Integrating IDOs with External Applications
## Contents

**Overview** .......................................................... 7
- About IDOs .......................................................... 7
- IDO Runtime .......................................................... 7
- IDO Requests and Responses ..................................... 8
  - Types of Requests/Responses ................................... 8
  - Structure of Requests ........................................... 8
  - Structure of Responses ......................................... 9
- Protocols ............................................................. 10
  - XML Request/Response Documents ............................. 10
  - IDO .NET Class Libraries, Command Interface, and Extension Classes ............................................. 10
  - Application Event System ...................................... 12
  - .NET Web Service Calls ....................................... 13
  - SharePoint Integration ........................................ 13
  - RSDAO Calls ...................................................... 13
- Integrating Directly with the Database (Dynamic IDOs) ....... 14
  - Standard vs. Dynamic IDOs ..................................... 14
  - Directly Accessing the Database Programmatically ....... 15
- Architecture of IDO Communications ............................. 15
  - Infor ERP SyteLine Client Application Communication with IDOs ..................................................... 15
  - Framework Services Communication with IDOs .......... 16
  - IDO Request/Response Communication with IDOs .......... 16
  - IDO Extension Class Communication with IDOs .......... 17
- User Middleware Permissions and Licensing ..................... 18
  - Permissions ....................................................... 18
  - Licensing ......................................................... 18
- For More Information ............................................... 19

**Using XML IDO Request and Response Documents** ............... 21
- When Would You Use This? ....................................... 21
- About XML IDO Requests and Responses ......................... 21
- Replicating XML Documents ...................................... 22
- Posting XML Documents ........................................... 22
- Request Documents ............................................... 23
- Response Documents ............................................... 23
Contents

XML Request/Response Header Information .............................................. 24
Request and Response Types ................................................................. 25
  GetConfigurations ............................................................................. 26
  OpenSession ....................................................................................... 30
  GetPropertyInfo ............................................................................... 33
  LoadCollection .................................................................................. 38
  UpdateCollection .............................................................................. 43
  Invoke ................................................................................................. 52
  CloseSession ...................................................................................... 54
Passing Dates to the IDO Runtime .......................................................... 55
Calling Dynamic IDOs in XML Documents .............................................. 55

Using the Event System ......................................................................... 57
  When Would You Use This? ................................................................. 57
  Event Action Types Available for IDO Integration ............................... 57
  Example ............................................................................................... 58
    Opening a Session and Requesting Data for a Remote Site .................. 58
  For More Information .......................................................................... 61

Using the .NET Web Service API ............................................................ 63
  When Would You Use This? ................................................................. 63
  About .NET WebService API Calls ...................................................... 63
  Prerequisite ......................................................................................... 63
  API Calls ............................................................................................. 64
    GetConfigurationNames ..................................................................... 65
    CreateSessionToken ......................................................................... 66
    LoadDataSet ...................................................................................... 67
    SaveDataSet ..................................................................................... 68
    CallMethod ....................................................................................... 70
  .NET Web Service Example ................................................................. 71
  Calling Dynamic IDOs Using .NET ....................................................... 73
  Infor .NET Web Service Test Utility .................................................... 73

Using Infor SyteLine SharePoint Web Parts ............................................ 75
  When Would You Use This? ................................................................. 75
  About the IDO Web Parts ................................................................... 75
  Prerequisites ....................................................................................... 76
  Deploying the Web Parts in SharePoint ............................................... 76
    Configuring the Connection to SyteLine at SharePoint Sites ............... 77
Adding the Web Parts to a SharePoint Page ................................................. 79
Configuring the Web Parts to Display SyteLine Data in SharePoint .................. 80
  Infor Chart Web Part ........................................................................... 80
  Infor List Web Part ........................................................................... 82
Exporting Web Parts ............................................................................ 83
Importing Web Parts ............................................................................ 84
Drilling Back to SyteLine Forms. ............................................................. 84

Directly Accessing the Database Programatically ................................. 85
  When Would You Use This? ................................................................. 85
    InitSessionContextSp ...................................................................... 85
    CloseSessionContextSp ................................................................... 86
About IDOs

An **Intelligent Data Object** (IDO) is a business 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.

For example, the SLItems IDO handles the data and logic shown on many of the Infor ERP SyteLine forms that deal with items (Items, ItemCosts, ChangeItemStatus, and so on).

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.

IDO definitions are stored as metadata in an objects database and are accessed through IDO editing forms in SyteLine. These forms are available to users with the SyteLineDev license.

Through configurations, application databases are linked with an objects database and a forms database.

IDO Runtime

Most access to the application database is through the IDO runtime. The primary responsibilities of the IDO runtime include querying data sets, saving data, and calling methods. In IDO terms as mentioned previously, these actions are known as `LoadCollection`, `UpdateCollection`, and `Invoke`. 
IDO Requests and Responses

Interactions with the IDO runtime are based on requests and responses. The caller builds an IDO request and sends it to the IDO runtime to be executed. The IDO runtime builds and returns a response to the caller containing the results of the requested action.

Types of Requests/Responses

Actions that can be requested include the following:

- **Open session** - The caller provides a user ID, password, and the name of the configuration you want to access. The system uses this information to select the utility server and site database to use for the session, and returns a session ID. This is essentially the same as a user logging into the application - it opens a session where the caller can communicate with the IDO.

- **Get configurations** - Get a list of the application configuration names available on a server and the relevant runtime configuration information associated with each. Use this action only if you do not already know the configuration or URL you want to use.

- **Load collection** - Query either an IDO 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.

- **Invoke method** - Execute the IDO method; this may be code in a custom assembly, or it may be a stored procedure.

- **Close session** - Close the session. This action is only required when accessing IDOs from an external application.

Structure of Requests

IDO requests have the following hierarchical structure:

- IDORequestEnvelope (contains 1 or more IDORequests)
  - IDO Request (contains zero or 1 Request Payloads)
    - Request Payload (contains data required to complete the request)

The **IDORequestEnvelope** is the top-level container for a collection of IDOResquests. It also contains header information such as the caller's Session ID.

Each **IDORequest** has a request type (OpenSession, LoadCollection, Invoke, etc.). It also contains a optional **request payload** that is dependent on the request type. For example, the payload for an OpenSession request contains login information for a user (user ID, password, and configuration).
The following example shows the structure of an Invoke IDO request in XML format. (This example asks the IDO runtime to validate whether "Developers" is a defined Group name.)

```
<IDORequest ProtocolVersion="5.00" SessionID="xxxx">
  <RequestHeader Type="Invoke">
    <Name>MGCore.GroupNames</Name>
    <Method>GroupValid</Method>
    <Parameters>
      <Parameter>Developers</Parameter>
      <Parameter ByRef="Y"/>
      <Parameter ByRef="Y"/>
      <Parameter ByRef="Y"/>
    </Parameters>
  </RequestData>
</RequestHeader>
</IDORequest>
```

Structure of Responses

Responses from the IDO runtime have a parallel hierarchy corresponding to the contents of the IDO request. The payload includes the requested data or return values. The response to our example request might be as follows:

```
<IDOResponse ProtocolVersion="5.00" SessionID="xxx">
  <ResponseHeader Type="Invoke">
    <ResponseData>
      <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>
      <ReturnValue VT="2">16</ReturnValue>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

TIP: A good way to become more familiar with XML IDO request and response document structure is to view the documents logged by the "Wire Tap" feature of the IDO Runtime Development Server. When the wire tap is turned on, everything a user does in a SyteLine session is recorded as a series of request/response documents.
Protocols

An application can use any of the following protocols to make IDO requests and receive responses.

XML Request/Response Documents

At the lowest level, interactions between callers and the IDO runtime are based on XML documents like the ones in the examples on the previous pages. The use, syntax, and examples of these XML documents are described in Chapter 2, "Using XML IDO Request and Response Documents." Generally XML documents are used for synchronizing data (through non-transactional replication) with another application that expects XML import(exports. This applies to the external financials interface and applications interfacing through Infor ESB (the "Bus").

IDO .NET Class Libraries, Command Interface, and Extension Classes

Developers can programmatically access the IDO runtime using the Mongoose.IDO.Client class library or the IDO .NET API. The Command Interface (IIDOCmds) can be used by standalone client applications, by WinStudio scripts, by user controls and within IDO extension classes. The Mongoose.IDO.Client implements this interface; it supports all IIDOCmds methods (LoadCollection, UpdateCollection, GetPropertyInfo and Invoke), plus additional methods that are necessary to create a stand-alone application but should not be called from a WinStudio form script or an IDO extension class (OpenSession, CloseSession, GetConfigurations).

Protocol Class Library

The Infor framework provides a class library (IDOProtocol) with classes that correspond to each level in the hierarchy of an IDO request and response. Thus a developer rarely, if ever, has to work directly with the XML. The top level of the hierarchy is implemented by the IDOREquestEnvelope class. The IDO request is implemented by the IDOREquest class. The request payload implementation, if any, varies depending on the request type. For example, an Invoke request is implemented by the InvokeRequestData class, and other similar classes are available for each of the request types that carry a payload. The protocol classes provide a ToXML method and a FromXML method to easily convert to and from XML text and protocol classes.
The following example shows the same Invoke request as in the previous example - but using code that includes an IDO protocol class. Compare the code to the XML.

```
<IDORequest ProtocolVersion="5.00" SessionID="xxxx">
    <RequestHeader Type="Invoke">
        <RequestData>
            <Name>MGCore.GroupNames</Name>
            <Method>GroupValid</Method>
            <Parameters>
                <Parameter>Developers</Parameter>
                <Parameter ByRef="Y"/>
                <Parameter ByRef="Y"/>
                <Parameter ByRef="Y"/>
            </Parameters>
        </RequestData>
    </RequestHeader>
</IDORequest>
```

IDO responses are implemented in classes that mirror the requests, with the IDOResponseEnvelope at the top of the hierarchy.

**IDO Command Interface**

A much simpler programmatic interface is also available: the IDO command interface (`IIDOCommand`), which provides wrapper methods that typically reduce the interaction to a single line of code. The following example shows the same Invoke request as in the previous example, this time using the IDO command interface.

```
Dim resp As InvokeResponseData
resp = Context.Commands.Invoke( 
    "MGCore.GroupNames", 
    "GroupValid", 
    "Developers", 
    IDONull.Value, 
    IDONull.Value, 
    IDONull.Value, )

group = resp.Parameters(1).Value
```

Although using IDO command interface wrappers in your code is easier than using the IDO protocol classes, the interface is limited to one response per command, and generally supports only the most common parameters for each type of command. Using the IDO...
Protocol class library is more flexible: you can build multiple requests in one "envelope" and you can build hierarchical LoadCollection or UpdateCollection requests.

Client Class Library

The IDOResquestClient .NET class library has one class, Client. When you create an instance of the Client class, it publishes the IDO access commands (LoadCollection, UpdateCollection, and Invoke), plus additional commands (OpenSession, GetConfiguration, CloseSession) that are only used for external application communications.

IDO Extension Classes

An IDO extension class is a .NET class that allows developers to extend the functionality of an IDO by adding methods and event handlers. IDO extension classes are compiled into a .NET class library assembly and stored in the objects (IDO metadata) database. The IDO runtime loads these assemblies on demand and calls methods and event handlers in the extension classes in response to IDO requests.

AppDB Class

The AppDB class provides direct access to the application database, plus access to common framework features such as application messages and session variables. AppDB is a class that may be used only from IDO extension classes. It is defined in the IDOCore class library.

For more information about the IDO command interface, class libraries, and extension classes, see the online book IDO Development Guide.

Application Event System

You can use SyteLine’s events forms to set up event handlers that communicate with IDOs. For example, to invoke the existing method GetSystemParms in the SLInvparms IDO, set up an Event Action of type Call IDO Method with the following parameters:

IDO("SLInvparms")
METHOD("GetSystemParm")
PARMS( RE(ParmValue), E(ParmName) )

For more information on how to set up events for IDO communication, see Chapter 3, “Using the Event System.”
.NET Web Service Calls

It is possible to use VB .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
IDOWebServiceReference.DOWebServiceSoapClient webService =
    new IDOWebServiceReference.DOWebServiceSoapClient();
DataSet IdoDataSet;
IdoDataSet = webService.LoadDataSet(
    SessionToken,
    "SL.SLItems",
    "Item, Description, UnitCost",
    "Item = N'FA-10000'",
    ",", ",", -1);
```

See Chapter 4, “Using the .NET Web Service API for more information.

SharePoint Integration

Infor SyteLine SharePoint web parts retrieve and display on a SharePoint site the IDO data from a specific SyteLine configuration. The web parts display SyteLine information as charts or tabular lists, based on the specified IDO, properties and filters.


RSDAO Calls

RSDAO is provided for backward compatibility but is not generally used in the current version, since the Mongoose.IDO.Client class library replaces the RSDAO COM library.

RSDAO is a set of components that implements a small set of 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. 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 RSDAO help file, available on the Infor support web site.
Integrating Directly with the Database (Dynamic IDOs)

IDO LoadCollection and UpdateCollection requests 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 Infor ERP SyteLine Guide for details on how to do this). Use the standard request 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:

```csharp
Imports Mongoose.IDO
Imports Mongoose.IDO.Protocol
Public Class SPSample
    Shared Sub Main()
        Using client As Client = New Client()
            Dim openResponse As OpenSessionResponseData
            openResponse = client.OpenSession("user", "password", "configuration")
            If openResponse.LogonSucceeded Then
                Dim invokeResponse As InvokeResponseData
                invokeResponse = client.Invoke("SP!", "MyStoredProc", "parm1", "parm2")
            End If
        End Using
    End Sub
End Class
```

Standard vs. Dynamic IDOs

Standard and dynamic IDOs behave exactly the same at runtime. However, standard IDOs are defined in the objects database (IDO metadata) and can support these additional features:

- A custom load method may be used to load a collection.
- 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.
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 VB or C#.

The IDO layer usually handles the session management for connections. Thus, if your program is bypassing the IDO layer, it must run procedures to perform session management. This is described more fully in Chapter 6, "Directly Accessing the Database Programmatically."

Architecture of IDO Communications

This section shows how the client, framework services, IDO request/response calls, and IDO extension classes communicate with the IDO runtime and the back-end databases.

Infor ERP SyteLine Client Application Communication with IDOs

When a user logs into a WinStudio (SyteLine) session on a client, the system sends a "Get Configurations" request to the Config Server page on a utility/web server.

Each time a request for a connection comes in, the Config Server returns the URL of the IDO Request Service page on a utility/web server to the client. The client then sends an "open session" request, and all IDO requests made during that session, to the specified IDO Request Service, which runs the request and sends the IDO response back to the client.
Framework Services Communication with IDOs

The framework services running on the utility/web server do not require a login, so there is no "open session" request. Instead, the services read an XML file on the utility server that contains the names of the configurations each service monitors. This XML is updated when an administrator updates the Service Configuration Manager utility.

IDO Request/Response Communication with IDOs

When an external application initiates communication with the IDO runtime, it has to first get the list of configurations. This requires communication with the Config Server. The application then opens a session. Once the session has been established, a Session ID is passed back to the application. That session ID is used by subsequent IDO requests (GetPropertyInfo, LoadCollection, UpdateCollection, or Invoke), which communicate with the proper configuration's IDO runtime. The IDO runtime is the middleman between the external application and the SyteLine databases.
Dynamic IDOs go through the IDO runtime to access the application database.

IDO Extension Class Communication with IDOs

IDO extension classes are compiled into a .NET class library assembly and stored in the objects database. The IDO runtime loads these assemblies on demand and calls methods and event handlers in the extension classes in response to IDO requests.
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 IDO Request Client and IDO Protocol libraries to get a connection and subsequent access to an IDO and its methods, the Client 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 framework. 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 protocol.

Licensing

Customer-created IDOs are assigned to the SyteLineAutomation license module. Imported or extended IDOs are associated with the license module(s) of the base IDO which they extend.

If you are accessing an IDO either through the XML IDO interface (replication) or through the programmatic IDO interface, make sure the user ID specified in the session login data is assigned to a license module that allows access to that IDO.
For More Information

For more information about IDOs and related topics, refer to the following:

<table>
<thead>
<tr>
<th>For Information About...</th>
<th>Look In...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating and editing IDOs and their methods and properties</td>
<td>SyteLine online help on IDO editing</td>
</tr>
<tr>
<td>The Application Event System</td>
<td>SyteLine online help on events Guide to the Application Event System online book</td>
</tr>
<tr>
<td>Overview of framework development tools, guidelines for customizing Infor ERP SyteLine, and information on some common development tasks</td>
<td>Modifying Infor ERP SyteLine online book</td>
</tr>
<tr>
<td>IDO .NET API, IDO .NET class library, IDO extension classes</td>
<td>IDO Development Guide (currently under construction)</td>
</tr>
<tr>
<td>Customizing forms</td>
<td>WinStudio Design Mode online help (from the SyteLine main menu, select Help&gt;Customizing Forms)</td>
</tr>
<tr>
<td>Licensing custom forms and IDOs</td>
<td>Multi-Site Planning Guide online book</td>
</tr>
<tr>
<td>Performing diagnostics on IDO use; viewing logged data</td>
<td>IDO Runtime Development Server online help or Log Monitor online help, on the utility server or an administrative client</td>
</tr>
<tr>
<td>Dealing with metadata for IDOs: synchronizing metadata from upgrades or add-on products; importing or exporting IDO-related metadata</td>
<td>App Metadata Sync utility online help on the utility server App Metadata Transport utility online help on the utility server</td>
</tr>
<tr>
<td>Defining configurations (named combinations of application, forms, and objects databases)</td>
<td>Configuration Manager online help on the utility server</td>
</tr>
</tbody>
</table>
Using XML IDO Request and Response Documents

When Would You Use This?

XML request and response documents provide the lowest-level interaction with IDOs. Some applications interfacing with Infor ERP SyteLine expect to receive and submit all data as XML documents. For example, the SL-SunSystems interface uses XML documents to submit requests and responses.

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.

A transaction is created for each UpdateCollection request, and optionally for each Invoke request if the method is identified as requiring a transaction.
Replicating XML Documents

If you are sending XML request and response documents to/from other applications by using replication, you must set up non-transactional replication, as described in the Multi-Site Implementation Guide. For a better understanding of how non-transactional replication works, see the Replication Reference.

Posting XML Documents

When posting XML documents:

1. Provide access to the RequestService ASPx page via a URL specification such as:
   http://domain_name/IDORequestService/RequestService.aspx
   or
   http://ip_address/IDORequestService/RequestService.aspx

   (IDORequestService is set up as a virtual directory during installation of the SyteLine utility server/web server component.) This URL specification is entered on the Intranets form in SyteLine. You may also want to make the ASPx available on the internet, which requires additional setup in IIS.

   TIP: You can use a GetConfigurations request (see page 26) to find the URL to be used by subsequent request documents.

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

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

   If you are sending XML documents directly to the IDORequestService (that is, not through the programmatic client class), be aware that the service supports GZip compression. Compression is handled automatically when using the client class.
Request Documents

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

For example:

```xml
<IDOResquest ProtocolVersion="x.yy" [SessionID="sessID"]>
  <RequestHeader Type="RequestType">
    <InitiatorType>InitiatorType</InitiatorType>
    <InitiatorName>InitiatorName</InitiatorName>
    <SourceName>Source</SourceName>
    <TargetName>Target</TargetName>
    <RequestData>
      ...request data...
    </RequestData>
  </RequestHeader>
  <RequestHeader Type="RequestType">
    ...</RequestHeader>
  </RequestHeader>
  ...
</IDOResquest>
```

Replace the information in italics with values as described in "XML Request/Response Header Information" on page 24.

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:

```xml
<IDOResponse ProtocolVersion="x.yy" SessionID="sessID">
  <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 Header Information" on page 24.
XML Request/Response Header 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>x.yy (Protocol version)</td>
<td>The protocol version supported by the client (on a request document) or the server (on a response document). The absence of this attribute means the client or server is running a version of WinStudio earlier than version 5.00. Version 5.00 was the first version to support this attribute. Some of the functionality described in this chapter does not apply for servers/clients running earlier versions.</td>
</tr>
<tr>
<td>SessionID</td>
<td>Unique session ID that is created when an OpenSession request succeeds and is returned in the OpenSession response. All subsequent requests should include the SessionID in the request. For backward compatibility you may omit this attribute and the server will use your web session to look up the SessionID; however it may be required in the future, so new code should include this attribute.</td>
</tr>
<tr>
<td>RequestType</td>
<td>The type of IDO request. The commonly used requests are: “GetConfigurations” on page 26 “OpenSession” on page 30 “GetPropertyInfo” on page 33 “LoadCollection” on page 38 “UpdateCollection” on page 43 “Invoke” on page 52 “CloseSession” on page 54 Note: 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: Form - signals that the request was generated by a form. InitiatorName then must contain the form name. Replication - signals that the replication process generated the request. InitiatorName is blank, and the SourceName and TargetName contain the source and target sites. 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. For example: &lt;InitiatorName&gt;PurchaseOrders&lt;/InitiatorName&gt;</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>Not used.</td>
</tr>
<tr>
<td>Request Data or Response Data</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>
</tbody>
</table>
Request and Response Types

The commonly used request and response document are described on these pages:

- “GetConfigurations” on page 26
- “OpenSession” on page 30
- “GetPropertyInfo” on page 33
- “LoadCollection” on page 38
- “UpdateCollection” on page 43
- “Invoke” on page 52
- “CloseSession” on page 54
GetConfigurations

This XML document requests a list of the application configuration names available on the server and the relevant "runtime" configuration information associated with each. Most of the information retrieved in the response was set through the ConfigManager utility.

This request should be made from a client to `IDORequestServer\ConfigServer.aspx`. The information that is returned includes the AppURL for the `IDORequestService\RequestService.aspx`. Use that RequestService.aspx URL for processing the rest of the requests during this session.

For programs integrating via IDORequest XMLs to a single utility server, if you already know the RequestService URL and configuration name, the caller could skip the GetConfigurations call and post requests directly to the RequestServer.

Request Data

none

Response Data

```xml
<ResponseData>
   <Config Name="name">
      <AppURL>appurl</AppURL>
      <AppName>name</AppName>
      <CompanyName>name</CompanyName>
      <Copyright>copyright</Copyright>
      <Version>version</Version>
      <SplashFilename>bmpfilename</SplashFilename>
      <IconFilename>iconfilename</IconFilename>
      <VBAFilename>bmpfilename</VBAFilename>
      <FormCLSID>clsid</FormCLSID>
      <FormFileExtension>extension</FormFileExtension>
      <AppURL>appurl</AppURL>
      <AppName>name</AppName>
      <CompanyName>name</CompanyName>
      <Copyright>copyright</Copyright>
      <Version>version</Version>
      <SplashFilename>bmpfilename</SplashFilename>
      <IconFilename>iconfilename</IconFilename>
      <VBAFilename>bmpfilename</VBAFilename>
      <FormCLSID>clsid</FormCLSID>
      <FormFileExtension>extension</FormFileExtension>
      <HelpBasePath>path</HelpBasePath>
      <BaseHelpFile>filename</BaseHelpFile>
      <WallpaperFilename>graphicfilename</WallpaperFilename>
      <WallpaperDisplayOption>option</WallpaperDisplayOption>
      <VertComponentSpacing>charunits</VertComponentSpacing>
      <HorzComponentSpacing>charunits</HorzComponentSpacing>
      <ObjectMenuName>stringname</ObjectMenuName>
      <HelpContentsIndexMenuText>text</HelpContentsIndexMenuText>
      <DeleteExistingPrompt>stringname</DeleteExistingPrompt>
      <DeleteNewPrompt>stringname</DeleteNewPrompt>
      <ReportViewerExecutable>exename</ReportViewerExecutable>
      <DefFormBGColor>colorset</DefFormBGColor>
      <DefFormFGColor>colorset</DefFormFGColor>
      <DefNormalBGColor>colorset</DefNormalBGColor>
      <DefNormalFGColor>colorset</DefNormalFGColor>
      <DefReadOnlyBGColor>colorset</DefReadOnlyBGColor>
      <DefReadOnlyFGColor>colorset</DefReadOnlyFGColor>
      <DefRequiredBGColor>colorset</DefRequiredBGColor>
      <DefRequiredFGColor>colorset</DefRequiredFGColor>
   </Config>
</ResponseData>
```
### Tag and Description

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigGroup</td>
<td>Optional parameter in the request header used to specify a configuration group (as defined in the Configuration Manager utility) whose configurations should be passed.</td>
</tr>
</tbody>
</table>
| Config          | Defines the environment (servers, databases, etc.) that are required for the presentation of an application. Each configuration references an application ID.  
Name - Configuration name, which is unique. |
<p>| AppURL          | The requestservice URL to use for processing subsequent IDO requests. |
| FormsURL        | URL to the runtime instance of FormServer (providing WinStudio access to a Forms database). |
| AppName         | Displayable name for this application (for example, SyteLine). |
| CompanyName     | Name of the application vendor. |
| Copyright       | Copyright notice for the application. |
| Version         | Version number of the application. |
| SplashFilename  | Filename and path for the application’s splash-screen graphic. |
| IconFilename    | Filename and path for the application icon. |
| VBAFilename     | If the application uses Microsoft VB, the filename and path to the icon displayed in the VB editor. |
| FormCLSID       | Not used. |
| FormFileExtension | Not used. |
| LoginOverrideProgID | Not used. |
| HelpBasePath    | HTTP path to the base help directory. The application takes this path and adds the Language/culture subfolder (for example, http://installation_folder/Language/en-US) plus the BaseHelpFile name to create the full path. |
| BaseHelpFile    | Name of the help file opened from the application menu command Help&gt;Contents and Index. |
| WallpaperFilename | Graphic file (of any supported .Net image file type) that displays as background wallpaper in the application window. |
| WallpaperDisplayOption | Specifies whether the background wallpaper image is cenetered, tiled, or scaled to fit the application window. |</p>
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VertComponentSpacing</td>
<td>The amount of vertical spacing between components, in character units. This is used by the application’s Arrange Column alignment buttons.</td>
</tr>
<tr>
<td>HorzComponentSpacing</td>
<td>The amount of horizontal spacing between components, in character units. This is used by the application’s Arrange Column alignment buttons.</td>
</tr>
<tr>
<td>ObjectMenuName</td>
<td>Caption of the application’s Objects (Actions) menu. This can be a literal string or the name of a string in the Strings table.</td>
</tr>
<tr>
<td>HelpContentsIndexMenuText</td>
<td>Alternative text to use for the Contents and Index command on the application’s Help menu.</td>
</tr>
<tr>
<td>DeleteExistingPrompt</td>
<td>Specifies text to display when a user tries to delete an existing record. This can be a literal string or the name of a string in the Strings table.</td>
</tr>
<tr>
<td>DeleteNewPrompt</td>
<td>Specifies text to display when a user tries to delete an unsaved record. This can be a literal string or the name of a string in the Strings table.</td>
</tr>
<tr>
<td>ProgIDRootAugment</td>
<td>Not used.</td>
</tr>
<tr>
<td>ReportViewerExecutable</td>
<td>Name of the report viewer executable program to use, if not the default.</td>
</tr>
<tr>
<td>Def...Color</td>
<td>Server-side default colors for form components, which can be overridden by the user.</td>
</tr>
<tr>
<td>DefFormFont</td>
<td>Server-side default font for forms, which can be overridden by the user.</td>
</tr>
<tr>
<td>UseHtmlHelp</td>
<td>Specifies the type of application help:</td>
</tr>
<tr>
<td></td>
<td>0 - Web-based help</td>
</tr>
<tr>
<td></td>
<td>1 - HTML Help (.chm)</td>
</tr>
<tr>
<td>ActiveScriptingOnly</td>
<td>Not used.</td>
</tr>
<tr>
<td>PrimaryCacheUsesFormCaption</td>
<td>Specifies what to use as the application’s Objects (Actions) menu caption if ObjectMenuName is not specified:</td>
</tr>
<tr>
<td></td>
<td>0 - Use the default name, Objects.</td>
</tr>
<tr>
<td></td>
<td>1 - Use the form caption as the menu caption.</td>
</tr>
<tr>
<td>ColSecurityPrimaryOnly</td>
<td>Specify whether insert and delete privileges apply to primary collection and/or subcollections:</td>
</tr>
<tr>
<td></td>
<td>0 - Security permissions apply to primary collection and to secondary or subcollections.</td>
</tr>
<tr>
<td></td>
<td>1 - Security permissions apply to primary collection only.</td>
</tr>
<tr>
<td>RightJustifyNewLabels</td>
<td>Specifies how new static components (labels) on forms are justified:</td>
</tr>
<tr>
<td></td>
<td>0 - Left justified</td>
</tr>
<tr>
<td></td>
<td>1 - Right justified</td>
</tr>
<tr>
<td>ValidationErrorIndicators</td>
<td>Not used.</td>
</tr>
<tr>
<td>DisableInvalidValuePrompt</td>
<td>If set to 1, the user is not prompted to change a value if the validation for a field fails.</td>
</tr>
<tr>
<td>XPVisualStyle</td>
<td>Not used.</td>
</tr>
</tbody>
</table>
Example - GetConfigurations

This request finds the available configurations:

```xml
<IDOREquest>
  <RequestHeader Type="GetConfigurations" ConfigGroup="Infor"/>
</IDOREquest>

<IDOResponse ProtocolVersion="5.00" SessionID="">
  <ResponseHeader Type="GetConfigurations">
    <InitiatorType /></InitiatorType>
    <InitiatorName /></InitiatorName>
    <SourceName /></SourceName>
    <TargetName /></TargetName>
    <ResponseData>
      <Config Name="SL8.01_OH_Prod">
        <AppName>Infor ERP SyteLine 8.01</AppName>
        <CompanyName>Infor Global Solutions Technology GmbH</CompanyName>
        ...additional configuration information...
      </Config>
    </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.

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

Request Data

```xml
<RequestData>
  <UserId>userid</UserId>
  <LanguageID>culture</LanguageID>
  <ConfigName>config</ConfigName>
  <MachineName>machinename</MachineName>
  <DomainUserName>userID</DomainUserName>
  <ApplicationName>appname</ApplicationName>
  <AllowCloseExistingSessions>False</AllowCloseExistingSessions>
  <Password Encrypted="N">pswd</Password>
  <Workstation>workstation</Workstation>
</RequestData>
```

Response Data

```xml
<ResponseData>
  <UserId>userid</UserId>
  <LanguageID>culture</LanguageID>
  <ProductVersion>version</ProductVersion>
  <DeadlockRetry>0</DeadlockRetry>
  <License Status="status">
    <Message>message</Message>
  </License>
  <RegionalSettings MessageLanguageID="langID" LocaleID="langID"
    DecimalSeparator="dec-sep" DigitGroupSeparator="dig-sep"
    DigitsInGroup="dig">
    <AdditionalFailureInformation>info</AdditionalFailureInformation>
  </RegionalSettings>
  <ServerDate>datetime</ServerDate>
  <LoginResult>status</LoginResult>
  <PrimaryGroupName>group</PrimaryGroupName>
  <DaysUntilPasswordExpires>days</DaysUntilPasswordExpires>
  <EditLevel>level</EditLevel>
  <SuperUser>su-indicator</SuperUser>
  <StartupMethods>startup-method-spec</StartupMethods>
  <AuditingEnabled>true|false</AuditingEnabled>
</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>LanguageID</td>
<td>Culture being used in this session. This corresponds to a value in the LanguageIDs table. Optional, default=en-US.</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>MachineName</td>
<td>Name of the computer being used in this session. Optional. Used primarily for diagnostics.</td>
</tr>
<tr>
<td>DomainUserName</td>
<td>Client's NT user name. Optional. Used primarily for diagnostics.</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>Application name being used in this session. Optional. Used primarily for diagnostics.</td>
</tr>
<tr>
<td>AllowCloseExistingSessions</td>
<td>If true, existing sessions may be closed if concurrent login limit prevents user from logging in. Optional, default = false.</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>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>DeadlockRetry</td>
<td>Number of times to retry connection if a deadlock occurs.</td>
</tr>
<tr>
<td>License</td>
<td>If login is successful, this indicates the status of the application database’s license (VALID, INVALID). If invalid, the message contains additional info such as License expired.</td>
</tr>
<tr>
<td>RegionalSettings</td>
<td>Regional settings and locale settings used in this session. This information comes from the LanguageIDs table.</td>
</tr>
<tr>
<td>AdditionalFailureInformation</td>
<td>If the login fails, this value may contain additional information about why.</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>LoginResult</td>
<td>Result of the specified user’s login attempt. Possible values: Success, InvalidCredentials, AccountDisabled, AccountLocked, PasswordExpired, PasswordWillExpire, SessionLimit, ConcurrentSessionLimit, InvalidConfiguration, UnknownFailure - should never happen, LicenseInconsistency - indicates that someone tampered with licensing</td>
</tr>
<tr>
<td>PrimaryGroupName</td>
<td>Primary group defined for the specified user on the Users form.</td>
</tr>
<tr>
<td>DaysUntilPasswordExpires</td>
<td>Days until the specified user’s current password expires</td>
</tr>
<tr>
<td>EditLevel</td>
<td>Form edit level from Users form: 0=None, 1=Basic, 2=Full, 3=Site (4=vendor).</td>
</tr>
<tr>
<td>SuperUser</td>
<td>Indicates whether the specified user is a SuperUser. 1=Yes, 0=No.</td>
</tr>
<tr>
<td>StartupMethods</td>
<td>Used to set some initial global application variables. Used only in WinStudio 5.00 and above.</td>
</tr>
<tr>
<td>AuditingEnabled</td>
<td>For use by WinStudio.</td>
</tr>
</tbody>
</table>
Example - OpenSession

This example opens a session using the configuration SL_OH for user jimbow.

```xml
<IDOResponse ProtocolVersion="5.00" SessionID="0a87afce-3a72-4faa-8b11-f26e72b4057d">
  <ResponseHeader Type="OpenSession">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <TargetName />
    <ResponseData>
      <UserID>jcurry</UserID>
      <LanguageID />
      <ProductVersion>8.00.00</ProductVersion>
      <DeadlockRetry>0</DeadlockRetry>
      <License Status="VALID">
        <Message />
      </License>
      <RegionalSettings MessageLanguageID="1033" LocaleID="1033"
        DecimalSeparator="." DigitGroupSeparator="," DigitsInGroup="3">
        ... list of regional settings ...
      </RegionalSettings>
      <AdditionalFailureInformation />
      <ServerDate>20070806 15:47:40.983</ServerDate>
      <LoginResult>Success</LoginResult>
      <PrimaryGroupName />
      <DaysUntilPasswordExpires>2147483647</DaysUntilPasswordExpires>
      <EditLevel>4</EditLevel>
      <SuperUser>1</SuperUser>
      <StartupMethods>... list of startup methods ...</StartupMethods>
      <AuditingEnabled>false</AuditingEnabled>
    </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="IDOName" IncludeClassNotesFlag="Y|N" />
</RequestData>
```

Response Data

```xml
<ResponseData>
  <GetPropertyInfo Name="IDOName"
                   IncludeClassNotesFlag="Y|N"
                   InsertPostedToWorkflow="Y|N"
                   ClassNotesExist="Y|N">
    <IDOProperties PrimaryBaseTable="tablename" PrimaryBaseTableAlias="alias">
      <Property Name="prop-name" PropertyClass="prop-class-name" Key="seq">
        <ProgID>IDOName</ProgID>
        <DataType DecimalPosition="n">type-name</DataType>
        <LinkBy><PropertyPair Parent="parentprop" Child="childprop" /></LinkBy>
        <Length>nn</Length>
        <ColumnType>sql-column-type</ColumnType>
        <ReadOnly />
        <Required />
        <RORecord />
        <Html />
        <Subcollection />
        <Identity />
        <Warning />
        <DefaultValue>default-value</DefaultValue>
        <Label>label-value</Label>
        <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>
              </DateFormat>
            </DefaultIMECharset>
          </InputMaskPromptChar>
        </InputMask>
      </Property>
    </IDOProperties>
  </GetPropertyInfo>
</ResponseData>
```
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
</table>
| GetPropertyInfo  | **Name** - Contains the name of the IDO for which the caller is requesting property information. Multiple GetPropertyInfo tags may be included in the RequestData, for multiple IDOs.  
**IncludeClassNotesFlag** - If set to Y in the request, the IDO runtime determines whether there are any class notes associated with the IDO’s primary base table and sets the **ClassNotesExist** attribute in the response to Y or N. Optional, default=N.  
**InsertPostedToWorkflow** - If set to Y in the response, there is a suspending event associated with this IDO for inserts. |
| IDOProperties    | Includes as attributes the primary base table and its alias, as defined in the IDO metadata. This information can be viewed from the IdoCollections form, on the Tables subcollection. It is primarily used internally.                                                                                                      |
| Property         | Includes these attributes:  
**Name** - Property name.  
**PropertyClass** - Property class associated with this property.  
**Key** - If the property is a key property, this attribute will be included and is the 1-based key sequence.  
**ProgID** - Included for sub-collection properties only. This is the IDO name of the subcollection property, for example, SL.SLItems.  
The rest of the elements under the <Property> tag correspond to fields on the IDO Properties form. |
| DataType         | Contains one of these values:  
Binary  
Byte  
CONTAINS  
Currency  
Date  
Decimal  
Float  
GUID  
Long Integer  
NumSortedString  
Object (pre-5.00 only)  
Short Integer  
String  
CONTAINS is used to specify relationships.  
The Length attribute specifies the length for String and Decimal datatypes. |
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
</table>
| LinkBy           | Valid only in CONTAINS relationship data. This is similar to the WinStudio LINKBY keyword. The information within this pair of tags tells how the CONTAINS relationship data is linked to the parent collection. For example:  
  `<LinkBy>`
  `<PropertyPair Child="Item" Parent="Item"/>
  <!-- Optional: more sets of PropertyPair tags -->
  </LinkBy>` |
| ColumnType       | Contains the SQL user-defined type of the column to which this property is bound. |
| ReadOnly         | Is included if the property has the read-only attribute set, and omitted otherwise. |
| Required         | Is included if the property has the required attribute set, and omitted otherwise. |
| RORecord         | Is included if the property has the read-only record flag attribute set, and omitted otherwise. |
| Html             | Is included if the property contains HTML text. |
| Subcollection    | Is included if the property is a subcollection. |
| Identity         | Is included if the property is bound to an identity (auto-incrementing) column, and omitted otherwise. |
| Warning          | Is included if this property contains warning messages for UpdateCollection responses, and omitted otherwise. |
| DefaultValue     | Returns the default value for the specified component. |
| Label            | Returns a string that is the component's label. |
| LabelStringID    | Not used. |
| DomainIDOName    | Contains the domain IDO where the property is found. For example, the DomainIDOName for a unit of measure code is SL.SLUMs. |
| BoolTrue         | Contains a value that represents True in Boolean evaluations of the component. |
| BoolFalse        | Contains a value that represents False in Boolean evaluations of the component. |
| InputMask        | Contains the pattern used for the formatting input mask. |
| InputMaskPromptChar | Contains the character used in the pattern to signify empty spaces. |
| DefaultIMECharset| Returns the character-set information for components which receive input from Asian language keyboards. |
| BinaryDataFormat | Indicates a graphic format:  
  **BMP** - bitmap  
  **ICO** - icon |
| Case             | Indicates the case of text:  
  **U** - uppercase  
  **L** - lowercase  
  **M** - mixed case |
Example 1: GetPropertyInfo

This request gets property information for SL.SLItems:

```xml
<IDORequest>
  <RequestHeader Type="GetPropertyInfo" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <RequestData>
      <GetPropertyInfo Name="SL.SLItems" />
    </RequestData>
  </RequestHeader>
</IDORequest>
```
<IDOResponse>
  <ResponseHeader Type="GetPropertyInfo" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <IDataRepresentation>
      <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...
    </IDataRepresentation>
  </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

```xml
<RequestData>
  <LoadCollection Name="collection" LoadCap="0">
    <LoadType>loadtype</LoadType>
    <ReadMode>mode</ReadMode>
    <Cursor />
    <PropertyList>
      ... list of properties...
    </PropertyList>
    <RecordCap>recordcap</RecordCap>
    <Filter>filter</Filter>
    <OrderBy>order</OrderBy>
    <PostQueryCmd>command</PostQueryCmd>
  </LoadCollection>
</RequestData>
```

Response Data

```xml
<ResponseData>
  <LoadCollection Name="collection" LoadCap="0">
    <ReadMode>mode</ReadMode>
    <Cursor>cursorvalue</Cursor>
    <PropertyList>
      ... list of properties...
    </PropertyList>
    <LoadType>loadtype</LoadType>
    <RecordCap>recordcap</RecordCap>
    <Filter>filter</Filter>
    <OrderBy>order</OrderBy>
    <PostQueryCmd>command</PostQueryCmd>
    <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>
</table>
| LoadCollection      | **Name** - This is the collection name (for example MGCore.GroupNames).  
|                     | **LoadCap** - Applies only to subcollections. This is the maximum number of subcollections to include in the result. Values for LoadCap are:  
|                     |   -1 - include all subcollections  
|                     |   1 (the default) - only query the subcollection records for the first parent item  
|                     | any other number - specify the maximum number of subcollections to retrieve.                                                                 |
| ReadMode            | Specify the collection read mode, which controls the isolation level used when executing queries (see the online help on Process Defaults). Options are:  
|                     |   ReadCommitted  
|                     |   ReadUncommitted  
|                     |   Default  
|                     | If the tag is empty or omitted, Default is assumed. The read mode appears in the response if the &lt;ReadMode&gt; tag in the request is anything but Default.  
|                     | This tag does not apply to custom load methods.                                                                                                |
| 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 &lt;cursor/&gt; tag in your FIRST request.  
|                     | The response document provides a cursor value. Include the response document's cursor value (as &lt;cursor&gt;value&lt;/cursor&gt;) 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 &lt;cursor/>.                                                                                       |
| PropertyList        | In request data, you only need to list the properties you want returned. The response data mimics the request list.  
|                     | For example:  
|                     |   &lt;PropertyList&gt;  
|                     |     &lt;Property&gt;name1&lt;/Property&gt;  
|                     |     ...  
<p>|                     |   &lt;/PropertyList&gt;                                                                                                                     |</p>
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordCap</td>
<td>This setting determines how many records are retrieved in one request.</td>
</tr>
<tr>
<td></td>
<td>Values are:</td>
</tr>
<tr>
<td></td>
<td>-1 (the default) - 200 records are retrieved</td>
</tr>
<tr>
<td></td>
<td>0 - uncapped - all records are retrieved</td>
</tr>
<tr>
<td></td>
<td>any other number - the specified number of records are retrieved.</td>
</tr>
<tr>
<td>Item</td>
<td>The ID attribute contains a value used within the system to identify the</td>
</tr>
<tr>
<td></td>
<td>item.</td>
</tr>
<tr>
<td>propertyvalue</td>
<td>For each item, this list contains values corresponding the properties</td>
</tr>
<tr>
<td></td>
<td>listed above.</td>
</tr>
<tr>
<td>LinkBy</td>
<td>Valid only in CONTAINS relationships. This is similar to the WinStudio</td>
</tr>
<tr>
<td></td>
<td>LINKBY keyword.</td>
</tr>
<tr>
<td></td>
<td>The information within this pair of tags tells how the CONTAINS</td>
</tr>
<tr>
<td></td>
<td>relationship is linked to the parent collection. For example:</td>
</tr>
<tr>
<td></td>
<td>&lt;LinkBy&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;PropertyPair Child=&quot;Item&quot; Parent=&quot;Item&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;!-- Optional: more sets of PropertyPair tags --&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/LinkBy&gt;</td>
</tr>
<tr>
<td>Distinct</td>
<td>Use this option to retrieve a set of data representing only the distinct</td>
</tr>
<tr>
<td></td>
<td>combinations of properties requested. This is similar to the WinStudio</td>
</tr>
<tr>
<td></td>
<td>option.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>&lt;LoadCollection Name=&quot;SL.SLItemcusts&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;LoadType&gt;FIRST&lt;/LoadType&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;PropertyList&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;CustNum/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/PropertyList&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Distinct/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/LoadCollection&gt;</td>
</tr>
<tr>
<td>Filter</td>
<td>The filter is used to restrict the result set, similar to the WinStudio</td>
</tr>
<tr>
<td></td>
<td>FILTERPERM keyword.</td>
</tr>
<tr>
<td></td>
<td>Any valid SQL where clause syntax is allowed.</td>
</tr>
<tr>
<td></td>
<td>Some examples:</td>
</tr>
<tr>
<td></td>
<td>&lt;Filter&gt;CoNum like 'ABC%'&lt;/Filter&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Filter&gt;QtyOnHand &lt; 20 AND Whse = 'OH'&lt;/Filter&gt;</td>
</tr>
<tr>
<td></td>
<td>In the response document, this element contains any IDO-level filter</td>
</tr>
<tr>
<td></td>
<td>specifications that were used.</td>
</tr>
<tr>
<td>OrderBy</td>
<td>Provides a comma-delimited list of properties that specify how the</td>
</tr>
<tr>
<td></td>
<td>response document result set should be sorted. The DESC keyword</td>
</tr>
<tr>
<td></td>
<td>may appear after a property name to sort that property descending.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>&lt;LoadCollection Name=&quot;SL.SLItemcusts&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;LoadType&gt;FIRST&lt;/LoadType&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;PropertyList&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;CustNum/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;CustItem/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/PropertyList&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;OrderBy&gt;CustNum, Item DESC&lt;/OrderBy&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/LoadCollection&gt;</td>
</tr>
</tbody>
</table>
Example 1: LoadCollection

This request retrieves the task type, task code, and description from the first 200 records in the MGCore.TaskTypes collection.

```xml
<IDORequest>
  <RequestHeader Type="LoadCollection" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <RequestData>
      <LoadCollection Name="MGCore.TaskTypes">
        <LoadType>FIRST</LoadType>
        <PropertyList>
          <TaskType/>
          <TaskCode/>
          <TaskDesc/>
          <NoteExistsFlag/>
        </PropertyList>
        <RecordCap>-1</RecordCap>
      </LoadCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>

<IDOResponse>
  <ResponseHeader Type="LoadCollection" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <ResponseData>
      <LoadCollection Name="MGCore.TaskTypes">
        <PropertyList>
          <TaskType/>
          <TaskCode/>
          <TaskDesc/>
          <NoteExistsFlag/>
        </PropertyList>
        <Items>
          <Item ID="TaskTypes @0.D='2007-04-18 16:31:53.730' @0.P='C444D3B2-FB2E-4416-97E2-D5BCA7F7FA74'">
            <P>11001</P>
            <P>JC2</P>
            <P>WebSvc test2</P>
            <P>0</P>
          </Item>
        </Items>
      </LoadCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

### Tag Description

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostQueryCmd</td>
<td>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: <code>&lt;PostQueryCmd&gt;MyPostQuerySp(Property1, Property2, REF Property3)&lt;/PostQueryCmd&gt;</code></td>
</tr>
</tbody>
</table>
Example 2: Custom LoadCollection Request

This is a variation of LoadCollection, allowing you to create a custom LoadCollection. It includes this tag:

```xml
<CustomLoadMethod Name="methodname"/>
```

and then a parameters list.

```xml
<IDOREquest>
  <RequestHeader Type="LoadCollection" SessionID="296d2410-4de7-4395-bd50">
    <RequestData>
      <LoadCollection Name="MGCore.UserNames" SessionID="296d2410-4de7-4395-bd50">
        <LoadType>FIRST</LoadType>
        <PropertyList>
          <Username/>
        </PropertyList>
        <Filter/>
        <RecordCap>-1</RecordCap>
      </LoadCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

Example 3: Nested LoadCollection Request

This request includes CONTAINS relationship records for all parent items.

```xml
<RequestData>
  <LoadCollection Name="SL.SLItemcusts" SessionID="296d2410-4de7-4395-bd50">
    <LoadType>FIRST</LoadType>
    <PropertyList>
      <CustNum/>
      <CustItem/>
      <Item/>
    </PropertyList>
    <Filter/>
    <RecordCap>0</RecordCap>
  </LoadCollection>
  <LoadCollection Name="SL.SLItems" LoadCap="-1">
    <LinkBy>
      <PropertyPair Child="Item" Parent="Item"/>
    </LinkBy>
    <LoadType>FIRST</LoadType>
    <PropertyList>
      <Item/>
      <ProductCode/>
      <DerQtyOnHand/>
    </PropertyList>
    <Filter/>
    <RecordCap>0</RecordCap>
  </LoadCollection>
</RequestData>
```
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**

```
<RequestData>
  <UpdateCollection Name="Collection" RefreshAfterUpdate="Y" [TxnScope="Item" | "Collection"]>
    <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**

```
<ResponseData>
  <UpdateCollection Name="Collection" RefreshAfterUpdate="Y">
    <Items>
      <Item Action="ActionType" ItemNo="recordnumber" ID=itemID
             Id.RecordDate='date'
             Id.RowPointer='row-pointer'>
        <Property Name="property" Modified="Y">data</Property>
        <Property Name="property" Modified="Y">data</Property>
        more properties...
      </Item>
      more items...
    </Items>
  </UpdateCollection>
</ResponseData>
```
### Tag: 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.

- **TxnScope** - This optional attribute can be set to Collection (the default value) or Item. When TxnScope is set to Item, each individual item in the UpdateCollection request is saved in a separate transaction. For a hierarchical (nested) UpdateCollection request, the value of the TxnScope attribute at the root level determines the behavior for the entire UpdateCollection request. If an exception occurs while processing an UpdateCollection request using the Item TxnScope, records that were saved before the exception are still committed, but no additional records are saved.

  If TxnScope is omitted or is set to Collection, the request is processed in a single transaction, as in previous versions.

  This attribute has no effect when it is included in an inner UpdateCollection within hierarchical requests.

- **CollectionID**

  Used when reporting errors that occur while processing items in an UpdateCollection request. Optional; if CollectionID is omitted in the request, IDOName is used instead in any error response.

  An UpdateCollection request can contain hierarchical (nested) requests. Having a collection ID allows the system response to include both the error message and which item caused the error. (See example 3 below)

### Tag: Item

The tag can contain these attributes:

- **Action** - The action type can be Insert, Update or Delete. For Inserts, if the InitiatorType (see page 24) 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.
TIP: 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.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 CONTAINS relationships. Similar to the WinStudio LINKBY keyword.  
The information within this pair of tags tells how the CONTAINS relationship is linked to the parent collection. For example:  
  <LinkBy>  
    <PropertyPair Child="Item" Parent="Item"/>  
  </LinkBy>  
  <!-- Optional: more sets of PropertyPair tags -->  
  </LinkBy> |
Example 1: UpdateCollection - Insert, Update and Delete Items

```xml
<IDORequest>
  <RequestHeader Type="UpdateCollection" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <RequestData>
      <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>
    </RequestData>
  </RequestHeader>
</IDORequest>

<IDOResponse>
  <ResponseHeader Type="UpdateCollection" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <ResponseData>
      <UpdateCollection Name="MGCore.TaskTypes" RefreshAfterUpdate="Y">
        <Items>
          <Item Action="Insert" ItemNo="0" ID="PBT=TaskTypes.TaskTypes.RecordDate='2007-04-18 16:31:53.730' TaskTypes.RowPointer='C444D3B2-FB2E-4416-97E2-D5BCA7F7FA74'">
            <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>
  </ResponseHeader>
</IDOResponse>
```
Example 2: UpdateCollection - Nested Insert of Group and Users

This request inserts a user-defined type and some values for it.

```
<IDORequest ProtocolVersion="5.00" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <RequestHeader Type="UpdateCollection">
        <InitiatorType>Form</InitiatorType>
        <InitiatorName>UserDefinedTypes</InitiatorName>
        <SourceName/>
        <TargetName/>
        <RequestData>
            <UpdateCollection Name="MGCore.UserDefinedTypes" RefreshAfterUpdate="Y">
                <CollectionID/>
                <Items>
                    <Item ID="" ItemNo="1" Action="Insert">
                        <Property Name="Name">ColorType</Property>
                        <Property Name="NoteExistsFlag" Modified="N"/>
                        <Property Name="RowPointer" Modified="N"/>
                        <Property Name="_ItemWarnings" Modified="N"/>
                        <Property Name="Description">New type</Property>
                        <UpdateCollection Name="MGCore.UserDefinedTypeValues" RefreshAfterUpdate="Y">
                            <LinkBy>
                                <PropertyPair Parent="Name" Child="TypeName"/>
                            </LinkBy>
                            <CollectionID>UserDefinedTypeValues</CollectionID>
                            <Items>
                                <Item ID="" ItemNo="0" Action="Insert">
                                    <Property Name="Value">Red</Property>
                                    <Property Name="TypeName">ColorType</Property>
                                    <Property Name="NoteExistsFlag" Modified="N"/>
                                    <Property Name="RowPointer" Modified="N"/>
                                    <Property Name="_ItemWarnings" Modified="N"/>
                                    <Property Name="Description">red</Property>
                                </Item>
                                <Item ID="" ItemNo="1" Action="Insert">
                                    <Property Name="Value">Green</Property>
                                    <Property Name="TypeName">ColorType</Property>
                                    <Property Name="NoteExistsFlag" Modified="N"/>
                                    <Property Name="RowPointer" Modified="N"/>
                                    <Property Name="_ItemWarnings" Modified="N"/>
                                    <Property Name="Description">green</Property>
                                </Item>
                            </Items>
                        </UpdateCollection>
                    </Item>
                </Items>
            </UpdateCollection>
        </RequestData>
    </RequestHeader>
</IDORequest>
```
<TargetName />
<ResponseData>
  <UpdateCollection Name="MGCore.UserDefinedTypes" RefreshAfterUpdate="Y">
    <CollectionID />
    <Items>
      <Item ID="PBT=[UserDefinedTypes] UserDefinedTypes.ID=[a955821a-67d2-4771-bf99-89ff949e9f6e]" ItemNo="1" Action="Insert">
        <Property Name="Name" Modified="N">ColorType</Property>
        <Property Name="NoteExistsFlag" Modified="N">0</Property>
        <Property Name="RowPointer" Modified="N">a955821a-67d2-4771-bf99-89ff949e9f6e</Property>
        <Property Name="_ItemWarnings" Modified="N" />
        <Property Name="Description" Modified="N">my new type</Property>
      </Item>
      <UpdateCollection Name="MGCore.UserDefinedTypeValues" RefreshAfterUpdate="Y">
        <LinkBy>
          <PropertyPair Parent="Name" Child="TypeName" />
        </LinkBy>
        <CollectionID />
        <Items>
          <Item ID="PBT=[UserDefinedTypeValues] UserDefinedTypeValues.ID=[6bbfb52d-20aa-40d6-ald6-f5596f934c30]" ItemNo="0" Action="Insert">
            <Property Name="Value" Modified="N">Red</Property>
            <Property Name="TypeName" Modified="N">ColorType</Property>
            <Property Name="NoteExistsFlag" Modified="N">0</Property>
            <Property Name="RowPointer" Modified="N">6bbfb52d-20aa-40d6-ald6-f5596f934c30</Property>
            <Property Name="_ItemWarnings" Modified="N" />
            <Property Name="Description" Modified="N">red</Property>
          </Item>
          <Item ID="PBT=[UserDefinedTypeValues] UserDefinedTypeValues.ID=[01ef0dd7-2c28-43bf-8c72-0d9afe742ef4]" ItemNo="1" Action="Insert">
            <Property Name="Value" Modified="N">Green</Property>
            <Property Name="TypeName" Modified="N">ColorType</Property>
            <Property Name="NoteExistsFlag" Modified="N">0</Property>
            <Property Name="RowPointer" Modified="N">01ef0dd7-2c28-43bf-8c72-0d9afe742ef4</Property>
            <Property Name="_ItemWarnings" Modified="N" />
            <Property Name="Description" Modified="N">green</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </Items>
  </UpdateCollection>
</ResponseData>
</ResponseHeader>
</IDOResponse>
Example 3: UpdateCollection Error Response

If you try to insert an item that already exists in the collection, the response returns an error indicating the item where the problem occurred. The ErrorMessage tag has an attribute named ItemNo, in this format:

```xml
<top-level-ItemNo>[<collection-id>.<child-collection-ItemNo>[…]]
```

If CollectionID is omitted in the request, IDOName is used instead.

```xml
<IDORequest ProtocolVersion="5.00" SessionID="11d39624-5d71-436f-8554-708922b0a1f4">
  <RequestHeader Type="UpdateCollection">
    ...header information...
  </RequestHeader>
  <RequestData>
    <UpdateCollection Name="MGCore.UserDefinedTypes" RefreshAfterUpdate="Y">
      <CollectionID />
      <Items>
          ItemNo="23" Action="Update">
          <Property Name="Name" Modified="N">ColorType</Property>
          <Property Name="NoteExistsFlag" Modified="N">0</Property>
          <Property Name="RowPointer" Modified="N">a955821a-67d2-4771-bf99-89ff949e9f6e</Property>
          <Property Name="_ItemWarnings" Modified="N" />
          <Property Name="Description" Modified="N">New type</Property>
        </Item>
        <UpdateCollection Name="MGCore.UserDefinedTypeValues"
          RefreshAfterUpdate="Y">
          <LinkBy>
            <PropertyPair Parent="Name" Child="TypeName" />
          </LinkBy>
          <CollectionID>UserDefinedTypeValues</CollectionID>
          <Items>
            <Item ID="" ItemNo="2" Action="Insert">
              <Property Name="Value">Red</Property>
              <Property Name="TypeName">ColorType</Property>
              <Property Name="NoteExistsFlag" Modified="N" />
              <Property Name="RowPointer" Modified="N" />
              <Property Name="_ItemWarnings" Modified="N" />
              <Property Name="Description">duplicate</Property>
            </Item>
          </Items>
        </UpdateCollection>
      </Items>
    </UpdateCollection>
  </RequestData>
</IDORequest>

<IDOResponse ProtocolVersion="5.00" SessionID="11d39624-5d71-436f-8554-708922b0a1f4">
  <ResponseHeader Type="UpdateCollection">
    ...header information
  </ResponseHeader>
  <ResponseData />
  <ErrorMessage ItemNo="23.UserDefinedTypeValues.2">
The Type Name, Value combination entered already exists.
  </ErrorMessage>
</IDOResponse>

Example 4: 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" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
  <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='32537FE4-4BD5-8C9B-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" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <ResponseData>
      <UpdateCollection Name="MGCore.TaskTypes" RefreshAfterUpdate="Y">
        <Items>
          <Item Action="Update" ItemNo="0" ID="TaskTypes.TaskTypes.TaskTypes TaskTypes.TaskTypes.RecordDate='2004-05-11 15:25:31.753' TaskTypes.TaskTypes.RowPointer='32537FE4-4BD5-8C9B-8A5EA27D6A33'">
            <Property Name="TaskDesc">Updated TaskCode</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
Example 5: UpdateCollection - Update Using Keys

This request updates existing records using the UseKeys attribute:

```
<IDORequest>
  <RequestHeader Type="UpdateCollection" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <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" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
    <ResponseData>
      <UpdateCollection Name="MGCore.TaskTypes" RefreshAfterUpdate="Y">
        <Items>
          <Item Action="Update" ItemNo="0" ID="PBT=TaskTypes TaskTypes.RecordDate='2004-05-11 15:25:31.753' TaskTypes.RowPointer='32537FE4-64D4-4B55-8C9B-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>PRIMARY</Parameter>
    <Parameter ByRef="Y"/>
    <Parameter ByRef="Y"/>
    <Parameter ByRef="Y"/>
  </Parameters>
</RequestData>
```

Response Data

```xml
<ResponseData>
  <Name>IDOName</Name>
  <Method>MethodName</Method>
  <Parameters>
    <Parameter>PRIMARY</Parameter>
    <Parameter ByRef="Y">n</Parameter>
    <Parameter ByRef="Y">parm</Parameter>
    <Parameter ByRef="Y"/>
  </Parameters>
  <ReturnValue>n</ReturnValue>
</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>ByRef - Parameters that are marked as ByRef in the IDO metadata will be returned in the response, regardless of the ByRef value in the request. If set to Y, the parameter is input/output. If set to N, or if the attribute is omitted, the parameter is input-only. Optional.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> Parameters should be listed in the order that the method is expecting them. If you use the obsolete Seq attribute, it will be ignored.</td>
</tr>
</tbody>
</table>
Example - Invoke

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

```xml
<IDOREquest ProtocolVersion="5.00" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
  <RequestHeader Type="Invoke">
    <RequestData>
      <Name>MGCore.GroupNames</Name>
      <Method>GroupValid</Method>
      <Parameters>
        <Parameter>Developers</Parameter>
        <Parameter ByRef="Y"/>
        <Parameter ByRef="Y"/>
        <Parameter ByRef="Y"/>
      </Parameters>
    </RequestData>
  </RequestHeader>
</IDOREquest>

<IDOResponse ProtocolVersion="5.00" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
  <ResponseHeader Type="Invoke">
    <ResponseData>
      <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>
      <ReturnValue VT="2">16</ReturnValue>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```
CloseSession

This XML request/response document closes the current session.

Request Data
none

Response Data
none

Example - CloseSession

```xml
<IDOResponse ProtocolVersion="5.00" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
  <ResponseHeader Type="CloseSession">
    <InitiatorType />  
    <InitiatorName />  
    <SourceName />  
    <TargetName />  
    <ResponseData />  
  </ResponseHeader>
</IDOResponse>
```

```xml
<IDOResponse ProtocolVersion="5.00" SessionID="296d2410-4de7-4395-bd50-58a89df3c3c6">
  <ResponseHeader Type="CloseSession">
    <InitiatorType />  
    <InitiatorName />  
    <SourceName />  
    <TargetName />  
    <ResponseData />  
  </ResponseHeader>
</IDOResponse>
```
Passing Dates to the IDO Runtime

When passing dates to the IDO runtime (through XML documents, .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.

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 Infor ERP SyteLine guide.
Using the Event System

When Would You Use This?

You can use Infor ERP SyteLine’s application event system to send IDO request/responses via framework events. For example, when an event occurs, it can trigger an update to a collection or execute an IDO method or stored procedure.

Event Action Types Available for IDO Integration

The following action types are available:

- **Dispatch IDO Request** - Dispatch an IDO request to a URL. (The result can be parsed using the Transform XML action type.) This type of action opens a session, performs the request, and closes the session. You could use this action to request communication with the IDO runtime at another “enterprise” (for example, one of your customers or vendors who also uses SyteLine) or with another site in your own enterprise. The URL on the receiving end must point to a WinStudio framework-based application.

- **Execute IDO Request** - Execute an IDO Request. (The result can be parsed using the Transform XML action type.) This is similar to Dispatch IDO Request, but it only goes to the current configuration’s IDO runtime (that is, it runs against the requester’s own application database).

- **Load Collection** - Load rows of an IDO collection, up to the record cap.

- **Load IDO Row** - Retrieve the first matching row of data from an IDO collection. There will be a scenario showing how this works in the Guide to the Application Event System.

- **Update Collection** - Update data in a collection.

- **Call IDO Method** - Invoke an IDO method.

- **Call Database Method** - Similar to a dynamic IDO (SP!), this directly calls a method in a database.

- **Call Web Service** - Invoke and run a particular .NET Web service. (See Chapter 4, “Using the .NET Web Service API,” for more information.)

- **Transform XML** - Perform an XSL transformation on the designated object. This can be used to parse elements from the result of a Dispatch. It also might be needed if, for example, the responding site is running a different version of the application (with a slightly different schema) than the requesting site.
Example

Opening a Session and Requesting Data for a Remote Site

You can use the application event system to call into another site or to an external Infor ERP SyteLine environment and return a specific, discrete piece of information from the remote application database. This example returns the On Hand quantity of an item in another site.

You will create a new event that accepts two input parameters:

- **Item** specifies the item whose On Hand quantity you want to see.
- **Site** specifies the site whose URL is to be asked for the results.

The event returns an output parameter called **OnHand** that contains the On Hand quantity as reported by the SLItems IDO for that site.

To set up and test this event, do the following:

1. For some flexibility, define the site URLs as **Event Global Constants**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiteA_URL</td>
<td>URL of SiteA’s IDO Request Processor, for example:</td>
</tr>
<tr>
<td></td>
<td><a href="http://mySiteAUtilServer/IDORequestService/RequestService.aspx">http://mySiteAUtilServer/IDORequestService/RequestService.aspx</a></td>
</tr>
<tr>
<td>SiteB_URL</td>
<td>URL of SiteB’s IDO Request Processor, for example:</td>
</tr>
<tr>
<td></td>
<td><a href="http://mySiteBUtilServer/IDORequestService/RequestService.aspx">http://mySiteBUtilServer/IDORequestService/RequestService.aspx</a></td>
</tr>
</tbody>
</table>

   Later, you will use the pre-parser keyword TV() to select the appropriate one of these URLs, based on the site from which you want to get data. If both sites are on the same intranet, they can use the same URL.

2. Create a text file containing a parameterized IDO request to retrieve the On Hand quantity of an item:

   ```xml
   <RequestHeader Type="LoadCollection">
     <InitiatorType />
     <InitiatorName />
     <SourceName />
     <TargetName />
     <RequestData>
       <LoadCollection Name="SL.SLItems" LoadCap="0">
         <PropertyList>
           <DerQtyOnHand />
         </PropertyList>
         <LoadType>FIRST</LoadType>
         <RecordCap>-1</RecordCap>
         <Filter>Item = {0}</Filter>
         <OrderBy />
         <PostQueryCmd />
       </LoadCollection>
     </RequestData>
   </RequestHeader>
   ```
Notice the special syntax {0} that gives us the ability to substitute a different Item key for each run. Notice also that we include other properties that might be retrieved by other events, so we can reuse this request file in those events' handlers.

3. Name this file `Load1ItemInfo.xml`, and copy (or share) it to each utility server that will run the IDO Runtime Service. In this example, we place it on a common network folder which is mapped everywhere as `V:\Scenarios`.

4. In the Event Handlers form, create a handler where the Event Name is `GetRemoteOnHandQty` and the Synchronous check box is selected.

5. Click the Event Actions button and specify the following actions for the new event handler:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>ActionType</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set Values</td>
<td>SETVARVALUES(SiteID=E(Site))</td>
</tr>
<tr>
<td>2</td>
<td>Dispatch IDO Request</td>
<td>URL(GC(TV(SiteID)_URL)) CONFIGNAME(E(Site)) SET(RV(IDO_Response)=RESULT) IDOREQUEST( SUBSTITUTE(FILECONTENTS(&quot;V:\Scenarios\Load1ItemInfo.xml&quot;), FE(Item)) )</td>
</tr>
</tbody>
</table>

These actions are interpreted as follows:

- Copy the value of the input parameter named Site into a variable named SiteID. We need this in order to use the TV() keyword in the next action.
- Construct and dispatch an IDO request as follows:
  - Evaluate the variable named SiteID, append the string "_URL", and look up the result as the name of an Event Global Constant. Use its value as the URL address to which to dispatch the request.
  - Signal the remote IDO Request Service to execute the request against its configuration named after the site. This configuration must exist there and point to that site’s application database.
  - Read the contents of the file `V:\Scenarios\Load1ItemInfo.xml`, substituting the value of the input parameter named Item wherever {0} appears therein. Use the resulting value as the IDO request to be dispatched.
  - Put the IDO response into a new variable named IDO_Response. The value will look similar to this (varies depending on the item requested):
60  Using the Event System

Infor ERP SyteLine Integrating IDOs with External Applications

Copyright © 2010 Infor

<ResponseHeader Type="LoadCollection">
  <InitiatorType />
  <InitiatorName />
  <SourceName />
  <TargetName />
  <ResponseData>
    <LoadCollection Name="SL.SLItems" LoadCap="0">
      <PropertyList>
        <Item />
        <Description />
        <UnitCost />
        <CurUCost />
        <LotSize />
        <UnitWeight />
        <WeightUnits />
        <DerQtyOnHand />
      </PropertyList>
      <LoadType>FIRST</LoadType>
      <RecordCap>-1</RecordCap>
      <Filter>Item = 'PT-70000'</Filter>
      <OrderBy />
      <PostQueryCmd />
      <Items>
        <Item ID="PBT=[item] itm.DT=[20070618 08:30:06.250] itm.ID=[c6cb54f1-258e-4ab6-ba0a-e182bc4d32c1]">
          <P>PT-70000</P>
          <P>Paint,Purple</P>
          <P>0.50000000</P>
          <P>0.50000000</P>
          <P>1.00000000</P>
          <P>1.00000</P>
          <P>LB</P>
          <P>279.00000000</P>
        </Item>
      </Items>
    </LoadCollection>
  </ResponseData>
</ResponseHeader>

- Perform an XSL transform on the contents of the variable named IDO_Response. The transform uses XPath to select all last children named P. (We know that DerQtyOnHand is the last property because we listed it last in the <PropertyList> of our IDO request. Instead of "[last()]" here, we could equivalently say "[8]".) Put the result of the transform into an output parameter called OnHand.

6. To see the use of this event interactively, construct a new form like this:

![Get Remote On Hand Quantity Form](image)

Infor ERP SyteLine Integrating IDOs with External Applications

Copyright © 2010 Infor
On the form’s **Get Data** button, define a form event handler of Response Type "Generate App Event" with these parameters:

7. On your new form, the Site and Item drop-downs can easily be populated by inline and IDO collection list sources.

   Additional Event Global Constants could be used for additional sites and/or external SL servers.

8. Test the form: after selecting a site and item, click the **Get Data** button to dispatch the IDO request to the desired site’s URL URL against the desired site’s configuration. It should return and display the On Hand quantity for that item in that site’s application database.

For More Information

Details about creating and using events are found in the online help and in the *Guide to the Application Event System*. 
Using the .NET Web Service API

When Would You Use This?

It is possible 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.

About .NET WebService API Calls

The .NET web service 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.

You can also call dynamic IDOs using .NET, as described on page 73.

Prerequisite

Before you use .NET Web APIs to transfer data between your external interface and Infor ERP SyteLine, make sure the Intranets form includes the appropriate URL information for both the external application and SL.
API Calls

Use the prototypes shown in this chapter as examples when creating your API calls.

All prototypes are using C# code.

Also, this code assumes you have added a reference to your project for the IDO .Net web service, as described below.

In Visual Studio 2008:

- Right-click the References node in the solution explorer, or select Add Service Reference from the Project menu.
- Enter the URL of your .NET Web Service:
  
  http://server_name/IDORequestService/IDOWebService.asmx

- Set Namespace to IDOWebServiceReference.
- Click OK.
- You can redirect calls at runtime (for instance, to a production server rather than a test server) by passing two parameters to the constructor, for example:

  IDOWebServiceReference.DOWebServiceSoapClient idoWS =
  new IDOWebServiceReference.DOWebServiceSoapClient( "IDOWebServiceSoap",
  "http://other_server_name/IDORequestService/IDOWebService.asmx" );

In Visual Studio 2005:

- Right-click the References node in the solution explorer, or select Add Web Reference from the Project menu.
- Enter the URL of your .NET Web Service:
  
  http://server_name/IDORequestService/IDOWebService.asmx. Or you can click the option to browse for WebServices on your local machine, if you have installed the SyteLine 8 utility server (or SyteLine 7 web server) on the machine where you are developing your application.
- Set Web reference name to IDO.
- Click Add Reference.
- You can redirect calls at runtime (for instance, to a production server rather than a test server) by setting the .Url property after constructing the instance, for example:

  webService.Url = "http://<web-service-url>";

NOTE: You can add a query parameter to the .NET Web Service URL to specify the configuration group, which applies to the GetConfigurationNames web service API:

  http://server_name/IDORequestService/IDOWebService.asmx?configgroup=group-name

where group-name corresponds to a configuration group created with the Configuration Manager utility.
GetConfigurationNames

Returns a list of valid configuration names. This call only has access to configurations in the Default group.

Syntax

public string[] GetConfigurationNames()

Parameters

None

Return Value

Returns all valid configuration names.

Example

// Visual Studio 2005:
IDO.IDOWebService webService = new IDO.IDOWebService();
webService.Url = "http://<web-service-url>");
string[] astrConfigNames;
astrConfigNames = webService.GetConfigurationNames();

// Visual Studio 2008:
IDOWebServiceReference.DOWebServiceSoapClient webService =
    new IDOWebServiceReference.DOWebServiceSoapClient();
string[] astrConfigNames;
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 Infor ERP SyteLine.

Syntax

```csharp
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

```csharp
// Visual Studio 2005:
IDO.IDOWebService webService = new IDO.IDOWebService();
webService.Url = "http://<web-service-url>";
SessionToken = webService.CreateSessionToken(
    "sa",
    "secret",
    "SyteLine" );

// Visual Studio 2008:
IDOWebServiceReference.DOWebServiceSoapClient webService =
    new IDOWebServiceReference.DOWebServiceSoapClient( );
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. The returned data may be filtered by any IDO-level filters. See the online help about IDO Filters.

Example

```csharp
// Visual Studio 2005:
IDO.IDOWebService webService = new IDO.IDOWebService();
webService.Url = "http://<web-service-url>";
DataSet IdoDataSet;
IdoDataSet = webService.LoadDataSet(
    SessionToken,
    "SL.SLItems",
    "Item, Description, UnitCost",
    "Item = N'FA-10000'",
    ",", ",", -1 );

// Visual Studio 2008:
IDOWebServiceReference.DOWebServiceSoapClient webService =
    new IDOWebServiceReference.DOWebServiceSoapClient( );
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

```csharp
// Visual Studio 2005:
IDO.IDOWebService webService = new IDO.IDOWebService();
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, "", "", "" );
```
// Visual Studio 2008:
IDOWebServiceReference.DOWebServiceSoapClient webService =
    new IDOWebServiceReference.DOWebServiceSoapClient();
string   sessionToken;
DataSet  custDataSet;
DataTable idoTable;
DataRow   newCust;
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. The strMethodParameters argument looks like this:</td>
</tr>
</tbody>
</table>

```xml
<Parameters>
    <Parameter ByRef="Y|N">value</Parameter>
    ...etc...
</Parameters>
```

Return Value

Returns the value that was returned to it by the method.

Example

```csharp
// Visual Studio 2005:
IDO.IDOWebService webService = new IDO.IDOWebService();
webService.Url = "http://<web-service-url>";
object ReturnValue;
string strParameters = "<Parameters><Parameter
ByRef="Y"/></Parameters>";
ReturnValue = webService.CallMethod(
    SessionToken,
    "SL.SLParms",
    "GetSessionIDSp",
    ref strParameters );

// Visual Studio 2008:
IDOWebServiceReference.DOWebServiceSoapClient webService =
    new IDOWebServiceReference.DOWebServiceSoapClient();
object ReturnValue;
string strParameters = "<Parameters><Parameter
ByRef="Y"/></Parameters>";
ReturnValue = webService.CallMethod(
    SessionToken,
    "SL.SLParms",
    "GetSessionIDSp",
    ref strParameters );
```
.NET Web Service Example

You could create an application form in C# with a display area and a Query Customers button, similar to this:

```
String sessionToken;
DataSet idoDataSet;
// Visual Studio 2005:
IDO.IDOWebService idoWS = new IDO.IDOWebService();
// Visual Studio 2008:
IDOWebServiceReference.DOWebServiceSoapClient idoWS =
    new IDOWebServiceReference.DOWebServiceSoapClient();
sessionToken = idoWS.CreateSessionToken(
    "user2",
    "",
    "SL800_Local" );
idoDataSet = idoWS.LoadDataSet(
    sessionToken,
    "SLCustomers",
    "Name, CustNum, CustSeq",
    "",
    "",
    "",
    0 );
dataGridView1.DataMember = idoDataSet.Tables[0].TableName;
dataGridView1.DataSource = idoDataSet;
```
When you click the **Query Customers** button, the form displays the SL customer name, number and sequence information, queried through the SLCustomers IDO:

When setting this up, you need to add a web reference to the Infor ERP SyteLine WebService.

To do this in Visual Studio 2005:

- Right-click the References node in the solution explorer, or select **Add Web Reference** from the Project menu.
- Enter the URL of your SyteLine .NET Web Service:
  
  http://server_name/IDORequestService/IDOWebService.asmx. Or you can click the option to browse for WebServices on your local machine, if you have installed the SyteLine 8 utility server (or SyteLine 7 web server) on the machine where you are developing your application.
- Set Web reference name to **IDO**.
- Click **Add Reference**.

To do this in Visual Studio 2008:

- Right-click the References node in the solution explorer, or select **Add Service Reference** from the Project menu.
- Enter the URL of your .NET Web Service:
  
  http://server_name/IDORequestService/IDOWebService.asmx
- Set Namespace to **IDOWebServiceReference**.
- Click **OK**.
Calling Dynamic IDOs Using .NET

IDO requests 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 “SPI!” 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 *Infor ERP SyteLine Installation Guide*. 
Using Infor SyteLine SharePoint Web Parts

When Would You Use This?

If your company uses SharePoint to share information among employees, the SharePoint pages can include charts and lists that display current SyteLine data. For example, you can display a chart showing sales year-to-date by salesperson, or a list of customers on credit hold. If a list is set up to allow it, you can also drill back to open the SyteLine form that contains the data.

Infor SyteLine SharePoint web parts retrieve and display on a SharePoint site the IDO data from a specific SyteLine configuration. You can download, deploy, and configure these web parts as described in this section, so that the web parts display specific SyteLine information. Administrators can include the configured web parts on shared pages, or they can export the configured web parts so that users can import them to their pages. Individual users can reconfigure each web part to point to a different SyteLine configuration or to log in with a different user name.

About the IDO Web Parts

These web parts are available:

- **Infor Chart Web Part** - displays a chart based on a SyteLine IDO. You can define some chart settings.

- **Infor List Web Part** - displays a list based on a SyteLine IDO. This is similar to the collection of information you see on many SyteLine forms.

The deployment also includes an administrative web page where you configure the connection settings for the web parts.

The web parts are described in detail later in this section. For more information about what web parts are, and how to set up or use SharePoint, go to the MSDN web site.
Prerequisites

These must be installed:

- SyteLine 8.00 or above
- .NET 3.5 SP1
- Microsoft Chart Controls for Microsoft .NET Framework 3.5. The executable to install the chart controls (MSChart.exe) is included in the MGWebParts.zip file.

Deploying the Web Parts in SharePoint

To load the web parts and then deploy and activate them in SharePoint:

1. Download the MGWebParts.zip file from the Infor365 downloads area to the SharePoint server. The ZIP file contains the MGWebParts.wsp SharePoint deployment package and the Microsoft Chart Controls installation executable file.
2. Extract the contents of the MGWebParts.zip file to a folder on your SharePoint server.
3. On the Sharepoint server, open a new command prompt window with administrator privileges:
   - For Windows 2008 Server, right-click on the Command Prompt menu item and select Run as administrator.
   - For Windows 2003 Server, right-click on the Command Prompt menu item and select Run as... to display the Run dialog, where you can choose to run with full administrative privileges.
4. In the command prompt window, change your current folder to the location where you unzipped the contents of the MGWebParts.zip file.
5. If you are upgrading from a previous version of the Infor web parts, retract and remove the old version:

   NOTE: The path to stsadm.exe shown in the commands below is specific to SharePoint 3.0. For SharePoint 2010, the 12 must be changed to 14.

   a. In the command prompt window, run this command to deactivate the feature:
      "C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN\stsadm.exe" -o deactivatefeature -filename mgwebparts\feature.xml -url SharepointServerName
   b. When that command completes, run this command to retract the solution:
      "C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN\stsadm.exe" -o retractsolution -name mgwebparts.wsp -immediate -url SharepointServerName
   c. When that command completes, run this command to delete the solution:
      "C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN\stsadm.exe" -o deletesolution -name mgwebparts.wsp
   d. Do not close the command prompt window, because you will use it later.
6. If the Microsoft Chart Controls for Microsoft .NET Framework 3.5 are not already installed, run `MSChart.exe` in the folder on the Sharepoint server where you extracted the contents of the zip file.

7. To add and deploy the solution and activate the feature in SharePoint:

**NOTE:** The path to `stsadm.exe` shown in the commands below is specific to SharePoint 3.0. For SharePoint 2010, the 12 must be changed to 14.

a. In the command prompt window, run this command to add the SyteLine SharePoint Web Parts solution to your SharePoint server:

   "C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN\stsadm.exe" -o addsolution -filename mgwebparts.wsp

b. When that command completes, run this command to deploy the solution:

   "C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN\stsadm.exe" -o deploysolution -name mgwebparts.wsp -immediate -allowgacdeployment -url SharepointServerName

c. When that command completes, run this command to activate the feature:

   "C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN\stsadm.exe" -o activatefeature -filename mgwebparts\feature.xml -url SharepointServerName

The web parts are now deployed and activated.

**Configuring the Connection to SyteLine at SharePoint Sites**

Any SharePoint user who displays SyteLine information in a web part uses SyteLine configuration and login information to connect to SyteLine. The connection lasts only long enough to retrieve current data.

At a SharePoint site where the web parts are deployed, use the **Site Settings > Infor Web Part Configuration** page to configure the default connection settings for all web parts. The URL you specify at each site must be used by all web parts on that site; for individual web parts, users can override the configuration name, user ID and password.

All web parts on a site connect to the same SyteLine configuration server, but individual web parts can override the configuration and user login set for a site. This allows you to
compare data from different SyteLine sites in side-by-side graphs on the same SharePoint page. The following example illustrates this:

Follow these steps to configure the SyteLine connection for a site:

1. From a SharePoint site where the web parts are deployed, select **Site Actions > Site Settings**.
2. On the Site Settings page, under Site Administration, click **Infor web part configuration**.
   
   **NOTE:** This page also displays the version number of the web parts.
3. Specify these settings:
   
   - **Config Server URL** - Specify the URL to the configuration server, that is, the utility or web server where the SyteLine configuration you want to access is defined. In the default value shown in the field, replace `{server-name}` with the appropriate server name, for example `myserver.infor.com`. This URL cannot be overridden for individual web parts. This field is validated upon entry.
   
   - **Config Name** - Select the SyteLine configuration to sign into. The drop-down list shows all available configurations on the configuration server you selected.
   
   - **Change User ID/Password** - For individual web parts, select this field if you do not want to use the default login. If this field is not selected, the default User ID and Password are disabled.
   
   - **User ID** - Specify a SyteLine user ID that exists in the specified SyteLine configuration. This user must be assigned a SyteLineAutomation license. In certain cases, it is recommended that this user be a multi-session user such as `SL_Internal`. See “SharePoint Licensing,” below.
   
   - **Password** - Specify the password for that SyteLine user.
ClickOnce Server URL - Specify the URL to the utility server where the ClickOnce client code is installed. In the default value shown in the field, replace {server-name} with the appropriate server name, for example myserver.infor.com.

This URL is required only if you plan to allow drillbacks from the web parts. If a user selects the View in SyteLine menu option on the web part’s edit menu, and the user’s computer does not already have a SyteLine client installed, the menu option uses this URL to install the thin ClickOnce client so the web part can open the SyteLine configuration specified above.

4. Click OK to save the settings. The user ID and password are verified for the selected configuration.

SharePoint Licensing

When you configure the Infor web parts, you must specify a valid SyteLine user ID, password, and configuration for the SyteLine site. This SyteLine user must be granted the SyteLineAutomation license module. All Infor web parts on a site can share a single default set of SyteLine login credentials (user ID, password, and configuration), or you can override the user ID, password, and configuration on specific web parts. For moderate use, a single dedicated user ID is sufficient because connections to SyteLine are created and released as data is queried for the various web parts.

However, if your SharePoint site is heavily used, and if most users will use the default login rather than overriding it with their own user IDs, then you might need a multi-session SyteLine user for the default login, to support higher levels of concurrent use. In that case, specify the SL_Internal user that is provided with each SyteLine license. SL_Internal is a multi-session user. When you grant the SyteLineAutomation license module to the SL_Internal user, it consumes the number of sessions defined for SL_Internal. Use the SyteLine License Management form to see how many sessions are defined for SL_Internal.

Adding the Web Parts to a SharePoint Page

To add web parts to a page in SharePoint:

1. Select Site Actions > Create to create a new page, or select Site Actions > Edit to edit an existing page.

   If you are creating a new page, click Web Part Page and specify the filename and layout template to use.

2. Click Add a Web Part in the area of the page where you want to add the web part.

3. Select one of the Infor web parts from the list. The web part displays on the page with a note that the IDO Name must be specified.

4. Click the edit down-arrow on the web part and select Modify Shared Web Part.

5. Configure the web part settings as described in the following sections.

After you configure a web part, you can export it so that other users can import it with all the configuration settings intact.
Configuring the Web Parts to Display SyteLine Data in SharePoint

To configure a web part, first add it as described above, and select Modify Shared Web Part from the edit menu on the web part.

If you are not sure which IDO to use in a web part, in SyteLine display a form that includes the information you want to include in the web part. In design mode on the form, view the form properties. The Collections tab lists the IDOs being used in the form.

Infor Chart Web Part

Use this web part to display a chart based on IDO data like the one below:

![Sales Year-To-Date Chart](chart_image)

This example displays two series. The legend displays the translated series property names.

In the edit dialog box, fill in these fields to set up the web part:

**Override Config Settings**

Select this check box if you want this web part to use a different SyteLine configuration or login and password than the default that was set at the site. When you select this check box, additional fields display. For information about these fields, see “Configuring the Connection to SyteLine at SharePoint Sites” on page 77.

**IDO Name**

Select the IDO whose properties you want to display.
X-Axis Property
Select the property from the IDO that you want to display on the X-axis (horizontal axis) in the chart. For example, in the chart above, the salesperson name is displayed on the X-axis. Note that some properties may not make sense to display in a chart format.

Series Properties
Select one or more properties to display in the chart. To add properties to the chart, click the >> button. To remove them from the chart, click the << button. Note that series properties are the actual values that are charted and must be numeric.

Filter
If you want to limit the information, for example, to a certain set of customer numbers, specify the filter information here. Any valid SQL where clause syntax is allowed. Here are some examples:
CoNum like ‘ABC%’
QtyOnHand < 20 AND Whse = ‘OH’

Order By
Specify a comma-delimited list of properties that specify how the results should be sorted.

Chart Title
Specify the title that appears over the chart.

X-Axis Title
Specify the title for the X-Axis (horizontal axis).

Y-Axis
Specify the title for the Y-Axis (vertical axis).

Chart Height
Specify the chart height, in pixels, or set a fixed height that fits the space defined for the web part.

Chart Width
Specify the chart width, in pixels, or set a fixed width that fits the space defined for the web part.

Chart Type
Select the chart type. Note that some chart types are inappropriate for some types of data.
Background Color and Secondary Background Color
Use these properties to create a color gradient in the chart.
Other self-explanatory chart options are available.

Infor List Web Part
Use this web part to display a tabular list based on a SyteLine IDO like the one below:

<table>
<thead>
<tr>
<th>Credit Hold</th>
<th>Name</th>
<th>Customer</th>
<th>Ship To</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Control Bicycles, Inc.</td>
<td>5</td>
<td>0</td>
<td>Rockford</td>
</tr>
<tr>
<td>Select</td>
<td>Larry’s Bicycles</td>
<td>6</td>
<td>0</td>
<td>Seattle</td>
</tr>
<tr>
<td>Select</td>
<td>Travel Time Bicycles</td>
<td>7</td>
<td>0</td>
<td>Bloomington</td>
</tr>
<tr>
<td>Select</td>
<td>Test</td>
<td>A000017</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Fill in the fields as described here:

Override Config Settings
Select this check box if you want this web part to use a different SyteLine configuration or login and password than the default that was set at the site. When you select this check box, additional fields display. For information about these fields, see “Configuring the Connection to SyteLine at SharePoint Sites” on page 77.

IDO Name
Select the IDO whose properties you want to display.

IDO Properties
Select the properties from the IDO that you want to include in the listing. To add properties to the list, click the >> button. To remove them from the list, click the << button.

IDO Filter
If you want to limit the information, for example, to a certain set of customer numbers, specify the filter information here. Any valid SQL where clause syntax is allowed. Here are some examples:

CoNum like ‘ABC%’
QtyOnHand < 20 AND Whse = ‘OH’

IDO Order By
Specify a comma-delimited list of properties that specify how the results should be sorted.
Grouping Column

Use this field to group information in the displayed list. For example, if the list contains a State column, you can set the Grouping Column to State and the grid displays data grouped by state.

Drillback Form Name

If you want users to be able to drill down on a row in the list to open a SyteLine form showing the currently selected data, select the form name here (for example, AccountingPeriods). For additional information about drillback, see “Drilling Back to SyteLine Forms” on page 84.

Drillback Filter

If you specified a drillback form name, and you want the form to open filtered on certain data, specify the filter expression here. This filter expression is similar to WinStudio collection filter expressions, except the only valid substitution keyword is FP (filter property). Only properties that are displayed in the web part can be included in the drillback filter.

For example, if your web part displays data from the SLCustomers IDO and includes the CustNum property, you can enable drillback to the Customers form by specifying Customers for the drillback form, and this drillback filter expression:

\[ \text{CustNum} = \text{FP(CustNum)} \]

For additional information about drillback, see “Drilling Back to SyteLine Forms” on page 84.

Appearance, Layout, Advanced

These sections are common to all web parts and are not specific to SyteLine web parts.

Exporting Web Parts

After a web part is configured, you can export the web part so that other users can import it rather than having to set up their own configurations.

To export a configured web part:

1. Select Export on the edit menu of the web part.
2. In the dialog box, browse to the area where you want to store the web part file and specify a filename. The file extension should be .webpart.
3. Click Save.
Importing Web Parts

To import a web part that is already configured:

1. Edit the page where you want to import the part.
2. Select Add Web Part.
3. From the Add Web Part dialog box, click the Advanced Web Part gallery and options link.
4. In the advanced Add Web Parts editor, click the down-arrow next to Browse and select Import.
5. Click Browse… and select the .webpart file to import.
6. Click Upload.
7. Click Import.

NOTE: The SyteLine configuration name, user and password is not exported along with the web part. When you import a web part, this information defaults to the global site-level settings.

Drilling Back to SyteLine Forms

If a web part is configured to allow drillback, then when you select a row by clicking the Select link, you can select View In SyteLine from the edit menu. This option opens a SyteLine client on your computer and displays the form and property data that is linked to that row in the web part. (If you are not currently logged into SyteLine, a login dialog box displays.)

In order for drillback to work, the ClickOnce Server URL must be specified in the default connection information (see “Configuring the Connection to SyteLine at SharePoint Sites” on page 77). A drillback form name must be specified during configuration of the web part. A drillback filter can also be specified during the configuration; then when the user selects the View in SyteLine menu option, the form opens with a filter created using the drillback filter expression and the property values from the selected row. If no filter is specified, the form opens unfiltered.
When Would You Use This?

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

The IDO layer usually handles the session management for connections. Thus, if your program is bypassing the IDO layer, it must 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.

```sql
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 value 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>
Database Settings

The IDO layer sets the following whenever a connection is opened. Your application or process should also make these settings whenever it opens a connection, if your code will access any SyteLine code:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET ANSI_WARNINGS, ANSI_NULLS, ANSI_PADDING, XACT_ABORT, CONCAT_NULL_YIELDS_NULL, ARITHABORT, QUOTED_IDENTIFIER, NOCOUNT ON</td>
<td>ANSI_WARNINGS, ANSI_NULLS, ANSI_PADDING, ARITHABORT, CONCAT_NULL_YIELDS_NULL, and QUOTED_IDENTIFIER are required for computed column indexes to work. NOCOUNT ON prevents every single operation in a trigger or stored procedure from returning a count to the client of how many rows were affected, which would be a performance problem given the large number of operations performed in the database. XACT_ABORT is required for distributed transaction support.</td>
</tr>
<tr>
<td>SET NUMERIC_ROUNDABORT OFF</td>
<td>NUMERIC_ROUNDABORT must be off for computed column indexes.</td>
</tr>
</tbody>
</table>

Your application or process must also set the following whenever a connection is opened:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET TRANSACTION ISOLATION LEVEL [READ COMMITTED</td>
<td>READ UNCOMMITTED]</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>
Index

Symbols

.NET
  CallMethod 70
  CreateSessionToken 66
  Dynamic IDOs 73
  GetConfigurationNames 65
  LoadDataSet 67
  SaveDataSet 68
  Test Utility 73
  Web Service API 63
  Web service example 71

A
  ActiveScriptingOnly tag 28
  AdditionalFailureInformation tag 31
  AllowCloseExistingSessions tag 31
  ANSI_NULLS 86
  ANSI_PADDING 86
  ANSI_WARNINGS 86
  App Metadata Sync utility 19
  App Metadata Transport utility 19
  AppDB class 12
  ApplicationName tag 31
  AppName tag 27
  AppURL tag 27
  ARITHABORT 86
  AuditingEnabled tag 31
  Authorizations 18

B
  BaseHelpFile tag 27
  BinaryDataFormat tag 35
  BoolFalse tag 35
  BoolTrue tag 35

C
  CallMethod, .NET API call 70
  Case tag 35
  ClassNotesExist attribute 34
  Client class library 12
  CloseSession, in XML documents 54
  CloseSessionContextSp 86
  Closing session 86
  ColSecurityPrimaryOnly tag 28
  ColumnType tag 35
  CompanyName tag 27
  CONCAT_NULL_YIELDS_NULL 86
  Config tag 27
  ConfigGroup tag 27
  ConfigName tag 30
  configserver.aspx 26
  Configuration Manager 19
  Configuration Manager utility 27
  Copyright tag 27

CreateSessionToken, .NET API call 66
  Cursor tag 39

D
  Database settings during programmatic database connection 86
  Database, accessing programmatically 85
  Database, used for virtual IDOs 55, 73
  DataType tag 34
  Date format, when passing to IDOs 55
  DateFormat tag 36
  DaysUntilPasswordExpires tag 31
  DeadlockRetry tag 31
  DefaultIMECharset tag 35
  DefaultValue tag 35
  DefFormFont 28
  DeleteExistingPrompt tag 28
  DeleteNewPrompt tag 28
  Direct access to database 85
  DisableInvalidValuePrompt tag 28
  DisplayDecimal Position tag 36
  Distinct tag 40
  DomainIDOName tag 35
  DomainUserName tag 30
  Dynamic IDO
    Definition 14
    XML documents 55

E
  EditLevel tag 31
  Error response for XML UpdateCollection 49
  Extension classes 12

F
  Filter tag 40
  FormCLSID tag 27
  FormFileExtension tag 27
  FormsURL tag 27
  FromXML method 10

G
  GetConfigurationNames, .NET API call 65
  GetConfigurations, in XML documents 22, 26
  GetPropertyInfo tag 34
  GetPropertyInfo, in XML documents 33

H
  HelpBasePath tag 27
  HelpContentIndexMenuText tag 28
  HorzComponentSpacing tag 28
  Html tag 35
  HTTP POST command 22
  HTTPS protocol 22
I
IconFilename tag 27
ID attribute, UpdateCollection 50
Identity tag 35
IDO
.NET class libraries 10
Command interface 11
Definition 7
Extension classes 12
Licensing 19
Passing dates to 55
Runtime Development Server 9
IDOProperties tag 34
IDOProtocol class library 10
IDORequest 8
IDORequest class 10
IDORequestEnvelope 8, 10
IDOWebService example 71
IncludeClassNotesFlag attribute 34
Infor Chart Web Part 75
Infor List Web Part 75
InitiatorName, in XML documents 24
InitiatorType, in XML documents 24
InitSessionContextSp 85
InputMask tag 35
InputMaskPromptChar tag 35
InsertPostedToWorkflow attribute 34
Invoke, in XML documents 52
Item tag 40

J
Justify tag 36

L
Label tag 35
LabelStringID tag 35
LanguageID tag 30
License tag 31
Licensing IDOs and user IDs 19
Licensing, SharePoint 79
LinkBy tag 35, 40, 45
LoadCollection tag 39
LoadCollection, in XML documents 38
LoadDataSet, .NET API call 67
LoadType tag 39
LoginOverrideProgID tag 27
LoginResult tag 31

M
MachineName tag 30
Method, definition 7
MGWebParts.zip file 76
Microsoft Chart Controls 77
Middleware permissions 18
Mongoose.IDO.Client class library 10
MSChart.exe 77

N
NOCOUNT 86
NumDigitsInGroup tag 36
NUMERIC_ROUNDBORT 86

O
Object authorizations for user 18
ObjectMenuName tag 28
OpenSession, in XML documents 30
OrderBy tag 40

P
Password tag 31
Payload 8
Permissions 18
Posting XML documents 22
PostQueryCmd tag 41
PrimaryCacheUsesFormCaption tag 28
PrimaryGroupName tag 31
ProductVersion tag 31
ProgIDRootAugment tag 28
Program access to database 85
Property tag 34
Property, definition 7
PropertyList tag 39
Protocol class library 10
Protocol version elements in XML documents 24

Q
Query Customers example 71
QUOTED_IDENTIFIER 86

R
ReadMode tag 39
ReadOnly tag 35
RecordCap tag 40
RegionalSettings tag 31
ReportViewerExecutable tag 28
RequestHeader elements in XML documents 23
RequestService.ASP 22
RequestType elements in XML documents 24
Required tag 35
ResponseHeader elements in XML documents 23
RightJustifyNewLabels tag 28
RORecord tag 35
RSDAO COM library 13

S
SaveDataSet, .NET API call 68
Secure transmissions 22
ServerDate tag 31
Session
  Closing 86
  Initialization 85
  Token 66
SessionID elements in XML document 24
SharePoint

SyteLine Integrating IDOs with External Applications
Copyright © 2010 Infor
Index | 89

Licensing 79
Web parts 75
SourceName, in XML documents 24
SPI format 14, 55
SplashFilename tag 27
Standard IDO, definition 14
StartupMethods tag 31
Subcollection tag 35

T
TABLE! format 55
TargetName, in XML documents 24
Token for session 66
ToXML method 10
TRANSACTION ISOLATION LEVEL 86
TransactionName tag 24

U
UpdateCollection
  Error response 49
  in XML documents 43
  Item ID example 50
  Nested Insert example 47
  UseKeys example 51
  UseHtmlHelp tag 28
UseKeys attribute, UpdateCollection 51
User middleware permissions 18
UserId tag 30
Users, licensing 19

V
ValidationErrorIndicators tag 28
VBAFilename tag 27
Version tag 27
VertComponentSpacing tag 28

Virtual IDOs 55

W
WallpaperDisplayOption tag 27
WallpaperFilename tag 27
Warning tag 35
web parts 75
Web service example 71
Wire Tap 9
Workstation tag 31

X
XACT_ABORT 86
XML
  CloseSession 54
  Dynamic IDOs 55
  GetConfigurations 22, 26
  Get PropertyInfo 33
  InitiatorName 24
  InitiatorType 24
  Invoke 52
  LoadCollection 38
  OpenSession 30
  Posting documents 22
  ProtocolVersion 24
  Request and response documents 21
  RequestHeader 23
  RequestType 24
  ResponseHeader 23
  SessionID 24
  SourceName 24
  TargetName 24
  UpdateCollection 43
  XPVisualStyle tag 28