Microsoft Dynamics GP 2013 R2

eConnect Programmer's Guide

Publication date

May 2014

Contents

INTRODUCTION

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\Microsoft Dynamics\eConnect 12\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.
Margin notes summarize important information.	Margin notes call attention to critical information and direct you to other areas of the documentation where a topic is explained.

Convention	Description
Part 2, **XML Documents**	Bold type indicates a part name.
Chapter 1, "Overview"	Quotation marks indicate a chapter name.
Getting started	Italicized type indicates a section name.
`using System.IO;`	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 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

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?

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, a 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.

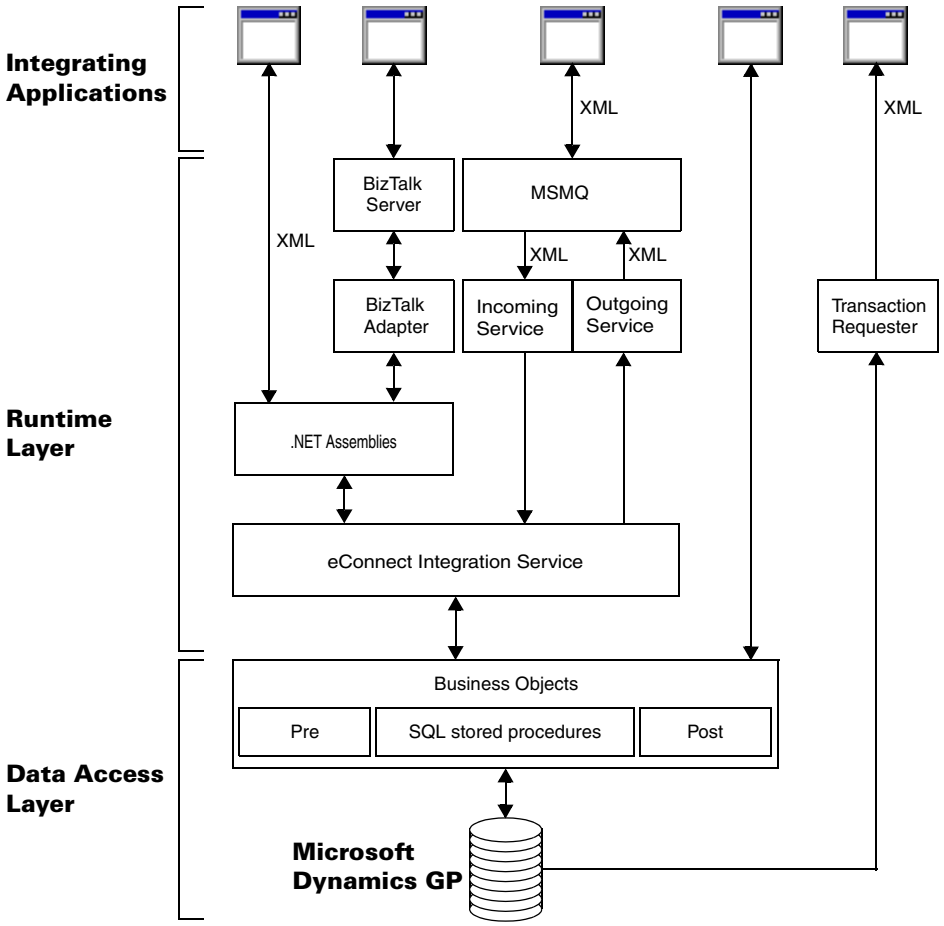
4. *Select the API for your project.*
   Once you select the eConnect API you intend to use, review the portion of the Programmer's Guide that discusses that API. For example, if you want to use eConnect with a .NET development project, review Part 3, **.NET Development** to learn how to add and use 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

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 runtime layer contains a collection of files, Windows services, and components that provide the application programming interfaces (API) you use to send or retrieve XML documents. The API enable your application to use the business objects. You must install the components of the runtime layer on the same computer as your integrating application. When you develop a new application, install and 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 you use to retrieve specified types of eConnect XML documents. Typically, the Transaction Requester uses the Outgoing Service to publish document to an MSMQ queue. The Transaction Requester includes the eConnect Requester Setup tool that you use to specify the types of transactions the Outgoing Service publishes as an XML document.

Business objects

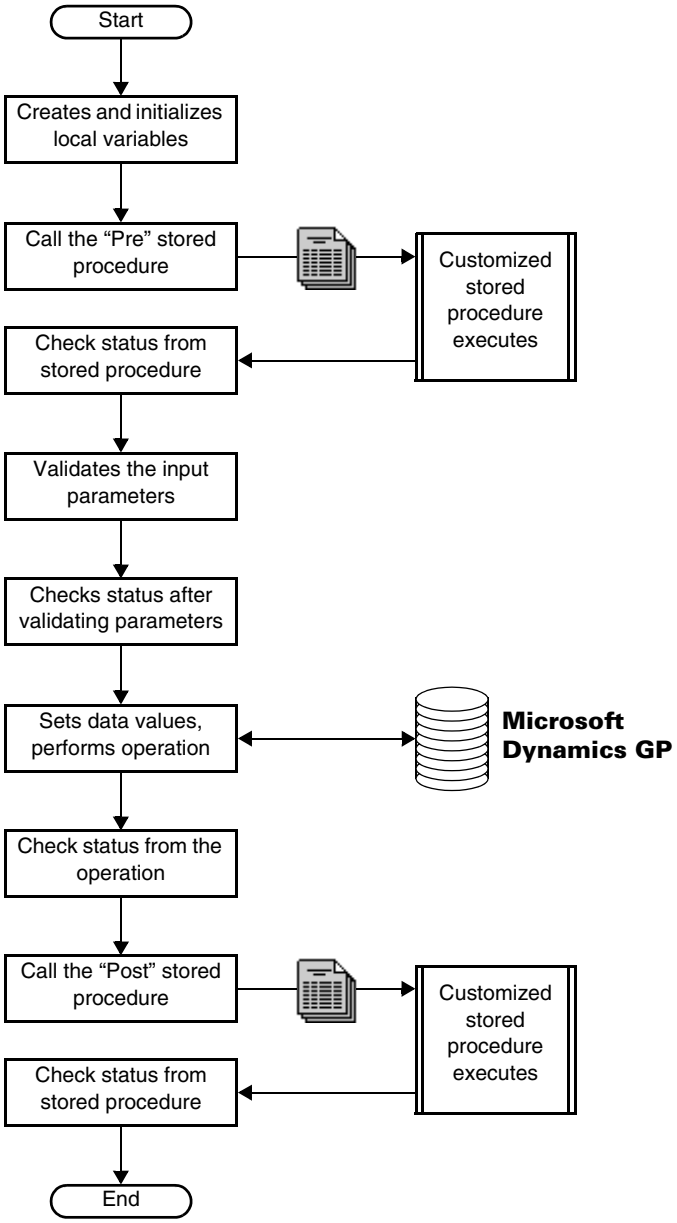
The stored procedures contain the business logic used by eConnect. Any integration that uses eConnect to query, create, update, or delete data from Microsoft Dynamics GP uses one or more of these stored procedures.

The eConnect business objects include the 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 edit or modify the business logic in an eConnect stored procedure. To enable the customization of business logic, eConnect includes a pre and post stored procedures for each eConnect stored procedure. The pre stored procedure runs immediately before the eConnect stored procedure, while the post stored procedure runs immediately after the eConnect stored procedure. To customize the business logic, add SQL queries and commands to one or both of these stored procedures.

For example, assume you want the eConnect stored procedure named taCreateTerritory to always populate a field with a specified value. To implement your customization, add custom SQL code to the stored procedure named taCreateTerritoryPre that populates the field with the specified value. Your custom code runs immediately before every execution of the taCreateTerritory stored procedure.

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



Notice how the business object validates data parameters, adds default values, and performs a status check after each event. The business object uses status checks to handle errors. When a status check detects an error, it immediately stops the stored procedure, initiates a rollback of the transaction, and returns an error message to the caller.

For example, a status check detects that an error occurred during the update operation of the business object. The status check immediately stops the stored procedure, rolls back the transaction, and then returns an error message. In this scenario, the business object never runs the post stored procedure.

The eConnect business objects are placed on your database server during the install of Microsoft Dynamics GP, and the creation of a new company database. There are two ways to access the business objects.

- Call individual business objects from the integrating application. However, direct calls require you to implement a connection to the database server, add security restrictions that prevent unauthorized access to data, and include proper error handling. For more information about direct calls, see *Calling the business objects*.

- Use one of the APIs that eConnect provides. The APIs offer a simpler approach to using the eConnect business objects from an integrating application. The types of eConnect APIs that are available are discussed in the sections that follow.

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 Windows service named eConnect for Microsoft Dynamics GP 2013 Integration Service. The eConnect Integration Service manages interaction between your application and the eConnect business objects. You must install the eConnect Integration Service in the Services on the computer you use to run your applicaiton. Refer to the eConnect Installation and Administration Guide for information about installing and configuring the eConnect Integration Service.

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 eConnect Integration Service:

Microsoft .NET

When you install the eConnect integration Service, the installer places two .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 Integration Service. The .NET assemblies enable 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 and .NET," 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 then uses the eConnect Integration Service 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.

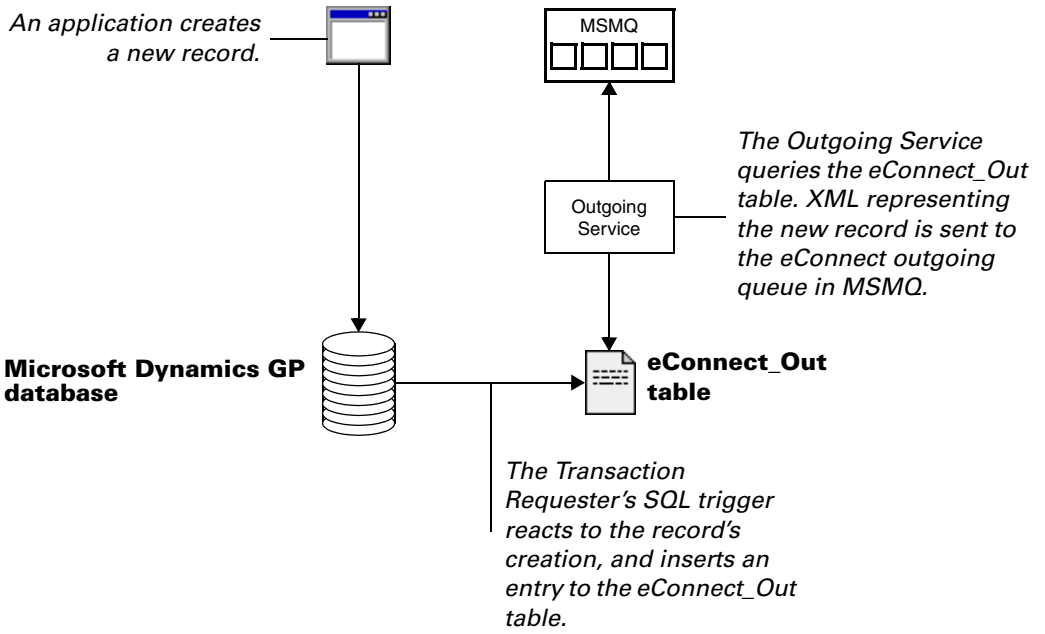
eConnect provides a BizTalk adapter that you can install on your BizTalk server. The BizTalk adapter allows you to use BizTalk to manage interaction with eConnect business objects.

The adapter supports the use of eConnect as a part of a BizTalk integration solution. BizTalk enables you to route messagess between applications, or to integrate applications with differing message formats.

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

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

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\Microsoft Dynamics\eConnect 12.0\XML Sample Documents\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\Microsoft Dynamics\eConnect 12.0\XML Sample Documents\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

An eConnect XML document is a text based data structure that represents a Microsoft Dynamics GP transaction or document. To submit or retrieve Microsoft Dynamics GP data, you send or receive data as an eConnect XML document.

You use an eConnect XML document to represent the data for a Microsoft Dynamics GP operation. For example, you have to create an XML document when you want to add or update a record in Microsoft Dynamics GP.

To help you create an eConnect XML document, you can use the classes in the Microsoft.Dynamics.GP.eConnect.Serialization .NET assembly. You can use the classes to create a document and then convert that document to XML. For information about the serialization classes, see Creating an eConnect document for a .NET project and Serializing an eConnect document object.

When you use eConnect to retrieve a Microsoft Dynamics GP document, the data appears as an eConnect XML document. To use the data, you get values from the XML document.

The following sections provide more detailed information about eConnect XML documents:

- *eConnect XML document structure*
- *Creating an eConnect XML document*
- *Using the eConnect XML document sample files*
- *Using eConnect to update existing data*
- *Automating document number assignment*
- *Special characters in eConnect XML documents*

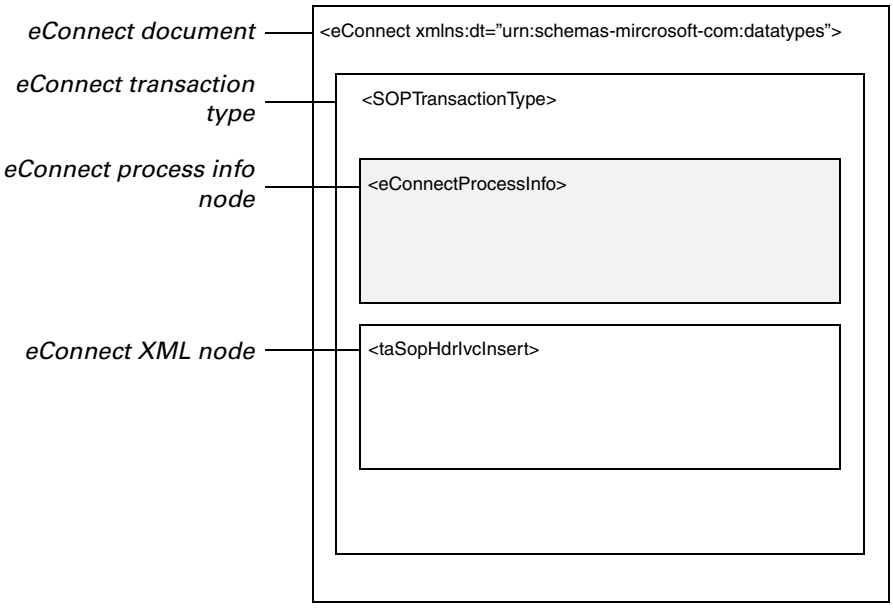
eConnect XML document structure

An eConnect XML document is collection of related XML components that represent a Microsoft Dynamics GP transaction or document. The XML components are in a parent/child hierarchy that define the data structure for the document type. The components and hierarchy for a document type is specified by an XML schema. For information about the schema, see *eConnect schema overview*.

The following table lists the basic components of an eConnect XML document:

Component type	Description
eConnect document	The root of the document. The eConnect document is the parent to one or more eConnect transaction type nodes.
eConnect transaction type	Specifies the type of the document or transaction. The transaction type is the parent to one or more eConnect XML nodes.
eConnect XML nodes	The parent to one or elements. The elements contain the data values for the document.

The following diagram illustrates a simple eConnect XML document. Notice how the eConnect document implements the parent/child relationships specified by the eConnect schema. Also notice how the eConnect transaction type is the parent to the <eConnectProcessInfo> and <taSopHdrIvcInsert> eConnect XML nodes.



eConnect document

An eConnect XML document always contains a single <eConnect> node. The <eConnect> node is the root node of the document. The <eConnect> node contains one or more transaction type nodes.

In addition, the <eConnect> node defines the scope of the SQL transaction for the specified operation. If any child component of the <eConnect> node fails, a rollback is performed that removes all the changes and updates specified by the XML document.

As a result, an eConnect XML document 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.

eConnect Transaction Type

An eConnect transaction type is an XML component that represents a Microsoft Dynamics GP document or operation. An eConnect XML document always includes one or more transaction type nodes. If you want your eConnect XML document to perform more than one operation, you have to add a transaction type node for each operation.

To create an eConnect XML document, you first add a transaction type node to the <eConnect> root node. For example, you add a <RMCustomerMasterType> transaction type node to a document that represents a new customer.

A transaction type node is the parent of one or more eConnect XML nodes. The eConnect schema files show the XML nodes you use to populate each transaction type. For information about the schema files, see *Using eConnect schema*.

eConnect XML node

An eConnect node is an XML component that contains a collection of elements. An element has a name and can contain a data value. You use the data values of the elements when you use eConnect to perform a Microsoft Dynamics GP operation. The eConnect schema files specify the elements contained in an XML node.

Some eConnect transaction types can include a collection of more than one XML node of a specified type. The schema identifies these nodes by appending "_Items" to the node name. For example, the <RMCustomerMasterType> can include more than one address for a customer. The <taCreateCustomerAddress_Items> node indicates that you can add more than one <taCreateCustomerAddress> XML node for a customer.

The following table describes the properties for an eConnect XML element:

Property	Description
Name	Specifies the name that identifies the element. You use the name to identify a specific element in a document.
Required	Specifies whether the element requires a value. If you do specify a value for the element, the transaction produces an error, In the schema files, a required element has both the minOccurs and maxOccurs property set to 1.
Type	Specifies the expected data type for the element value. For example, an element that contains text will have a data type of string.
Default	Specifies the default value for an element.
Length	Specifies the maximum data length of the value you can use to populate an element. The length is determined by the business object input parameter that handles the specified element.
Constraint	Specifies the data value or values that the element can accept. To populate the element, you must specify a value that satisfies the data constraint. If you specify a value that does not meet the data constraint, an error occurs and the transaction fails.

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 shows how to use the <eConnectProcessInfo> node to override the default eConnect connection string:

```
<eConnectProcessInfo>
    <ConnectionString>
        Integrated Security=SSPI;
        Persist Security Info=False;
        Initial Catalog=TWO;
        Data Source=machinename
    </ConnectionString>
</eConnectProcessInfo>
```

Creating an eConnect XML document

An eConnect XML document is a text-based representation of a Microsoft Dynamics GP document or transaction. As a result, you can use many XML or text tools to produce an eConnect XML document.

However, you must structure the XML in the document to comply with the hierarchy of nodes specified by the eConnect schema for the specified eConnect transaction type. If you do not provide the XML nodes and elements in the expected order, the document will be rejected.

To manually create an XML document, find the eConnect transaction type in the XML Schema Reference of the eConnect help file and add the XML nodes in the specified order. Next, find each XML node in the XML Node Reference of the eConnect Help file and add the XML elements in the specifed order.

You can also use the schema file for the specified transaction type to validate the structure of your XML document.

After you create the XML document, you can use MSMQ or the .NET assembly (Microsoft.Dynamics.GP.eConnect.dll) to submit the XML document to eConnect. Any XML document that you submit keeps the XML nodes and element in the order that you provided. MSMQ and the classes in the eConnect assembly do not change the order of the XML nodes and elements.

To create an eConnect document for a .NET project, you can use the eConnect serialization assembly. The Microsoft.Dynamics.GP.eConnect.Serialization assembly is a .NET assembly that you can use to programmatically produce eConnect XML documents. To learn more about how to use the serialization classes, see *Serializing an eConnect document object*.

The serialization assembly does automatically order the XML nodes and elements to comply with the eConnect schema requirements.

Using the eConnect XML document sample files

To assist you with developing and testing eConnect solutions, the eConnect SDK includes several files that contain sample XML documents. Typically, the eConnect install places these files in the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\XML Sample
Documents\Incoming
```

The XML files represent several types of eConnect documents and include test data. You can use the sample documents as an example for an XML document you want to use. You can also use the sample documents to test an eConnect application.

Using eConnect to update existing data

Many eConnect XML documents allow you to update existing Microsoft Dynamics GP 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

Microsoft Dynamics GP uses document numbers to uniquely identify each document. Many types of documents have document numbers. Each document type has a specified series of numbers and a new document is given the next available number from the series. When you use eConnect to create a document, you have to give that document a unique document number.

To simplify the numbering of new documents, several types of eConnect XML documents automate the assignment of the document number. The document gets the next available number at the time that it is created in Microsoft Dynamics GP.

The following table shows the eConnect documents and schemas that support the automated assignment of a document number:

Document Type	Schema name	XML element
General Ledger	GLTransaction	<JRNENTRY></JRNENTRY>
Inventory	IVInventoryTransaction	<IVDOCNBR></IVDOCNBR>
Inventory	IVInventoryTransfer	<IVDOCNBR></IVDOCNBR>
Purchase Order Processing	POPReceivings	<POPRCTNM></POPRCTNM>
Purchase Order Processing	POPTransaction	<PONUMBER></PONUMBER>
Purchasing	PMTransaction	<VCHNUMWK></VCHNUMWK>
Receivables	RMTransaction	<DOCNUMBR></DOCNUMBR>
Sales Order Processing	SOPTransaction	<SOPNUMBE></SOPNUMBE>

To have eConnect supply a 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 value of the element must remain empty.

The following XML example shows how to use automated number assignment for a new GL transaction document. Notice how the value of the <JRNENTRY> element is empty. When the business object encounters the empty element, the business object performs a query that gets the next available GL journal entry number. The business object uses the query result to populate the <JRNENTRY> element.

```
<taGLTransactionHeaderInsert>
    <BACHNUMB>TEST14</BACHNUMB>
    <JRNENTRY></JRNENTRY>
    <REFRENCE>General Transaction</REFRENCE>
    <TRXDATE>2007-01-21</TRXDATE>
    <RVRSNGDT>1900-01-01</RVRSNGDT>
```

```
    <TRXTYPE>0</TRXTYPE>
    <SQNCLINE>16384</SQNCLINE>
</taGLTransactionHeaderInsert>
```

However, SOP documents do not require an empty element. When you create a SOP document, eConnect assigns a document number even if the <SOPNUMBE> element is excluded from the XML document.

There are several types of SOP documents and each type uses a separate series of document numbers. To specify the type of a SOP document, you use the <SOPTYPE> and <DOCID> elements.

For example, you want to create a new sales order document. You add a <taSopHdrIvcInsert> XML node to you document. To get the correct type of document number, you populate the <SOPTYPE> with 2 and the <DOCID> with STDORD. You do not include the <SOPNUMBE> element. When you submit the document, it gets the next available document number for a sales order.

You can override the behavior that adds the document number. If you populate the document number element with a value, eConnect always uses the specified value in the Microsoft Dynamics GP document.

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>
            <REFRENCE>General Transaction</REFRENCE>
            <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>
```

You can also use a CDATA tag to remove data from a field. To clear data from a field, create an eConnect XML document that updates the targeted record. Use a CDATA tag that contains a blank space to populate the eConnect element that represents the field.

The following example uses a CDATA tag to clear the Short Name field of a customer. Notice how the CDATA tag contains a single blank space.

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <RMCustomerMasterType>
        <eConnectProcessInfo>
        </eConnectProcessInfo>
        <taUpdateCreateCustomerRcd>
            <CUSTNMBR>AARONFIT0001</CUSTNMBR>
            <SHRTNAME>
                <![CDATA[ ]]>
            </SHRTNAME>
            <UpdateIfExists>1</UpdateIfExists>
        </taUpdateCreateCustomerRcd>
    </RMCustomerMasterType>
</eConnect>
```

Chapter 5: XML Document Examples

This portion of the documentation contains XML examples that show how to use an eConnect document to create, retrieve, update and delete a record. The following sections include XML examples for each type of operation:

Create a customer

The following XML example uses an eConnect XML document to create a customer. Note the following characteristics of the document:

- The document contains a single <RMCustomerMasterType> transaction type schema. The schema specifies the XML nodes you use to create a customer.
- You use the <taUpdateCreateCustomerRcd> node to describe the customer record.
- Notice how the elements of the <taUpdateCreateCustomerRcd> node are populated with values that identify the new customer.

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <RMCustomerMasterType>
        <eConnectProcessInfo>
        </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>
            <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

The following XML example shows how to use an eConnect XML document to delete a customer address. Note the following characteristics of the document:

- The document contains a single <RMDeleteCustomerAddressType> transaction type schema. The schema specifies the XML node you use to identify an address record.
- Notice how the <RMDeleteCustomerAddress> schema includes a single <taDeleteCustomerAddress> XML node.
- Notice how the elements of the <taDeleteCustomerAddress> XML node are populated with values that identify the customer address you want to delete.

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

Retrieve a customer

The following XML example shows how to use an eConnect XML document to retrieve a customer record. Note the following characteristics of the document:

- The document contains a single <RQeConnectOutType> transaction type schema. The schema contains XML nodes you use to specify an existing customer record.
- Notice how the <RQeConnectOutType> schema includes <eConnectProcessInfo> and <eConnectOut> nodes. You use these nodes to specify the customer and how you want to retrieve the data for the customer record.
- Notice how the <Outgoing> element of the <eConnectProcessInfo> node is set to TRUE. You use TRUE to specify that this is a request for an existing record. Also notice how the <MessageID> element describes the record type.
- Notice that the <DOCTYPE> element of the <eConnectOut> node specifies the value Customer. The value identifies the type of document in the eConnect_Out_Setup table that you want to use to retrieve the record.

- Notice how the <OUTPUTTYPE> element specifies the value of 2. A value of 2 instructs eConnect to return complete customer record for the specified customer.

- Notice how the <INDEX1TO> and <INDEX1FROM> elements specify the ID of the customer you want to retrieve. To retrieve a customer, the INDEX1 column in the eConnect_Out_Setup table requires you to submit a CUSTNMBR value. To retrieve a single record, you use the same ID value to populate the <INDEX1TO> and <INDEX1FROM> elements.

- The values that populate the <FORLOAD>, <FORLIST>, and <ACTION> elements instruct eConnect to return the document to the caller and not create a record in the eConnect_Out table.

```
<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 to a sales order

The following XML example shows how to use an eConnect XML document to create a sales order. In addition, the document uses eConnect to query Microsoft Dynamics GP for a sales order number. The query result populates the <SOPNUMBE> element of the document. Note the following characteristics of the document:

- The document contains a single <SOPTransactionType> transaction type schema. The schema specifies the XML nodes you use to create a sales order.

- Notice how the document uses a <taSopLineIvcInsert> node to create a line item and a <taSopHdrIvcInsert> node for the header of the sales order. The <taSopLineIvcInsert_Items> node enables the document to include one or more line items.

- Notice how the <SOPNUMBE> element of the <taSopLineIvcInsert> and <taSopHdrIvcInsert> nodes do not specify a value. The element will be populated when the query obtains the new sales order number.

- Notice how the <SOPTYPE> element of the <taSopLineIvcInsert> and <taSopHdrIvcInsert> nodes are populated the with the value of 2. The value specifies that the document is a sales order.

- Notice that the <DOCID> element of the <taSopHdrIvcInsert> node is populated with the value STDORD. The value specifies the type of sales order you want to create.

- The values you use in the <SOPTYPE> and <DOCID> elements determine the type of Microsoft Dynamics GP document number that is requested for this sales order.

```
<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>
                <COMMNTID>TEST</COMMNTID>
                <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>
            <FREIGTBLE>1</FREIGTBLE>
            <MISCTBLE>1</MISCTBLE>
        </taSopHdrIvcInsert>
    </SOPTransactionType>
</eConnect>
```

PART 3: .NET DEVELOPMENT

Part 3: .NET Development

This section of the documentation discusses how to include eConnect in a .NET development project. The eConnect runtime includes .NET assemblies that enable you to view, create, update, and delete or void Microsoft Dynamics GP data from a Microsoft .NET solution. The following sections describe how to add and use the eConnect .NET assemblies:

- Chapter 6, ".NET Development Overview," introduces the eConnect assemblies and namespaces. You use these assemblies and namespaces to add eConnect to your .NET development project.

- Chapter 7, "eConnect and .NET," describes how to use classes in the Microsoft.Dynamics.GP.eConnect namespace. You use classes from the Microsoft.Dynamics.GP.eConnect namespace to send and request eConnect XML documents from your .NET solution.

- Chapter 8, "Serialization," describes how to use the classes in the Microsoft.Dynamics.GP.eConnect.Serialization namespace. You use the serialization classes to create .NET objects that represent eConnect XML documents.

- Chapter 9, "eConnect Integration Service," describes how to use the eConnect Integration Service from a .NET application. You use the service to send and request eConnect XML documents from your .NET solution.

Chapter 6: .NET Development Overview

This section of the documentation describes how to add an eConnect .NET assembly to a Microsoft Visual Studio development project. To access the eConnect business objects from a .NET solution, you must add one or more of the eConnect .NET assemblies to your development project. The following topics describe how to add an eConnect assembly and namespace to a project in Visual Studio:

eConnect and .NET

A .NET assembly is the fundamental building block of all .NET applications. An assembly includes the types and resources that produce a logical unit of functionality. In eConnect, an assembly is stored as a .dll file.

To add eConnect functionality to a .NET solution, you use one or more of the following assemblies:

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

The eConnect installer typically places these files in the directory c:\Program Files\Microsoft Dynamics\eConnect 12.0\Objects\Dot Net.

Each eConnect assembly contains an eConnect namespace. The values of each namespace matches the name of the assembly where it is found. For example, the Microsoft.Dynamics.GP.eConnect.dll assembly is where you find the Microsoft.Dynamics.GP.eConnect namespace.

Each eConnect namespace is a collection of related classes and enumerations. For example, the Microsoft.Dynamics.GP.eConnect namespace includes the eConnectMethods class as well as other classes and enumerations.

To use an eConnect class in a .NET development project, use Visual Studio to add a reference to the assembly that contains the namespace for that class.

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

Adding a reference

To add an eConnect class or enumeration to your .NET development project, use Visual Studio to add a reference to the eConnect assembly. To add a reference, complete the following steps:

1. **Open the Add Reference window.**
   From the Visual Studio Project menu, click Add Reference. The Add Reference window opens and displays the .NET tab.

2. **Find the assembly.**
   In the .NET tab, scroll the list of assemblies and click the eConnect assembly name. For example, click Microsoft.Dynamics.GP.eConnect.

3. **Add a reference to the assembly.**
   To add the reference, click OK. The specified assembly is added to the list of references for your Visual Studio project. The Add Reference window closes.

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 a effort to reduce the chance of a name collision. The eConnect namespaces are as follows:

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

You typically includes the namespace when you specify the type of an eConnect object. To demonstrate the use of a namespace, the following Visual Basic example instantiates a GetSopNumber object. Notice how the namespace and class name are used to specify the object type:

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

To simplify your code, use the Visual Basic **Imports** statement or C# **using** statement to specify the namespace from a referenced assembly. These statements eliminate the need to include the namespace when you specify an object type.

The following Visual Basic example uses the **Imports** statement to add the Microsoft.Dynamics.GP.eConnect namespace to the .vb file of a project. You typically add the Imports statement to the top of the file where you are using the members of that namespace:

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

The following C# example shows how to add a **using** statement to include namespace information in the .cs file of a project. You typically add the using statement to the top of the file:

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

After you use the Imports or using statements to include a namespace, you can use the eConnect class name to specify the type of an object. The following Visual Basic example shows how to import the Microsoft.Dynamics.GP.eConnect namespace and then instantiate a GetSopNumber object.

```
Imports Microsoft.Dynamics.GP.eConnect

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

If you are using the eConnect Integration Service you can use Imports or using statements for the service reference. The following Visual Basic example shows how to import the service reference for an application named VbTestApp.

```
Imports vbServiceTestApp.eConnectIntegrationService
```

The following C# code example shows how to add a using statement for a service reference to an application named ServiceTestApp.

```
using ServiceTestApp.eConnectIntegrationService;
```

Specifying configuration settings

To optimize the performance of your application, you might want to customize how your application interacts with the eConnect Integration Service. For example, if you use eConnect to send large eConnect XML documents, you might want to increase the settings that specify maximum message size.

To specify configuration settings, you add one or more key nodes to the <appSetting> node of the configuration file. The configuration file you update depends upon whether you are using a reference to the Microsoft.Dynamics.GP.eConnect assembly or a service reference to the eConnect Integration Service.

Reference type	Configuration file	Description
Reference	<ApplicationName>.exe.config	Add configuration keys to the application configuration file of your .NET application.
Service reference	Microsoft.Dynamics.GP.eConnect.Service.exe.config	Add configuration keys to the configuration file of the eConnect Integration Service. The configuration file for the service is typically found in the folder C:\Program Files\Microsoft Dynamics\eConnect 12.0\Service

To specify a configuration setting, you add a <key> node to the appSetting section of the configuration file. In the key node, you specify the name of the configuration setting and the value to use. The eConnect Integration Service enables you to supply custom values for the following configuration settings.

Name	Description
MaxReadQuotaSize	Specifies the maximum size of read messages.
MaxReceivedMessageSize	Specifies the maximum size of an XML document.
ProcTimeOut	Specifies the number of seconds to wait before a time out occurs
RequireProxyService	Specifies whether to use the identity of the logged on user or the service identity when accessing SQL server. Set the value to true when you want the application to use the service identity. If your application includes a service reference to eConnect, the application always uses the service identity to access SQL. The value of this configuration setting is ignored.
SendTimeout	Specifies the length of time to wait when connecting to the eConnect Integration Service. Used when the RequiredProxyService is set to True.
ServiceAddress	Specifies the URL of the eConnect Integration Service.
ServiceOperationTimeout	Specifies the length of time to wait for an operation to complete. Used when the RequiredProxyService is set to True.

Name	Description
TransactionIsolationLevel	Specifies the isolation level of the transaction. The typical transaction isolation level for eConnect is ReadUncommitted.
TransactionTimeoutSeconds	Specifies the length of time to wait for a SQL transaction to complete.

For information about the data type and default value of each configuration setting, see Configuration Class in the eConnect .NET Reference section of the eConnect help file.

If you add configuration settings to the configuration file of your application, you can also use the Configuration class in the Microsoft.Dynamics.GP.eConnect namespace to programmatically set configuration settings. Any configuration value set by the Configuration class override the value set in your application configuration file.

How to specify configuration setting for an eConnect enabled .NET application.

The following steps show how to add configuration settings to an eConnect .NET application. To begin, open the configuration file in a text editor.

Before you edit the Microsoft.Dynamics.GP.eConnnect.Service.exe.config file, make a copy of the file and store the file in a safe location. If you encounter problems with the service, use the saved copy to restore the existing service configuration.

Adding appSettings to the configuration file

To use custom configuration settings, add an <appSettings> node to the <configuration> of the application or service configuration file.

The following XML example shows how to add appSettings to an application configuration file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    </appSettings>
</configuration>
```

Adding configuration values

To specify custom configuration settings, add individual key and value information for each setting.

The following XML example adds configuration information for the SendTimeout, MaxReadQuotaSize, and MaxReceivedMessageSize. Notice how MaxReadQuotaSize and MaxReceivedMessageSize use the maximum value for those configuration settings.

```
<appSettings>
    <add key="SendTimeout" value="60" />
    <add key="MaxReadQuotaSize" value="2147483647" />
    <add key="MaxReceivedMessageSize" value="2147483647" />
</appSettings>
```

Saving the configuration

To use your custom configuration settings, save the configuration file and restart your application.

If you edited the Microsoft.Dynamics.GP.eConnnect.Service.exe.config file. Stop and restart the eConnect Integration Service before restarting your application.

Tracing an eConnect .NET application

When using eConnect with your .NET application, you might want to use tracing to monitor the execution of your application. The classes in the Microsoft.Dynamics.GP.eConnect namespace support .NET tracing.

To enable tracing, you add the eConnectTraceSource and trace listeners to the application configuration file. The configuration file you update depends upon whether you are using a reference to the Microsoft.Dynamics.GP.eConnect assembly or a service reference to the eConnect Integration Service.

Reference type	Configuration file	Description
Reference	<ApplicationName>.exe.config	Add the trace source and listeners to the application configuration file of your application.
Service reference	Microsoft.Dynamics.GP.eConnect.Service.exe.config	Add the trace source and listeners to the configuration file of the eConnect Integration Service. The configuration file for the service is typically found in the folder C:\Program Files\Microsoft Dynamics\eConnect 12.0\Service

When you add eConnectTraceSource to the configuration file, the eConnect .NET interface generates trace messages that can be collected and recorded in specified logs, or files.

To collect and record tracing information, you must specify the type of output you want to record. The eConnect trace source has a property named switchValue that you use to specify the type of tracing data to collect. Set switchValue to one of the following values.

switchValue	Description
Error	All exceptions are recorded.
Information	Records important and successful milestones of application execution regardless of whether the application is working properly or not.
Off	Disable tracing for the application.
Verbose	Records events that mark successful milestones. Includes low level events for both user code and the service. Useful for debugging or for application optimization.

Tracing in eConnect supports the same trace listeners as other .NET applications. A trace listener is the mechanism that collects and records trace information. To specify where trace information is recorded, use one of the following trace listeners.

Name	Description
ConsoleTraceListener	Directs tracing output to either the standard output or the standard error stream.
DelimitedListTraceListener	Directs tracing output to a text writer, such as a stream writer, or to a stream, such as a file stream. The trace output is in a delimited text format that uses a specified delimiter.
EventLogTraceListener	Directs tracing output to a specified event log.

Name	Description
TextWriterTraceListener	Directs output to an instance of the TextWriter class or to anything that is a Stream class. It can also write to the console or to a file.
XmlWriterTraceListener	Directs tracing output as XML-encoded data to a TextWriter or to a Stream, such as a FileStream. Typically, output is recorded in an XML file.

The most commonly used trace listeners are the TextWriterTraceListener and the XmlWriterTraceListener. These listeners record trace information to a file. For more information about these or the other trace listeners, refer to the documentation for the .NET framework.

For information about how to get tracing information from the eConnect Integration Service, see eConnect Integration Service tracing in the Troubleshooting section of the eConnect Installation and Administration Guide.

How to enable tracing for an eConnect enabled .NET application.

The following steps show how to add tracing to an eConnect .NET application. To begin, open the configuration file in a text editor.

Before you edit the Microsoft.Dynamics.GP.eConnnect.Service.exe.config file, make a copy of the file and store the file in a safe location. If you encounter problems with the service, use the saved copy to restore the existing service configuration.

Adding a trace source

To enable eConnect tracing, add a trace source named eConnectTraceSource to the system. diagnostics section of the configuration file. Use the switchValue attribute of the source to specify the type of information you want logged. The eConnectTraceSource logs trace information that is available in the classes and methods of the Microsoft.Dynamics.GP.eConnect namespace.

Typically, the Microsoft.Dynamics.GP.eConnect.Service.exe.config file includes the eConnectTraceSource. To enable tracing, change the switchValue of the eConnectTraceSource to Error, Information, or Verbose.

The following XML example adds the eConnectTraceSource to an application configuration file. Notice how switchValue is set to Verbose.

```
<system.diagnostics>
    <sources>
        <source name="eConnectTraceSource" switchValue="Verbose">
    </source>
    <trace autoflush="true" />
</system.diagnostics>
```

Adding listeners

Add the listener you want to use to log the trace results. Add a <sharedListeners> node to the <system.diagnostics> section of you application file. Add and configure the listeners you want to use.

Typically, the Microsoft.Dynamics.GP.eConnect.Service.exe.config file includes listeners. Update the initializeData attribute of the eConnectTextTracelistener or the eConnectXmlTracelistener to specify a folder and file name for the trace file.

The following XML example shows how to add a shared listener to your application configuration file. Notice how a TextWriterTraceListener is specified. The intializeData attribute specifies the location and name of the log file.

```
<sharedListeners>
    <add initializeData="c:\TEMP\eConnect.log"
        type="System.Diagnostics.TextWriterTraceListener,
        System, Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089"
        name="eConnectTextTracelistener">
            <filter type="" />
    </add>
</sharedListeners>
```

Adding a listener to the source

To use the listener, add the listener to the listeners of the eConnectTraceSource. Specify the name of the listener you added to the SharedListeners.

Typically, the Microsoft.Dynamics.GP.eConnect.Service.exe.config file includes a listener named eConnectTextTracelistener with the eConnectTraceSource. You do not need to make any changes to use this listener. To use the eConnectXmlTracelistener, add that listener to listeners node of the eConnectTraceSource.

The following XML example adds the shared listener named eConnectTextTracelistener to the eConnectTraceSource in an application configuration file.

```
<source name="eConnectTraceSource" switchValue="Verbose">
    <listeners>
        <add name="eConnectTextTracelistener">
            <filter type="" />
        </add>
    </listeners>
</source>
```

Collecting trace data

Save the changes to the application configuration file. If you made change to the Microsoft.Dynamics.GP.eConnect.Service.exe.config, stop and restart the eConnect Integration Service.

To generate trace data, start your application and perform the operations you want to trace. To view the result, open the specified log or file and review the trace information.

Stopping the trace

When you are done collecting trace information. You should disable the eConnectTraceSource. In the configuration file, set the switchValue attribute of the eConnectTraceSource to Off.

```
<source name="eConnectTraceSource" switchValue="Off">
```

Example

The following XML example shows the system.diagnostics node from an application configuration file. Notice how eConnectTraceSource has been added

with a switchValue set to Verbose. Also notice how two listeners have been specified for eConnectTraceSource. The eConnectTextTraceListener logs trace information as text to a file name eConnect.log. The eConnectXmlTracelistener logs trace information as XML in a file named eConnectLog.xml.

```
<system.diagnostics>
    <sources>
        <source name="eConnectTraceSource" switchValue="Verbose">
            <listeners>
                <add name="eConnectXmlTracelistener">
                    <filter type="" />
                </add>
                <add name="eConnectTextTracelistener">
                    <filter type="" />
                </add>
            </listeners>
        </source>
    </sources>
    <sharedListeners>
        <add initializeData="c:\TEMP\eConnect.log"
            type="System.Diagnostics.TextWriterTraceListener,
            System, Version=2.0.0.0, Culture=neutral,
            PublicKeyToken=b77a5c561934e089"
            name="eConnectTextTracelistener">
                <filter type="" />
        </add>
        <add initializeData="c:\TEMP\eConnectLog.xml"
            type="System.Diagnostics.XmlWriterTraceListener,
            System, Version=2.0.0.0, Culture=neutral,
            PublicKeyToken=b77a5c561934e089"
            name="eConnectXmlTracelistener">
                <filter type="" />
        </add>
    </sharedListeners>
    <trace autoflush="true" />
</system.diagnostics>
```

Chapter 7: eConnect and .NET

This section of the documentation describes how to use the classes in the Microsoft.Dynamics.GP.eConnect assembly. The following topics show how to use these classes to enable your .NET application to create, update, retrieve, and delete or void Microsoft Dynamics GP data.

Microsoft.Dynamics.GP.eConnect

To use eConnect to retrieve or update data from Microsoft Dynamics GP, add the Microsoft.Dynamics.GP.eConnect assembly and namespace to your .NET project.

To use the classes in the Microsoft.Dynamics.GP.eConnect namespace, you must add a reference to the Microsoft.Dynamics.GP.eConnect assembly to your .NET project. For information about how to add a reference, see Adding a reference.

The Microsoft.Dynamics.GP.eConnect namespace includes the following classes:.

Class	Description
Configuration	Specifies the properties you use to configure an eConnect application.
DocumentRollback	Enables you to create a collection of Microsoft Dynamics GP document numbers. You use the collection with the RollBackDocumentList method of the GetNextDocNumbers class to return unused numbers.
eConnectException	The eConnectException class enables you to catch or throw eConnect-specific errors. If a method in the eConnectMethods class encounter an error, the method throws an eConnectException.
eConnectMethods	The eConnectMethods class allows you to send and receive XML that represent eConnect documents.
EnumTypes	The EnumTypes class defines enumerations that you use to connect to the eConnect business objects. You typically use these enumerations as parameters for the eConnect_EntryPoint and eConnect_Requester methods of the eConnectMethods class.
GetNextDocNumbers	Enables you to get the next available number for several types of Microsoft Dynamics GP documents.
GetSopNumber	Enables you to retrieve the next available number for a sales document. This class also allows you to return a SOP number that was retrieved but not used.
RollBackDocument	Specifies a single Microsoft Dynamics GP document number. Use a RollBackDocument object when you need access to an individual record in the arraylist produced by the RollBackDocuments method of the DocumentRollback class.

Using CreateEntity for new records

The eConnectMethods class enables your .NET application to add new data records in Microsoft Dynamics GP. To create a record, use the following methods:

Method	Description
CreateEntity	Creates a record using information from an eConnect XML document. Use CreateEntity to add data entities like customers or vendors. You specify the type of record to create with a parameter that represents an eConnect XML document. If the operation succeeds, the method returns True as a boolean value.
CreateTransactionEntity	Create a transaction using information from an eConnect XML document. Use CreateTransactionEntity for transaction documents like sales orders or inventory transactions. You specify the type of transaction to create with a parameter that represents an eConnect XML document. If the operation succeeds, the method returns an XML string that represents the eConnect XML document that was created.

The eConnectMethods class also includes methods to update or delete existing records in Microsoft Dynamics GP. To use these methods, you must supply the eConnect XML document for the specified operation and document type.

To create, update, and delete or void a record in Microsoft Dynamics GP, you must create an XML string for your eConnect XML document. The following are the most common techniques for creating this string.

- Construct a string that contains the eConnect XML schema and node tags that are required for your operation. In addition, include the XML tags and values for the XML elements that you want to populate with data.
- Load text from a file or other data store that contains an existing eConnect XML document into a .NET XmlDocument object. Use the OuterXML property of the XmlDocument class to convert the XML document to a string.

To work with eConnect, the text that is loaded into the XmlDocument object must be a valid eConnect XML document.

- Use classes from the eConnect Serialization namespace to construct an object that represents an eConnect XML document. Use a .NET XmlSerializer and a XmlDocument object to convert your document object to a string.

For more information about eConnect XML documents, see *eConnect XML document structure*.

The following steps show how to use the CreateEntity method to create a record using XML from a textbox control.

How to add a new record to Microsoft Dynamics GP.

Instantiate an eConnectMethods object

To begin, instantiate an eConnectMethods object. The following Visual Basic example shows how to instantiate an eConnectMethods object:

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

Load the eConnect XML document

The following Visual Basic example shows how to use the XML text from a textbox control as the source of the eConnect XML document. Notice how text from the specified control is loaded into a .NET XmlDocument object.

```
'Load the text from the textbox control into an XmlDocument object
Dim xmlDoc As XmlDocument
xmlDoc.LoadXml(XmlDoc_TextBox.Text)
```

Create an eConnect connection string

The CreateEntity method requires an eConnect connection string. You use the connection string to specify the Dynamics GP data server and the company database.

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

The following Visual Basic example shows how to creates a connection string:

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

Submit the eConnect document

Use the CreateEntity method to submit the document to the eConnect business objects. To call the CreateEntity method, supply parameters that specify the connection string, and the eConnect XML document string.

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

The following Visual Basic example uses the CreateEntity method to submit an eConnect XML document. Notice the following:

- The ConnectionString parameter specifies the Dynamics GP data server and the company database.
- The XML string parameter is created using the OuterXml property of the XmlDocument object.

```
'If eConnectResult is TRUE, the XML document was successfully submitted
Dim eConnectResult As Boolean
eConnectResult = eConnectObject.CreateEntity(ConnectionString, _
    xmlDoc.OuterXml)
```

Retrieving XML documents with GetEntity

The eConnectMethods class includes a method named GetEntity that enables you to retrieve data from Microsoft Dynamics GP. The GetEntity method gives you programmatic access to the eConnect Transaction Requester. You use the GetEntity method to get a string that represents a Transaction Requester XML document.

The Transaction Requester limits the types of documents you can retrieve with the GetEntity method. To retrieve a document, that document type must be specified in the eConnect_Out_Setup table of your company database. To see the default set of document types, see *Requester document types*.

To specify the data to retrieve, create an XML string that represents a Transaction Requester query document. The following are the most common techniques for creating a string that represents a Transaction Requester query document.

- Construct a string that contains the XML tags and values required by the Transaction Requester for the specified document type.
- Load text from a file or other data store that contains an existing Transaction Requester query into a .NET XmlDocument. Use the OuterXML property of the XmlDocument class to convert the XML document to a string.

To work with eConnect, the text that is loaded into the XmlDocument object must be a valid Transaction Requester query. For more information about the structure of a Transaction Requester query document, see Retrieve a customer.

- Use the eConnectOut, RQeConnectOutType, and eConnectType classes from the eConnect Serialization namespace to construct an object that represents a Transaction Requester query. Use a .NET XmlSerializer and a XmlDocument object to convert the query document to a string.

The following steps show how to use the GetEntity method to retrieve a transaction requester document for a customer.

How to retrieve an eConnect Transaction Requester XML document.

Instantiate an eConnectMethods object

To begin, instantiate an eConnectMethods object. The following Visual Basic example shows how to instantiate an eConnectMethods object:

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

Create a Transaction Requester document

The following Visual Basic example shows how to use classes from the eConnect serialization namespace to construct an object that requests a single customer record. Notice how the INDEX1FROM and INDEX1TO fields specify the ID of the customer, the OUTPUTTYPE field specifies the customer master document, and the FORLIST field specifies to return the document to the caller. Also notice how the .NET XmlSerializer is used to load the document object into a .NET XmlDocument.

```
'**Create an eConnect requestor document that specifies a single customer**
'Create the requestor node
Dim myRequest As New eConnectOut()
With myRequest
    .DOCTYPE = "Customer"
    .OUTPUTTYPE = 1
    .INDEX1FROM = "AARONFIT0001"
    .INDEX1TO = "AARONFIT0001"
    .FORLIST = 1
End With
```

```
'Create the requestor schema document type
'Since the eConnect document requires an array, create an
‘array of RQeConnectOutType
Dim econnectOutType() As RQeConnectOutType = _
    New RQeConnectOutType(0) {New RQeConnectOutType}
econnectOutType(0).eConnectOut = myRequest

'Create the eConnect document type
Dim eConnectDoc As New eConnectType()
eConnectDoc.RQeConnectOutType = econnectOutType

'**Serialize the eConnect document**
'Create a memory stream for the serialized eConnect document
Dim memStream As New MemoryStream()

'Create an Xml Serializer and serialize the eConnect document
‘to the memory stream
Dim serializer As New XmlSerializer(GetType(eConnectType))
serializer.Serialize(memStream, eConnectDoc)

'Reset the position property to the start of the buffer
memStream.Position = 0

'**Load the serialized Xml into an Xml document**
Dim xmldoc As New XmlDocument()
xmldoc.Load(memStream)
```

Create an eConnect connection string

The GetEntity method requires that you supply an eConnect connection string. Use the connection string to specify the Dynamics GP data server and the company database.

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

The following Visual Basic code example shows how to creates a connection string:

```
'Create an eConnect connection string
Dim connectionString As String
connectionString = "data source=localhost; initial catalog=TWO; “ _
    & “integrated security=SSPI; persist security info=False; “ _
    & “packet size=4096"
```

Retrieve the XML document

Use the GetEntity method to populate a string that represents the XML document that the Transaction Requester retrieved. To call the method, supply parameters that specify the connection string, and an XML string that specifies the data you want to retrieve.

The following Visual Basic example uses the GetEntity method to retrieve a customer XML document. Notice the following:

- The ConnectionString parameter specifies the Dynamics GP data server and the company database.

- The XML string parameter is created using the OuterXml property of the XmlDocument object.

```
'Retrieve the specified document
Dim customerDoc = eConnectMethods.GetEntity(connectionString, _
    EnumTypes.ConnectionStringType.SqlClient, xmldoc.OuterXml)
```

Retrieving a document number

The GetNextDocNumbers class includes methods that enable you to retrieve several types of Microsoft Dynamics GP document numbers. Typically, you use the GetNextDocNumbers class when you use eConnect to create a new record in Microsoft Dynamics GP.

How to retrieve Microsoft Dynamics GP document numbers.

The following Visual Basic example shows how to use the GetNextDocNumbers class to get document numbers for several types of documents.

```
'Use the connection string to connect to the TWO database
‘ on the local computer
Dim connectionString As String
connectionString = "Data Source=localhost;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"

'Instantiate a GetNextDocNumbers object
Dim getDocNumbers As New GetNextDocNumbers()

'Use the GetNextDocNumbers object to retrieve a series of Microsoft Dynamics
‘ GP document numbers
Dim poNumber = getDocNumbers.GetNextPONumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, connectionString)
Dim invNumber = getDocNumbers.GetNextIVNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, _
    GetNextDocNumbers.IVDocType.IVAdjustment, connectionString)
Dim rmNumber = getDocNumbers.GetNextRMNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, _
    GetNextDocNumbers.RMPaymentType.RMReturn, connectionString)
Dim pmNumber = getDocNumbers.GetPMNextVoucherNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, connectionString)
Dim sopNumber = getDocNumbers.GetNextSOPNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, "STDINV", _
    GetNextDocNumbers.SopType.SOPInvoice, connectionString)
```

Returning a document number

After you get a Microsoft Dynamics GP document number, you might find that you do not need to use that number. Typically, you return an unused document number so the number can be used later with another document.

To return a document number, you use the DocumentRollback class. The DocumentRollback class enables you to build a list of document numbers. You use this list with the RollBackDocumentList method of GetNextDocNumbers class to return unused document numbers to Microsoft Dynamics GP.

How to restore unused document numbers.

The following steps show how to retrieve a collection of document numbers and how to return those numbers to Microsoft Dynamics GP. You will also see how to view the individual members of a DocumentRollback arraylist.

Create the connection string

To retrieve and return document numbers, you need a connection string that specifies the data server and the company database.

The following Visual Basic example shows how to create a connection string.

```
'Create a connection string that connects to the TWO database
‘ on the local computer
Dim connectionString As String
connectionString = "Data Source=localhost;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"
```

Get the document number

To retrieve a document number, use the GetNextDocNumbers class.

The following Visual Basic example uses the GetNextDocNumbers class to retrieve several types of Microsoft Dynamics GP document numbers.

```
'Instantiate a GetNextDocNumbers object
Dim getDocNumbers As New GetNextDocNumbers()

'Use the GetNextDocNumbers object to retrieve a series of
‘ Microsoft Dynamics GP document numbers
Dim poNumber = getDocNumbers.GetNextPONumber( _
GetNextDocNumbers.IncrementDecrement.Increment, connectionString)
Dim invNumber = getDocNumbers.GetNextIVNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, _
    GetNextDocNumbers.IVDocType.IVAdjustment, connectionString)
Dim rmNumber = getDocNumbers.GetNextRMNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, _
    GetNextDocNumbers.RMPaymentType.RMReturn, connectionString)
Dim pmNumber = getDocNumbers.GetPMNextVoucherNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, connectionString)
Dim sopNumber = getDocNumbers.GetNextSOPNumber( _
    GetNextDocNumbers.IncrementDecrement.Increment, "STDINV", _
    GetNextDocNumbers.SopType.SOPInvoice, connectionString)
```

Specify the document number to return

To specify the document numbers to return, create a DocumentRollback object. Use the Add method of the DocumentRollback class to build a list of unused document numbers. To use the Add method, you must specify the document type and the document number. The Add method creates a RollBackDocument object and adds that object to an internal arraylist.

The following Visual Basic example shows how to use the Add method of the DocumentRollback class. Notice how the TransactionType enumeration specifies the type of each document.

```
'Instantiate a DcoumentRollback object
Dim docRollBack As New DocumentRollback()

'Add the document numbers to the DocumentRollback object
docRollBack.Add(TransactionType.POP, poNumber)
docRollBack.Add(TransactionType.IVTrans, invNumber)
```

```
docRollBack.Add(TransactionType.RM, rmNumber)
docRollBack.Add(TransactionType.PM, pmNumber)
docRollBack.Add(TransactionType.SOP, sopNumber)
```

Search the document number arraylist (optional)

To find information about each document number in a DocumentRollback object, search the arraylist of that object. Since each member of the arraylist is a RollBackDocument object, you can use the document number, document type, and other properties to find important information about each document number in the collection.

The following Visual Basic examples shows how to search for document numbers in a DocumentRollback arraylist. Notice how the ArrayList, the ForEach loop, and the RollBackDocument type are used to create a list of document numbers.

```
'**Iterate through the document numbers to return**
'Determine whether the document rollback object contains any document numbers
If docRollBack.CollectionContainsDocuments() = True Then

    'Instantiate an arraylist
    Dim aList As New ArrayList()

    'Load the arraylist with the information about the document numbers
    aList = docRollBack.RollBackDocuments()

    'Instantiate a list of strings
    Dim returns As New List(Of String)

    'Iterate the array list and inspect the document numbers
    'Use the RollBackDocument type to provide access to the DocumentNumber
    ' property of each object in the arraylist
    For Each number As RollBackDocument In aList

        returns.Add(number.DocumentNumber)

    Next

End If
```

Return the document number to Microsoft Dynamics GP

To return one or more document numbers, use the RollBackDocumentList method of the GetNextDocNumbers class. The RollBackDocumentList method requires you to provide an arraylist that contains one or more Microsoft Dynamics GP document numbers.

To provide the arraylist, use the RollBackDocuments method of your DocumentRollback object. The method returns an arraylist that includes the document numbers you previously added to your DocumentRollback object..

The following Visual Basic example shows how to use the RollBackDocumentList method of the GetNextDocNumbers class. Notice how the RollBackDocuments method of the DocumentRollback object supplies the arraylist parameter for the RollBackDocumentList method.

```
'Use the RollBackDocumentList method of the GetNextDocNumbers
‘ object to return the unused document numbers
'Use the RollBackDocuments method of the DocumentRollback object
‘ to specify the document numbers

getDocNumbers.RollBackDocumentList(docRollBack.RollBackDocuments(), _
    connectionString)
```

Retrieving a sales document number

To retrieve a Microsoft Dynamics GP sales document number, use the GetNextSopNumber method of the GetSopNumber class. To use the GetNextSopNumber method you specify the type of sale document, the document ID, and an eConnect connection string. The method returns the next available document number for the specified type of sales document.

How to retrieve a sales document number.

The following Visual Basic example shows how to use the GetNextSopNumber method of the GetSopNumber class. Notice how the GetNextDocNumbers.SopType enumeration specifies the type of sales document. Also notice the use of STDINV as the document ID.

```
'Create a connection string that connects to the TWO database
‘ on the local computer
Dim connectionString As String
connectionString = "Data Source=localhost;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"

'Instantiate a GetSopNumber object
Dim getSopNumber As New GetSopNumber

'Get the next available sales invoice document number
Dim salesInvoiceNumber As String
salesInvoiceNumber = getSopNumber.GetNextSopNumber( _
    GetNextDocNumbers.SopType.SOPInvoice, _
    "STDINV", connectionString)
```

Returning a sales document number

To return an unused Microsoft Dynamics GP sales document number, you use the RollBackSopNumber method of the GetSopNumber class. To use the RollBackSopNumber method you specify the document number, the type of sale document, the document ID, and an eConnect connection string.

How to restore an unused sales document number.

The following Visual Basic example shows how to use the RollBackSopNumber method of the GetSopNumber class to return an unused sales invoice document number. Notice how the GetSopNumber class is used to first retrieve the sales invoice document number. Also notice the use of STDINV as the document ID when the document is retrieved and when it is returned.

```
'Create a connection string that connects to the TWO database on the local
‘ computer
Dim connectionString As String
connectionString = "Data Source=localhost;Integrated Security=SSPI;" _
    & "Persist Security Info=False;Initial Catalog=TWO;"

'Instantiate a GetSopNumber object
Dim getSopNumber As New GetSopNumber

'Get the next available sales invoice document number
Dim salesInvoiceNumber As String
salesInvoiceNumber = getSopNumber.GetNextSopNumber( _
    GetNextDocNumbers.SopType.SOPInvoice, _
    "STDINV", connectionString)

'Return the sales invoice number to Microsoft Dynamics GP
Dim returnSucceeded = getSopNumber.RollBackSopNumber(salesInvoiceNumber, _
    GetNextDocNumbers.SopType.SOPInvoice, _
    "STDINV", connectionString)
```

eConnect exception handling

The eConnectException classes produces eConnect-specific error information. You typically use the eConnectExceptions in the following situations.

- You add code to your .NET project that detects eConnect-specific errors. You then add code that specifies the actions to take when an eConnect error occurs.

- You use the eConnectException class to create and throw a new exception object. For example, you use the errMessage parameter of the eConnectException class to add error information that specifies where the error occurred in your .NET application.

If you use the classes in Microsoft.Dynamics.GP.eConnect, you should include code to catch and handle eConnect exceptions from the methods of those classes. The most common exception handling technique is the Try/Catch block. For example, you place a Try block around a call to the CreateEntity method. You then use a Catch block to handle the eConnectException type. Typically, you add code to the Catch block that attempts to correct the error, reports the error to the user, or records error information to a log.

How to handle eConnect exceptions.

The following Visual Basic example shows how to use a Try/Catch block to handle an eConnectException. Notice how the first Catch statement handles eConnectExceptions while the second Catch handles all other exception types. In this example, the application displays the error information from the message property of the exception in a textbox control.

```
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 = "Data Source=localhost;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.CreateEntity(ConnectionString,
        xmlDoc.OuterXml)

'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
```

Chapter 8: Serialization

This section of the documentation describes how to use classes from the Microsoft.Dynamics.GP.eConnect.Serialization namespace. You use serialization classes to create .NET objects that represent eConnect XML documents. The following items are discussed:

Microsoft.Dynamics.GP.eConnect.Serialization

To create .NET objects that represent Connect XML document, add the Microsoft.Dynamics.GP.eConnect.Serialization assembly and namespace to your project.

To use the classes in the Microsoft.Dynamics.GP.eConnect.Serialization namespace, you must add a reference to the Microsoft.Dynamics.GP.eConnect.Serialization assembly to your .NET project. For information about how to add a reference, see Adding a reference.

The Microsoft.Dynamics.GP.eConnect.Serialization namespace includes the following types of classes.

Category	Description
Node types	The node classes represent the data nodes of an eConnect XML document. The node classes have fields that specify Dynamics GP data values.
Transaction types	The transaction type classes represent the document type and operation for an eConnect XML document. The transaction type classes have fields that you populate with node classes.
Document type	The eConnectType class represents the root node of an eConnect XML document. To complete a document, you populate the fields of the eConnectType class with one or more transaction type classes.

To see the list of serialization classes or to find more information about a specific class, see the .NET Programming Reference section of the eConnect help. You might also use the Visual Studio Object Browser to view the fields associated with a serialization class.

Creating an eConnect document for a .NET project

The serialization classes enable you to use .NET to create an object that represents an eConnect XML document. The following sections show how to use the serialization classes to construct a document object. You typically use these objects with the the Create, Update, or Delete method of the eConnectMethods class.

To create an eConnect document object using serialization classes, complete the following steps.

Create an eConnect node object

To begin, use the serialization classes to instantiate the objects that represent the XML nodes for your type of transaction and operation. Populate the fields of the object with the data values you want to use.

The following Visual Basic example shows how to instantiate a taSopHdrIvcInsert object. Notice how the fields of the object are populated with data 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
```

Create an eConnect transaction type object

An eConnect XML document uses transaction type schemas to group related nodes for a specified operation. To perform this step, you instantiate a transaction type object. You then populate the fields of the transaction type object with node objects for that specified transaction type.

The following Visual Basic example shows how to instantiate a SOPTransactionType object. Notice how the taSopHdrIvcInsert field is populated with the salesHdr object from the the previous step:

```
Dim salesOrder As New SOPTransactionType

salesOrder.taSopHdrIvcInsert = salesHdr
```

Create an eConnect document object

An eConnect XML document uses a document node to package transaction types for the eConnect business objects. To perform this step in .NET, you instantiate an eConnectType object. You then populate the fields of the document object with your transaction type objects.

The following Visual Basic example instantiates an eConnectType object. Notice how the SOPTransactionType field is populated with salesOrder object from the previous step:

```
Dim eConnect As New eConnectType

eConnect.SOPTransactionType = salesOrder
```

Serialize the eConnect document

To use or store your .NET eConnect document object, use the .NET XmlDocument and XmlSerializer classes to convert the .NET document object into an eConnect XML document. You typically serialize your .NET document for the following scenarios:

- You use the XmlDocument to supply the XML string parameter for the Create, Update, or Delete method of the eConnectMethods class. You use these methods when you use eConnect to create, update, and delete or void Dynamics GP data records.

- You write the serialized eConnect object to an XML file. You use XML files when you use Microsoft message queuing (MSMQ) and the eConnect Incoming Service to create, update, and delete or void Dynamics GP data records. You can also use the files to archive your transactions to a disk.

For information about how to serialize an eConnect document object, see *Serializing an eConnect document object*.

Using serialization flags

Several classes in the eConnect serialization namespace include fields called serialization flags. A serialization flag is a boolean member of the class that you use to specify whether to use or discard the value assigned to a related field.

The serialization flag fields in an eConnect serialization class always append the word "Specified" to the name of the field that the boolean flag targets.

To update a field that has a serialization flag, you assign a value to the field and set the value of the related serialization flag to True. The serialization flag instructs the business object to use the value you assigned to the field to update the record in the database. If you set the serialization flag to False or do not include the serialization flag, the value you supply in the class field is not used.

You use serialization flags when the underlying eConnect business object supports update functionality. Update functionality enables you to update a record by submitting a document that specifies a few fields that have new values. Fields not explicitly included in the update document retain their existing value. For more information about eConnect update functionality, see *Using eConnect to update existing data*.

The following Visual Basic example illustrates the use of a serialization flag in a .NET development project. Notice the use of the HOLD and HOLDSpecified fields of the taUpdateCreateCustomerRcd class. The value of the HOLD field places a hold on the customer. The value of the HOLDSpecifed field enables the update of the customer hold status. If HOLDSpecified is omitted or is not set to True, the value assigned to HOLD is discarded and the hold status of the customer remains unchanged.

```
public sub UseSerializationFlag()
    Try

        '**Create a customer document**
        'Use the taUpdateCreateCustomerRcd class to specify
        ' the customer update
        Dim customer As New taUpdateCreateCustomerRcd
        With customer
            'Specify the customer
            .CUSTNMBR = "AARONFIT0001"

            'Use the HOLD field to place a hold on the customer
            'Set the serialization flag HOLDSpecifed to True.
            .HOLD = 1
            .HOLDSpecified = True
        End With

        'Add the customer object to the RMCustomerMasterType transaction type
        Dim customerTransactionType As New RMCustomerMasterType
        customerTransactionType.taUpdateCreateCustomerRcd = customer

        'Create an array of RMCustomerMasterType and add
        ' the customer transaction type object to the array
        Dim mySMCustomerMaster(0) As RMCustomerMasterType
        mySMCustomerMaster(0) = customerTransactionType

        'Add the array of transaction type objects to an
        ' eConnect document object
        Dim eConnectDoc As New eConnectType
        eConnectDoc.RMCustomerMasterType = mySMCustomerMaster

        '**Serialize the eConnect document**
        'Create a memory stream for the serialized eConnect document
        Dim memStream As New MemoryStream()

        'Create an Xml Serializer and serialize the eConnect document
        ' to the memory stream
        Dim serializer As New XmlSerializer(GetType(eConnectType))
        serializer.Serialize(memStream, eConnectDoc)

        'Reset the position property to the start of the buffer
        memStream.Position = 0

        '**Load the serialized Xml into an Xml document**
        Dim xmldoc As New XmlDocument()
        xmldoc.Load(memStream)

        'Instantiate an eConnectMethods object
        Dim eConnectObject As New eConnectMethods

        '**Set the connection string**
        'The connection string targets the TWO database on the local computer
        Dim ConnectionString As String
        ConnectionString="Data Source=localhost; Integrated Security=SSPI;" _
            & "Persist Security Info=False; Initial Catalog=TWO;"
```

```
        '**Update the customer record**
        'If eConnectResult is TRUE, the XML document was
        ' successfully submitted
        Dim eConnectResult As Boolean
        eConnectResult=eConnectObject.UpdateEntity(ConnectionString, _
            xmlDoc.OuterXml)

    Catch eConnectError As eConnectException
        Console.Write(eConnectError.ToString())

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

Serializing an eConnect document object

The serialization classes enable you to programmatically create eConnect document objects from your .NET application. However, to submit your document to the eConnect business objects, you must convert the .NET document object to the XML format that the business objects require. To convert a .NET document object to XML, use the .NET XmlSerializer and XmlDocument classes to produce a serialized version of your document.

In .NET, serialization is the process of converting an object into a form that can be persisted or transported. For eConnect, you typically convert your document object to a string. For more information about .NET serialization, refer to the .NET Framework SDK.

The following Visual Basic example shows how to create an eConnect sales invoice object and serializes the sales invoice to an XML file. The serialized information is then used with the CreateTransactionEntity method to create the sales invoice in Dynamic GP. As you review the example, note the following actions:

- The SerializeSalesOrderObject subroutine uses several eConnect serialization classes to create an eConnect sales order document object. Notice how the two taSopLineIvcInsert objects and the taSopHdrIvcInsert object populate the SOPTransactionType object. Also notice how the SOPTransactionType populates the SOPTransactionType field of the eConnectType document object.

- The SerializeSalesOrderObject subroutine shows how to use a .NET XmlSerializer, FileStream, and XmlTextWriter to serialize the eConnect document object to a file. The code example writes an XML representation of the sales order document object to the SalesOrder.xml file.

- Notice how the Main subroutine loads the XML from the SalesOrder.xml file into a .NET XmlDocument object.

- The example shows how to use the OuterXml property of the XmlDocument to populate a string with the XML for the sales order document.

- The example shows how to instantiate an an eConnectMethods object and how to use the CreateTransactionEntity method. Notice how the sales order document string is used as a parameter for the CreateTransactionEntity method. The example then uses the CreateTransactionEntity method to create

the sales order document in the Dynamic GP company database specified by the connection string.

```
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
  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=localhost;" _
                & "initial catalog=TWO;integrated security=SSPI;" _
                & "persist security info=False; packet size=4096"

            'Create the invoice in Microsoft Dynamics GP
            eConCall.CreateTransactionEntity(sConnectionString, _
                    salesOrderDocument)

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

    End Sub

    'This subroutine creates an eConnect invoice XML document and
    'writes the XML to a file
  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"
        .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.Write(ex.ToString)
        End Try

    End Sub
End Class
```

If you use the example code to create the SalesOrder.xml file. the file should contain 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>
    <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>
```

Deserializing a Transaction Requester document

This section of the documentation shows how to convert the string returned by the GetEntity method of the eConnectMethods class to an eConnect serialization object that you can use in your .NET development project. In .NET, the process of converting a string to an object is called deserialization. To deserialize the eConnect XML to an object from the eConnect Serialization classes, you use the following eConnect and .NET components.

- To retrieve eConnect data as an XML string, use the GetEntity method of the eConnectMethods class. The GetEntity method uses the eConnect Transaction Requester to retrieve data and create an XML string.

The eConnect Transaction Requester supports a limited number of eConnect document types. Always check whether the Transaction Requester can retrieve the document type that you want to use. Also check that the Transaction Requester returns the data fields you need to create a valid object.

- You use .NET StringReaders and XmlTextReaders to reformat the XML string you receive from the GetEntity method. Typically, you use the data fields from the Transaction Requester XML string to populate the data fields of an XML string that represents a serialized eConnect document type.

- Use the Deserialize method of the .NET XmlSeralizer to create an instance of an eConnect serialization class. The Deserialize method converts the XML string into an eConnect serialization object you can use with your .NET project.

To illustrate the deserialization procedure, the following steps show how to retrieve customer information as an XML string and how to convert that string into a taUpdateCreateCustomerRcd object.

Use GetEntity to retrieve a record

Use the GetEntity method to retrieve an XML string that includes the data fields for the specified Transaction Requester document.

To view an example of how to use the GetEntity method to retrieve a customer record, see *Retrieving XML documents with GetEntity*.

The following Visual Basic example shows how to use the GetEntity method to obtain a customer record. Notice how the return result populates the customerDoc string.

```
'Retrieve the customer document
Dim customerDoc = eConnectMethods.GetEntity(connectionString,
xmldoc.OuterXml)
```

Retrieve the data fields for the record

Use a .NET StringReader and XmlTextReader to retrieve the data fields from the Transaction Requester XML string. Create a new XML string that contains the data fields.

The following Visual Basic example retrieves the field values contained in the customerDoc string. Notice how customerDoc is loaded into the StringReader and XmlTextReader objects. Also notice how the XmlTextReader uses the name of the Customer node in customerDoc to identify the parent node of the data fields. The ReadInnerXml method of the XmlTextReader returns a string that contains all the data fields that were retrieved by the Transaction Requester.

```
'**Retrieve the customer XML**
'Load the customer document string into a StringReader
Dim requestReader As New StringReader(customerDoc)

'Use the StringReader to populate an XML text reader
Dim xmlTextReader As New XmlTextReader(requestReader)

'Use the XML text reader to find the customer XML in the eConnect
' Requester response document
'The eConnect_Out_Setup table shows that the customer XML data will
' be enclosed in <Customer> tags
xmlTextReader.ReadToFollowing("Customer")
Dim customerXml = xmlTextReader.ReadInnerXml()
```

Create a serialized eConnect object

Create a string that represents a serialized version of the object you want to create. For example, to create a taUpdateCreateCustomerRcd you create a string that includes the XML for the customer object.

The following Visual Basic example shows how to create a string that represents a serialized taUpdateCreateCustomerRcd object. Notice how the customerXml string supplies the XML for the data fields. The customerObjectXml string is then added to a .NET StringReader object.

```
'Create a string that places the customer XML into an
taUpdateCreateCustomerRcd XML node
Dim customerObjectXml = String.Concat("<?xml version=""1.0""?> _
    <taUpdateCreateCustomerRcd>", customerXml, _
    "</taUpdateCreateCustomerRcd>")

'Use a StringReader to read the XML for the taUpdateCreateCustomerRcd XML node
Dim customerReader As New StringReader(customerObjectXml)
```

Deserialize the eConnect XML string

Use your XML string with the Deserialize method of the .NET XmlSerializer to create an instance of an eConnect serialization object. The Deserialize method uses the XML in the string to populate the fields of the object.

The following Visual Basic example shows how to use a .NET XmlSerializer to deserialize the customerObjectXml in the customerReader object. Notice how CType is used to specify the type of the object.

```
'**Deserialize the taUpdateCreateCustomerRcd XML node from the StringReader**
Dim deSerializer As New XmlSerializer(GetType(taUpdateCreateCustomerRcd))

'Cast the deserialized object to a taUpdateCreateCustomerRcd
' serialization object
Dim testObject = CType(deSerializer.Deserialize(customerReader), _
    taUpdateCreateCustomerRcd)
```

Chapter 9: eConnect Integration Service

This section of the documentation describes how to use the eConnect Integration Service in a .NET application. The eConnect integration service is a Windows Communication Foundation (WCF) service that enables you to perform specified operations using eConnect XML documents. The following items are discussed:

eConnect for Microsoft Dynamics GP 2013 Integration Service

The eConnect Integration Service is a Windows service that enables you to use the eConnect business objects from a .NET application. You can use the eConnect Integration Service instead of the Microsoft.Dynamics.GP.eConnect assembly.

To use the eConnect Integration Service, you use a WCF service from your application. A WCF service gives you more flexibility but configuring and using the service can become complex. Before you attempt to use the eConnect Integration Service, you should be familiar with WCF .

The eConnect Integration Service includes classes, methods, and enumerations that you use to perform operations. The classes of the eConnect Integration Service include many of the same methods that you find in the classes of the Microsoft.Dynamics.GP.eConnect namespace. For more information, see the .NET Programming Reference section of the eConnect help.

To add eConnect functionality to an application, you use the following eConnect Integration Service classes.

Class	Description
DocumentNumberRollbackClient	Specifies a collection of Microsoft Dynamics GP document numbers that were not used. This class contains methods found in the DocumentRollback class of the Microsoft.Dynamics.GP.eConnect assembly.
eConnectClient	Provides access to Microsoft Dynamics GP data. This class contains methods found in the eConnectMethods and GetNextDocNumbers classes of the Microsoft.Dynamics.GP.eConnect assembly.
eConnectFault	Specifies an eConnect exception that occurred during a service operation. This class is similar to eConnectException class of the Microsoft.Dynamics.GP.eConnect assembly.
eConnectSqlFault	Specifies one or more SQL exceptions that occurred during a service operation. You use this class to identify SQL exceptions during an eConnect operation. This class is similar to eConnectSqlFault class of the Microsoft.Dynamics.GP.eConnect assembly.
RollBackDocument	Specifies a single Microsoft Dynamics GP document number that was not used. This class contains the same properties found in the RollBackDocument class of the Microsoft.Dynamics.GP.eConnect assembly.

Class	Description
TransactionRecordIdsClient	Retrieves the next document number for several types of Microsoft Dynamics GP documents. This class contains methods found in the GetNextDocNumbers class of the Microsoft.Dynamics.GP.eConnect assembly.

The also includes the following enumerations

Enumeration	Description
IncrementDecrement	Specifies whether to retrieve or return a document number.
IVDocType	Specifies the type of an inventory document.
RMPaymentType	Specifies the type of a receivables document.
SopType	Specifies the type of a sales order document.
TransactionType	Specifies the type of a transaction.

Adding a service reference

To use the eConnect Integration Service with a .NET application, you have to first add a service reference to the Visual Studio project for the application.

To add the service reference, your Visual Studio project must specify the target framework as .NET Framework 3.5.

To add a service reference to a Visual Studio project, complete the following steps.

1. **Add a Service Reference.**
   In Visual Studio, click the the Project menu, and then click Add Service Reference. The Add Service Reference window opens.

2. **Specify the service URL.**
   In the Address box of the Add Service Reference window, enter the URL of eConnect Integration Service. Use the following format for the URL:

   net.pipe://<server name>/Microsoft/Dynamics/GP/eConnect/

   The following example shows a URL for the eConnect Integration service. Notice how the URL specifies localhost for the server name.

   ```
   net.pipe://localhost/Microsoft/Dynamics/GP/eConnect/
   ```

   Click Go.

3. **Specify the eConnect service.**
   In the Services box, click eConnect. In the Namespace box, enter a name for the service. Click OK. The service reference is created and the service configuration information is added to the configuration file of the application.

Client constructors

To use the eConnect Integration Service, you first initialize one or more of the following eConnect client classes:

- DocumentNumberRollbackClient
- eConnectClient

- TransactionRecordIdsClient

Each class includes one or more constructors you can use to initialize a client object. Typically, you use the default constructor. The default constructor uses the endpoint specified in Microsoft.Dynamics.GP.eConnect.Service.exe.config file. When you install eConnect, the configuration file is added to the folder:

c:\Program Files\Microsoft Dynamics\eConnect 12.0\Service\

The following C# code example use the default constructor to initialize an eConnectClient object:

```
// Initialize a new instance of the eConnectClient
eConnectClient eConnectObject = new eConnectClient();
```

However, you can use other constructor methods to specify the eConnect Integration Service endpoint you want to use. All of the client classes inherit from the System.ServiceModel.ClientBase class. The ClientBase class includes overloaded constructors that you can use to specify endpoint information.

For example, the following table lists all the constructor methods for the eConnectClient class

The DocumentNumberRollbackClient and TransactionRecordIdsClient include overloaded constructors with the same parameters as the eConnectClient class.

You use one of these constructors when you want to target a different endpoint then the default endpoint..

Name	Description
eConnectClient()	Initializes a new instance of the eConnectClient class using the default target endpoint.
eConnectClient(string endpointConfigurationName)	Initializes a new instance of the eConnectClient class using the specified endpoint configuration. You have to add the endpoint configuration information to the configuration file of your application.
eConnectClient(string endpointConfigurationName, string remoteAddress)	Initializes a new instance of the eConnectClient class using the specified endpoint configuration. You have to add the endpoint configuration information to the configuration file of your application. You also specify the service address you want to use.
eConnectClient(string endpointConfigurationName, System.ServiceModel.EndpointAddress remoteAddress)	Initializes a new instance of the eConnectClient class using the specified endpoint configuration. You have to add the endpoint configuration information to the configuration file of your application. You also specify the service address you want to use.
eConnectClient(System.ServiceModel.Channels.Binding binding, System.ServiceModel.EndpointAddress remoteAddress)	Initializes a new instance of the eConnectClient class using the specified binding and endpoint address.

The constructors use the following parameters.

Name	Type	Description
binding	System.ServiceModel.Channels.Binding	Specifies the binding to use to make calls to the service.
endpointConfiguationName	string	Specifies the name of the endpoint in the application configuration file.
remoteAddress	string	Specifies the address of the eConnect Integration Service.
remoteAddress	System.ServiceModel.EndpointAddress	Specifies the address of the eConnect Integration Service.

Using the CreateEntity method to add a record

The eConnectClient class of the eConnect Integration Service includes methods that create, update, delete, and retrieve Microsoft Dynamics GP data. In addition, the class includes methods you can use to retrieve and restore Microsoft Dynamics GP document numbers.

To learn how to use the eConnectClients class, use the following steps to create a customer in Microsoft Dynamics GP. The example shows how to use the CreateEntity method of the eConnectClient class.

Add a service reference to the project

To begin, use Visual Studio to add a service reference to the eConnect Integration Service. In the Solution Explorer, right-click References, and then click Add Service Reference. In the Add Service Reference window, enter the following in Address, and then click Go.

```
net.pipe://localhost/Microsoft/Dynamics/GP/eConnect
```

Click eConnect in the list of Services and type a Namespace value that specifies a name for the service reference. For example, enter eConnectIntegrationService.

Add the service namespace to the project

Use the service reference name you entered in the previous step to add the namespace in your Visual Studio project. To add the namespace in C#, you add a using statement. To add the namespace in Visual Basic you add an Imports statement.

The following C# example adds a using statement that specifies a namespace from the service reference. In this example, ServiceTestApp is the name of the application project and eConnectIntegrationService is the name specified for the service reference.

```
using ServiceTestApp.eConnectIntegrationService;
```

Instantiate an eConnectClient object

To use the integration service, instantiate an eConnectClient object. The object includes the CreateEntity method you use to add a customer to Microsoft Dynamics GP.

The following C# example shows how to use the eConnectClient constructor to create the object:

```
// Instantiate an eConnectClient object
eConnectClient eConnectObject = new eConnectClient();
```

Use an XML file to load the customer document

To create a customer you use an eConnect XML document that includes the data for the customer. One way to get the customer document is to create an XML file that contains the customer information. For information about how to create an eConnect XML document for a customer, see *Create a customer*.

The following C# example shows how to load an eConnect XML document from a file. Notice how the text from the specified file is loaded into a .NET XmlDocument object. Also notice how an XML string is created using the OuterXml property of the XmlDocument object.

```
// Use the XML document in the specified file to create
// a string representation of the customer
XmlDocument newCustDoc = new XmlDocument();
newCustDoc.Load("CustomerCreate.xml");
string newCustomerDocument = newCustDoc.OuterXml;
```

Create an eConnect connection string

The CreateEntity method requires an eConnect connection string. You use the connection string to specify the Dynamics GP data server and the company database.

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

The following C# example shows how to create a connection string:

```
//Create a connection string
string connectionString = "Data Source=localhost;Integrated
Security=SSPI;Persist Security Info=False;Initial Catalog=TWO;";
```

Use CreateEntity to submit the customer document

Use the CreateEntity method to submit the document to the eConnect business objects. To use the CreateEntity method, you have to supply parameters that specify the connection string, and the customer.

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

The following C# example uses the CreateEntity method to submit an eConnect XML document. Notice that the first parameter is the connection string. Also notice that the second parameter is the string that represents the eConnect XML document for the customer.

```
// If eConnectResult is TRUE, the XML document was successfully submitted
bool result = eConnectObject.CreateEntity(connectionString,
newCustomerDocument);
```

eConnect Integration Service exception handling

The eConnect Integration Service includes the following exception classes that you can use to catch and handle eConnect errors:

- eConnectFault
- eConnectSqlFault

You use these classes together with the FaultException generic class from the System.ServiceModel namespace. You use eConnectFault and eConnectSqlFault to specify the exception type. The following example shows how to use eConnectFault with the FaultException class.

```
FaultException<eConnectFault>
```

The most common exception handling technique is the Try/Catch block. For example, you place a Try block around a call to the GetEntity method. You then use one or more Catch blocks to handle exceptions. You can add code to each Catch block that attempts to correct the error, reports the error to the user, or records the error in a log.

The following C# example shows a console application that uses eConnectFault, and eConnectSqlFault together with FaultException to handle eConnect exceptions. To follow this example, you use Visual Studio to add a service reference to the eConnect Integration Service. Notice that the application does not require a reference to either of the eConnect .NET assemblies.

To create the parameters for GetEntity, the sample includes two methods:

- The ConnectionString method creates an eConnect connection string that specifies the Microsoft Dynamics GP database you want to query.
- The SpecifyCustomer method returns an XML string. The XML represents an eConnect requestor document that specifies the customer to retrieve. Notice how you can use the method to specify the ID of the customer you want to retrieve.

You could also use the classes in the eConnect Serialization namespace to create the XML string. For more information, see Retrieving XML documents with GetEntity.

Finally, notice how the Dispose method is called to release the resources used by the eConnectClient.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using ExceptionHandling.eConnectIntegrationService;

namespace ExceptionHandling
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
    eConnectClient client = null;

    try
    {
        // Instantiate an eConnectClient object
        client = new eConnectClient();

        // Get a requestor document for a specified customer
        string customer = client.GetEntity(
            ConnectionString("localhost", "TWO"),
            SpecifyCustomer("AARONFIT0001"));

        // Show customer XML document in the console window
        Console.WriteLine(customer);
        Console.WriteLine("\n\nTo continue, press any key");
        Console.ReadKey(false);
    }
    catch (FaultException<eConnectFault> eFault)
    {
        Console.WriteLine(eFault.ToString());
        Console.WriteLine("\n\nTo continue, press any key");
        Console.ReadKey(false);
    }
    catch (FaultException<eConnectSqlFault> sqlFault)
    {
        Console.WriteLine(sqlFault.ToString());
        Console.WriteLine("\n\nTo continue, press any key");
        Console.ReadKey(false);
    }
    catch (Exception err)
    {
        Console.WriteLine(err.Message);
        Console.WriteLine("\n\nTo continue, press any key");
        Console.ReadKey(false);
    }
    finally
    {
        if (client != null)
        {
            client.Dispose();
        }
    }
}

// Return a string that represents a transaction requestor XML
// document for the specified customer
static string SpecifyCustomer(string custID)
{
    return String.Format(@""<?xml version=""1.0"" ?>
        <eConnect
        xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""
        xmlns:xsd=""http://www.w3.org/2001/XMLSchema"">
        <RQeConnectOutType><eConnectProcessInfo xsi:nil=""true"" />
        <taRequesterTrxDisabler_Items xsi:nil=""true"" />
        <eConnectOut><DOCTYPE>Customer</DOCTYPE>
        <OUTPUTTYPE>1</OUTPUTTYPE><INDEX1TO>{0}</INDEX1TO>
```

```
                    <INDEX1FROM>{1}</INDEX1FROM><FORLIST>1</FORLIST>
                    </eConnectOut></RQeConnectOutType></eConnect>",
                    custID, custID);
            }

            // Return a string that represents an eConnect connection string
            // to the specified server and database
            static string ConnectionString(string dataSource, string catalog)
            {
                return String.Format(@"data source={0}; initial catalog={1};
                    integrated security=SSPI; persist security info=False;
                    packet size=4096", dataSource, catalog);
            }
        }
    }
```

PART 4: MSMQ DEVELOPMENT

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

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

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" 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_incoming**. The following Visual Basic .NET code creates the object and specifies the **eConnect_incoming** queue on the local server:

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

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_incoming** is a transactional queue. A transactional queue requires a MesssageQueueTransaction object. The following Visual Basic .NET code shows how to create and use MesssageQueueTransaction to send a message to the **eConnect_incoming** queue:

```
Dim MyQueTrans As New MessageQueueTransaction
MyQueTrans.Begin()
    MyQueue.Send(MyMessage, MyQueTrans)
MyQueTrans.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 compnents
        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_incoming")
        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_outgoing.

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

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_outgoing. 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_outgoing. 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_outgoing")
myQueue.Formatter = New ActiveXMessageFormatter
```

2. **Create a transaction object.**
   The econnect_outgoing 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_outgoing")

    '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

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

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 it's 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

eConnect allows you to add XML nodes to the XML document schema. Custom XML nodes enable you to use new data elements in an eConnect XML document. You also use custom XML nodes to trigger the business logic in a custom SQL stored procedure.

To begin, specify a name for your XML node. The name must be unique and must match the name of a SQL stored procedure. In addition, the name of the XML node cannot end with the word Items.

In the eConnect transaction type schema documents, XML nodes that end with Items indicate the node contains one or more child nodes. If you attempt to use a single XML node with a name that ends with Items, your transaction will fail.

When eConnect processes an XML document, it uses the name of each XML node to find the SQL stored procedure that contains the business logic for that XML node. For example, a document that includes a custom XML node named <eConnectCustomProcedure> requires the target database to include a SQL stored procedure named eConnectCustomProcedure.

After you have a name for your XML node, you must specify the data elements for your XML node. The data elements contain the values for the XML document. To add data elements to an XML node, use the following guidelines:

- Define the data elements of your custom XML node. Typically, the number of data elements match the input parameters of the SQL stored procedure.
- Specify a unique name for each data element. Typically, the data element names match the names used for the input parameter of the SQL stored procedure.
- Determine the data type of each element. Use the data type and data length constraints from the input parameters of the SQL stored procedures to specify the type of data for each element.

The SQL stored procedure uses the data elements of the XML node to complete the business logic associated with that XML node.

To show a custom XML node, the following XML example defines a node named <eConnectCustomProcedure>. Notice how the XML node includes a single data element named <CUSTNMBR> that holds a customer ID value:

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

After you define your custom XML node, add your new XML node to an existing eConnect transaction type schema. To use your XML node, you need to include that node in an eConnect XML document.

The following XML example adds the <eConnectCustomProcedure> XML node to an eConnect XML document. Notice how the <eConnectCustomProcedure> node has been added to the <RMCustomerMasterType> transaction type.

```
<eConnect xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <RMCustomerMasterType>
        <eConnectProcessInfo>
        </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, you also have to create a SQL stored procedure for that node. You use the stored procedure to create, update, or delete a database record using the values in the data elements of the node. To associate the stored procedure with the node, the stored procedure and the custom node must have the same name.

For example, the previous section adds an <eConnectCustomProcedure> node to an XML document. To process the new node, you have to create a stored procedure named eConnectCustomProcedure. To process the data elements in the node, the stored procedure must include the following parameters:

- You add an input parameter for each element of your custom node. For example, the <eConnectCustomProcedure> node requires you to add a CUSTNMBR parameter to the eConnectCustomProcedure stored procedure.

- You add the input parameters in the same order that the element appears in the custom XML node.

- To use the eConnect error handling process, you add output parameters that can hold an ErrorState value and an ErrString message. ErrorState specifies whether an error occurred and ErrString includes error codes and other information.

To add a stored procedure to your database server, refer to the SQL Server help documentation for information about installing SQL stored procedures.

The following SQL example shows a stored procedure for the <eConnectCustomProcedure> XML node. Notice that the eConnectCustomProcedure name matches the name of the node. Also notice how the input parameter named I_vCUSTNMBR maps to the CUSTNMBR element of the node. Finally, notice that the procedure includes output parameters named O_iErrorState and oErrString that enable the procedure to return error information.

```
/* 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

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 that document. When the stored procedure executes, it also executes the pre and post stored procedures for that businss object. The following example describes the sequence of actions initiated by a call to the **taSopHdrIvcInsert** stored procedure..

- The pre stored procedure runs prior to the core stored procedure. In this example, the **taSopHdrIvcInsertPre** executes. After the pre stored procedure completes, the business object checks for errors. If it detects an error, the stored procedure stops, initiates a rollback, and returns an error message to the caller. If no error is detected, the stored procedure continues.

- The **taSopHdrIvcInsert** stored procedure runs. When the stored procedure completes, it checks for errors. If it detects an error, the stored procedure stops, initiates a rollback, and returns an error message to the caller. If no error is detected, the stored procedure continues with a call to the post stored procedure.

- The post stored procedure runs immediately after the core stored procedure. In this example, the **taSopHdrIvcInsertPost** stored procedure executes. After the post stored procedure completes, the business object checks for errors. If it detects an error, the stored procedure stops, initiates a rollback, and returns an error message to the caller. If no error is detected, the stored procedure ends and returns to the caller.

For additional information about the sequence of events in an eConnect stored procedure, refer to the *Business objects* on page 14.

To alert the eConnect business object of an error in your pre or post stored procedure, use the output parameters for that stored procedure. To report the status of your pre or post procedure, use the output parameters as follows:

- If the pre or post stored procedure completes successfully, set the value of the ErrorState output parameter to zero.

- If the pre or post stored procedures encounter an error, set the ErrorState parameter of the stored procedure to a non-zero value. If the eConnect business

object finds a non-zero value in the ErrorState parameter, the stored procedure halts and initiates a rollback of the transaction.

- When an error occurs in the pre or post stored procedures, set a value in the ErrString output parameter that describes 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 folder C:\Program Files\Microsoft Dynamics\eConnect 12.0\Custom Procedures. This folder contains a subfolder for each transaction type schema. Open the subfolder 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\Microsoft Dynamics\eConnect 12.0\Custom Procedures\Receivables folder. 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
*
* SANSCRIPT 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_vSTMGRMNM char(15) output, /*Sales Terr Managers Middle Name <Optional>*/
@I_vSTMGRLNM 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

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:

Transaction Requester Overview

The eConnect Transaction Requester enables you to retrieve XML that represents a document and operation. You typically use the Transaction Requester to publish XML documents to an MSMQ. There are two components to the Transaction Requester:

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 about how to retrieve documents from a queue, see Chapter 12, "Outgoing Service."

The Transaction Requester also enable you to retrieve XML documents from a .NET application. To retrieve XML that represents a document, add code to your .NET application that uses the GetEntity method of the eConnectMethods class. For information about how to use the GetEntity method, see Chapter 7, "eConnect and .NET."

Requester document types

The Transaction Requester enables you to request an XML document related to a create, update, or delete operation for the following document types:

- Cash_Receipt
- Customer
- Customer_Balance
- Employee
- GL_Accounts
- GL_Hist_Trans
- GL_Open_Trans

- GL_Work_Trans
- Item
- Item_ListPrice
- ItemPriceLevels
- 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
- RM_SalesPerson
- Sales_History_Transaction
- Sales_Transaction
- UOFM
- Vendor
- VendorItem

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 Address Internet_Address	RM00101 RM00102 SY01200
Customer_Balances	Customer Balance	RM00101 RM00103
Employee	Employee Address	UPR00100 UPR00102
GL_Accounts	GL_Accounts Details	GL00105 GL00100
GL_Hist_Trans	GL_Hist_Trans	GL30000
GL_Open_Trans	GL_Open_Trans	GL20000
GL_Work_Trans	GL_Work_Trans	GL10000
Item	Item Quantities	IV00101 IV00102

Document type	Alias	Tables used
Item_ListPrice	Item ListPrice	IV00101 IV00105
ItemPriceLevels	ItemPriceLevels	IV00108
Payables_History_Transaction	PM_Hist_Trans Tax	PM30200 PM30700
Payables_Posted_Transaction	PM_Posted_Trans Tax	PM2000 PM10500
Payables_Transaction	PM_Trans Tax	PM1000 PM10500
PO_History_Transaction	PO_Hist_Trans Line Comment	POP30100 POP30110 POP10150
PO_Receiving_Hist_Trans	PO_Receiving_Hist Line Quantities	POP30300 POP30310 POP10500
PO_Receiving_Transaction	PO_Receiving Line Quantities	POP10300 POP10310 POP10500
Project_Acct_Contract	PA_Contract Cont_Bill_Cycle PA_Project	PA01101 PA02401 PA01201
Project_Acct_Contract_Template	PA_Contract_Temp Cont_Bill_Cycle_Temp	PA41501 PA42901
Project_Acct_Cost_Category	Cost_Category	PA01001
Project_Acct_Employee_Rate	PA_Employ_Rate Line	PA01402 PA01403
Project_Acct_EmployeeExpense	PA_EmpExp Line Tax	PA10500 PA10501 PA10502
Project_Acct_Equipment_Rate	PA_Equip_Rate Line	PA01406 PA01407
Project_Acct_MiscLog	PA_MiscLog Line	PA10200 PA10201
Project_Acct_Position_Rate	PA_Employ_Rate Line	PA01404 PA01405
Project_Acct_Project	PA_Project Bill_Cycle Budget Budget_IVItems Fee Fee_Schedule Access_List Equip_List	PA01201 PA61020 PA01301 PA01303 PA02101 PA05200 PA01408 PA01409
Project_Acct_Project_Access	Proj_Acct_List Employee_Detail Address	PA01408 UPR00100 UPR00102
Project_Acct_Project_Template	PA_Project_Temp Proj_Bill_Cycle_Temp Budget Budget_Items Fee Fee_Schedule Equip_List Access_List	PA41601 PA60020 PA40201 PA40202 PA60040 PA40203 PA41409 PA41401

Document type	Alias	Tables used
Project_Acct_Timesheet	PA_Time Line	PA10000 PA10001
Purchase_Order_Transaction	Header Line Comment	POP10100 POP10110 POP10150
Receivables_Hist_Trans	Header Invoice_Tax	RM30101 RM30601
Receivables_Posted_Transaction	Header Invoice_Tax	RM20101 RM10601
Receivables_Transaction	Header Invoice_Tax	RM10301 RM10601
RM_SalesPerson	RM_SalesPerson	RM00301
Sales_History_Transaction	SO_Hist_Trans Line Line_Tax Commissions Distribution Payments Holds UserDefined Deposit Notes	SOP30200 SOP30300 SOP10105 SOP10101 SOP10102 SOP10103 SOP10104 SOP10106 SOP30201 SY03900
Sales_Transaction	SO_Trans Commissions Line LineTax Distribution Payments Holds UserDefined Deposit Notes	SOP10100 SOP10101 SOP10200 SOP10105 SOP10102 SOP10103 SOP10104 SOP10106 SOP30201 SY03900
UOFM	UOFM	IV40201
Vendor	Vendor Vendor_Addr	PM00200 PM00300
VendorItem	VendorItem	IV00103

Using the RequesterTrx element

Many eConnect XML nodes include an element named RequesterTrx. The RequesterTrx element enables you to specify whether the Transaction Requester should publish a transaction that creates, updates, or deletes Microsoft Dynamics GP record as an XML document.

To specify that an incoming eConnect XML document should not publish the transaction to the eConnect_Out table, set the RequesterTrx element of each XML node in the document to 0. A RequesterTrx element with a value of 0 instructs the Transaction Requester to not publish the current transaction in the eConnect_Out table.

For example, you use the eConnect Requester Setup utility to publish all customer inserts, updates, or deletes to the eConnect_Out table. To prevent a specific eConnect customer XML document from being published, you set the RequesterTrx element of the <taUpdateCreateCustomerRcd> XML node to 0.

For most XML nodes, the default value of the RequesterTrx element is 0. The default prevents the Transaction Requester from recognizing creates, updates, and deletes that originate from eConnect.

To ensure an eConnect XML document you use to create, update, or delete a record in Microsoft Dynamics GP is handled by the Transaction Requester, set the RequesterTrx element of that XML node to 1.

Using the <taRequesterTrxDisabler> XML node

The <taRequesterTrxDisabler> XML node gives you the ability to disable core and third-party Transaction Requester document types for individual transactions. All the eConnect transaction type schemas allow you to add one or more <taRequesterTrxDisabler> XML nodes.

You use <taRequesterTrxDisabler> with the classes in Microsoft.Dynamics.GP.eConnect .NET assembly. You typically use <taRequesterTrxDisabler> when submitting documents through the Create, Update or Delete method of the eConnectMethods class.

You typically use the <taRequesterTrxDisabler> node with core or third-party document types that do not support the use of the RequesterTrx element. If the document you are using has XML nodes that have RequesterTrx elements, use the RequesterTrx elements to specify whether the Transaction Requester publishes your transaction.

To disable the Transaction Requester for a single transaction, add a <taRequesterTrxDisabler> XML node to the transaction type schema. To disable multiple Transaction Requester document types, add separate <taRequesterTrxDisabler> nodes to the transaction type schema. To specify the Transaction Requester document type, add a document type name to the DOCTYPE element.

Use the Index elements to identify a specific document. For example, the Transaction Requester requires the Customer document type to include a CUSTNUMBR value. To disable the Transaction Requester for a document that updates a specific customer, you populate the INDEX1 field with the Customer ID.

The following XML example uses two <taRequesterTrxDisabler> nodes to disable the “Customer” and “Sales_Transaction” Transaction Requester document types. Notice how the INDEX values are used to specify the individual transactions.

```
<?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>
        <taRequesterTrxDisabler_Items>
            <taRequesterTrxDisabler>
                <DOCTYPE>Customer</DOCTYPE>
                <INDEX1>CONTOSOL0001</INDEX1>
            </taRequesterTrxDisabler>
            <taRequesterTrxDisabler>
```

```
            <DOCTYPE>Sales_Transaction</DOCTYPE>
            <INDEX1>INV2001</INDEX1>
            <INDEX2>3</INDEX2>
        </taRequesterTrxDisabler>
    </taRequesterTrxDisabler_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>
            <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>
```

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:

Creating a Transaction Requester document type

The Transaction Requester allows you to create custom Transaction Requester document types. You can use the Transaction Requester to define new document types that support your specific business needs. For example, you can create a document type to retrieve data from tables that the original Transaction Requester document types do not include.

To create a custom Transaction Requester Service, you first identify the tables and other information that the Transaction Requester requires. You need to identify a document type name, the tables to use, the index columns, the key fields used to join tables, and the data fields you want to include in your document.

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 define your document type, you need to add the information you gathered to the eConnect_Out_Setup table. The Transaction Requester uses the information in this table to retrieve and publish eConnect XML documents.

The following table describes the columns of the eConnect_Out_Setup table.

Column name	Data type	Description
DOCTYPE	**varchar**	Identifies the service.
INSERT_ENABLED	**int**	Determines whether the service is enabled or disabled for an insert action. Enabled=1, Disabled=0
UPDATE_ENABLED	**int**	Determines whether the service is enabled or disabled for an update action. Enabled=1, Disabled=0
DELETE_ENABLED	**int**	Determines whether the service is enabled or disabled for a delete action. Enabled=1, Disabled=0
TABLENAME	**varchar**	Physical name of table in SQL Server.
ALIAS	**varchar**	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	**varchar**	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.
DATACNT	**int**	Defines the number of columns that are specified for the data columns.
DATA1-180	**varchar**	Specifies names of the data columns that you want to appear in the XML document. You can specify 1 to 180 columns.

The Transaction Requester cannot support SQL Server data columns that are defined as either text or binary.

To add a document type to the Transaction Requester, use a SQL query to insert a new record to the eConnect_Out_Setup table of your Microsoft Dynamics GP company database. Use the query to populate the columns that uniquely define your document type. To ensure the Transaction Requester can use your document type, your SQL query must include the following required information:

- The number of tables you use to retrieve data defines the number of inserts you use to add your document type to the eConnect_Out_Setup table.
- When you create a custom document type, you must define a single table as the main table. You specify the main table by setting the value of MAIN to 1.
- Additional tables for the document type have a value of MAIN greater than one. In addition, these tables supply a value for the JOINTABLE column.
- Take care to match the columns specified by the JOIN1-10 and JOINTO1-10 fields. Use these columns to define the relationships between the joined tables. For example, the column specified by JOIN1 must contain values that match values in the column specified by JOINTO1.

The following SQL example uses two insert statements to define a Transaction Requester document type for employees. The new Transaction Requester document type 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,
    DATACNT,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
```

After you add your document type to the eConnect_Out_Setup table, the Transaction Requester works the same as the existing Transaction Requester document types. For example, the Outgoing Service publishes the data fields in an XML document to MSMQ.

The following XML examples show the XML documents that the Transaction Requester produces for the employee document type. The first example shows the master document you get when you set OutputType to 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>
            <DEPRTMNT>CONS</DEPRTMNT>
            <JOBTITLE>CONS1</JOBTITLE>
            <SUPERVISORCODE_I></SUPERVISORCODE_I>
            <LOCATNID></LOCATNID>
            <WCACFPAY>0</WCACFPAY>
        </Employee>
    </eConnect_Out>
</root>
```

The following example shows the complete document you get when you set the OutputType to 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>
            <DEPRTMNT>CONS</DEPRTMNT>
            <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>
                <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>
```

Implementing the RequesterTrx element

To control whether the Transaction Requester publishes your document type for a create, update or delete, use the RequesterTrx element of the XML node. The RequesterTrx element enables you to specify whether the Transaction Requester should publish the current transaction as an XML document.

To use the RequesterTrx in a document type, you need to use an eConnect stored procedure named eConnectOutVerify. This stored procedure prevents the Transaction Requester from publishing a document where the RequesterTrx flag is set to 0.

To use the eConnectOutVerify stored procedure, call the eConnectOutVerify from the eConnect pre and post stored procedure of the business object associated with your XML node. To use the eConnectOutVerify stored procedure, your pre and post procedures must supply the following values:

- Use the DOCTYPE parameter to specify the Transaction Requester document type. Use the same DOCTYPE value you used to add the document type to the eConnect_Out_Setup table. In the following example, the Transaction Requester document type is Employee.

- Use the INDEX parameters (INDEX1 - 15) to identify the specific document you want to exclude from the Transaction Requester. You must supply a value for each index you defined in the eConnect_Out_Setup table for your document

type. In the following example, the value of @I_vEMPLOYID is assigned to INDEX1 to specify the ID of the employee.

The following SQL examples shows how to use eConnectOutVerify from the pre and post procedures for the Employee example. Notice how the pre procedure uses eConnectOutVerify to identify document type and the employee. Also, notice how the post procedure sets the eConnectOutVerify Delete parameter to 1. The Delete parameter restores the original configuration of the Transaction Requester.

Pre procedure example

```
/*** Call eConnectOutVerify proc ***/
if (@I_vRequesterTrx =0)
begin
    exec @iStatus = eConnectOutVerify
        @I_vDOCTYPE ='Employee',
        @I_vINDEX1=@I_vEMPLOYID ,
        @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
```

Post procedure example

```
/*** Call eConnectOutVerify proc ***/
if (@I_vRequesterTrx =0)
begin
    exec @iStatus = eConnectOutVerify
        @I_vDOCTYPE ='Employee',
        @I_vINDEX1=@I_vEMPLOYID ,
```

```
            @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
```

PART 7: ECONNECT SAMPLES

Part 7: eConnect Samples

This portion of the documentation describes the sample applications you get when you install the Help and Samples feature. The samples are a collection of Visual Studio projects you can use to build applications that work with eConnect. The eConnect install places the folders and files for each project in the following location:

c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples

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

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 and the .NET Framework installed on your computer. To find project files for the C# application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\CSHARP
Console Application
```

To find the project files for the Visual Basic .NET application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\VB DOT NET
Console Application
```

Running the sample application

To run this sample application, perform the following steps:

1. **Start Visual Studio 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 CreateEntity method to create the customer. The CreateEntity method accesses 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 converts the XML contents of the file to a string and passes that string to the CreateEntity 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

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 and the .NET Framework installed on your computer. To find project files for the C# application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\CSHARP
SalesOrder Console Application
```

To find the project files for the Visual Basic .NET application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\VB DOT NET
SalesOrder Console Application
```

Running the sample application

To run this sample application, perform the following steps:

1. **Start Visual Studio 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 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 CreateTransactionEntity method to create the sales order. The CreateTransactionEntity method needs the sales order XML document and the eConnect connection string.

The CreateTransactionEntity method returns a string. The string contains the XML of the document that was created. You can use the string to verify the document was created or to view values that were generated by eConnect. For example, use the XML in the string to view the document ID assigned to a new sales document.

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_ItemsTaSopLineIvc-Insert, and taSopHdrIvcInsert classes.

The application instantiates two taSopLineIvcInsert_ItemsTaSopLineIvcInsert objects. It populates each properties of the object 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.

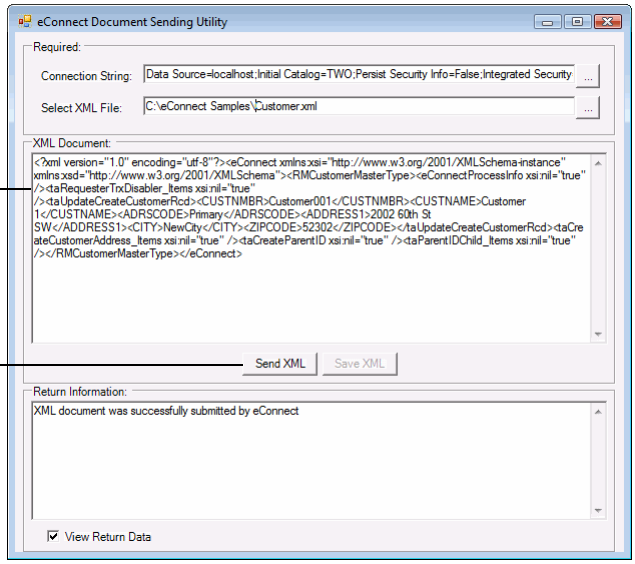
The application uses a .NET XMLSerializer to write the eConnect XML document to the SalesOrder.xml file. The application passes the XML contents of this file to the eConnect_EntryPoint method.

Chapter 20: XML Document Manager

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

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 sample application is available in both C# and Visual Basic .NET. To build this application, you must have Visual Studio and the .NET Framework installed on your computer. To find project files for the C# application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\CSHARP
DirectDocSender
```

To find the project files for the Visual Basic .NET application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect
Samples\XMLDocumentSender
```

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 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, or 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 create, 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 GetEntity method. The GetEntity 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 CreateEntity method. The CreateEntity method uses the eConnect XML to perform the specified operation and returns a boolean value that specifies whether the operation was successful.

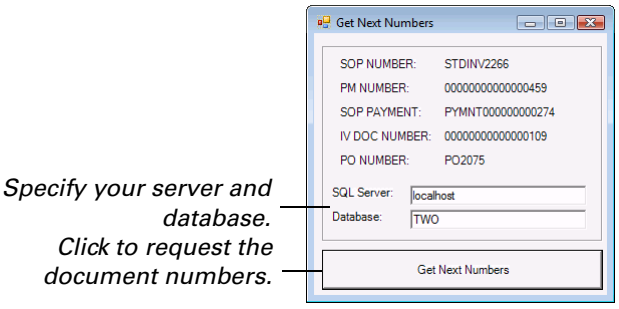
This sample application does not validate the eConnect XML document before sending its XML element values to the business objects.

Chapter 21: Get a Document Number

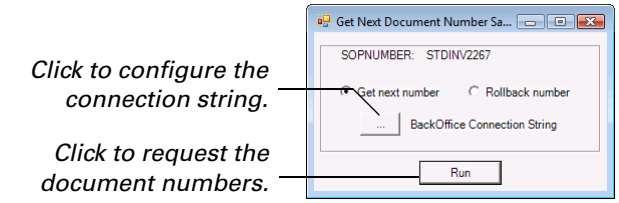
The Document Number samples include two .NET applications that retrieve the next available document numbers from Microsoft Dynamics GP. The following topics are discussed:

Overview

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 and the .NET Framework installed on your computer. To find project files for the C# application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\CSHARP
GetNextDocumentNumber
```

To find the project files for the Visual Basic .NET application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect
Samples\NextNum_TestHarness
```

Running the sample applications

To run either sample application, perform the following steps:

1. **Start Visual Studio 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 assembly.

Microsoft.Dynamics.GP.eConnect

The two applications vary in how they use the Microsoft.Dynamics.GP.eConnect 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

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.

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

To build this application, you must have Visual Studio and the 3.5 .NET Framework installed on your computer. To find project files for the C# application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\CSHARP
Requester Console Application
```

Running the sample application

To run this sample application, perform the following steps:

1. **Start Visual Studio 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 GetEntity method to retrieve the specified customer data. The GetEntity method requires two strings. One represents an eConnect XML request document and the other is the 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.

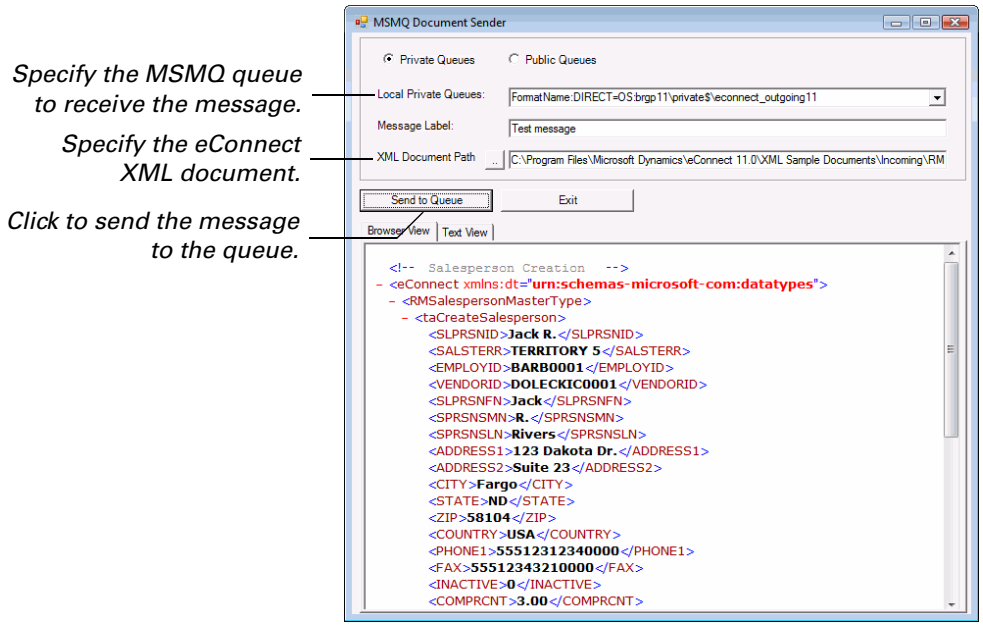
The application uses a .NET XmlSerializer to write the eConnectType object as a string. The application uses the XML string as parameter of the eConnect_Requester method.

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

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 and the 2.0 .NET Framework installed on your computer. You must also have MSMQ installed and running on your computer.

To find the project files for this Visual Basic .NET application, open the following folder:

```
c:\Program Files\Microsoft Dynamics\eConnect 12.0\eConnect Samples\VB DOT NET
Queue Client
```

Running the sample application

To run this sample application, perform the following steps:

1. **Start Visual Studio 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.

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 Integration Service

A Microsoft Windows service that enables applications to send XML documents to the eConnect business objects (SQL stored procedures) Also, enables applications to retrieve specified XML documents from 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.

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.

Windows Communication Foundation (WCF)

A runtime and a set of APIs for creating systems that send messages between services and clients. Used to create applications that communicate with other applications on the same system or a remote system.

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

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

X