Infor SyteLine ERP

Integrating IDOs with External Applications

Version 7.05

This document is available for reference only. It is not a published API reference, and it does not imply that the end user is licensed to use the features described here. This document was created for the benefit of Infor employees and Infor affiliates. Although we have attempted to make this document accurate, Infor does not warrant that it is free from errors and is not responsible for any adverse results from its use, nor does Infor provide end user support for these features. Infor may modify and change these components and access methods without notice. The use of these features does not negate the requirement for users to be properly licensed to access the application.
Integrating IDOs with External Applications

Published in the USA March 2007

Copyright © 2007, Infor Global Solutions Technology GmbH and/or its affiliate(s) and subsidiaries. All rights reserved. The word and design marks set forth herein are trademarks and/or registered trademarks of Infor Global Solutions Technology GmbH and/or its affiliates and subsidiaries. All rights reserved. All other trademarks listed herein are the property of their respective owners.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and Rights in Data-General at FAR 52.227.14, as applicable.

Name of Contractor: Infor Global Solutions Technology GmbH, 13560 Morris Road Suite 4100 Alpharetta, Georgia 30004-8508

Changes made to this document since the previous version:

<table>
<thead>
<tr>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/24/2005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/11/2005</td>
<td>3-6</td>
<td>Added description for ServerDate.</td>
</tr>
<tr>
<td>3/16/2005</td>
<td>3-8</td>
<td>Updated Request Data for GetPropertyInfo</td>
</tr>
<tr>
<td>3/16/2005</td>
<td>2-3, 2-4</td>
<td>Removed permission information about dynamic IDOs; added references for license information.</td>
</tr>
<tr>
<td>8/5/2005</td>
<td>3-18</td>
<td>Changed ItemNo attribute on UpdateCollection to a 32-bit signed integer</td>
</tr>
<tr>
<td>9/22/05</td>
<td>2-1, 2-2</td>
<td>Removed some old information in Intranets and added/changed information in Intranets and Sites/Entities form descriptions</td>
</tr>
<tr>
<td>9/22/05</td>
<td>1-6</td>
<td>Added info about using custom load method with standard IDO</td>
</tr>
<tr>
<td>11/1/05</td>
<td>4-10</td>
<td>Corrected step to include all available message classes</td>
</tr>
<tr>
<td>3/20/06</td>
<td>chapter 3</td>
<td>Added information about BeginTrans, CommitTrans, RollbackTrans. Revised Posting XML Documents section</td>
</tr>
<tr>
<td>3/20/06</td>
<td>chapter 2</td>
<td>Revised information about web server setup, Intranets form, Sites/Entities form</td>
</tr>
<tr>
<td>3/20/06</td>
<td>1-5</td>
<td>Added reference to Modifying SL guide for more information about using direct IDOs</td>
</tr>
<tr>
<td>3/30/06</td>
<td>3-28</td>
<td>Added information about format for passing dates</td>
</tr>
<tr>
<td>3/30/06</td>
<td>6-1</td>
<td>Added information on session initialization when programmatically directly accessing the database</td>
</tr>
<tr>
<td>5/8/06</td>
<td>5-4</td>
<td>Added better example of SaveDataSet API call</td>
</tr>
<tr>
<td>5/9/06</td>
<td>1-5, chap 5</td>
<td>Changed example API calls slightly (removing unnecessary qualifications and &quot;FS&quot; in variables)</td>
</tr>
<tr>
<td>5/9/06</td>
<td>4-10</td>
<td>Added information about changing properties to Boolean in Workflow AppData</td>
</tr>
<tr>
<td>6/9/06</td>
<td>5-2</td>
<td>Added description for CreateSessionToken</td>
</tr>
<tr>
<td>6/9/06</td>
<td>A-7</td>
<td>Corrected information on which installations include the Workflow templates</td>
</tr>
<tr>
<td>6/13/06</td>
<td>3-2</td>
<td>Added information about using HTTPS</td>
</tr>
<tr>
<td>6/13/06</td>
<td>4-3, 4-4</td>
<td>Clarified examples for using 1 request per step vs. multiple requests per step</td>
</tr>
<tr>
<td>7/24/06</td>
<td>4-8</td>
<td>Deleted information in Caution about not using carriage returns in message strings</td>
</tr>
<tr>
<td>12/27/06</td>
<td>A-15</td>
<td>Changed wording about running stored procedures manually.</td>
</tr>
<tr>
<td>1/2/07</td>
<td>3-28</td>
<td>Added information about decimal settings (SL7 only; not applicable to future versions of toolset)</td>
</tr>
<tr>
<td>3/7/07</td>
<td>5-2</td>
<td>Added note about user module</td>
</tr>
</tbody>
</table>
## Contents

**Overview** ................................................................. 1-1
   Integrating with a Form ............................................ 1-1
   Integrating with the Middleware ............................ 1-2
      About IDOs ......................................................... 1-2
      Protocols ......................................................... 1-2
      Examples - Interfacing with Standard IDOs .......... 1-3
   Integrating Directly with the Database (Dynamic IDOs) 1-5
      Standard vs. Dynamic IDOs ............................... 1-5
      Examples ....................................................... 1-6

**Web, Intranet, and User Setup** .............................. 2-1
   Web Server Setup .............................................. 2-1
   Workflow Setup .................................................. 2-1
   Intranets Form ................................................... 2-1
   Sites/Entities Form ............................................. 2-2
   Replication Categories and Rules .......................... 2-3
   User Middleware Permissions and Licensing ............ 2-3

**Using XML IDO Request and Response Documents** ..... 3-1
   About XML IDO Requests and Responses ................ 3-1
   Prerequisites - XML .......................................... 3-1
   Posting XML Documents ....................................... 3-2
   Request Documents ............................................ 3-2
   Response Documents .......................................... 3-3
   XML Request/Response Information ........................ 3-4
   Request Types .................................................. 3-5
      OpenSession .................................................. 3-6
      GetPropertyInfo .............................................. 3-8
      LoadCollection .............................................. 3-12
      UpdateCollection ............................................ 3-17
      Invoke ........................................................... 3-23
      CloseSession ................................................ 3-25
      BeginTrans, CommitTrans, RollbackTrans ............ 3-26
   Passing Dates to the IDO Runtime ......................... 3-28
   Decimal Values .................................................. 3-28
   Calling Dynamic IDOs in XML Documents ................. 3-28

**Using Workflow to Access IDOs** ............................. 4-1
   Prerequisites - Workflow .................................... 4-1
   Using the Workflow IDO Declarative Interface in Templates 4-2
      Creating Templates ........................................ 4-2
      About IDO Message Classes in Workflow Steps .... 4-3
      Finding the IDO Collection or Method ................. 4-4
      Creating the Workflow IDO Message Step ............. 4-7
      Examples: Using Methods or Collections in Steps .. 4-11
      Accessing Dynamic IDOs in Workflow Templates ..... 4-12
      Communicating with User-Extended Tables and User-Defined Fields through Workflow 4-12
      Loading Old Values ......................................... 4-13
      Viewing Error Messages ................................... 4-13
      Sample Templates .......................................... 4-13
   Launching a Workflow ......................................... 4-14
      Launching a Workflow from a Form ............... 4-14
Launching a Flow: Process Diagram ................................................................. 4-16
Posting from a Flow: Process Diagram .......................................................... 4-16

Using the .NET Web Service API ................................................................. 5-1
Prerequisites - .NET ......................................................................................... 5-1
API Calls .............................................................................................................. 5-1
  GetConfigurationNames .................................................................................. 5-1
  CreateSessionToken ......................................................................................... 5-2
  LoadDataSet ...................................................................................................... 5-3
  SaveDataSet ...................................................................................................... 5-4
  CallMethod ........................................................................................................ 5-5
Calling Dynamic IDOs Using .NET ................................................................. 5-6
Infor .NET Web Service Test Utility ............................................................... 5-6

Directly Accessing the Database Programmatically ................................. 6-1
  InitSessionContextSp ....................................................................................... 6-1
  CloseSessionContextSp ................................................................................... 6-1

Sample Workflow Templates for Automating SyteLine Business Processes.
  A-1
  Overview .......................................................................................................... A-1
  Prerequisites ..................................................................................................... A-1
  About IDOs and Workflow ............................................................................... A-2
  Licensing ........................................................................................................... A-2
  About the Templates ........................................................................................ A-3
    Customer Credit Check .................................................................................. A-3
    Item Pricing (Price Calculation) ................................................................... A-4
    Order Invoicing (EInvoice) .......................................................................... A-5
    Item Availability .............................................................................................. A-5
    Item Reservation for Order (Reserve Order) ............................................... A-6
    Print Pick List .................................................................................................. A-6
  Importing a Template ....................................................................................... A-7
  Adding a Workflow Template to SyteLine ...................................................... A-10
  Running a Workflow From a Template .......................................................... A-11
  If You Change a Template ............................................................................... A-14
  Troubleshooting ............................................................................................... A-15
    Checking the Event Viewer ............................................................................ A-15
    Running Stored Procedures Manually ......................................................... A-15
Overview

This document describes types of integration between SyteLine and external applications, as well as the prerequisites, setup, and protocols needed.

Integration between SyteLine and an external application can occur at any of these levels:

- Between the external application and the SyteLine form
- Between the external application and the SyteLine middleware (IDOs) through the Web Server or through COM
- Between the external application and the SyteLine application database, through direct access to the database tables.

Integrating with a Form

An external application can integrate with SyteLine at the form level. Two simple examples:

- Add a custom button to a form, and set up the button to open an Excel spreadsheet and export certain information from the form to the spreadsheet.
- Use an Excel spreadsheet macro to open a form.

You can use Visual Basic for Applications to access the WinStudio automation API and the Excel automation API. For information about scripting with the WinStudio automation API, see the online help for WinStudio (in SyteLine, select Help>Customizing Forms).

This document will not go into any further detail on form-level integration.
Integrating with the Middleware

About IDOs
An Intelligent Data Object (IDO) is a COM object that encapsulates units of information and logic that are called from the client layer to interact with data in the database. The IDO’s job is to transport collections of data back and forth, with any validation or rules needed, between the client and the database.

An IDO consists of the following elements:

- A set of properties. A property may represent persistent data stored in the application database, derived data, or temporary data used to communicate information to the middleware. A property may also represent a whole subcollection of data.

- A set of standard methods. All IDOs implement LoadCollection, UpdateCollection, GetPropertyInfo, and Invoke. LoadCollection retrieves a collection of rows from the database. UpdateCollection takes a set of rows marked for insert, update, or delete, and executes the appropriate SQL code against the database. GetPropertyInfo returns detailed information about the properties supported by the IDO. Invoke allows you to execute a custom method.

- A set of custom methods. Created by the developers of the application, these methods execute stored procedures and perform routine data maintenance tasks.

For more information about IDOs, refer to the online help for ObjectStudio (included as part of the developer installation of SyteLine).

Protocols
An external application could use any of the following protocols to access IDOs:

- Workflow message posting
- XML documents
- .NET web service
- RSDAO calls to the IDOs

Access to an IDO is similar for all of the protocols, allowing the same types of actions:

- **Log in** - provide a user ID, password, and the name of the configuration you want to access (this determines the site and database being used). Logging in opens a session where you can communicate with the IDO.
Overview

Integrating IDOs with External Applications

- Load collection - query either a collection or a database table
- Get property information - get a list of the properties and attributes of a collection, or the columns in a database table
- Update collection - insert, add, or delete records from a collection or a table
- Run method - invoke a method through the IDO, or run a database stored procedure
- Log off - close the session. This action is implied in some protocols.

Specific information about how to use each protocol is provided later in this document.

Examples - Interfacing with Standard IDOs

Using RSDAO Calls in External Applications

RSDAO is the automation API for IDOs. It is a set of components that implements a small set of COM objects that wrap the functionality provided by the IDOs. Using RSDAO, the programmer can create instances of any IDO, retrieve data, update data, and call methods of the IDO.

In the following example, VBA code in an Excel spreadsheet macro is used to call RSDAO methods that update values in a collection of customer orders.

```vba
Dim objSession
Dim objCollection

Set objSession = CreateObject("SYMIX.SessionInterface")
objSession.Logon "<UID>", "<PWD>", "<Config Name>"

' Load SLCos collection and define properties and filter
Set objCollection = objSession.LoadCollection("SL.SLCos", _
   "CoNum, coUfColor", "CoNum LIKE '%Test%'", _
   -1, "")

' Update the "coUfColor" UET property value to "RSDAO TEST".
While (Not objCollection.EOF)
   MsgBox objCollection.GetProperty("CoNum") & ", " & _
   objCollection.GetProperty("coUfColor")
   objCollection.SetProperty "coUfColor", "RSDAO TEST"
   objCollection.MoveNext
Wend

objCollection.SaveChanges True
objSession.Logoff
```

Since RSDAO allows users to directly access IDOs, the system checks the license on those requested IDOs. The IDOs must be properly licensed before access is allowed.

For more information about accessing IDOs through RSDAO, see the ObjectStudio online help.

Using XML IDO Request and Response Documents

An external financial application could use XML request and response documents to download a collection of customer data from SyteLine and then make updates to that data. A request to update the credit limit for customer #2500 could include a set of tags like these:
<IDORequest>
  <RequestHeader Type="UpdateCollection">
  </RequestHeader>
  <RequestData>
    <UpdateCollection Name="SL.SLCustomers">
      <Items>
        <Item Action="Update" ItemNo="0">
          <Property Name="CustNum" Modified="N">2500</Property>
          <Property Name="CreditLimit" Modified="Y">40000</Property>
        </Item>
      </Items>
    </UpdateCollection>
  </RequestData>
</IDORequest>

For more information about accessing IDOs through XML request and response documents, see Chapter 3, "Using XML IDO Request and Response Documents."

Use HTTP POST protocol to post the request and response documents to the ASP pages.

**Using Workflow Messages and the IDO Declarative Interface**

Workflow templates can use VBScript to access IDOs via COM, but a better technique is to use the IDO Declarative Interface provided with SyteLine. A workflow template gets control only when a user on a particular form saves changes to records on that form.

To create a workflow template that accesses IDOs, you don’t need to know how to write programming code - but you will need to understand the logic used in SyteLine, and you will need to know how to find the collections and methods you want. You can then copy and modify them for use with Workflow.

SyteLine’s implementation of Workflow includes a SyteLine queue and queue listener to support the IDO Declarative Interface. Also, a SyteLine mailbox is set up to monitor emails. When it receives one of the special IDO request message classes (for example, InvokeMethod or LoadCollection), the system knows how to handle the message in order to get the request to the system.

This allows you to automate a process flow. For example, if an order comes in from company X, a workflow can be set up to automatically call the IDO methods used to approve the order, ship the order, and so on.

The example shown here is the message text from a Workflow step whose Message Class is "InvokeMethod." The message text calls the ItemwhseGetDetailsSp method in the SLItemwhses collection:

```
SL.SLItemwhses.ItemwhseGetDetailsSp(
  FARMS(
    VAR APPDATA(Item),
    VAR APPDATA(Whse),
    RVAR APPDATA(QtyOnHand),
    RVAR APPDATA(QtyReorder),
    RVAR APPDATA(CntInProc),
    RVAR APPDATA(CycleFlag),
    RVAR APPDATA(Loc), MESSAGE))
```

See Chapter 4, "Using Workflow to Access IDOs," for more information.
Using .NET Web Service Calls

It is possible to use Visual Basic .NET, C#, or any .NET CLR-compatible language to make calls over the Internet that load or update IDO collections, or run methods. The .NET web service is another layer on top of the Infor Web Service (XML) layer.

The following declaration would return a dataset containing the item, description, and unit cost for item "FA-1000" from the SLItems collection:

```csharp
WebService webService = new WebService();
DataSet IdoDataSet;
webService.Url = "http://<web-service-url>";
IdoDataSet = webService.LoadDataSet(
  SessionToken,
  "SL.SLItems",
  "Item, Description, UnitCost",
  "Item = N'FA-1000'",
  "", "", -1);
```

See Chapter 5, "Using the .NET Web Service API," for more information.

Integrating Directly with the Database (Dynamic IDOs)

COM calls, XML documents, and Workflow messages can directly query or update the SQL database tables through "dynamic IDOs." Using dynamic IDOs, you can also directly invoke stored procedures from the SQL database, or from custom events defined for SyteLine forms (see the Modifying SyteLine ERP Guide for details on how to do this). Use the standard XML or COM syntax, except:

- For collection requests, instead of the collection name, you specify the table name (in the format TABLE!table_name).
- When invoking a method, instead of the IDO name, specify "SP!" and instead of the method name, specify the stored procedure name. For example:

```vbscript
Dim objSession
Dim objMyIDO
Dim result
Dim param1, param2

Set objSession = CreateObject("SYMIX.SessionInterface")
objSession.Logon "<UID>", "<FWD>", "<Config Name>"
' Create the dynamic IDO for calling procedures
Set objMyIDO = objSession.CreateInstance("SP!")
' Call the MyStoredProc stored procedure
Result = objMyIDO.MyStoredProc(param1, param2)
objSession.Logoff
```

Standard vs. Dynamic IDOs

Standard and dynamic IDOs behave exactly the same at runtime. However, standard IDOs are predefined in ObjectStudio and can support these additional features:

- The IDO can publish data from multiple database tables.
- A property name may be different than the name of the column it is bound to. (In SyteLine, this is almost always the case.
- Properties can be calculated values that are not represented by a column in the database.
• The IDO can publish non-persistent properties (often used for passing user input as parameters to a process).
• The IDO can publish subcollection properties (necessary to use WinStudio’s subcollection features).

A dynamic IDO simply binds to a table or a view and publishes each column as a property. The replication service sets some flags in the database to control whether the business logic (triggers) executes. Just as you could allow business logic to fire using dynamic IDOs, you could suppress it using dynamic IDOs.

Any method that returns result sets must be called as a custom load method, which requires a standard IDO and is not possible to do using a dynamic IDO (SI).

You can design your interface protocol to take advantage of either the standard IDO behavior or the dynamic IDO behavior - whichever you need.

Examples

Use of Dynamic IDOs in Non-Transactional XML Replication
Replication, as set up in SyteLine, uses dynamic IDOs.

A company has three sites: OHIO, MICH, and CALIF. OHIO and MICH are on the same intranet; CALIF is on a separate intranet. An OHIO user updates and saves a customer record in the OHIO database. When the customer is saved, an SQL trigger copies the data to the “customer_all” table. (Tables with a suffix of “_all” are special: they contain records that are shared between multiple sites and associate data with the site that owns it. In a multi-site environment, certain “_all” tables are shared via replication so that, for example, the OHIO site knows what customers are defined in the MICH site.)

If customer data is to be shared between multiple sites, a “replication trigger” will be defined on the customer_all table. When the customer data is copied to the customer_all table in OHIO, the replication trigger fires. The replication trigger’s job is to take the updated data and initiate the process of sending that data to the sites that need it. MICH and CALIF both need the customer update. The replication trigger writes the updated data to a set of staging tables; separate rows are created for the data that needs to go to the MICH site and the data that needs to go to the CALIF site. The object name associated with the updated data takes the dynamic IDO form “TABLE!customer_all”.

On the SyteLine utility server, a set of replication services run. These are processes (which run as Services) that are responsible for packaging/delivering and receiving/processing replication updates. Each SyteLine intranet has its own utility server, and the replication services on that utility server handle the replication processing for all sites within that intranet.

There are two replication services:
• Replicator is responsible for polling its intranet’s replication staging tables and packaging the data updates found in those tables into XML documents. These XML documents are then forwarded to the second service, Replication Listener.
• Replication Listener has two jobs: delivering the replication XML documents to remote intranets, and processing incoming replication XML documents from remote intranets. Replication Listener has two separate queues, one for outgoing data deliveries and one for incoming data.
If the replication XML document is due to be delivered to a site on a remote intranet, Replicator will drop the document in Replication Listener’s outgoing queue; if the document is due to be delivered to a site on the local intranet, Replicator will drop the document in Replication Listener’s incoming queue. In our case, the replication update to CALIF will be placed in Replication Listener’s outgoing queue, and the replication update to MICH will be placed in Replication Listener’s incoming queue.

Delivery of replication XML documents to remote intranets by Replication Listener is performed via HTTP. The replication XML document is delivered to the remote intranet’s URL (as defined in the data found in the local site’s Intranets form) via an HTTP POST operation. The ASP page that receives this POST on the remote intranet places the posted XML document into its own Replication Listener’s incoming queue. Replication Listener picks up the document and imports the data into the appropriate site’s database. In our case, MICH’s intranet’s Replication Listener will accept the incoming customer_all update and create or update the record in MICH’s site database. Since the update has an object name of “TABLE!customer_all”, a dynamic IDO configured to work on the customer_all table will be created to perform the update.

**Using Replication to Integrate with Other Systems**

You can use the replication method to push data from SyteLine to other systems (for example, an application on a Biztalk website). To do this, use SyteLine to create a new intranet, define the URL, define the site and assign the site to an intranet. You then create a special replication category for the data in a specific table, and write a rule that submits a table update to the URL any time something changes in that table. This can be set to occur at a trigger level, and the replication can deliver both the old and new values to the other system.

To push data from another system to SyteLine, you need to understand the other application’s multi-site protocol. You can then choose either to directly update the application’s table information (using dynamic IDOs) or submit standard IDO requests that go through validation rules.
Web, Intranet, and User Setup

Before an external interface can access the middleware through the Web Server, make sure the following areas of SyteLine are set up properly. Follow the instructions in the *SyteLine ERP Installation Guide* and the *Multi-Site Implementation Guide*. If you plan to use the XML IDO interface, set up your system to include *non-transactional replication*.

### Web Server Setup

The Web Server installation’s **SyteLine Web** option lays down two ASP pages in the virtual directory you specify:

- `virtual_directory/SL/IDO/SessionManager.asp`, which is used for synchronous communications
- `virtual_directory/SL/InboundQueue.ASP`, which is used for non-transactional communications.

The Utility Server installation’s **Utility Server COM+ Components** option lays down COM+ applications that are required to run the IDOs.

The easiest way to be sure both of these options are installed on the same machine is to install the Web Server component on the utility server. This is described in the *Multi-Site Implementation Guide*. However, if you are concerned that one utility server will have too much traffic in this case, you can set up a separate web server that has both of these options installed.

### Workflow Setup

If you will be using Workflow, set up your Utility Server as described in these chapters of the *Installation Guide*:

- Setting Up the Utility Server
- Setting Up Workflow Automation

### Intranets Form

The Intranets form specifies connection information used in data requests between the current site and other sites (or other applications defined as sites). If you are integrating SyteLine with another application by using non-transactional replication, the other application must be defined as a site on an "external" intranet - see the External field description below. The important fields for our purposes include:

- **Queue Server** - Enter the name of the MSMQ server for the intranet. If left blank, this field defaults to the utility server running this application, and the private queue area. If you have multiple intranets on different servers, include the utility server machine name in the path (for example, `machinename\PRIVATE$`).
• **Private Queue** - We recommend that you select this field, since queues in non-transactional replication are used only by SyteLine services on the same machine. Setting up public MSMQ instances requires further ActiveDirectory setup, which is not required by the SyteLine services.

• **URL** - This is the web address to which the XML request/response documents will be posted. When replication occurs between sites on different intranets, XML documents are routed to this URL for the target site. It may also be used by system integrators for programmatic access to a site on this intranet.

This URL could be an ASP page on the external system that receives and processes the XML from the SyteLine message queue. The processing done at this URL is up to you; for example, an ASP page might map the data into the proper format for the external system, or it might write the XML documents to a location on its server for later processing.

In many cases, this field provides the address of the default InboundQueue.asp in the virtual directory that was installed with the web server components (page 2-1).

• **Direct IDO URL** - This is the web address of the synchronous XML interface to this intranet. It is used primarily by application clients which are configured to connect via the internet. It may also be used by system integrators for programmatic access to a site on this intranet.

In many cases, this field provides the address of the default SessionManager.asp in the virtual directory that was installed with the web server components (page 2-1).

• **External** - Select this field if the intranet is an external intranet, meaning that it allows applications other than those based on the SyteLine (WinStudio) toolset. Replication done on an external intranet must be non-transactional, and an external intranet cannot have a master site.

If you are using Workflow, you should also fill in these fields on the Workflow tab for each Intranet you define. The values default from those entered through the SyteLine Configuration Wizard on the utility server when setting up Workflow Automation.

• **Exchange Server** - Enter the path to your MS Exchange server.

• **Exchange Mailbox Alias** - Enter the Exchange mailbox used to post workflows. Usually this mailbox name is **Syteline** - see the Installation Guide’s chapter on setting up Workflow Automation for more information.

• **Workflow Templates Public Folder Path** - Enter the path where the Workflow templates folder is located. This must match the path that was created in MS Outlook.

### Sites/Entities Form

This form lists SyteLine sites (and other applications that are defined as sites) and defines the relationships between the sites. For our purposes, these are the important fields:

• **System Type** - The system type is used by the non-transactional replication infrastructure to provide XSL transforms on XML data being replicated between sites. See the online help on the System Types form for information about how this works.

• **Intranet Name** - Specify the intranet to which each site belongs. Sites on the same intranet perform non-transactional replication by placing XML documents in the intranet’s inbox. Sites on separate intranets perform non-transactional replication by looking up the URL for the target site and performing an HTTP POST to the URL. This places the XML documents in the other intranet’s mailbox.
Web, Intranet, and User Setup

- **Workflow Enabled** - Select this field to allow workflows to be run from the application at this site.
- **To Site and Linked Server Name** on the Live Linked Sites tab - The Live Linked relationship is a SQL server concept where SQL code from one site can run on the other site “live” (real-time).
- **From Site and User Name** on the Site User Map tab - Used for non-transactional replication, these fields set up the SQL user name under which the From Site (the local site) communicates with the To Site (the site whose record is currently displayed). The user name and the password for that user must be the same on both the source and target systems. This information has to agree in the source and target sites, if non-transactional replication will be used between the two sites.

### Replication Categories and Rules

If you will be calling dynamic IDOs from your external application, set up replication categories and rules for transferring the data directly to or from the SyteLine database. Use the Replication Categories and Replication Rules forms to do this.

When your replication rules are complete, display the Replication Management form and click **Regenerate Replication Triggers** to put your external application’s data replication into play.

See the online help on the forms and the Replication Reference guide for more information about the replication forms and fields.

### User Middleware Permissions and Licensing

You should create one or more user IDs that can be used for remote sessions that access SyteLine IDOs through clients other than WinStudio. For instance, when you use RSDAO to get a connection and subsequent access to an IDO and its methods, RSDAO looks up the permissions on the calling user ID to see if it is permitted to access that IDO, and if so, which types of access are allowed.

**Permissions**

The steps to set up the proper permissions are as follows:

1. In the Users form, create and save the user record.
2. Click the **User Authorizations** button to display the Object Authorizations for User form with the user ID selected.
3. In the Object Authorizations for User form, select Object Type **MIDDLEWARE**.
4. In the Object Name field, click the down arrow to display a list of IDOs that are registered for the current site. This list includes IDOs that are registered as part of the core toolset. Select a name from the list, or type in any IDO name (with prefix).
5. Specify the privileges that are granted or revoked for this user ID:
   - **Edit** privilege allows the user to insert, update, or delete records while accessing an IDO.
   - **Execute** privilege allows the user to run IDO methods.
• **Read** privilege allows the user to view data (for example, run a LoadCollection request).

  The middleware permissions have no effect when the user is running SyteLine (WinStudio), which disables all middleware security and secures everything at the form level. They are in effect when this user ID is used to access IDOs through some other method.

## Licensing

Make sure that:

• The IDO you are accessing is licensed, and

• The user ID being used to access the IDO is assigned to a license module containing that IDO.

See the SyteLine help topic "List of SyteLine Modules," as well as the help on the License Management form, for more information.

  Due to a limitation in the current licensing mechanism for IDO access, use "super-user" accounts, rather than special licensing, for your applications that access the IDOs programmatically.
Using XML IDO Request and Response Documents

This information applies to applications based on toolset changes (protocol version 5.00 and later).

About XML IDO Requests and Responses

The IDO web service accepts XML request documents, processes them, and returns any results in XML response documents.

The following requests are discussed in this chapter:

- **GetConfigurations** - Get a list of available configurations for a server.
- **OpenSession** - Open a session where you can then send additional requests.
- **GetPropertyInfo** - Get a list of properties and their attributes from an IDO. For example, if you ask for the properties of SL.SLItems, the response document returns information like the name of the base table used by the collection, and for each property, its data type, label string, class, and so on.
- **LoadCollection** - Retrieve a set of records from a collection.
- **UpdateCollection** - Insert, update, or delete records from a collection.
- **Invoke** - Invoke a method from a collection. The response returns any values and parameters from the method.
- **CloseSession** - Closes the session.
- **BeginTrans, CommitTrans, and RollbackTrans** - Use these requests together to perform transaction management; that is, to wait until you verify that an UpdateCollection request is successful before you commit it to the database.

Prerequisites - XML

Before you can use XML request and response documents, you must set up the system as described in Chapter 2, "Web, Intranet, and User Setup."
Posting XML Documents

When posting the XML documents:

1. Provide access to the SessionManager.asp via a URL specification such as:
   http://domain_name/SL/IDO/SessionManager.asp or
   http://ip_address/SL/IDO/SessionManager.asp

   The SessionManager.asp was set up in the SL/IDO virtual directory during installation
   of the SyteLine Web Server component (see page 2-1). This ASP is specified in the
   URL field on the Intranets form (page 2-1). You may also want to make the ASP
   available on the internet, which requires additional configuration in IIS.

2. Post the XML documents to the URL, using the HTTP POST protocol.

   Use HTTPS if you require secure transmissions - for example, when transmitting
   passwords.

Request Documents

A request document consists of a top level IDOResult element containing one or more
RequestHeader elements.

For example:

```xml
<IDOResult>
  <RequestHeader Type="RequestType">
    <InitiatorType>InitiatorType</InitiatorType>
    <InitiatorName>InitiatorName</InitiatorName>
    <SourceName>Source</SourceName>
    <TargetName>Target</TargetName>
    <RequestData>
      ...request data...
    </RequestData>
  </RequestHeader>

  <RequestHeader Type="RequestType">...
</IDOResult>
```

Replace the information in italics with values as described in ”XML Request/Response
Information” on page 3-4.
Response Documents

The response documents have a structure very similar to the request documents. The top level element in the response document is IDOResponse, and it contains one or more ResponseHeader elements corresponding to the RequestHeader elements in the request document.

For example:

```
<IDOResponse>
  <ResponseHeader Type="RequestType">
    <InitiatorType>InitiatorType</InitiatorType>
    <InitiatorName>InitiatorName</InitiatorName>
    <SourceName>Source</SourceName>
    <TargetName>Target</TargetName>
    <ResponseData>
      ...response data...
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

The information in italics is replaced with values as described in the Request/Response Information table on “XML Request/Response Information” on page 3-4.
## XML Request/Response Information

Use the following table when inserting information into an XML request or response document that accesses the IDOs.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestType</td>
<td>The type of IDO request. The commonly used requests are:</td>
</tr>
<tr>
<td></td>
<td><em>“GetConfigurations”</em> on page 3-5</td>
</tr>
<tr>
<td></td>
<td><em>“OpenSession”</em> on page 3-6</td>
</tr>
<tr>
<td></td>
<td><em>“GetPropertyInfo”</em> on page 3-8</td>
</tr>
<tr>
<td></td>
<td><em>“LoadCollection”</em> on page 3-12</td>
</tr>
<tr>
<td></td>
<td><em>“UpdateCollection”</em> on page 3-17</td>
</tr>
<tr>
<td></td>
<td><em>“Invoke”</em> on page 3-23</td>
</tr>
<tr>
<td></td>
<td><em>“CloseSession”</em> on page 3-25</td>
</tr>
<tr>
<td></td>
<td>&quot;BeginTrans&quot;, &quot;CommitTrans&quot;, and &quot;RollbackTrans&quot; on page 3-26</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>These attribute names are case-sensitive.</td>
</tr>
<tr>
<td>InitiatorType</td>
<td>The type of process that initiated this document. Optional. Examples of InitiatorType include:</td>
</tr>
<tr>
<td></td>
<td><em>Workflow</em> - signals to the receiver of the document that this request was generated through Workflow. <em>InitiatorName</em> can then be either a form name or a flow name.</td>
</tr>
<tr>
<td></td>
<td><em>Form</em> - signals that the request was generated by a form. <em>InitiatorName</em> then must contain the form name.</td>
</tr>
<tr>
<td></td>
<td><em>Replication</em> - signals that the replication process generated the request. <em>InitiatorName</em> is blank, and the <em>SourceName</em> and <em>TargetName</em> contain the source and target sites.</td>
</tr>
<tr>
<td></td>
<td>For UpdateCollection requests, if InitiatorType is set to Replication, an insert is treated as an update if the record already exists.</td>
</tr>
<tr>
<td>InitiatorName</td>
<td>The name associated with the initiator, if any. This could be a form name or a flow name. Optional.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td>Source</td>
<td>The name of the site where the request originated. Optional.</td>
</tr>
<tr>
<td>Target</td>
<td>The name of the site the request is targeting. Optional.</td>
</tr>
<tr>
<td>TransactionName</td>
<td>In pre-5.00 versions of WinStudio, the <em>&lt;TransactionName&gt;</em> tag specified a named transaction the request was to be executed under. This tag is not valid for versions 5.00 and above.</td>
</tr>
<tr>
<td>Request Data or</td>
<td>The contents of this element depends on the data required by the RequestHeader Type attribute. See the following sections describing the specific request types for details.</td>
</tr>
<tr>
<td>Response Data</td>
<td></td>
</tr>
</tbody>
</table>
Request Types

The commonly used request types are:
- “GetConfigurations” on page 3-5
- “OpenSession” on page 3-6
- “GetPropertyInfo” on page 3-8
- “LoadCollection” on page 3-12
- “UpdateCollection” on page 3-17
- “Invoke” on page 3-23
- “CloseSession” on page 3-25

GetConfigurations

This XML document requests a list of application configurations available on the server.

Request Data
none

Response Data

<ResponseData>
  <ConfigName1/>
  <ConfigName2/>
  ...etc...
</ResponseData>

Example - GetConfigurations

This request finds the available configurations:

<IDORequest>
  <RequestHeader Type="GetConfigurations"/>
</IDORequest>

<IDOResponse>
  <ResponseHeader Type="GetConfigurations">
    <ResponseData>
      <SLColumbus/>
      <SLMaster/>
      <AppOne/>
      <MyConfig/>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
**OpenSession**

This XML request/response document validates your identity and creates a user session. If the OpenSession is successful, it pulls some information from the configuration file on the server.

NOTE You must submit an OpenSession request before you can submit other requests (except for GetConfigurations).

**Request Data**

```xml
<RequestData>
  <UserId>userid</UserId>
  <Password Encrypted="N">pswd</Password>
  <ConfigName>config</ConfigName>
  <Workstation>workstation</Workstation>
</RequestData>
```

**Response Data**

```xml
<ResponseData>
  <UserId>userid</UserId>
  <ProductVersion>version</ProductVersion>
  <ServerDate>datetime</ServerDate>
  <DeadlockRetry>0</DeadlockRetry>
</ResponseData>
```

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserId</td>
<td>The user ID, which must be defined for the specified configuration.</td>
</tr>
<tr>
<td>Password</td>
<td>The user’s password. If Encrypted=&quot;Y&quot;, the password is encrypted; however, Encrypted is usually set to &quot;N&quot;.</td>
</tr>
<tr>
<td>ConfigName</td>
<td>The configuration where the session will run. You can get a list of configurations by using the request.</td>
</tr>
<tr>
<td>Workstation</td>
<td>Workstation name, currently only used as an encrypted value by WinStudio. Optional.</td>
</tr>
<tr>
<td>ProductVersion</td>
<td>The version of the application.</td>
</tr>
<tr>
<td>ServerDate</td>
<td>The server date. WinStudio’s CURDATE() and CURTIME() keywords need to yield values that match the server for the application, not the client machine. This ServerDate value is used to compute the difference between the client machine’s date and time and the server’s, so that each time CURDATE() and CURTIME() is evaluated, WinStudio can produce the correct server-based result.</td>
</tr>
<tr>
<td>DeadlockRetry</td>
<td>Number of times to retry connection if a deadlock occurs.</td>
</tr>
</tbody>
</table>
Example - OpenSession
This example opens a session using the configuration MyConfig for user jimbow.

<IDORequest>
  <RequestHeader Type="OpenSession">
    <RequestData>
      <UserID>jimbow</UserID>
      <Password Encrypted="N">Bob1245</Password>
      <ConfigName>MyConfig</ConfigName>
    </RequestData>
  </RequestHeader>
</IDORequest>

<IDOResponse>
  <ResponseHeader Type="OpenSession">
    <ResponseData>
      <UserID>jimbow</UserID>
      <ProductVersion>7.05.00</ProductVersion>
      <ServerDate>2005-01-21 10:02:18</ServerDate>
      <DeadlockRetry>0</DeadlockRetry>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
**GetPropertyInfo**

This XML request/response document retrieves property information published by an IDO. For each property in the IDO, the response returns the property’s attributes.

**Request Data**

```xml
<RequestData>
  <GetPropertyInfo Name="ProgID"/>
</RequestData>
```

**Response Data**

```xml
<ResponseData>
  <GetPropertyInfo Name="ProgID">
    <Properties>
      <Property Name="prop-name" PropertyClass="prop-class-name" Key="seq" ProgID="prog-id">
        <DataType DecimalPosition="n">type-name</DataType>
        <Length>n</Length>
        <ColumnType>sql-column-type</ColumnType>
        <ReadOnly />
        <Required />
        <RORecord />
        <Html />
        <Subcollection />
        <Identity />
        <Warning />
        <DefaultValue>default-value</DefaultValue>
        <Label>label-value</Label>
        <LabelStringID>label-id</LabelStringID>
        <DomainIDOName>ido-name</DomainIDOName>
        <DomainProperty>prop-name</DomainProperty>
        <DomainAdditionalProperties>property-list</DomainAdditionalProperties>
        <BoolTrue>true-value</BoolTrue>
        <BoolFalse>false-value</BoolFalse>
        <InputMask>input-mask</InputMask>
        <InputMaskPromptChar>input-mask-prompt</InputMaskPromptChar>
        <DefaultIMECharset>ime-charset</DefaultIMECharset>
        <BinaryDataFormat>binary-format</BinaryDataFormat>
        <Case>case-format</Case>
        <Justify>justify-format</Justify>
        <DateFormat>date-format</DateFormat>
        <DisplayDecimalPosition>n</DisplayDecimalPosition>
        <NumDigitsInGroup>n</NumDigitsInGroup>
      </Property>
    </Properties>
  </GetPropertyInfo>
</ResponseData>
```

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPropertyInfo</td>
<td>Name - Contains the Prog ID of the IDO for which the caller is requesting property information. Multiple GetPropertyInfo tags may be included in the RequestData, for multiple IDOs.</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Property</td>
<td>Includes these attributes:</td>
</tr>
<tr>
<td></td>
<td>Name - Property name.</td>
</tr>
<tr>
<td></td>
<td>PropertyClass - Property class associated with this property.</td>
</tr>
<tr>
<td></td>
<td>Key - If the property is a key property, this attribute will be included and is the 1-based key sequence.</td>
</tr>
<tr>
<td></td>
<td>ProgID - Included for sub-collection properties only. This is the ProgID of the subcollection property, for example, SL.SLItems.</td>
</tr>
<tr>
<td></td>
<td>The rest of the elements under the &lt;Property&gt; tag correspond to fields on the ObjectStudio property editor.</td>
</tr>
<tr>
<td>DataType</td>
<td>Contains one of these values:</td>
</tr>
<tr>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td>Byte</td>
</tr>
<tr>
<td></td>
<td>Currency</td>
</tr>
<tr>
<td></td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td>Decimal</td>
</tr>
<tr>
<td></td>
<td>Float</td>
</tr>
<tr>
<td></td>
<td>GUID</td>
</tr>
<tr>
<td></td>
<td>Long Integer</td>
</tr>
<tr>
<td></td>
<td>NumSortedString</td>
</tr>
<tr>
<td></td>
<td>Object</td>
</tr>
<tr>
<td></td>
<td>Short Integer</td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td>Length</td>
<td>Specifies the length for String and Decimal datatypes.</td>
</tr>
<tr>
<td>ColumnType</td>
<td>Contains the SQL user-defined type of the column to which this property is bound.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Is included if the property has the read-only attribute set, and omitted otherwise.</td>
</tr>
<tr>
<td>Required</td>
<td>Is included if the property has the required attribute set, and omitted otherwise.</td>
</tr>
<tr>
<td>RORecord</td>
<td>Is included if the property has the read-only record flag attribute set, and omitted otherwise.</td>
</tr>
<tr>
<td>Html</td>
<td>Is included if the property contains HTML text.</td>
</tr>
<tr>
<td>Subcollection</td>
<td>Is included if the property is a subcollection.</td>
</tr>
<tr>
<td>Identity</td>
<td>Is included if the property is bound to an identity (auto-incrementing) column, and omitted otherwise.</td>
</tr>
<tr>
<td>Warning</td>
<td>Is included if this property contains warning messages for UpdateCollection responses, and omitted otherwise.</td>
</tr>
<tr>
<td>DefaultValue</td>
<td>Returns the default value for the specified component.</td>
</tr>
<tr>
<td>Label</td>
<td>Returns a string that is the component’s label.</td>
</tr>
<tr>
<td>LabelStringID</td>
<td>Returns the ID used to retrieve a label string.</td>
</tr>
<tr>
<td>DomainIDOName</td>
<td>Contains the domain IDO where the property is found. For example, the DomainIDOName for a unit of measure code is SL.SLUMs.</td>
</tr>
<tr>
<td>BoolTrue</td>
<td>Contains a value that represents True in Boolean evaluations of the component.</td>
</tr>
<tr>
<td>BoolFalse</td>
<td>Contains a value that represents False in Boolean evaluations of the component.</td>
</tr>
<tr>
<td>InputMask</td>
<td>Contains the pattern used for the formatting input mask.</td>
</tr>
<tr>
<td>InputMaskPromptChar</td>
<td>Contains the character used in the pattern to signify empty spaces.</td>
</tr>
</tbody>
</table>
### Example 1: GetPropertyInfo

This request gets property information for SL.SLItems:

```xml
<IDOResquest>
  <RequestHeader Type="GetPropertyInfo">
    <GetPropertyInfo Name="SL.SLItems" />
  </RequestData>
</RequestHeader>
</IDOResquest>
```
<IDOResponse>
  <ResponseHeader Type="GetPropertyInfo">
    <ResponseData>
      <GetPropertyInfo>
        <IDOProperties PrimaryBaseTable="item" Count="272" PrimaryBaseTableAlias="itm">
          <Property Key="1" Name="Item" PropertyClass="ItemBase">
            <DataType Length="30">String</DataType>
            <ColumnType>ItemType</ColumnType>
            <Required/>
            <LabelStringID>sItem</LabelStringID>
          </Property>
          <Property Name="AbcCode" PropertyClass="AbcCode">
            <DataType Length="1">String</DataType>
            <ColumnType>AbcCodeType</ColumnType>
            <LabelStringID>sABCCode</LabelStringID>
          </Property>
          <Property Name="AcceptReq" PropertyClass="ListYesNo">
            <DataType>Byte</DataType>
            <ColumnType>ListYesNoType</ColumnType>
            <LabelStringID>sAcceptRequirements</LabelStringID>
          </Property>
          <Property Name="AltItem" PropertyClass="Item">
            <DataType Length="30">String</DataType>
            <ColumnType>ItemType</ColumnType>
            <LabelStringID>sAlternateItem</LabelStringID>
            <DomainIDOName>SL.SLItems</DomainIDOName>
            <DomainProperty>Item</DomainProperty>
            <DomainAdditionalProperties>Description</DomainAdditionalProperties>
          </Property>
          ...more properties...
        </IDOProperties>
      </GetPropertyInfo>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
LoadCollection
This XML request/response document uses the LoadCollection method of an IDO to query for information, and returns the results.

LoadCollection requests and responses can be hierarchical. For example, you can get order header and order lines information in a single request, by nesting the order line LoadCollection requests inside the order header LoadCollection request.

Request Data

```
<RequestData>
    <LoadCollection Name="collection">
        <LoadType>loadtype</LoadType>
        <Cursor />
        <PropertyList>
            ... list of properties...
        </PropertyList>
        <RecordCap>recordcap</RecordCap>
    </LoadCollection>
</RequestData>
```

Response Data

```
<ResponseData>
    <LoadCollection Name="collection">
        <Cursor>cursorvalue</Cursor>
        <PropertyList>
            ... list of properties...
        </PropertyList>
        <Items>
            <Item ID="itemID">
                <P>propertyvalue</P>
                <P>propertyvalue</P>
                <P>...etc...</P>
            </Item>
            <Item ID="itemID">
                <P>propertyvalue</P>
                <P>propertyvalue</P>
                <P>...etc...</P>
            </Item>
        </Items>
    </LoadCollection>
</ResponseData>
```

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadCollection</td>
<td>Name - This is the collection name (for example MGCore.GroupNames).</td>
</tr>
<tr>
<td></td>
<td>LoadCap - Applies only to subcollections. This is the maximum number of subcollections to include in the result. Values for LoadCap are:</td>
</tr>
<tr>
<td></td>
<td>-1 - include all subcollections</td>
</tr>
<tr>
<td></td>
<td>1 (the default) - only query the subcollection records for the first parent item</td>
</tr>
<tr>
<td></td>
<td>any other number - specify the maximum number of subcollections to retrieve.</td>
</tr>
</tbody>
</table>
LoadType | Specify one of these types:
---|---
FIRST  
NEXT  
PREV  
LAST  

The first time you submit a LoadCollection request, it must be called FIRST or LAST. For subsequent requests, you can use NEXT or PREV.

Cursor | The response data can include a cursor value that is a “bookmark” into the collection to tell where the retrieval stopped. If you will be making multiple requests using NEXT or PREV to retrieve data, include a single `<cursor />` tag in your FIRST request.

The response document provides a cursor value. Include the response document’s cursor value (as `<cursor>value</cursor>`) in the NEXT request document, so the system can tell where to start the next retrieval request.

Continue updating the cursor value as you provide additional NEXT or PREV requests.

A LAST request should use the empty tag `<cursor />`.

PropertyList | In request data, you only need to list the properties you want returned. The response data mimics the request list.

For example:

```
<PropertyList>
  <Property>
    name1
  </Property>
  ...
</PropertyList>
```

RecordCap | This setting determines how many records are retrieved in one request. Values are:

-1 (the default) - 200 records are retrieved

0 - uncapped - all records are retrieved

any other number - the specified number of records are retrieved.

Item | The ID attribute contains a value used within the system to identify the item.

propertyvalue | For each item, this list contains values corresponding the properties listed above.

LinkBy | Valid only in subcollections. This is similar to the WinStudio LINKBY keyword.

The information within this pair of tags tells how the subcollection is linked to the parent collection. For example:

```
<LinkBy>
  <PropertyPair Child="Item" Parent="Item"/>
  <!-- Optional: more sets of PropertyPair tags -->
</LinkBy>
```
Using XML IDO Request and Response Documents

### Tag Description

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
</table>
| Distinct    | Use this option to retrieve a set of data representing only the distinct combinations of properties requested. This is similar to the WinStudio WinStudio option. For example:  
  ```xml
  <LoadCollection Name="SL_SLItemcusts">
    <LoadType>FIRST</LoadType>
    <PropertyList>
      <CustNum/>
    </PropertyList>
    <Distinct/>
  </LoadCollection>
  ``` |
| Filter      | The filter is used to restrict the result set, similar to the WinStudio FILTERPERM keyword. Any valid SQL where clause syntax is allowed. Some examples:  
  ```xml
  <Filter>CoNum like 'ABC%'</Filter>
  <Filter>QtyOnHand < 20 AND Whse = 'OH'</Filter>
  ``` |
| OrderBy     | Provides a comma-delimited list of properties that specify how the response document result set should be sorted. The DESC keyword may appear after a property name to sort that property descending. For example:  
  ```xml
  <LoadCollection Name="SL_SLItemcusts">
    <LoadType>FIRST</LoadType>
    <PropertyList>
      <CustNum/>
      <CustItem/>
      <Item/>
    </PropertyList>
    <OrderBy>CustNum, Item DESC</OrderBy>
  </LoadCollection>
  ``` |
| PostQueryCmd| Specifies a method to execute once for each row in the result set after the query is completed. This is the equivalent of the WinStudio PQ option in LoadSave overrides and uses the same syntax. For example:  
  ```xml
  <PostQueryCmd>MyPostQuerySp( Property1, Property2, REF Property3 )</PostQueryCmd>
  ``` |
Example 1: LoadCollection
This request retrieves the task type, task code, and description from the first 200 records in the MGCore.TaskTypes collection.

```xml
<IDOResponse>
  <ResponseHeader Type="LoadCollection">
    <ResponseData>
      <LoadCollection Name="MGCore.TaskTypes">
        <PropertyList>
          <TaskType/>
          <TaskCode/>
          <TaskDesc/>
          <NoteExistsFlag/>
        </PropertyList>
        <RecordCap>-1</RecordCap>
        <Items>
          <Item ID="TaskTypes @0.D='2003-04-18 16:31:53.730' @0.P='C444D3B2-FB2E-4416-97E2-D5B6A7F7F7A74'">
            <P>11001</P>
            <P>JC2</P>
            <P>WebSvc test2</P>
            <P>0</P>
          </Item>
        </Items>
      </LoadCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
Example 2: Custom LoadCollection Request
This is a variation of LoadCollection, allowing you to create a custom LoadCollection. It includes this tag:

\[
\text{<CustomLoadMethod Name="methodname"}>
\]

and then a parameters list.

\[
\text{<LoadCollection Name="MGCore.UserNames"}>
\]

Example 3: Nested LoadCollection Request
This request includes subcollection records for all parent items.

\[
\text{<LoadCollection Name="SL.SLItemcusts"}>
\]

Integrating IDOs with External Applications
© 2007, Infor Global Solutions Technology GmbH
**UpdateCollection**

This XML request/response document modifies a collection (inserting, updating, or deleting records) using the UpdateCollection method of an IDO.

When performing an update or deletion request, identify the record(s) you want to update or delete. There are two ways to do this:

- **By item ID.** In order to update or delete by item ID, you will need to perform a LoadCollection first to retrieve the item ID, and then include it in the `<Item>` node’s ID attribute when performing the UpdateCollection update or delete.

- **By the key data.** In this case, specify the attribute UseKeys="Y" on the `<Item>` node and set the appropriate properties to the key data when performing the update or delete.

For insert requests, this information is not needed.

UpdateCollection requests and responses can be hierarchical. For example, you can update the order header and order lines in a single request, by nesting the lines’ UpdateCollection requests inside the header UpdateCollection request.

**Request Data**

```xml
<RequestData>
  <UpdateCollection Name="Collection" RefreshAfterUpdate="Y">
    <Items>
      <Item Action="ActionType" ItemNo="recordnumber" ID="itemID" |
      UseKeys="Y">"
        <Property Name="property" Modified="Y">data</Property>
        <Property Name="property" Modified="Y">data</Property>
        more properties...
      </Item>
      more items...
    </Items>
  </UpdateCollection>
</RequestData>
```

**Response Data**

```xml
<ResponseData>
  <UpdateCollection Name="Collection">
    <Items>
      <Item Action="ActionType" ItemNo="recordnumber" ID="itemID
      Id.RecordDate='date'
      Id.RowPointer='row-pointer'">"
        <Property Name="property">data</Property>
        <Property Name="property">data</Property>
        more properties...
      </Item>
      more items...
    </Items>
  </UpdateCollection>
</ResponseData>
```
### Tag | Description
--- | ---
**UpdateCollection** | The tag can contain these attributes:
- **Name** - This contains the name of the IDO collection.
- **RefreshAfterUpdate** - Y indicates that the response document should show information for inserted or updated items that was refreshed after the update was done. (Deleted items are not refreshed.) N indicates that the response document information is not refreshed after the update.

**Item** | The tag can contain these attributes:
- **Action** - The action type can be INSERT, UPDATE or DELETE. For INSERTs, if the InitiatorType (see page 3-4) is set to "Replication" the insert is treated as an update if the record already exists.
- **ItemNo** - This number is user-defined and optional. It must be a valid 32-bit signed integer. If RefreshAfterUpdate is "Y" and this number is specified, the response includes the record number value, so you can match updated records with the originals and see more easily what has changed.
- **ID** - This value is used within the system to identify the item. When requesting an update or deletion of existing records, include either the item ID or the UseKeys attribute (see below). Omitting the ID attribute defaults to the same behavior as UseKeys="Y".
- **UseKeys** - Y (the default value) indicates that the update or delete request will use key data rather than item IDs. N indicates that the request will include item IDs.

**Property** | Include in your request any properties that you plan to update, as well as all nonnullable properties that do not have a default value. You can omit properties that have a default value and that will not change as a result of the update.
- The tag can contain these attributes:
  - **Name** - This contains the name of the property.
  - **Modified** - This value specifies whether the property was updated. Only values flagged as "Modified=Y" will be updated in the database.

**LinkBy** | Valid only in subcollections. Similar to the WinStudio LINKBY keyword.
- The information within this pair of tags tells how the subcollection is linked to the parent collection. For example:
  ```xml
  <LinkBy>
    <PropertyPair Child="Item" Parent="Item"/>
    <!-- Optional: more sets of PropertyPair tags -->
  </LinkBy>
  ```

---

**NOTE** You may want to include in your request some items or properties that you are not directly updating. For example, a total related to a quantity that was modified may change as a result of the quantity changing. Your request should therefore include the total, as well as the quantity.
Example 1: UpdateCollection - Insert, Update and Delete Items

<IDOResponse>
  <ResponseHeader Type="UpdateCollection">
  </ResponseHeader>
  <ResponseData>
    <UpdateCollection Name="MGCore.TaskTypes"
      RefreshAfterUpdate="Y">
      <Items>
        <Item Action="Insert" ItemNo="0">
          <Property Name="TaskType" Modified="Y">11001</Property>
          <Property Name="TaskCode" Modified="Y">JC2</Property>
          <Property Name="TaskDesc" Modified="Y">WebSvc test2</Property>
        </Item>
        <Item Action="Update" ItemNo="1">
          ... properties...
        </Item>
        <Item Action="Delete" ItemNo="2">
          ... properties...
        </Item>
      </Items>
    </UpdateCollection>
  </ResponseData>
</IDOResponse>
Example 2: UpdateCollection - Nested Insert of Group and Users
This request inserts a group and the users in the group. Note also the error message in the response.

```xml
<IDOResponse>
  <ResponseHeader Type="UpdateCollection">
    <ResponseData>
      <UpdateCollection Name="MGCore.GroupNames" RefreshAfterUpdate="Y">
        <Items>
          <Item Action="Insert" ItemNo="0">
            <Property Name="GroupId" Modified="N"></Property>
            <Property Name="GroupName" Modified="Y">TestUsers</Property>
            <Property Name="GroupDesc" Modified="Y">a test group of users</Property>
          </Item>
          <Item Action="Insert" ItemNo="1">
            <Property Name="GroupId" Modified="N"></Property>
            <Property Name="UserId" Modified="Y">7</Property>
            <Property Name="Username" Modified="N"></Property>
          </Item>
          <Item Action="Insert" ItemNo="2">
            <Property Name="GroupId" Modified="N"></Property>
            <Property Name="UserId" Modified="Y">8</Property>
            <Property Name="Username" Modified="N"></Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
Example 3: UpdateCollection - Update Using Item IDs

This request updates existing records using the "ID" attribute. To find a specific ID, perform a LoadCollection request on the data before performing the UpdateCollection.

```xml
<IDORequest>
  <RequestHeader Type="UpdateCollection">
    <RequestData>
      <UpdateCollection Name="MGCore.TaskTypes" RefreshAfterUpdate="Y">
        <Items>
          <Item Action="Update" ItemNo="0" ID="TaskTypes @0.D='2004-03-08 15:18:35.897' @0.P='32537FE4-64D5-4BD5-8C9B-8A5EA27D6A33'">
            <Property Name="TaskType" Modified="N">11001</Property>
            <Property Name="TaskCode" Modified="N">T001</Property>
            <Property Name="TaskDesc" Modified="Y">Updated TaskCode</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>

<IDOResponse>
  <ResponseHeader Type="UpdateCollection">
    <ResponseData>
      <UpdateCollection Name="MGCore.TaskTypes">
        <Items>
          <Item Action="Update" ItemNo="0" ID="PBT=TaskTypes TaskTypes.RecordDate='2004-05-11 15:25:31.753' TaskTypes.RowPointer='32537FE4-64D4-4BD5-8C9B-8A5EA27D6A33'">
            <Property Name="TaskDesc">Updated TaskCode</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
Example 4: UpdateCollection - Update Using Keys
This request updates existing records using the UseKeys attribute:

```xml
<IDOREquest>
  <RequestHeader Type="UpdateCollection">
    <RequestData>
      <UpdateCollection Name="MGCore.TaskTypes" RefreshAfterUpdate="Y">
        <Items>
          <Item Action="Update" ItemNo="0" UseKeys="Y">
            <Property Name="TaskType" Modified="N">11001</Property>
            <Property Name="TaskCode" Modified="N">T001</Property>
            <Property Name="TaskDesc" Modified="Y">Updated TaskCode</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </RequestData>
  </RequestHeader>
</IDOREquest>

<IDOResponse>
  <ResponseHeader Type="UpdateCollection">
    <ResponseData>
      <UpdateCollection Name="MGCore.TaskTypes">
        <Items>
          <Item Action="Update" ItemNo="0" ID="PBT=TaskTypes TaskTypes.RecordDate='2004-05-11 15:25:31.753' TaskTypes.RowPointer='32537FE4-64D4-4BD5-8C9B-8A5EA27D6A33'">
            <Property Name="TaskDesc">Updated TaskCode</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
Invoke

This XML request/response document invokes a method published by an IDO.

Request Data

```xml
<RequestData>
  <Name>IDOName</Name>
  <Method>MethodName</Method>
  <Parameters>
    <Parameter Seq="0">PRIMARY</Parameter>
    <Parameter Seq="1" ByRef="Y"/>
    <Parameter Seq="2" ByRef="Y"/>
    <Parameter Seq="3" ByRef="Y"/>
  </Parameters>
</RequestData>
```

Response Data

```xml
<ResponseData>
  <Name>IDOName</Name>
  <Method>MethodName</Method>
  <ReturnValue>n</ReturnValue>
  <Parameters>
    <Parameter>PRIMARY</Parameter>
    <Parameter ByRef="Y">n</Parameter>
    <Parameter ByRef="Y">parm</Parameter>
    <Parameter ByRef="Y"/>
  </Parameters>
</ResponseData>
```

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the IDO that contains the method</td>
</tr>
<tr>
<td>Method</td>
<td>The name of the method to invoke</td>
</tr>
<tr>
<td>ReturnValue</td>
<td>In the response, this contains the return value from the method.</td>
</tr>
<tr>
<td>Parameter</td>
<td>The name of a parameter for the method. The tag can contain these attributes:</td>
</tr>
<tr>
<td></td>
<td>VT - The variant data type of the parameter (see Microsoft documentation for more information). Optional.</td>
</tr>
<tr>
<td></td>
<td>Seq - The parameter sequence</td>
</tr>
<tr>
<td></td>
<td>ByVal - If Y, the parameter is input/output. If N, or if the attribute is omitted, the parameter is input-only.</td>
</tr>
</tbody>
</table>
**Example - Invoke**

This request determines whether the Group ID "Developers" is valid.

```xml
<IDORequest>
  <RequestHeader Type="Invoke">
    <RequestData>
      <Name>MGCore.GroupNames</Name>
      <Method>GroupValid</Method>
      <Parameters>
        <Parameter Seq="0">Developers</Parameter>
        <Parameter Seq="1" ByRef="Y"/>
        <Parameter Seq="2" ByRef="Y"/>
        <Parameter Seq="3" ByRef="Y"/>
      </Parameters>
    </RequestData>
  </RequestHeader>
</IDORequest>

<IDOResponse>
  <ResponseHeader Type="Invoke">
    <ResponseData>
      <ReturnValue VT="2">16</ReturnValue>
      <Parameters>
        <Parameter VT="8" Seq="0">Developers</Parameter>
        <Parameter VT="1" ByRef="Y" Seq="1"/>
        <Parameter VT="1" ByRef="Y" Seq="2"/>
        <Parameter VT="8" ByRef="Y" Seq="3">The Group Name entered is not valid.</Parameter>
      </Parameters>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
**CloseSession**

This XML request/response document closes the current session.

**Request Data**
none

**Response Data**
none

**Example - CloseSession**

```xml
<IDOREquest>
    <RequestHeader Type="CloseSession"/>
</IDOREquest>

<IDOREsponse>
    <ResponseHeader Type="CloseSession"/>
</IDOREsponse>
```
**BeginTrans, CommitTrans, RollbackTrans**

Use the BeginTrans, CommitTrans and RollbackTrans requests together to perform transaction management; that is, to wait until you verify that a request is successful before you commit it to the database. If some part of the request fails, the RollbackTrans request rolls back the entire update.

A single request such as UpdateCollection or Invoke already runs in a transaction, so the Begin/Commit/Rollback transaction requests are not necessary. However, when you want to make multiple requests run in a single transaction (for example, UpdateCollection, Invoke, UpdateCollection, and so on), you can use the Begin/Commit/Rollback transaction requests.

**Use These Requests in an Interactive Session**

These request types are designed to work best as part of an interactive application with a single request per document. For example, you may not know whether you want to commit an update until you know that the update request was successful.

An interactive session could send the requests individually, inspecting the results of each request before sending the next. That way if you had several requests following a BeginTrans request, your code could decide if it should send a CommitTrans or RollbackTrans.

Any request can be sandwiched in a set of transaction requests. Some requests, such as Get PropertyInfo, may not be affected by the transaction, but it does no harm. It is possible for a LoadCollection request to include a custom load method that actually updates the database, so it may be desirable to include that type of request in a batch of other requests in the same transaction.

**Request Data**

```xml
<RequestHeader Type="BeginTrans"/>
<RequestHeader Type="CommitTrans"/>
<RequestHeader Type="RollbackTrans"/>
```

**Response Data**

```xml
<ResponseHeader Type="BeginTrans"/>
<ResponseHeader Type="CommitTrans"/>
<ResponseHeader Type="RollbackTrans"/>
```

**Example - Transaction Commit/Rollback**

An interactive application sends an UpdateCollection request and an Invoke request as a single transaction:

```xml
<RequestHeader Type="BeginTrans"/>
<RequestHeader Type="UpdateCollection">
  ...update collection request...
</RequestHeader>
<RequestHeader Type="Invoke">
  ...invoke request...
</RequestHeader>
```
The response document indicates the outcome of both of the requests in the transaction:

```xml
<ResponseHeader Type="BeginTrans"/>
<ResponseHeader Type="UpdateCollection">
  ...update collection response...
</ResponseHeader>
<ResponseHeader Type="Invoke">
  ...invoke method response...
</ResponseHeader>
```

The application checks the values in the response. If the responses indicate success, the next request from the application would be:

```xml
<RequestHeader Type="CommitTrans"/>
```

If the responses indicate failure, the entire transaction should be rolled back, so the next request from the application would be:

```xml
<RequestHeader Type="RollbackTrans"/>
```
Passing Dates to the IDO Runtime

When passing dates to the IDO runtime (through XML documents, Workflow, .NET or programmatically through another application) always use this locale-neutral format:

YYYYMMDD HH:MM:SS  (.mmm is optional)

Other formats may work for all or some regional settings, depending on how they are used. But the above format will always work.

Decimal Values

Decimal properties generated by the IDO runtime are always formatted using a period for the decimal place, and no separators. It is up to the receiving program to transform data to the proper locale settings.

SyteLine users should not redefine the decimal character for US English regional settings. Instead they should choose a non-US regional setting that matches the display formats they want to use.

Calling Dynamic IDOs in XML Documents

XML documents can directly query or update the SQL database tables by calling dynamic IDOs. Through dynamic IDOs, you can also directly invoke stored procedures from the SQL database. To access a dynamic IDO, use the standard XML syntax, except:

- For collection requests, instead of the collection name, you specify the table name (in the format TABLE!table_name). For example, the XML UpdateCollection request could contain a tag like this:

  `<UpdateCollection Name="TABLE!item">
  The properties specified in the calls are actually the column names in the table.

- When invoking a method, instead of the IDO name, specify "SP!" and instead of the Method name, specify the stored procedure name. For example, the XML Invoke request would contain these tags:

  `<Name>SP!</Name>
  `<Method>CalcHrsSp</Method>

- When you open the session where you make these requests, you specify a configuration. The configuration points to an application database. This is the database where the requests will look for the tables and stored procedures.

You can also invoke stored procedures by using dynamic IDOs in custom events defined for SyteLine forms. For more information about this, see the Modifying SyteLine ERP guide.
Using Workflow to Access IDOs

SyteLine’s implementation of Workflow includes a SyteLine MSMQ queue (fspost) and queue listener service (WFPost.exe) to support the Workflow IDO Declarative Interface. Also, an Exchange mailbox (default alias: syteline) is set up to monitor emails. When the mailbox receives one of the special IDO message classes described on page 4-3, it knows how to handle the message in order to get the request to the system.

Workflow allows you to intervene and automate certain processes. For example, you could create a workflow to handle this business rule: whenever an order line is created for a certain customer, notify the sales manager. Or, if an order line is more than $10,000, do not take the order until a manager signs off on it.

Using Workflow, you can query and update data or invoke IDO methods. For example, if your company needs additional information about a customer that is not available from the Customer Orders form, and an action in that form triggers a workflow, your can create a step in the workflow to query the SLCustomers IDO for the additional data.

Using Workflow message classes and the Workflow IDO Declarative Interface allows you to interact with SyteLine IDOs without scripting. The rest of this chapter describes how to do this.

NOTE
Sample Workflow templates are provided that show how to perform certain common business processes using Workflow and IDO requests. For more information, see Appendix A, “Sample Workflow Templates for Automating SyteLine Business Processes.”

Prerequisites - Workflow

Before you can use Workflow templates to pass data to and from IDOs, you must set up the system as described in Chapter 2, “Web, Intranet, and User Setup.”

You must also set up Workflow as described in the SyteLine ERP Installation Guide.

You may also want to read the SyteLine Workflow Training Guide, available through the Infor KnowledgeZone web site.
Using the Workflow IDO Declarative Interface in Templates

**Creating Templates**

From a Workflow-enabled client, open Microsoft Outlook. Navigate to the All Public Folders\MAPICS\Workflow\Templates folder and select it. Select **Actions>New Flow Template** from the Outlook menu. In the Flow Template Properties dialog, specify at least this information:

- **Title** - the name of the template. This should match the name you specify in the Workflows form (page 4-14).
- **Start flows using this server** - the MAPI profile which points to the Keyflow Server mailbox. This is **not** the Exchange server listed in the Intranets form. By convention, the name of the Keyflow server mailbox is "workflow."
- **Place started flows in this folder** - the path to the folder where active flows are stored. This should match the Active Flows folder field in the Work Flows form (page 4-14). By convention, this folder is set to All Public Folders\MAPICS\Workflow\Active Flows.
- **When flows are completed** - what to do with the flow when it is complete. This is up to you; you may want to save the flows in a "Completed Flows" folder. By convention, this folder is set to All Public Folders\MAPICS\Workflow\Completed Flows.
- **Send flow manager’s mail to** - where to send e-mail from this flow. Generally, you should select **Originator of flows from this template**. The originator is the SyteLine user who causes a flow to be launched by saving a record on a form for which Workflow is enabled (enabling a form is handled through the Work Flows form; see page 4-14). However, you could also set this flow’s e-mail to go to a specific mailbox, if an administrator wishes to get all manager messages instead of the originator receiving them.

Refer to the Flow Designer Guide online help for more information about creating a workflow template and adding steps.

To function properly, this template must be stored in the path specified in the Intranets form’s Workflow Templates Public Folders path (see page 2-2). By convention, this is set to All Public Folders\MAPICS\Workflow\Templates).
About IDO Message Classes in Workflow Steps
You can create Workflow steps that call any method on any IDO in the SyteLine system, or that query or update any data in the system. Return data is placed in the Application Data tab of the Workflow message.

One Method/Collection Per Step
You can use the following types of message classes in Workflow IDO steps. With these message classes, you can call only one method or collection per step. You must use "Response" type steps.

- **LoadCollection** - This class is used for querying and retrieving data from an IDO into a workflow. The LoadCollection class includes:
  - IDO name (example: SL.SLVendors)
  - a list of properties
  - a filter - can be multi-row, but only the first row is returned. This filter should be meaningful for the data. A sorting clause like "ORDER BY field DESC" can be used to return the first or last row. Filters can also use the SQL wildcard % character. For example, the following phrase would find the last order number beginning with W: FILTER(CoNum LIKE 'W%' ORDER BY CoNum DESC)
  - SET - tells which values to update or create in the flow’s AppData. The AppData will be created with the proper data type for returned values based on what is entered here. If AppData is needed for input, it must be defined in the flow before it is needed.

  Following is an example LoadCollection request:

  ```
  SL.SLVendors(
  PROPERTIES(VadName, VadCity, VadZip)
  FILTER(VendNum = APPDATA(VendNum))
  SET(APPDATA(VendorName)=VadName,APPDATA(VendorZipCode)=VadZip))
  ```

- **InvokeMethod** - This class is used for doing calculations or starting processes, or updating some types of data in the SyteLine application. Success or failure is indicated through AppData.

  Following is an example InvokeMethod request:

  ```
  SL.SLVendlogs.FormatVendorAddressSp(
  PARMS(
    APPDATA(VendNum),
    RVAR APPDATA(VendAddress),
    MESSAGE))
  ```

- **fsSLWorkflow** - This class is used for task steps that will update SyteLine based on an attached IDO request XML. The flow in which this task runs normally is launched when saving a record in a Workflow-enabled SyteLine form.

- **UpdateCollection** - This class updates data in a collection.

  Following is an example UpdateCollection request:

  ```
  SL.SLVendors(
  ACTION(UPDATE)
  SET(VendNum=APPDATA(VendNum),VadName=APPDATA(VendorName),
  VadZip=APPDATA(VendorZipCode)))
  ```

When SyteLine receives a Workflow email in its inbox, the Workflow Posting Service (WFPost.exe) knows what type of action to perform on these message types.
Multiple IDO Calls per Step
The message class IDORequest allows you to perform multiple IDO calls from a single template step. The IDORequest message class has a body composed of multiple instances of IDO requests, in this format:

```
REQUEST(InvokeMethod | UpdateCollection | LoadCollection)
{
    Appropriate request body
}
```

Following are some examples of multiple requests that could be used in one IDORequest message class:

```
REQUEST(InvokeMethod)
{
    SL.SLVendlogs.FormatVendorAddressSp(
        FARMS(
            APPDATA(VendNum),
            RVAR APPDATA(VendAddress),
            MESSAGE))
}

REQUEST(LoadCollection)
{
    SL.SLVendors(
        PROPERTIES(VadName, VadCity, VadZip)
        FILTER(VendNum = APPDATA(VendNum))
        SET(APPDATA(VendorName)=VadName,APPDATA(VendorZipCode)=VadZip))
}

REQUEST(UpdateCollection)
{
    SL.SLVendors(
        ACTION(UPDATE)
        SET(VendNum=APPDATA(VendNum),VadName=APPDATA(VendorName),
            VadZip=APPDATA(VendorZipCode)))
}
```

Each request may build on the AppData generated by previous requests. For example, a LoadCollection may fill in a VendNum AppData variable given an order number, and a subsequent UpdateCollection may update an SLVendors row using the VendNum previously fetched as a key.

Finding the IDO Collection or Method
Finding an IDO collection or method that does what you want may be the most difficult part of this process. There are several ways to approach this, as described below.

Using ObjectStudio
If ObjectStudio is available to you, find the proper IDO and then list the collections, properties, and methods.

Using the DataMap
If the DataMap is available, use its spreadsheets to find the properties and methods in a collection, and then to find the forms where that collection is used.
Using Workflow to Access IDOs

Using Edit Mode on a Form to Find a Method and its Parameters
1. Figure out which SyteLine form uses the method you want. For example, if you want to create a workflow that does the same thing as the Pricing button on the Pricing form, go to that form. Sometimes a method doesn’t appear on any form; in that case, find a form that includes a collection associated with the method.

2. Once you find the proper form, select Edit>Event Handler, which displays all the methods and other events associated with the form.

3. Figure out which method is the one you need. This can be tricky if there are several similar methods on the form; you may need to examine the method’s parameters or the underlying stored procedure in order to determine which method you want. If you are not experienced in reading code, contact Infor Services for assistance.

4. Select the method and click Edit.

5. In the Event Handler Properties dialog, select the entire string from theParms field. This is the information you want to copy into your Workflow step’s message text area, as a starting point for your new step. (The text will need to be massaged as described in “Creating the Workflow IDO Message Step” on page 4-7.)

Using Edit Mode on a Form to Find a Collection and its Properties
When creating a LoadCollection message, you need to include the name of the collection and the properties to be returned. To build a query that can be used in a message:

1. Figure out which form includes the collection you want. In this example, the Customers form uses the SLCustomers collection.

2. Open the form and go into Edit mode.

3. Double-click on a component with a drop-down arrow.
4. In the Form Component Properties dialog, select an appropriate Component Class (in this case, Item) and click **Derive From**.

5. In the Component Class Properties dialog's Data tab, click **List Source**.

6. In the Edit List Source Specification dialog, make sure Type is set to **IDO Collection** and the IDO you want (in this case, SL.SLCustAddrs) is selected.

7. Click **Properties**.

8. Select properties and click **Add** or **Remove** to set the properties you want to include in your query.
9. Click OK to return to the Edit List Source Specification dialog.

10. If you want to set up a filter, click Filter, etc. and specify what you want to filter for. Our example will filter by Customer Sequence.

11. Click OK twice to return to the Component Class Properties dialog.

12. Copy the entire string from the List Source field into your system clipboard, and cancel out of the Edit mode dialogs without saving anything. This is the information you want to copy into your Workflow step’s message text area, as a starting point for your new step. (The text will need to be massaged as described later in “Creating the Workflow IDO Message Step.”)

**Creating the Workflow IDO Message Step**

1. Open the Workflow template where this step will be placed.

2. Create the new step. Use these step naming conventions:
   - For methods: IM MethodName where MethodName is the method name minus the Sp ending (for example, IM Pricing).
   - For load or update collections: LC Description or UC Description where Description is a brief description of the type of information in the collection (for example, LC CO Info).
   - For IDO requests: ID Task where task describes the overall task being handled by the methods and Load/Update collections in the IDO request.
Using Workflow to Access IDOs

3. In the step properties Message tab, enter the message string, using this syntax:

```
For LoadCollection messages:
Collection{
  PROPERTIES(Property, Property, ..., Property)
  FILTER(any SQL Filter - can include "LIKE", ",", and "ORDER BY")
  SET(APPDATA(Property)=InputAppDataColumn, APPDATA(Property)=
      InputAppDataColumn, APPDATA(Property)=InputAppDataColumn,
      APPDATA(RetValue)=OutputAppDataColumn)

Property will be created as AppData or updated if it exists.

Tip: Sometimes you may want to have more than one AppData field hold information
such as the before and after value. For example, SET(APPDATA(OldName) = name,
APPDATA(NewName) = name) would create AppData for "OldName" and
"NewName".

For InvokeMethod messages:
MethodSp{
  PARMS(VAR APPDATA(InputParameter), VAR APPDATA(InputParameter), ..., 
         RVAR APPDATA(OutputParameter), MESSAGE))

For UpdateCollection messages:
Collection{
  ACTION(UPDATE | INSERT | DELETE)
  SET(Property1=APPDATA(InputParameter1),
       Property2=APPDATA(InputParameter2), ...)

  The keys for an update or delete are specified within the SET statement in the same
  manner as the properties to be updated. For example:

  SL.SLCustomers{
    ACTION(UPDATE)
    SET(CustNum=APPDATA(MyCustomer),
         City=APPDATA(NewCity))

  Since CustNum is a key field in SL.SLCustomers, the AppData value of MyCustomer
  will be used to look up the customer, and the customer’s City will be updated with the
  AppData value of NewCity.

For IDORequest messages:
See the syntax examples for IDORequest messages on page 4-4.

Data types and case-sensitivity are critical; "Item" and "item" are different
AppData variables.
Editing Strings Copied from SyteLine Forms

If you are pasting a string you copied from within a SyteLine form in edit mode, make the following changes to the string:

- Edit the string to make it appropriate for a workflow step, adding SET and APPDATA statements as needed to indicate initial and return values
- Replace each input variable (VAR P, C, or V) with VAR APPDATA. For example:

  VAR P(CoLine) becomes
  VAR APPDATA(CoLine)

- Replace each return variable (RVAR P, C, or V) with RVAR APPDATA. For example:

  RVAR P(RefNum) becomes
  RVAR APPDATA(RefNum)

  Input and return variables may not have the same name.

- If the string contains types (such as RSTR, BSTR, or INT), they need to be changed to either VAR, for input variables, or RVAR, for return variables.
- Remove any ERRORMESSAGE or SUCCESSMESSAGE parameters at the end of the method string.
- MESSAGE parameter - Stored procedures have an infobar or return parameter. If the call returns an error, it populates this parameter. The error shows up as a failure in the Workflow status. You can display the comments (pencil icon) and see the error message that was returned.
- All parameters, both input and return, must be specified in the call.

NOTE: Ask Infor Services about a utility that simplifies this process.

Specifying Literal Values in Messages

AppData is not the only way to specify values when updating collections. You can use the LIT keyword as well. For example:

```
SL.SLCustomers(
  ACTION(UPDATE)
  SET(CustNum=APPDATA(MyCustomer),
      City=LIT(Poughkeepsie) )
)
```

This example updates the customer specified by the AppData value of MyCustomer, setting the City property to “Poughkeepsie”. LIT is supported in:

- non-RVAR parameters in InvokeMethod requests
- the FILTER clause in LoadCollection requests
- the SET clause in UpdateCollection requests
4. Set the To: field to SyteLine’s mailbox (in this case, “SyteLine”). The completed Message tab should look like this example:

5. In the Possible Responses area, include Success and Failure responses. This is optional, but it is useful if other steps rely on the completion of this step as a prerequisite.

6. In the Forms tab, specify the message class: InvokeMethod, LoadCollection, UpdateCollection, or IDORequest.

7. Click OK to close and save the properties.

8. In the template’s Application Data tab, list each of the input APPDATA variables from the string you specified in Step 3. (Ask Infor Services about a utility that will simplify this step.)
   - The input parameters must be in the App Data tab before the step executes, but the return parameters will be dynamically created if they do not already exist in App Data. Return variables are not populated until data is returned from the call.
   - Also, you can go to a completed workflow, export its AppData as XML, and then import the variables you want into this template.
   - Boolean properties in SyteLine (for example, the ListYesNoType) are type tinyint and will now appear as integers in the workflow's AppData. To force them to appear as Yes/No in the AppData, you need to add them manually to the template's AppData with the correct (Boolean) data type. Then their values will be updated correctly from SyteLine, while being handled as Boolean in the workflow.

9. For an InvokeMethod, LoadCollection, or UpdateCollection request, if there is no attached IDORequest document, you must specify the following in the AppData:
• IDOResultConfiguration - the configuration (site) name against which the request will be executed.

• IDOResultUserName - the SyteLine user ID used to execute the request.

• IDOResultPassword - the password for the user, encrypted or unencrypted (if encrypted, this uses the same format/algorithm as is used to store the password in the UserNames table).

• IDOResultIsPasswordEncrypted - "Yes" or "No" to specify whether the IDOResultPassword is encrypted; defaults to "No" if not specified.

10. Test this step before creating the rest of the workflow.

11. Create and link the other steps and test the workflow.

Examples: Using Methods or Collections in Steps

The following examples show collection or method parameters from SyteLine IDOs used in a Workflow step.

Query Customer Order Information (LC CO Info)

This example creates a query against the SLCos collection that finds the first record that matches the order number specified in AppData and returns the customer number, contact, phone, and order number. The message properties are:

Filter to Find a Specific Customer Order

This example uses a SQL filter to find the last customer order that starts with W:
Method to Get Item Warehouse Details
This example takes the item and warehouse specified in AppData and returns the quantity on hand, safety stock, location, and whether a cycle count is in process or has been triggered.

Method to Delete a Customer Order Line
This example deletes the customer order line specified by the App Data. It does not return anything to the App Data.

Accessing Dynamic IDOs in Workflow Templates
Workflow messages can directly query or update the SQL database tables by calling dynamic IDOs. Through dynamic IDOs, you can also directly invoke stored procedures from the SQL database. To access a dynamic IDO, use the standard message syntax, except:

- For collection requests, instead of the collection name, you specify the table name (in the format TABLE!table_name). The properties specified in the calls are actually the column names in the table.
- When invoking a method, instead of the IDO name, specify "SP!" and instead of the method name, specify the stored procedure name.

Communicating with User-Extended Tables and User-Defined Fields through Workflow
Any UET field on a form can be sent as App Data when a workflow requesting that data launches from a SyteLine form.
Loading Old Values

With some older versions of SyteLine, the original AppData was placed into AppData fields with the extension ‘_old’. This was done so the user could compare the new value to the original (‘_old’) value. However, it is best not to unnecessarily overload AppData with data that might not be used.

When you use the IDO LoadCollection call, the old value remains in the database and can be easily retrieved. If required, the LoadCollection call could even create the ‘_old’ AppData variables as in this example (see the last statement):

```plaintext
APPDATA(old_Contact)=Contact).
SL.SLCos(
  PROPERTIES(CoNum, Contact)
  FILTER(CoNum='APPDATA(CoNum)' )
  SET(APPDATA(CoNum)=CoNum), APPDATA(old_Contact)=Contact))
```

Viewing Error Messages

Errors for IDO calls, like any other posting errors, are placed in the comments of the response reply and can be viewed in the running or complete flow.

Sample Templates

Sample Workflow templates that handle some of the most requested SyteLine functions are included with your SyteLine installation. You can copy the steps from these sample templates to modify and use in your workflows.

For more information about the sample templates, see Appendix A, “Sample Workflow Templates for Automating SyteLine Business Processes.”
Launching a Workflow

There are several ways to launch a workflow. The following examples should help you better understand how to launch your own workflows.

Launching a Workflow from a Form

Whenever a user takes a standard action (insert, update, or delete) on a record in a SyteLine form, a Workflow template can be initiated automatically. This method of launching is defined on the Work Flows form.

However, if your system has customizations that bypass the standard actions — for example, if you have set up custom methods instead of using insert, update, or delete — this standard launch method will not be available.

Setting Up Workflow Rules

Use the Work Flows form to set up rules for launching the template. In the example screen shown below, the workflow template “sellsteps” will be launched whenever a user on the Customer Orders form adds a record. Since Suspend is selected, the workflow takes control of the data before the record is saved to the database.

This form has the following fields:

- **Form Name** - The name of the form where the workflow will be launched.
- **Action** - The action a user takes on the form that triggers the workflow (add, update or delete records)
- **Active Flows Folder** - The path to the Outlook public folder to use for this type of active flows. (You may want to have separate Active Flows folders for different areas; for example, one for PO activity and another for Customer Order activity.) Right-click in this field and select Find to open a dialog box showing a list of available public folders on the Exchange server. This should match the information you entered in the template’s Properties dialog (page 4-2).
- **Suspend** - Select this field to prevent the application from immediately saving changes to the database when a user changes a record on the form. Select the Suspend field for each appropriate action type (add, update, or delete), to indicate which types of changes you want to suspend.

For a workflow that posts data to the application database, you must select the Suspend field. This suspends the data from being posted until the workflow steps complete. (Actually, behind the scenes, the system commits the data to validate it, then rolls it back.) For example, the workflow may include steps that prevent an order from being saved until a supervisor approves and posts it. If Suspend was not selected, the application would save the order immediately.

Clear this field if you want the application to save changes to the database immediately when a user adds, updates, or deletes a record on the associated application form. The workflow is then launched asynchronously.
• **Flow Name** - The name of the Workflow template that the application will launch when a user adds, updates, or deletes (depending on the action type selected) a record on the associated form. This should match the title you entered in the template’s Properties dialog (page 4-2).

**Setting Up User E-mails**
Be sure that a valid e-mail address is set up in the Users form for every user who may perform actions on any form where a workflow might be invoked.

**Registering Workflow Templates**
SyteLine includes a Workflow Templates form where system administrators can register the Workflow templates they want to allow to access the IDO Declarative Interface. You can click on the down-arrow to display a list of the templates available in the Exchange public folders.

Any templates not in your list will be denied access. This, along with the MS Exchange security measures that Administrators should implement when setting up the system, protects your system from intruders.

> **NOTE**

These templates must be moved to the folder specified on the Intranets form in order to be registered, but they then could be moved to subfolders to run.
Using Workflow to Access IDOs

**Launching a Flow: Process Diagram**

1. Record saved on Workflow-enabled SyteLine form
2. SyteLine IDO creates XML document and places it in queue
3. XML Request Document
4. Workflow Queue Listener reads document and uses the FLOW COM based APIs to launch a flow
5. Workflow runs. If Workflow posts data back to SyteLine, see Posting diagram

**Posting from a Flow: Process Diagram**

1. Workflow performs a posting request, placing the message in the inbox
2. Exchange Inbox (usually alias "sytleine")
3. Exchange Inbox Agent puts the message ID in fspost queue
4. Workflow Posting Service (WfPost):
   1. sees the message ID, then makes a CDO (MAPI) call to the Exchange server and mailbox (specified on the SyteLine Intranets form)
   2. using the message ID, queries the message for class
   3. if class is LoadCollection, UpdateCollection, InvokeMethod, IDORequest, or fsSLworkflow, reads the rest of the attributes from the message and processes it. Otherwise the message is ignored
   4. takes the data from the message, parses it, and communicates with the requested SyteLine IDO
   5. if the IDO request produces an error, rolls back changes. If the request is successful, sends "success" reply to Exchange Server so the workflow can continue.
Using the .NET Web Service API

It is possible to use Visual Basic 7 or C# code to make calls over the Internet that load or update SyteLine’s IDO collections, or that run IDO methods. The .NET web service accesses SyteLine through the XML interface. It allows the following API calls to the IDO:

- **GetConfigurationNames** - returns a list of valid configuration names
- **CreateSessionToken** - validates your user ID password and configuration and return a session token
- **LoadDataSet** - Returns data through an IDO.
- **SaveDataSet** - Updates data through an IDO.
- **CallMethod** - Calls an IDO method.

Use the following prototypes when creating your API calls.

**NOTE**

All prototypes are using C# code.

**Prerequisites - .NET**

Before you can use .NET Web APIs to transfer data between your external interface and SyteLine, you must set up the system as described in Chapter 2, "Web, Intranet, and User Setup."

**API Calls**

**GetConfigurationNames**

Returns a list of valid configuration names.

**Syntax**

```csharp
public string[] GetConfigurationNames()
```

**Parameters**

None

**Return Value**

Returns all valid configuration names.

**Example**

```csharp
WebService webService = new WebService();
string[] astrConfigNames;
webService.Url = "http://<web-service-url>";
astrConfigNames = WebService.GetConfigurationNames();
```
CreateSessionToken
Validates your user ID password and configuration and returns a session token.

When any of the other web service methods (LoadDataSet, SaveDataSet, or CallMethod) are called, the session token that was returned from CreateSessionToken is a required parameter. This token is used to open a SyteLine session for each call to a web service method, and the session is closed before the web service method returns.

The SyteLine user that is passed as a parameter to the CreateSessionToken must have the user module called "SyteLineAutomation" for the API call to work. This can be added on the User Modules form in SyteLine.

Syntax

```
public string CreateSessionToken(string strUserId,
                                 string strPswd,
                                 string strConfig )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strUserId</td>
<td>A valid User ID for SyteLine.</td>
</tr>
<tr>
<td>strPswd</td>
<td>The password for the application user.</td>
</tr>
<tr>
<td>strConfig</td>
<td>A valid configuration name used in the application.</td>
</tr>
</tbody>
</table>

Return Value

Returns a session token string, required by other web service methods.

Example

```
WebService webService = new WebService();

    webService.Url = "http://<web-service-url>";
    SessionToken = webService.CreateSessionToken(
        "sa",
        "secret",
        "SyteLine" );
```
LoadDataSet
Queries data through an IDO.

Syntax

```csharp
public DataSet LoadDataSet(string strSessionToken,
                           string strIDOName,
                           string strPropertyList,
                           string strFilter,
                           string strOrderBy,
                           string strPostQueryMethod,
                           int iRecordCap)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strSessionToken</td>
<td>A session token string created by calling CreateSessionToken.</td>
</tr>
<tr>
<td>strIDOName</td>
<td>The name of the IDO used to load the data set.</td>
</tr>
<tr>
<td>strPropertyList</td>
<td>A comma-delimited list of property names to include in the data set.</td>
</tr>
<tr>
<td>strFilter</td>
<td>A filter used to query records for the DataSet.</td>
</tr>
<tr>
<td>strOrderBy</td>
<td>A comma delimited list of property names used to order the DataSet.</td>
</tr>
<tr>
<td>strPostQueryMethod</td>
<td>The name of a method to run for each row in the result set after the query is completed.</td>
</tr>
<tr>
<td>iRecordCap</td>
<td>Caps the number of records to returned by the query:</td>
</tr>
<tr>
<td></td>
<td>0 = unlimited</td>
</tr>
<tr>
<td></td>
<td>-1 = default (200)</td>
</tr>
</tbody>
</table>

Return Value

Returns a System.Data.DataSet containing the results of the query.

Example

```csharp
WebService webService = new WebService();
DataSet IdoDataSet;

webService.Url = "http://<web-service-url>";
IdoDataSet = webService.LoadDataSet(
    SessionToken,
    "SL.SLItems",
    "Item, Description, UnitCost",
    "Item = N'FA-10000'",
    ",", ",", -1 );
```
**SaveDataSet**

Updates data through an IDO.

**Syntax**

```csharp
public DataSet SaveDataSet(string strSessionToken,
    DataSet UpdateDataSet,
    bool bRefreshAfterSave,
    string strCustomInsert,
    string strCustomUpdate,
    string strCustomDelete)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strSessionToken</td>
<td>A session token string created by calling CreateSessionToken.</td>
</tr>
<tr>
<td>UpdateDataSet</td>
<td>A DataSet containing rows to insert, update and delete.</td>
</tr>
<tr>
<td>bRefreshAfterSave</td>
<td>If True, the UpdateDataSet rows are refreshed and returned to the caller.</td>
</tr>
<tr>
<td>strCustomInsert</td>
<td>A comma-delimited list of methods and/or instructions which override the default save behavior.</td>
</tr>
<tr>
<td>strCustomUpdate</td>
<td>A comma delimited list of methods and/or instructions which override the default save behavior.</td>
</tr>
<tr>
<td>strCustomDelete</td>
<td>A comma delimited list of methods and/or instructions which override the default save behavior.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns a dataset containing rows refreshed after the save, or null if bRefreshAfterSave is False.

**Example**

The following code fragment adds a new customer using the SaveDataSet method.

```csharp
WebService webService = new WebService();
string      sessionToken;
DataSet     custDataSet;
DataTable   idoTable;
DataRow     newCust;
webService.Url = "http://<web-service-url>";
sessionToken = webService.CreateSessionToken( "sa", "", "PWDEMO" );
custDataSet = new DataSet( "SL.SLCustomers" );
idoTable = custDataSet.Tables.Add( "IDO" );
idoTable.Columns.Add( "CustNum", typeof(string) );
idoTable.Columns.Add( "CurrCode", typeof(string) );
idoTable.Columns.Add( "BankCode", typeof(string) );
idoTable.Columns.Add( "_ItemId", typeof(string) );
newCust = idoTable.Rows.Add( new object[] { "TEST123", "USD", "BK1", "" } );
webService.SaveDataSet( sessionToken, custDataSet, true, "", "", "" );
```
**CallMethod**
Calls an IDO method.

**Syntax**

```csharp
public Object CallMethod(string strSessionToken,
                          string strIDOName,
                          string strMethodName,
                          ref string strMethodParameters)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strSessionToken</td>
<td>A session token string created by calling CreateSessionToken.</td>
</tr>
<tr>
<td>strIDOName</td>
<td>The name of the IDO that publishes the method.</td>
</tr>
<tr>
<td>strMethodName</td>
<td>The IDO method name to call.</td>
</tr>
<tr>
<td>strMethodParameters</td>
<td>An XML formatted string containing the parameters to pass to the method.</td>
</tr>
</tbody>
</table>

**Return Value**
Returns the value that was returned to it by the method.

**Example**

```csharp
WebService webService = new WebService();
object ReturnValue;
string strParameters = "<Parameters><Parameter
ByRef="/Y"></Parameter>
...etc...
</Parameters>"

webService.Url = "http://<web-service-url>";
ReturnValue = oWS.CallMethod(
    SessionToken,
    "SL.SLParms",
    "GetSessionIDSp",
    ref strParameters );
```
Calling Dynamic IDOs Using .NET

COM calls can directly query or update the SQL database tables calling dynamic IDOs. Through dynamic IDOs, you can also directly invoke stored procedures from the SQL database. To access a dynamic IDO, use the standard API call syntax, except:

- For collection requests, instead of the collection name, you specify the table name (in the format TABLE!table_name).
  The properties specified in the calls are actually the column names in the table.
- When invoking a method, instead of the IDO name, specify "SP!" and instead of the Method name, specify the stored procedure name.
- When you open the session where you make these requests, you specify a configuration. The configuration points to an application database. This is the database where the requests will look for the tables and stored procedures.

Infor .NET Web Service Test Utility

On the utility server, there is an executable file that provides a test client for the .NET Web Service API. If you install the .NET web service on your SyteLine web server and want to verify that it is installed and configured correctly without writing your own client, you can use this utility.

For more information, see the Infor .NET Web Service Test Utility appendix in the SyteLine ERP Installation Guide.
Directly Accessing the Database Programmatically

You can programmatically access the SyteLine application database directly (without going through the SyteLine framework's IDO layer) using a development tool such as Visual Basic or C#.

The IDO layer usually handles the session management for connections. However, if your program is bypassing the IDO layer, it will need to run the following procedures to perform session management.

**InitSessionContextSP**

Before you execute any code on a database connection that may modify any SyteLine tables or call any SyteLine stored procedure, you must first initialize your session by calling this procedure.

```
EXEC dbo.InitSessionContextSp
    @ContextName = 'caller-context'
    , @SessionID   = @MySessionID OUTPUT
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ContextName</td>
<td>This parameter can be anything. It is helpful to put something meaningful in this field so that it is obvious what process initiated the call.</td>
</tr>
<tr>
<td>@SessionID</td>
<td>If this parameter is null, a new session ID will be created by the InitSessionContextSp procedure and returned as an output parameter. If it is not null, the value that is passed is used for the session ID. You may use the same @SessionID to initialize multiple connections. All connections that are initialized using the same @SessionID value will share a common set of session variables. However, you should never use the same session ID for two connections that are running in different transactions simultaneously.</td>
</tr>
</tbody>
</table>

**CloseSessionContextSP**

When your application or process is finished (assuming you did not roll back the entire transaction), you should call this procedure to clean up any session variables that may have been created.

```
EXEC CloseSessionContextSp @SessionID = @MySessionID
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@SessionID</td>
<td>This should be the session ID that was specified in the call to InitSessionContextSp</td>
</tr>
</tbody>
</table>
Sample Workflow Templates for Automating SyteLine Business Processes

Overview

SyteLine includes six sample Workflow templates that you can modify and then use to automate the following standard business processes:

- Performing a customer credit check
- Pricing an item for a customer
- Creating an invoice for a customer order
- Checking item availability
- Reserving items for an order
- Creating an order pick list

The sample templates call SyteLine IDO methods in order to perform the processes. Input and output data is stored in the workflow’s AppData.

These templates are intended as examples to be modified or to assist you in building your own templates.

NOTE: These templates are provided as a development tool. If you modify these templates, Infor is not responsible for any problems you encounter with, or because of, your modifications.

Prerequisites

In order to use these templates, you must have the following applications installed and configured as described in the SyteLine ERP Installation Guide:

- Microsoft Outlook and Microsoft Exchange Server (These templates were tested against Exchange Server 2003)
- SyteLine ERP 7.04 or above
- Workflow Automation (These templates were tested against Lexign version 7.1, Service Pack 3)

Permissions on Public Folders

When you set up Workflow Automation, you should have created a folder hierarchy similar to the following in Outlook. By convention, the Workflow folders are usually named Active Flows, Completed Flows, and Templates:
Users who import and run the sample templates must have Ownership permission for these public folders:

- Active Flows
- Completed Flows
- Templates

**Access Rights to Mailboxes**

In order to watch the progress of the flow through the syteline and workflow mailboxes, we also recommend that users developing the templates have full mailbox access rights to those mailboxes, at least during testing of the templates. To set this up in Outlook:

1. Create a distribution list called, for example, "development users" and add to that list any users who will develop or test templates. (This allows you to maintain a distribution list instead of having to update the mailboxes each time you add or remove a user who needs these rights.)

2. Add the distribution list as a user with mailbox access rights to the syteline and workflow mailboxes. These steps may vary slightly in different versions of Exchange.
   a. On the Exchange Server, run **Active Directory Users and Computers**.
   b. Select **Users** and double-click on the syteline object.
   c. Select the **Exchange Advanced** tab.
   d. Click the **Mailbox Rights** button.
   e. In the Permissions dialog, click **Add** and select the distribution list created in step 1.
   f. Click **OK**.
   g. Select **Allow Access** for all fields, including Full Mailbox Access.
   h. Click **OK**.
   i. Repeat steps b-h to add the distribution list as a user for the workflow mailbox.

3. Each user who is developing templates should add secondary mailboxes (for syteline and for workflow) to their Outlook account. (See the Outlook online help for more information.)

**About IDOs and Workflow**

For background information on using Workflow to access SyteLine IDOs, see Chapter 4, "Using Workflow to Access IDOs."

**Licensing**

The Workflow Templates are not licensed separately. The following license modules are allowed to access and execute the templates:

- SyteLine Transaction Users
- SyteLine Entity User
- SyteLine Multi Site Transaction User
- SyteLine Workflow Reply User
- SyteLine Automation User
# About the Templates

Each sample template has three steps: Start, Method Call, and Finish. The Finish step displays any messages that may have been returned by the IDO method.

## Customer Credit Check

The Customer Credit Check template will not work on a system where SyteLine is integrated with the Sun Systems financial package.

Use this template to retrieve credit information for a specific customer using the proposed order amount that is input as AppData. The method returns the following information:

- Whether the customer’s credit limit will be exceeded
- What the credit limit is
- The current credit balance
- The customer’s currency code
- Whether the customer is on credit hold (and the reason why)
- Whether a Letter of Credit is required for the customer
- Any error messages from the IDO

### Method/Stored Procedure

SLCustomers.CustomerCreditCheckSp

### Input App Data

In addition to the standard input App Data (see step 6 on page A-12), the following input parameters are required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pAmount</td>
<td>AmountType</td>
<td></td>
</tr>
<tr>
<td>pCustNum</td>
<td>CustNumType</td>
<td>Length 7 - value must be padded on the left with spaces, for example 103</td>
</tr>
<tr>
<td>pSite</td>
<td>SiteType</td>
<td></td>
</tr>
</tbody>
</table>

### Output App Data

The following information is returned in the flow’s Application Data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Mapping or Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infobar</td>
<td>InfobarType</td>
<td>contains any error message</td>
</tr>
<tr>
<td>pCreditHoldReason</td>
<td>ReasonCodeType</td>
<td>(custaddr/corp).credit_hold_reason</td>
</tr>
<tr>
<td>pCreditLimit</td>
<td>AmountType</td>
<td>(custaddr/corp).credit_limit</td>
</tr>
<tr>
<td>pCreditLimitBalance</td>
<td>AmountType</td>
<td>(custaddr/corp).credit_limit - (custaddr/corp).order_bal - (custaddr/corp).posted_bal - @pAmount</td>
</tr>
<tr>
<td>pCreditLimitExceeded</td>
<td>ListYesNoType</td>
<td>IF @pCreditLimitBalance &lt;= 0 THEN @pCreditLimitExceeded = 1 (ELSE 0).</td>
</tr>
<tr>
<td>pCurrencyCode</td>
<td>CurrCodeType</td>
<td>(custaddr/corp).curr_code</td>
</tr>
<tr>
<td>pLCRRequired</td>
<td>ListYesNoType</td>
<td>customer.lcr_reqd</td>
</tr>
<tr>
<td>pOnCreditHold</td>
<td>ListYesNoType</td>
<td>(custaddr/corp).credit_hold</td>
</tr>
</tbody>
</table>
**Item Pricing (Price Calculation)**

Use this template to find the currency code and the unit price that would be charged if an order was placed for a specified customer, item and quantity on a specified date.

**Method/Stored Procedure**

SLItems.PriceCalculationSp

**Input App Data**

In addition to the standard input App Data (see step 6 on page A-12), the following input parameters are required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCustNum</td>
<td>CustNumType</td>
<td>Length 7 - value must be padded on the left with spaces, for example 103</td>
</tr>
<tr>
<td>pItem</td>
<td>ItemType</td>
<td></td>
</tr>
<tr>
<td>pOrderDate</td>
<td>DateType</td>
<td></td>
</tr>
<tr>
<td>pQuantityOrdered</td>
<td>QtyUnitNoNegType</td>
<td></td>
</tr>
<tr>
<td>pSite</td>
<td>SiteType</td>
<td></td>
</tr>
<tr>
<td>pUOM</td>
<td>UMType</td>
<td></td>
</tr>
</tbody>
</table>

**Output App Data**

The following information is returned in the flow’s Application Data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Mapping or Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infobar</td>
<td>InfobarType</td>
<td>Contains any error messages</td>
</tr>
<tr>
<td>pCurrencyCode</td>
<td>CurrCodeType</td>
<td>custaddr_all.curr_code</td>
</tr>
<tr>
<td>pPrice</td>
<td>CostPrcType</td>
<td>Convert Quantity Ordered to Item Base U/M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get the Currency Exchange rate for the order date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get the price using PriceCalSp</td>
</tr>
</tbody>
</table>
Order Invoicing (EInvoice)
Use this template to generate and print an invoice for a specified customer order. The invoice is not translated into the customer language.

Method/Stored Procedure
SLCos.EInvoiceSp

Input App Data
In addition to the standard input App Data (see step 6 on page A-12), the following input parameters are required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCoNum</td>
<td>CoNumType</td>
<td>Length 10 - value must be padded on the left with spaces, for example 6321</td>
</tr>
<tr>
<td>pSite</td>
<td>SiteType</td>
<td></td>
</tr>
</tbody>
</table>

Other options that you normally select on the Order Invoicing Credit Memo form are set to their default values.

Output App Data
The following information is returned in the flow’s Application Data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Mapping or Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infobar</td>
<td>InfobarType</td>
<td>Contains any error messages</td>
</tr>
</tbody>
</table>

Item Availability
Use this template to take an Item and calculates its quantity on hand, quantity available ((On Hand + On Order + Qty WIP) - Qty Allocated to Orders and Jobs), Qty WIP, Qty Allocated to Orders, Qty Allocated to Jobs, and the Quantity On Order.

Method/Stored Procedure
SLItems.ItemAvailabilitySp

Input App Data
In addition to the standard input App Data (see step 6 on page A-12), the following input parameters are required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pItem</td>
<td>ItemType</td>
<td></td>
</tr>
<tr>
<td>pSite</td>
<td>SiteType</td>
<td></td>
</tr>
</tbody>
</table>

Output App Data
The following information is returned in the flow’s Application Data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Mapping or Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infobar</td>
<td>InfobarType</td>
<td>Contains any error messages</td>
</tr>
<tr>
<td>pQtyAllocCo</td>
<td>QtyTotlType</td>
<td>SUM(itemwhse_all.qty_alloc_co)</td>
</tr>
<tr>
<td>pQtyAllocMfg</td>
<td>QtyTotlType</td>
<td>item_all.qty_allocjob</td>
</tr>
<tr>
<td>pQtyOnHandAvailable</td>
<td>QtyTotlType</td>
<td>pQtyOnHandPhysical + pQtyOrdered + pQtyWip - pQtyAllocCo - pQtyAllocMfg</td>
</tr>
<tr>
<td>pQtyOnHandPhysical</td>
<td>QtyTotlType</td>
<td>SUM(itemwhse_all.qty_on_hand)</td>
</tr>
<tr>
<td>pQtyOnOrder</td>
<td>QtyTotlType</td>
<td>SUM(itemwhse_all.qty_ordered)</td>
</tr>
<tr>
<td>pQtyWip</td>
<td>QtyTotlType</td>
<td>SUM(itemwhse_all.qty_wip)</td>
</tr>
</tbody>
</table>
### Item Reservation for Order (Reserve Order)

Use this template to create inventory reservations for all line items of the specified order which have items that are defined as reservable.

**Method/Stored Procedure**

SLCos.ReserveOrderSp

**Input App Data**

In addition to the standard input App Data (see step 6 on page A-12), the following input parameters are required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCoNum</td>
<td>CoNumType</td>
<td>Length 10 - value must be padded on the left with spaces, for example 6321</td>
</tr>
<tr>
<td>pSite</td>
<td>SiteType</td>
<td></td>
</tr>
</tbody>
</table>

**Output App Data**

The following information is returned in the flow’s Application Data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Mapping or Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infobar</td>
<td>InfobarType</td>
<td>Contains any error messages</td>
</tr>
</tbody>
</table>

### Print Pick List

Use this template to print a pick list for the specified customer order number.

**Method/Stored Procedure**

SLCos.WfPrintPickListSp

**Input App Data**

In addition to the standard input App Data (see step 6 on page A-12), the following input parameters are required:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCoNum</td>
<td>CoNumType</td>
<td>Length 10 - value must be padded on the left with spaces, for example 6321</td>
</tr>
<tr>
<td>pSite</td>
<td>SiteType</td>
<td></td>
</tr>
</tbody>
</table>

Other options that you normally select on the Generate Order Pick List form are set to their default values.

**Output App Data**

The following information is returned in the flow’s Application Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Mapping or Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infobar</td>
<td>InfobarType</td>
<td>Contains any error messages</td>
</tr>
</tbody>
</table>
Importing a Template

Where to Import and Run the Templates

The sample Workflow templates are installed as part of the Utility Server or Exchange Server installations, if the Message Posting component is selected. The templates are stored as XML files in the SyteLine folder (usually c:\Program Files\MAPICS\SyteLine), under the subfolder Custom Outlook Forms.

You can import the templates and run the flows on either the admin client or the server. For ease of use, you may want to set up the templates on the server and then log in remotely to use or update the templates.

Which Templates to Import

You probably will want to create your own templates based on the ideas shown in these examples.

NOTE
We recommend that you install only those sample templates that you plan to modify and use. Start with one of the sample templates and get it working in a demo environment before you modify it, or before you load any of the others.

Steps to Import a Template

1. In Microsoft Outlook, navigate to the Templates folder. (By convention, the Workflow folders are usually set up using the hierarchy and names shown here.)

2. With the Templates folder selected, choose Actions>New Flow Template from the Exchange menu. This creates a shell template that you will overwrite.

3. Click Cancel to close the Properties dialog.

4. In the new template, select File>Import>Template as XML.

5. Browse to the path where the sample templates were installed (usually c:\Program Files\MAPICS\SyteLine\Custom Outlook Forms).

6. Select one of the templates and click Open. The following steps show, as an example, how to update the ItemAvailability template for use.
7. When Workflow displays the following message, click Yes.

8. The Flow Template Properties dialog displays in the foreground:

9. In the Flow Template Properties dialog, update these field values:

   - **Start flows using this server** - Select your Keyflow Server mailbox (usually named "workflow"). Even if your mailbox name exactly matches the default value, you should reselect it.

   - **Place started flows in this folder** - Browse to your system’s Active Flows folder. Even if your folder hierarchy exactly matches the default path, you should re-browse and reselect the folder.

   - **When flows are completed, move them to this folder** - Browse to your system’s Completed Flows folder. Even if your folder hierarchy exactly matches the default path, you should re-browse and reselect the folder.

10. Click **OK** to save the changes and close the dialog.
11. In the Flow Template dialog, right-click on the response step (the second step in the flow) and select **Properties**:

   ![Flow Template Diagram]

   a. In the Message tab, clear the existing "To" value.

   ![Message Tab]

   b. Click the **To** button and select the SyteLine mailbox set up for Workflow (that is, the user mailbox that SyteLine monitors for IDO requests). Even if that name exactly matches the default name shown here, you should browse and reselect the mailbox.

   c. Click **OK** to save the change.

12. Save and close the template.
13. In the Templates folder, you should now see your new template listed:

Adding a Workflow Template to SyteLine

Since this process involves starting and stopping all MAPICS services, make sure no other users are running SyteLine while you perform these steps.

After you have imported and updated a template, you must add it to the SyteLine Workflow Templates list, using the following steps:

1. Log in as an administrator on the SyteLine utility server.
2. Stop and restart all MAPICS services:
   - Select Start>Settings>Control Panel> Administrative Tools> Services.
   - Stop these MAPICS services, in this order:
     - MAPICS Workflow Posting Service
     - MAPICS Workflow Queue Listener
     - MAPICS Validation Service
     - MAPICS Task Manager
     - MAPICS Replicator
     - MAPICS Replication Listener
   - Restart the services in the opposite order.
3. Log into SyteLine and open the Workflow Templates form:
4. Add the name of the new template and save the record. (Click on the down-arrow and select the new template, which should appear in the drop-down list.)

If a record already exists for this template, see the steps under “If You Change a Template” on page A-14 to delete the record and add a new one.

5. Log out of SyteLine.

6. On the SyteLine utility server, stop and restart the services again, as described in step 2 above.

**Running a Workflow From a Template**

Once a template is imported into Outlook and added to SyteLine as described above, it is ready to run:

1. Go to the Templates folder in Outlook. (By convention, this is usually set to Public Folders>All Public Folders>MAPICS>Workflow>Templates.) Open the template.

2. Select **File>Run**.

3. You are prompted to initialize the flow:

4. Enter a name for this instance of the flow, and click **Next**.
5. Click **Next**.

![Diagram of the Input Application Data window]

6. In this dialog, provide the following information:

   - **IDORequestConfiguration** - the SyteLine configuration name to which this request should log in. (This is the "Data Source Config" value you select in the Logon dialog when logging into SyteLine.)

   - **IDORequestIsPasswordEncrypted** - indicates whether the password (if any) is encrypted. This value defaults to No but can be changed.

   - **IDORequestPassword** - the password for the SyteLine user name, if any. This value defaults to nothing (no password).

   - **IDORequestUserName** - the SyteLine user name that will be logged in for this request. This value defaults to the SL_Internal user (with no password), but can be changed to any valid SyteLine user name.

     Your security setup may affect the user name and password you choose, and whether you choose to encrypt the password.

   - **Application input data** - The data that is input to the workflow depends on the template and the business process. For example, the ItemAvailability template shown above requires you to enter an item number (PItem) and the name of the site to be checked for availability (PSite).

7. Click **Next**.

![Diagram showing a completed workflow setup]
8. Click **Finish** to save the changes and run the flow.

9. You should be able to see the flow move into the Active Flows folder. When it completes, it appears in the Completed Flows folder:

You can also watch the progress of the flow through the mailboxes:

- While the flow is sitting in the **Active Flows** folder, a message first appears in the **workflow** mailbox.
- After Workflow processes the message, a message appears in the **syteline** mailbox.
- Once SyteLine finishes working on the response step request, a message appears in the **workflow** mailbox again (to process the last step of the flow).
- At that point, the flow moves to the **Completed Flows** folder.

If the flow encounters an error at any point along the way, a message may stop in one of the mailboxes, and the flow will not complete. This will give you a clue about the point where the error occurred.

10. Open the flow in the Completed Flows folder and look at the Status tab. The Response column indicates Success or Failure.

If the response is Failure, you can double-click on the step to open the comment and find out more information about why the step failed.
11. Then look at the flow’s Application Data tab. It should return the requested data. In the case of the ItemAvailability template, the App Data for the completed flow would look similar to this:

![Application Data Table]

If You Change a Template

Any time you make a change to a template, you must delete the template from SyteLine’s Workflow Templates form and re-add it, using these steps:

1. In SyteLine, delete the template record from the Workflow Templates form.
2. Exit SyteLine.
3. Stop and start the MAPICS services, in the order described on page A-10.
4. In SyteLine, add the template record in the Workflow Templates form.
5. Stop and start the MAPICS services, in the order described on page A-10.

The order in which you do this is very important.
Troubleshooting

Checking the Event Viewer

You may be able to determine the problem by checking the Event Viewer on the utility server (Administrative Tools>Event Viewer>Application). If there are WF errors, right-click on an error and view its Properties:

The information at the end of the Description (highlighted above) may indicate the problem. In the example shown above, the user probably did not do one of these:

- stop/start the services in the proper order before and after adding or changing a template (see “Adding a Workflow Template to SyteLine” on page A-10”).
- delete and re-add the template to SyteLine’s Workflow Templates form, stopping and starting the services between steps, as described in “If You Change a Template” on page A-14.

Running Stored Procedures Manually

If you want to create a custom workflow template that uses an existing SyteLine stored procedure, first try to execute the stored procedure manually as a SQL query. Run the SP in a demo environment, providing the required input and output parameters. If the stored procedure will not run as a standalone query, it will not run as part of a workflow.
## Symbols

**.NET**
- CallMethod 5-5
- CreateSessionToken 5-2
- Dynamic IDOs 5-6
- Example 1-5
- GetConfigurationNames 5-1
- LoadDataSet 5-3
- SaveDataSet 5-4
- Test Utility 5-6
- Web Service API 5-1

**A**
- Action field on Work Flows form 4-14
- Active Flows folder 4-2
  - Field on Work Flows form 4-14
- ActiveDirectory 2-2
- APPDATA 4-10
- ASP page 2-2
- ASP pages 2-1
- Authorizations 2-3
- Availability, Workflow template A-5

**B**
- BeginTrans, in XML documents 3-26
- BinaryDataFormat tag 3-10
- Boolean properties in workflows 4-10
- BoolFalse tag 3-9
- BoolTrue tag 3-9

**C**
- CallMethod, .NET API call 5-5
- Case tag 3-10
- CloseSession, in XML documents 3-25
- CloseSessionContextSp 6-1
- Closing session 6-1
- ColumnType tag 3-9
- CommitTrans, in XML documents 3-26
- Completed Flows folder A-8
- Component Class Properties dialog 4-6
- ConfigName tag 3-6
- CreateSessionToken, .NET API call 5-2
- Credit check, Workflow template A-3
- Cursor tag 3-13
- Customer credit check, Workflow template A-3

**D**
- Database, accessing programmatically 6-1
- Database, used for virtual IDOs 3-28, 5-6
- DataMap 4-4
- DataType tag 3-9
- Date format, when passing to IDOs 3-28
- DateFormat tag 3-10
- DeadlockRetry tag 3-6
- decimal values, properties of 3-28
- DefaultIMECharset tag 3-10
- DefaultValue tag 3-9
- Direct access to database 6-1
- Direct IDO URL 2-2
- DisplayDecimal Position tag 3-10
- Distinct tag 3-14
- DomainIDOName tag 3-9
- Dynamic IDO
  - Definition 1-5
  - in Workflow 4-12
  - XML documents 3-28

**E**
- Edit List Source Specification dialog 4-6
- EInvoice, Workflow template A-5
- Error Messages from Workflow 4-13
- Event Handler 4-5
- Event Viewer A-15
- Exchange Mailbox Alias field on Intranets form 2-2
- Exchange Server field on Intranets form 2-2
- External intranet 2-2

**F**
- Filter tag 3-14
- Flow Name field on Work Flows form 4-15
- Form Component Properties dialog 4-6
- Form level integration 1-1
- Form Name field on Work Flows form 4-14
- From Site field on Intranets form 2-3
- fsslWorkflow message class 4-3

**G**
- GetConfigurationNames, .NET API call 5-1
- GetConfigurations, in XML documents 3-5
- Get PropertyInfo tag 3-8
- Get PropertyInfo, in XML documents 3-8

**H**
- Html tag 3-9
- HTTP POST command 2-2, 3-2
- HTTPS protocol 3-2

**I**
- ID attribute, UpdateCollection 3-21
- Identity tag 3-9
- IDO
  - Collection, finding in SyteLine 4-4
  - Definition 1-2
  - Licensing 2-4
  - Message classes in Workflow 4-3
  - Method, finding in SyteLine 4-4
  - Passing dates to 3-28
  - IDORequest message class, in Workflow 4-4
  - IDORequestConfiguration 4-11, A-12
Index

IDORequestIsPasswordEncrypted 4-11, A-12
IDORequestPassword 4-11, A-12
IDORequestUserName 4-11, A-12
InboundQueue.ASP 2-1
InboundQueue.asp 2-2
InitiatorName, in XML documents 3-4
InitiatorType, in XML documents 3-4
InitSessionContextSp 6-1
InputMask tag 3-9
InputMaskPromptChar tag 3-9
Integration
  Examples 1-3
  forms 1-1
  Levels 1-1
  Middleware 1-2
  Replication 1-7
Intranet Name field on Sites/Entities form 2-2
Intranets form 2-1
Invoice, Workflow template A-5
Invoke, in XML documents 3-23
InvokeMessage class in Workflow 4-3
Item Availability, Workflow template A-5
Item Pricing, Workflow template A-4
Item Reservation for Order, Workflow template A-6
Item tag 3-13

J
Justify tag 3-10

L
Label tag 3-9
LabelStringID tag 3-9
Launching a workflow 4-14, 4-16
Length tag 3-9
Licensing IDOs and user IDs 2-4
LinkBy tag 3-13, 3-18
Linked Server Name field on Intranets form 2-3
LIT keyword 4-9
Literal Values in Workflow messages 4-9
Live linked relationship 2-3
LoadCollection
  in XML documents 3-12
  Message class in Workflow 4-3
LoadCollection tag 3-12
LoadDataSet, .NET API call 5-3
LoadType tag 3-13

M
Mailbox alias 2-2
Message Posting component A-7
Message string syntax 4-8
Method, definition 1-2
Middleware level integration 1-2
Middleware Permissions 2-3
MSMQ 2-1

N
NumDigitsInGroup tag 3-10

O
Object Authorizations for User 2-3
OpenSession, in XML documents 3-6
Order Invoicing, Workflow template A-5
OrderBy tag 3-14

P
Password tag 3-6
Permissions 2-3
Pick List, Workflow template A-6
Place started flows in this folder, field in Workflow template 4-2, A-8
Posting XML documents 3-2
PostQueryCmd tag 3-14
Price calculation, Workflow template A-4
Print Pick List, Workflow template A-6
Private Queue 2-2
ProductVersion tag 3-6
Program access to database 6-1
Property tag 3-9
Property, definition 1-2
PropertyList tag 3-13
Protocols 1-2

Q
Queue Server 2-1

R
ReadOnly tag 3-9
RecordCap tag 3-13
Replication Categories form 2-3
Replication Management form 2-3
Replication Rules form 2-3
Replication, example 1-6
RequestHeader elements in XML documents 3-2
RequestType elements in XML documents 3-4
Required tag 3-9
Reserve Order, Workflow Template A-6
ResponseHeader elements in XML documents 3-3
RollbackTrans, in XML documents 3-26
RORecord tag 3-9
RSDAO, definition and example 1-3

S
SaveDataSet, .NET API call 5-4
Secure transmissions 3-2
Send Flow Manager Mail field in Workflow template 4-2
ServerDate tag 3-6
Session initialization 6-1
  session token 5-2
Session, closing 6-1
SessionManager.ASP 2-1, 3-2
SessionManager.asp 2-2, 3-2
Sites/Entities form 2-3
SourceName, in XML documents 3-4
SP! format 1-5, 3-28
SQL Query Analyzer, Troubleshooting Workflows A-15
SQL user name 2-3
Standard IDO, definition 1-5
Index

Start Flows Using field in Workflow template 4-2, A-8
Subcollection tag 3-9
Suspend field on Work Flows form 4-14
SyteLine Configuration Wizard 2-2

T
TABLE! format 3-28
TargetName, in XML documents 3-4
Templates
   Creating in Workflow 4-2
   Path 2-2
   Registering 4-15
Title field in Workflow template 4-2
To Site field on Intranets form 2-3
token for session 5-2
TransactionName tag 3-4

U
UpdateCollection
   in XML documents 3-17
   Item ID example 3-21
   Message class in Workflow 4-3
   Nested Insert example 3-20
   UseKeys example 3-22
URL 2-2
UseKeys attribute, UpdateCollection 3-22
User E-mails, setting up for Workflow 4-15
User Middleware Permissions 2-3
User Name field on Intranets form 2-3
Userld tag 3-6
Users, Licensing 2-4

V
Virtual directory, on Web Server 2-1
Virtual IDOs 3-28

W
Warning tag 3-9
Web Server setup 2-1
When Flows are Completed field in Workflow template 4-2
WinStudio automation API 1-1
Work Flows form 4-14
Workflow
   Active Flows folder 4-2, 4-14
   Changing a template A-14
   Completed Flows folder 4-2
   Configuration to use for IDO Request 4-11
   Creating a message step 4-7
   Creating templates 4-2
   Distribution list A-2
   Dynamic IDOs 4-12
   Editing strings copied from SyteLine forms 4-9
   E-mail receiver 4-2
   Error Messages 4-13
   Example - Method to delete a customer order line 4-12
   Example - Method to get item warehouse details 4-12
   Example - Query customer order information 4-11
   Flow name 4-15
   Flow Template Properties A-8
   Folder name 4-14
   IDO Declarative Interface 4-1
   InvokeMethod message class 4-3
   Launching 4-14
   Literal values 4-9
   LoadCollection message class 4-3
   Loading Old Values 4-13
   Mailbox Access Rights A-2
   Message classes 4-3
   Message string syntax 4-8
   Password to use for IDO Request 4-11
   Posting process 4-16
   Process flow 4-16
   Registering templates 4-15
   Sample Templates A-1
   Sample templates 4-13
   Server 4-2
   Setting rules 4-14
   Setup 2-1, 2-2
   Step naming conventions 4-7
   Suspend 4-14
   Template, adding to SyteLine A-10
   Template, sample Application Data A-14
   Templates, Importing A-7
   Templates, licensing A-2
   Trigger on form 4-14
   Troubleshooting A-15
   UETs and UDFs 4-12
   UpdateCollection 4-3
   User e-mails 4-15
   User ID to use for IDO Request 4-11
   Watching progress of the flow A-13
Workflow Enabled field on Intranets form 2-3
Workflow Templates form A-10
Workflow Templates Public Folder Path field on Intranets form 2-2
Workstation tag 3-6

X
XML
   BeginTrans 3-26
   CloseSession 3-25
   CommitTrans 3-26
   Dynamic IDOs 3-28
   Example 1-3
   GetConfigurations 3-5
   GetPropertyInfo 3-8
   InitiatorName 3-4
   InitiatorType 3-4
   Invoke 3-23
   LoadCollection 3-12
   OpenSession 3-6
   Posting documents 3-2
   Request and response documents 3-1
   RequestHeader 3-2

Integrating IDOs with External Applications
© 2007, Infor Global Solutions Technology GmbH
Index

RequestType 3-4  SourceName 3-4
ResponseHeader 3-3  TargetName 3-4
RollbackTrans 3-26  UpdateCollection 3-17