCustomerSpecficItemPriceForSellingU  
OFM method 68

GetItemPriceForAllPriceLevelsAndAllUn  
itsOfMeasure method 69

GetItemPricePerPriceLevelAndAllUnitsO  
fMeasure method 69

GetNextDocNumbers class 59  
GetNextGLJournalEntryNumber  
method 59

GetNextIVNumber method 60

GetNextPMPaymentNumber method  
60

GetNextPONumber method 60

GetNextPOPReceiptNumber method  
61

GetNextRMNumber method 61

GetNextSOPNumber method 62  
methods 59-62

RollBackDocumentList method 62

GetNextGLJournalEntryNumber method  
59

GetNextIVNumber method 60

GetNextPMPaymentNumber method 60

GetNextPONumber method 60

GetNextPOPReceiptNumber method 61

GetNextRMNumber method 61

GetNextSOPNumber method 62

GetNextSopNumber method 67

GetSopNumber class 66  
GetNextSopNumber method 67  
methods 67-68

RollBackSopNumber method 67

## I

Imports statement, namespaces 40

Incoming Service  
chapter 75-77  
create a document 75  
create a message 75  
defined 129  
described 73, 75  
example 76  
validation 75

installation  
sample XML documents 28  
schema files 23

## L

light bulb symbol 3

## M

margin notes 3

Microsoft .NET  
assemblies 39  
described 16  
framework 39  
namespaces 40  
references 39

Microsoft message queuing  
*see also* MSMQ  
defined 129

Microsoft SQL Server Management Studio  
92

Microsoft Visual Studio, required for .NET  
development 39

Microsoft.Dynamics.GP.eConnect  
assembly 41  
eConnectException class 44  
eConnectMethods class 41  
EnumTypes class 44

Microsoft.Dynamics.GP.eConnect.MiscRo  
utines  
assembly 59  
DocumentRollback class 62  
GetNextDocNumbers class 59  
GetSopNumber class 66  
PricingMethods class 68  
RollBackDocument class 65

Microsoft.Dynamics.GP.eConnect.Serializ  
ation

assembly 47  
serialization class 47  
middleware, defined 129

Miscellaneous Routines Assembly,  
chapter 59-69

MSMQ  
chapter 73-74  
described 17, 73  
eConnect MSMQ Control 73  
Incoming Service 73  
monitoring queues 73  
Outgoing Service 73  
retrieving messages 79  
sending a new message 75  
Windows services 73

MSMQ Development, part 72-81

MSMQ Document Sender  
chapter 127-128  
sample application 127

## N

namespaces  
adding 40  
described 40  
example 40  
Imports statement 40  
using statement 40

.NET

assemblies 39  
described 16  
namespaces 40  
references 39

.NET Development, part 38-69

.NET Development Overview, chapter  
39-40

nodes, *see* XML nodes 7

## O

Outgoing Service

*see also* Transaction Requester  
chapter 79-81  
default queue 79  
defined 129  
described 73, 79  
eConnect\_Out 79  
example 80  
publishing documents 79  
Requester Enabler/Disabler 79  
retrieving MSMQ messages 79  
Transaction Requester 97

Overview, chapter 7-11

## P

post procedure

defined 129  
described 91  
ErrorState 91  
ErrString 91  
files 91  
output paramaters 91  
RequesterTrx 100

pre procedure

defined 129  
described 91  
ErrorState 91  
ErrString 91  
example 92  
files 91  
output parameters 91  
RequesterTrx 100

PricingMethods class 68

GetCustomerSpecificItemPriceForSell  
ingUOFM method 68  
GetItemPriceForAllPriceLevelsAndA  
llUnitsOfMeasure 69  
GetItemPricePerPriceLevelAndAllUn  
itsOfMeasure 69  
methods 68-69

product support, for Microsoft Dynamics  
GP eConnect 3

## Q

queues

*see also* MSMQ  
retrieving MSMQ messages 79  
sending a new message 75

## R

Replication Service

defined 129  
described 14

Requester, *see* Transaction Requester

Requester Enabler/Disabler 79

RequesterTrx

described 100

RequesterTrx (*continued*)  
 examples 100  
 post procedure example 101  
 pre and post procedures 100  
 pre procedure example 100  
 requesting data, serialization example 54  
 Retrieve Data  
 chapter 125-126  
 sample application 125  
 RollBackDocument class 65  
 constructors 65-66  
 DocumentNumber property 66  
 DocumentTransactionType property 66  
 DocumentType property 66  
 methods 65-66  
 properties 66  
 RollBackDocument(transactionType, documentType, documentId, documentNumber) method 65  
 RollBackDocument(transactionType, documentType, documentNumber) method 65  
 RollBackDocument(transactionType, documentType, documentId, documentNumber) method 65  
 RollBackDocument(transactionType, documentType, documentNumber) method 65  
 RollBackDocumentList method 62  
 RollBackDocuments method 64  
 RollBackSopNumber method 67

## S

samples  
 Create a Customer 113  
 Create a Sales Order 115  
 Get a Document Number 121  
 MSMQ Document Sender 127  
 Retrieve Data 125  
 XML Document Manager 117  
 schema  
 defined 129  
 described 16, 23  
 install 23  
 uses 23  
 validation 16  
 schema validation, *see* validation  
 serialization 47  
 class 47  
 eConnectOut 54  
 example 48  
 Serialization Assembly, chapter 47-58  
 serialization flag, defined 129  
 serialization flags  
 described 53  
 use 53  
 serializaton flags, example 53  
 services  
*see also* Incoming Service, Outgoing Service, Replication Service

services (*continued*)  
 defined 129  
 described 73  
 special characters  
 described 30  
 in eConnect XML documents 30  
 SQL trigger  
 defined 129  
 eConnect Requester Setup 18  
 stored procedures  
*see also* business objects  
 business logic 85  
 custom 88  
 customizing pre and post procedures 91  
 defined 129  
 diagram 14  
 ErrorState 86  
 modifying 85  
 output parameters 91  
 using 85  
 support, for Microsoft Dynamics GP eConnect 3  
 symbols in documentation 3

## T

tables  
 eConnect\_Out 18  
 eConnect\_Out\_Setup 18  
 eConnectOutTemp 18  
 taRequesterTrxDisabler  
 described 109  
 example 109  
 technical support, for Microsoft Dynamics GP eConnect 3  
 Transaction Requester  
 architecture 18  
 components 97  
 custom  
 described 103  
 disabling 109  
 eConnect\_Out\_Setup 103  
 elements 103  
 example 105  
 install 103  
 query requirements 103  
 defined 129  
 described 14, 18  
 diagram 18  
 disabling 100  
 document tables 98  
 document types 97  
 eConnect\_Out table 18  
 eConnect\_Out\_Setup table 18  
 eConnectOutTemp table 18  
 example 18  
 Outgoing Service 97  
 part 96-110  
 publishing documents 97  
 RequesterTrx element 100  
 taRequesterTrxDisabler 109

transaction type, described 26  
 transactions  
 document 26  
 rollback 26  
 trigger, *see* SQL trigger

## U

using statement, namespaces 40  
 Using the Transaction Requester, chapter 97-102  
 utilities  
 eConnect MSMQ Control 73  
 eConnect Requester Setup 97  
 Requester Enabler/Disabler 79

## V

validation  
 Incoming Service 75  
 schema 16  
 Visual Studio, required for .NET development 39

## W

warning symbol 3  
 Windows services, *see* services

## X

XML  
 defined 129  
 described 8  
 XML document  
 automated numbering 28  
 create 27  
 create a customer 31  
 defined 129  
 delete a customer address 32  
 described 26  
 diagram 25  
 retrieve customer data 32  
 rollbacks 26  
 sample files 28  
 serialization 27  
 special characters 30  
 structure 25  
 updating 28  
 XML document examples, chapter 31-35  
 XML Document Manager  
 chapter 117-119  
 sample application 117  
 XML nodes  
 adding a custom node 87  
 elements 27  
 handling custom nodes 88  
 taRequesterTrxDisabler 109  
 XML schema  
 described 23  
 install 23  
 uses 23  
 XSD  
 files 23  
*see* schema  
 XSL, defined 129

