



# Infor EAM Programmer's Guide

Web Services Toolkit

---

**Copyright © 2016 Infor**

### **Important Notices**

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

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

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

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

### **Trademark Acknowledgements**

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

### **Publication Information**

Release: Infor EAM Programmer's Guide Web Services Toolkit Release 11.3

Publication date: December 11, 2016

---

# Contents

<b>About this guide</b> .....	<b>7</b>
Related documents .....	7
Contacting Infor.....	7
<b>Chapter 1 Using Web Services with .NET</b> .....	<b>8</b>
Introduction .....	8
Infor consulting.....	8
Infor training.....	8
Requirements.....	9
Adding a service reference to a WSDL file.....	9
<b>Chapter 2 Sample applications</b> .....	<b>11</b>
AddWorkOrder sample.....	11
InitializeWOService routine .....	12
CreateWorkOrder routine .....	14
User interface.....	14
AddAndGetWorkOrder sample.....	15
Adding a work order .....	16
Getting a work order.....	17
User interface.....	18
BrowseWorkOrders sample .....	18
Using the EWSSGridDisplayManager component.....	19
Using the EWSSGridDisplayControl component.....	21
User interface.....	22
AddWorkOrderUsingLookups sample .....	23
Date and DateTime lookups .....	24
Code List lookups.....	24
Add a work order.....	26
User interface.....	28

<b>Chapter 3    Datastream .NET library .....</b>	<b>29</b>
Datastream namespace .....	30
Configuration.....	30
Methods .....	30
NameValuePair structure .....	30
Properties .....	30
NameValuePairList .....	30
Properties .....	30
Methods .....	31
Datastream.EWS namespace .....	31
EWSException .....	31
EWSSoapException.....	31
Grid .....	31
Constructors .....	31
Properties .....	32
Methods .....	32
GridData .....	33
Properties .....	33
GridFilterItem .....	33
Properties .....	33
GridList.....	34
Constructors .....	34
Methods .....	34
GridLOV .....	35
Constructors .....	35
Methods .....	35
GridParameter structure.....	35
Properties .....	35
GridParameterList.....	36
Properties .....	36
Methods .....	36
GridParameterType .....	36
GridToolbar .....	36
Properties .....	36
Methods .....	36
Session .....	37
Properties .....	37
Methods .....	37

---

Events.....	37
Datastream.EWS.Util namespace .....	38
SoapExceptionHandler .....	38
Methods .....	38
SoapExceptionInfo.....	38
SoapMessageHelper .....	38
Methods .....	38
Datastream.Reflection namespace .....	39
FieldAccess.....	39
Methods .....	39
Datastream.WinForms namespace .....	39
EWSFilterInputBox.....	39
Properties .....	40
Methods .....	40
Events.....	40
EWSGridDisplayControl.....	41
Properties .....	41
Methods .....	44
Events.....	44
EWSGridDisplayManager .....	45
Properties .....	45
Methods .....	46
Events.....	47
FilterLine .....	47
Properties .....	47
Methods .....	48
Events.....	48
MultilineFilter .....	48
Properties .....	48
Methods .....	49
Events.....	49
ObjectFieldLinkProvider.....	50
Methods .....	51
Datastream.WinForms.Dialogs namespace .....	51
CalendarDlg.....	51
Properties .....	51
Methods .....	52
LookupDlg.....	52

**Contents**

---

Constructors .....	52
Properties .....	52
Methods .....	53
RecordViewDlg .....	53
Constructors .....	53
Properties .....	53
Methods .....	53
<b>Chapter 4 Java Web Services .....</b>	<b>54</b>
AddWorkOrder Java sample .....	54
Prerequisites for creating the sample.....	54
Generating the proxy classes.....	54
Building the AddWorkOrder project.....	55
Running the AddWorkOrder project .....	55
Notes on the application.....	56
<b>Appendix A Application Config file.....</b>	<b>57</b>
<b>Appendix B Diagrams .....</b>	<b>59</b>
<b>Appendix C Known issues.....</b>	<b>60</b>

---

## About this guide

The Infor EAM Web Services Toolkit includes Web Service Definition Language (WSDL) files that define the web services exposed by the Connector component of Infor EAM. This document describes how to use the WSDL files from the Microsoft .NET Framework.

## Related documents

You can find the documents in the product documentation section of the Infor Xtreme Support portal, as described in "Contacting Infor" on page 7.

## Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at [www.infor.com/inforxtreme](http://www.infor.com/inforxtreme).

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact [documentation@infor.com](mailto:documentation@infor.com).

# Chapter 1 Using Web Services with .NET

# 1

## Introduction

The Infor EAM Web Services Toolkit includes Web Service Definition Language (WSDL) files that define the web services exposed by the Connector component of Infor EAM. This document describes how to use the WSDL files from the Microsoft .NET Framework.

## Infor consulting

Infor's Professional Services Group has helped thousands of companies with asset management implementation. Our Web Services consultants can help you set up your asset management software to utilize the features available through web services and prepare your employees to operate the system efficiently. As part of Infor's commitment to customer satisfaction, our services are focused on helping companies leverage their assets so that they can become more profitable. For more information about Infor's consulting services, contact your Infor sales representative.

## Infor training

Infor offers a variety of intensive training classes consisting of in-depth product demonstrations, instructor lectures, and hands on exercises. Infor training classes provide an opportunity to learn the advanced techniques and shortcuts to get the most out of your asset management software. Web Services training is technical in nature and recommended for persons with development experience in XML and object-oriented programming. Descriptions and schedules of Infor's full range of classes are available on the Web at [www.infor.com](http://www.infor.com). Infor also offers our customers on-site instruction to help with specific training needs.



# Requirements

The following items are required to use the Web Services Toolkit in .NET:

- Infor EAM Web Services Toolkit, which includes the WSDL files and the data schema (.xsd) files
- Microsoft .NET Framework v4.5
- Microsoft Visual Studio 2015 or other .NET-compatible development environment.

**Note:** Steps described in this document may be different if using an IDE other than Visual Studio 2015 or a different version of the .NET Framework.

## Adding a service reference to a WSDL file

In order to use a web service, the .NET project must add a service reference to the appropriate WSDL file. This section illustrates how to add a service reference in Visual Studio.

**Note:** If it is required to transmit cookies to the server, it may be necessary to add a Web Reference instead of a Service Reference. Transmitting cookies is essential when the EAM server is hosted by Infor, or when the server contains more than one JVM.

To add a service reference to a WSDL file:

- 1 Create a new project or open an existing project in Visual Studio.
- 2 Choose Project | Add Service Reference. The system displays the Add Service Reference form.
- 3 **Address**—Enter the full name of the WSDL file, including the path. This entry can either be a URL or a path to a file.  
**Note:** Since WSDL files reference the Schema files using relative paths, both folders must reside in a common parent folder, i.e., if the WSDL files reside in c:\WSToolKit\wsdl, then the schemas must reside in c:\WSToolkit\schemas.
- 4 Click **Go**, and then wait while Visual Studio reads the WSDL file. Once it has finished reading the file, the system displays a list of the available methods on the Add Service Reference form.
- 5 **Namespace**—Enter a namespace. This name will be the namespace of the web service, so each web service added should have a unique name.  
**Note:** In order to match the sample applications and the documentation for the samples, Infor recommends that the namespace start with "WebServices" and end with the function number of the WSDL being added. The function number includes the four digits preceded by "MP" in the name of the WSDL. For example, if adding MP0023\_AddWorkOrder\_001.wsdl, the namespace would be WebServices.MP0023.
- 6 Click **OK**. Visual Studio creates the appropriate code and adds references to any required schema (.xsd) files.
- 7 The project should now show a "Service References" section in the Visual Studio Solutions Explorer. It will also add a project reference to System.Web.Services namespace if it is not

already there. The generated code will be in the Reference.vb file. To view the new objects added to the project, open the “Class View” tab.

## Chapter 2 Sample applications

# 2

### AddWorkOrder sample

The AddWorkOrder sample application demonstrates how to import a WSDL file that describes an Infor web service, how to call an Infor web service, and how to use the results of that call. The web service that is used in the sample is AddWorkOrder (Function #MP0023). This web service is described in the MP0023\_AddWorkOrder\_001.wsdl file.

After creating a project in Visual Studio, add a web reference to the desired WSDL files. See "Adding a Web Reference to a WSDL File" in Chapter 1 Using Web Services with .NET for an explanation of adding the WSDL file to a .NET project.

Once the WSDL file has been added to the project, you will have access to various classes in the "WebServices.MP0023" namespace. The first class of interest is "AddWorkOrderService". This class contains properties used to construct the SOAP header, as well as the methods needed to actually call the service. "MP0023\_AddWorkOrder\_001" and "MP0023\_AddWorkOrder\_001\_Result" are also very important. The former is used to provide the data required by the web service, while the latter is where the web service will return information to the caller.

The sample's primary form (AddWorkOrderForm) contains input fields to gather all the information needed to add a work order.

After entering the information, the user would click Submit to trigger the call to the web service. The "Submit" button's click event handler calls the "CallAddWorkOrderService" routine, which will set up and call the web service. The routine performs the following steps:

- Calls the "AreAllRequiredFieldsFilled" routine to validate that all fields have been entered.
- If empty required fields are found, the code attempts to set focus to the offending control.
- Prompts for the login details.
- Defines variables needed to call the web service.
- A reference to the actual web service class:

```
Dim addWOService As WebServices.MP0023.AddWorkOrderService
```

An instance class that will hold the information that must be passed to the web service call:

```
Dim serviceData As New WebServices.MP0023.MP0023_AddWorkOrder_001
```

A reference to the class that will hold the result of the call to the web service:

```
Dim result As WebServices.MP0023.MP0023_AddWorkOrder_001_Result
```

- Calls the "InitializeWOService" routine and assigns the result to the "addWOService" variable. This routine is broken down later in this chapter.

- The "serviceData" object has a property "WorkOrder" that will contain the information that was gathered on the form. This property is set by a call to the "CreateWorkOrder" routine. This routine is broken down later in this chapter.
- Calls the "AddWorkOrderOp" method of the web service instance. This is the routine that was generated from the WSDL file and is the actual call to the web service.  
Always make the call to a web service in a Try/Catch block, since any problems will be raised by the web service as a "SoapException".
- Assigns the result to the "result" variable. The work order number is returned in the "result.ResultData.JOBNUM" property.  
result = addWOService.AddWorkOrderOp(serviceData)

Before accessing the result variable and its properties, it must be tested to ensure that it is not set to "Nothing" (null). This may occur if the web service call returns an invalid or corrupt response. Also note that the web service call may still have been successful in this case.

## InitializeWOService routine

This routine creates an instance of the web service object (AddWorkOrderService) and sets all of the properties that are required by the web service to create the SOAP header. The routine performs the following steps:

- Creates an instance of the web service object and assigns it to a variable "ws".  
Dim ws As New WebServices.MP0023.AddWorkOrderService
- Attempts to read the URL for the web service from the application config file. See Appendix A for details. If a URL is defined in the config file, it will be used to override the default by setting the "Url" property of "ws" to the new value. If not overridden, the web service classes created from the WSDL file will point to the default URL of the Infor web services, usually localhost.  
' Read the URL for the web services from the app config file  
Dim url As String = AppSettings.Get(WEBSERVICES\_URL\_SETTINGS\_KEY)  
  
If Not url Is Nothing AndAlso url.Length > 0  
Then url = url.Trim()  
' May throw System.UriFormatException if url is invalid  
If url.Length > 0 Then ws.Url = url  
Else  
' if the URL is not defined in the app config file,  
' the web service will attempt to use the default  
' URL defined by the WSDL file  
End if
- Sets the "OrganizationValue" property of "ws". The type of this property is "Organization", which is a class that was created from the WSDL file. Since it is a class, it must first be instantiated,  
ws.OrganizationValue = New WebServices.MP0023.Organization

and then its properties must be set. The "Organization" class has one property named "Text" that takes an array of organization strings.

```
ws.OrganizationValue.Text = New String() {org}
```

- Creates a "UserNameToken" xml element that holds the username and password as elements and stores in the "token" variable.

```
Dim token As XmlElement = New
XmlDocument().CreateElement("UserNameToken
") Dim node As XmlNode

token.SetAttribute("Id", "MyId")

node = token.OwnerDocument.CreateElement("Username")

node.InnerText =
username
token.AppendChild(node)

node = token.OwnerDocument.CreateElement("Password")

node.InnerText =
password
token.AppendChild(node)
```

The element is defined in the secext.xsd schema file and looks like the following:

```
<UserNameToken Id="MyID" xmlns="">
  <Username>user</Username>
  <Password>password</Password>
</UserNameToken>
```

If we had been working with a previously established session, we would have created a "Session" element instead. See "Session Handling" in the *Web Services Toolkit User's Guide* for more information.

- Assigns the created "UserNameToken" element to the "SecurityValue" property of "ws". The type of this property is "Security", which is a class that was created from the WSDL file. Since it is a class, it must first be instantiated,

```
ws.SecurityValue = New WebServices.MP0023.Security
```

and then its properties must be set. The "SecurityValue" class has one property named "Text" that takes an array of strings.

```
ws.SecurityValue.Any = New XmlElement() {token}
```

- Sets the "SessionScenarioValue" property of "ws". The type of this property is "SessionScenario", which is a class that was created from the WSDL file. Since it is a class, it must first be instantiated,

```
ws.SessionScenarioValue = New WebServices.MP0023.SessionScenario
```

and then its properties must be set. The "SessionScenario" class has one property named "Text" that takes an array of strings.

```
ws.SessionScenarioValue.Text = New String() {"terminate"}
```

The "terminate" text tells the web service not to establish a session to be maintained between web service calls. See "Session Handling" in the *Web Services Toolkit User's Guide* for more information.

- Returns the created instance.

## CreateWorkOrder routine

This routine creates an instance of the "WorkOrder" class created from the WSDL file and sets its properties from the information entered into the sample's main form. The routine performs the following steps:

- Creates an instance of the "WorkOrder" class and assigns it to a variable "wo".

```
Dim wo As New WebServices.MP0023.WorkOrder
```

- Sets the "WORKORDERID" property of "wo". The type of this property is "WOID\_Type" which is a class that was created from the WSDL file. Since it is a class, it must first be instantiated,

```
wo.WORKORDERID = New WebServices.MP0023.WOID_Type
```

and then its properties must be set. The properties are set directly from the controls on the main form of the sample. Note that the "WOID\_Type" class has a property named "ORGANIZATIONID", which is an object reference to a class created from the WSDL file. As has been done with similar properties, it must first be instantiated before setting its own properties.

```
With wo.WORKORDERID
    .JOBNUM = "99999"
    .DESCRIPTION = WODescTextBox.Text.Trim()
    .ORGANIZATIONID = New WebServices.MP0023.ORGANIZATIONID_Type
    .ORGANIZATIONID.ORGANIZATIONCODE =
    WOOrgTextBox.Text.Trim() End With
```


The "STATUS", "TYPE", "DEPARTMENTID", "EQUIPMENTID", and "CREATEDBY" properties are object references and must be instantiated before being set. The "FIXED" property is required and needs to be hard-coded to the value "V".

- Returns the "WorkOrder" instance.

## User interface

The user interface of the application is composed of three forms. The main form "AddWorkOrderForm" provides for the entry of the fields required to create a work order. The "LoginForm" is used to gather the User ID and Password, and the "AboutForm", accessed by pressing F1, displays the application title and the copyright information.

The AddWorkOrder web service is triggered by clicking **Submit** on the input form. Of course, the button first displays the "LoginForm" to prompt for the login details. Also note that the only validation performed on the input is for required fields.

Some conventions that are used include a Blue-Gray background for all required fields and use of an "ErrorProvider" component on each form that has any input fields. This component is what displays the red exclamation mark  to the right of the input field when it is not valid (required in this case).

Any of the input fields, including User ID but not Password, can have a default value. This value is set in the application config file. The configuration file is fully commented as to what fields it supports. These values will appear when the application is started and when the "Reset" button is pressed.

## AddAndGetWorkOrder sample

The AddAndGetWorkOrder sample demonstrates how to use some of the components provided by the Datastream .NET Library. The demonstrated components include the Datastream.WinForms.ObjectFieldLinkProvider and the Datastream.EWS.Session. See "ObjectFieldLinkProvider" and "Session" descriptions, respectively, in Chapter 3 Datastream .NET Library.

The sample builds upon the AddWorkOrder sample by replacing how some steps were performed in that sample with how the steps can be accomplished using the library components. The sample also demonstrates how to use another Infor web service. This is the GetWorkOrder (Function #MP0024) and is described in the MP0024\_GetWorkOrder\_001.wsdl file.

After creating a project in Visual Studio, add a web reference to the desired WSDL files. See "Adding a Web Reference to a WSDL File" in Chapter 1 Using Web Services with .NET for an explanation of adding the WSDL file to a .NET project.

Once the WSDL files have been added to the project, you will have access to various classes in the "WebServices.MP0023" and "WebServices.MP0024" namespaces. See the previous sample for information on the "WebServices.MP0023" namespace. For the "WebServices.MP0024" namespace, the first class of interest is "GetWorkOrderService", which contains properties used to construct the SOAP header, as well as the methods needed to actually call the service.

"MP0024\_GetWorkOrder\_001" and "MP0024\_AddWorkOrder\_001\_Result" are also very important. The former is used to provide the data that the web service will use while the latter is where the web service will return information to the caller.

Since the sample uses components from the Datastream .NET Library, a reference to the library file must be added to the project. This reference will point to the Datastream.EWS.dll file which is the Datastream .NET Library. For details on how to do this, please search the Visual Studio help for "Adding and Removing References."

Some of the components in the library are designed to be used in the Visual Studio Forms Designer. These components should be added to the Visual Studio Toolbox. For details on how to do this, please search the Visual Studio help for "Managing Tabs and Items in the Toolbox" and "Customize Toolbox Dialog Box."

The sample's primary form (AddAndGetWorkOrderForm) contains input fields to gather and display information about a work order.

It also contains an instance of the `Datastream.WinForms.ObjectFieldLinkProvider` named "ObjFieldLinkProvider". See "ObjectFieldLinkProvider" in Chapter 3 Datastream .NET Library. This component was dragged from the Toolbox and shows up in the tray area at the bottom of the Form Designer. It allows mapping the value stored in a control with an object property.

The name of the object property to map to is set in the "ObjectFieldName" property of the control. For example, the "WOOrgTextBox" has its "ObjectFieldName" property set to "WORKORDERID.ORGANIZATIONID.ORGANIZATIONCODE". The mappings can be used by the "CopyValuesFromControls" and "CopyValuesToControls" methods of "ObjFieldLinkProvider" to move data between an object and the controls.

The form allows two primary actions: adding a work order and retrieving an existing work order. These actions are triggered by the "Submit" button and the "Retrieve" button, respectively. The following sections will describe these actions in more detail.

## Adding a work order

The "Submit" button's click event handler calls the "CallAddWorkOrderService" routine, which will set up and call the "AddWorkOrderService" web service. The routine behaves basically the same as the previous sample with the following differences:

- Instead of writing a routine like "InitializeWOService" (See "AddWorkOrder Sample") to set all of the properties required by the web service to create the SOAP message header, we use the "PrepareServiceRequest" method of the global Session object.

```
Globals.Session.PrepareServiceRequest(addWOService)
```

```
result = addWOService.AddWorkOrderOp(serviceData)
```

"PrepareServiceRequest" takes an instance of the web service class and sets all of the appropriate properties of the instance needed by the web service call.

The sample uses a global instance of the `Datastream.EWS.Session` class. This instance is accessed through the "Session" property of the "Globals" module that can be found in the `Globals.vb` file.

- Instead of writing a routine like "CreateWorkOrder" (See "AddWorkOrder Sample") to set the "WorkOrder" property of the "serviceData" object, we use the "CopyValuesFromControls" method of "ObjFieldLinkProvider" to copy the values from the controls to the "serviceData.WorkOrder" object.

```
serviceData.WorkOrder = New WebServices.MP0023.WorkOrder
serviceData.WorkOrder.FIXED = "V" ' Required and hard coded
value
```

```
Me.ObjFieldLinkProvider.CopyValuesFromControls(serviceData.WorkOrder
)
```

The "Fixed" property is set directly in code because there is no control that maps its contents to this property.

- The result of the call to the web service is passed to the "CompleteServiceRequest" method of the Session object. This method extracts the session ID returned by the web service call and stores it in the "SessionId" property of the Session object for later use.



## Getting a work order

The "Retrieve" button's click event handler calls the "CallGetWorkOrderService" routine, which will set up and call the "GetWorkOrderService" web service. The routine performs the following steps:

- Uses the "WorkOrderPromptForm" to prompt for the Work Order Number and Organization to get.
- Prompts for the login details.
- Defines variables needed to call the web service.

A reference to the actual web service class:

```
Dim getWOService As WebServices.MP0024.AddWorkOrderService
```

An instance class that will hold the information that must be passed to the web service call:

```
Dim serviceData As New WebServices.MP0024.MP0024_GetWorkOrder_001
```

A reference to the class that will hold the result of the call to the web service:

```
Dim result As WebServices.MP0024.MP0024_AddWorkOrder_001_Result
```

- The "serviceData" object has a property "WORKORDERID" that will contain the work order number and organization gathered on dialog.

```
serviceData.WORKORDERID = New
WebServices.MP0024.WOID_Type
serviceData.WORKORDERID.JOBNUM = workOrderNumber
serviceData.WORKORDERID.ORGANIZATIONID = New
```

```
WebServices.MP0024.ORGANIZATIONID_Type
```

```
serviceData.WORKORDERID.ORGANIZATIONID.ORGANIZATIONCODE =
organization
```

Note that, as in the previous sample, any property that has a reference type must be instantiated before its properties can be set.

- Calls the "PrepareServiceRequest" method of the global Session object. This method takes an instance of the web service class and sets all of the appropriate properties of the instance needed by the web service call.

```
Globals.Session.PrepareServiceRequest(getWOService)
```

"Globals.Session" is a global instance of the Datastream.EWS.Session class. It is accessed through the "Session" property of the "Globals" module that can be found in the Globals.vb file.

- Calls the "GetWorkOrderOp" method of the web service and assigns the result to the "result" variable. This is the routine that was generated from the WSDL file and is the actual call to the web service.

```
result = getWOService.GetWorkOrderOp(serviceData)
```

Always make the call to a web service in a Try/Catch block, since any problems will be raised by the web service as a "SoapException".

- The result of the call to the web service is passed to the "CompleteServiceRequest" method of the Session object. This method extracts the session ID returned by the web service call and stores it in the "SessionId" property of the Session object for later use.

```
Globals.Session.CompleteServiceRequest(getWOService)
```

- If the result is valid, the form controls are populated with the work order data from the result property "ResultData.WorkOrder" by the call to "ObjFieldLinkProvider.CopyValuesToControls". See "ObjectFieldLinkProvider" in Chapter 3 Datastream .NET Library.

```
Me.ObjFieldLinkProvider.CopyValuesToControls(result.ResultData.WorkOrder)
```

## User interface

The user interface of the application is composed of four forms. The main form "AddAndGetWorkOrderForm" is almost identical to the one in the prior sample, and like the main form from the prior sample it provides for the entry of the fields required to create a work order. It also displays the work order data when a work order has been retrieved. The "LoginForm" is used to gather the User ID and Password. The "WorkOrderPromptForm" is used to gather the Work Order Number and Organization for retrieval, and the "AboutForm", accessed by pressing F1, displays the application title and the copyright information.

The "Retrieve" button triggers the GetWorkOrder web service, while prompting for any necessary data. Like in the prior sample, the AddWorkOrder web service is triggered by the "Submit" button.

See "AddWorkOrder Sample" for conventions that are used in the user interface.

## BrowseWorkOrders sample

The BrowseWorkOrders sample application demonstrates how to use components from the Datastream .NET Library to access "Grid" data. The two primary components that are demonstrated are Datastream.WinForms.EWSGridDisplayManager and Datastream.WinForms.EWSGridDisplayControl. See "EWSGridDisplayManager" and "EWSGridDisplayControl" in Chapter 3 Datastream .NET Library for detailed descriptions of the components.

Unlike the previous sample applications, adding a Web Reference to the appropriate WSDL files is not necessary. The components already include the appropriate references and expose their functionality through .NET components. These components consume the GetGridHeaderData, GetGridDataOnlyCache, and GetGridDataOnly web services (Function #MP0018, #MP0017, and MP0016 respectively).

Since the sample uses components from the Datastream .NET Library, a reference to the library file must be added to the project. This reference will point to the Datastream.EWS.dll file, which is the Datastream .NET Library. For details on how to do this, please search the Visual Studio help for "Adding and Removing References."

The components demonstrated here are designed to be used in the Visual Studio Forms Designer. These components should be added to the Visual Studio Toolbox. For details on how to do this,

please search the Visual Studio help for "Managing Tabs and Items in the Toolbox" and "Customize Toolbox Dialog Box."

The sample's primary form (MainForm) presents buttons to select which demonstration to view. For clarity, each component is demonstrated on a separate Windows form that is triggered from a button on the primary form. The "Browse Using Standard Controls with Display Manager" button opens the BrowseWithManagerForm form and demonstrates the EWSGridDisplayManager component.

The "Browse Using the Display Control" button opens the BrowseWithControlForm form and demonstrates the EWSGridDisplayControl component.

The following sections will describe how to use each component in more detail.

## Using the EWSGridDisplayManager component

The BrowseWithManagerForm form demonstrates the EWSGridDisplayManager component. This component handles retrieving the "Grid" data and uses existing controls to display and manage the data. It does not present any visual controls of its own.

To use this component, you should first add the controls that will display the data. These controls will be assigned to the appropriate properties of the component so that it may use them to display and manage the data. The following table lists the EWSGridDisplayManager component properties that can be assigned to existing form controls:

Property	Description
DataspysComboBox	List of available Dataspy's
DataspysApplyButton	Apply the selected Dataspy
FilterFieldsComboBox	List of available filterable fields
OperatorComboBox	List of filter comparison operators
FilterValueInputBox*	A text box to enter the value to compare against the selected filter field.
LookupButton	Bring up the lookup dialog to select the value that will be entered in the above TextBox
FilterApplyButton	Apply the specified filter to the data.
GridControl	DataGrid control to display the actual "Grid" data that was returned.

\*This sample uses the EWSFilterInputBox control in place of a standard text box. See "EWSFilterInputBox" in Chapter 3 Datastream .NET Library. Since this control has a built-in lookup button, the "LookupButton" property is not set to anything.

The following steps outline how to use the component on a Windows form.

- Add all the controls that will display the "Grid" data.

- Drag the EWSSGridDisplayManager from the Visual Studio Toolbox onto the form. The component will show up in the tray area at the bottom of the Form Designer.
- Click on the EWSSGridDisplayManager component to select it and have its properties shown in the Property Editor.
- Assign the appropriate controls to the properties of the component shown in the table above.
- Set the required GridName and FunctionName properties for the desired "Grid" data.
- The form's "Activated" event handler calls the "InitialLoad" routine to load the "Grid" data.

```
Static dataLoaded As
    Boolean
Static dataLoading As Boolean

If Not dataLoaded Then
    If Not dataLoading
        Then dataLoading
            = True
        Me.Refresh()           ' Force form to draw fully
        Me.InitialLoad()
        dataLoaded = True     ' Only do this once
        dataLoading = False
    End If
End If
```

The handler uses static variables to keep the routine from being called multiple times.

- The "InitialLoad" routine first attempts to fill in any missing properties by reading the values from the application config file.
- To load the data, the Session property must be set to an instance of the Datastream.EWS.Session class. The sample uses a global instance that is stored in Globals.Session and was created by the "MainForm". The Session property can either be set directly or the instance can be passed as an argument to the LoadData method. The call to LoadData will attempt to fetch the data into an instance of Datastream.EWS.GridData, exposed through the GridData property, and will populate the associated controls with the appropriate data.

```
GridDisplayManager.Session = Globals.Session
GridDisplayManager.LoadData()
```

Always make the call to LoadData in a Try/Catch block, since any problems encountered by the component will be raised as exceptions.

The EWSSGridDisplayManager component automatically handles pertinent events from the controls that have been assigned to it. An example of this is if any of the "Apply" buttons are clicked, they will attempt to gather the appropriate information and reload the data with that new information. Another example is if more records remain cached and the DataGrid is scrolled to the bottom, it will attempt to load more "Grid" data.

## Using the EWSSGridDisplayControl component

The BrowseWithControlForm form demonstrates the EWSSGridDisplayControl component. In contrast to the EWSSGridDisplayManager component, this component contains all the controls needed to display and navigate the grid data. This removes the need to add multiple controls to the form and then associate them with the component. However, in order to provide flexibility on how the constituent controls will appear, many more properties are exposed by this control.

The following steps outline how to use the component on a Windows form.

- Drag the EWSSGridDisplayControl from the Visual Studio Toolbox onto the form. Since this is a visual component, it will appear directly on the form. Resize the control to the desired dimensions.
- Click on the EWSSGridDisplayControl component to select it and have its properties shown in the Property Editor.
- Set the required GridName and FunctionName properties for the desired "Grid" data.
- The form's "Activated" event handler calls the "InitialLoad" routine to load the "Grid" data.

```

Static dataLoaded As Boolean
Static dataLoading As Boolean

If Not dataLoaded Then
    If Not dataLoading
        Then dataLoading
            = True
        Me.Refresh()           ' Force form to draw fully
        Me.InitialLoad()
        dataLoaded = True     ' Only do this
                              once dataLoading = False
    End If
End If

```

The handler uses static variables to keep the routine from being called multiple times.

- The "InitialLoad" routine first attempts to fill in any missing properties by reading the values from the application config file.
- To load the data, the Session property must be set to an instance of the Datastream.EWS.Session class. The sample uses a global instance that is stored in Globals.Session and was created by the "MainForm". The Session property can either be set directly or the instance can be passed as an argument to the LoadData method. The call to LoadData will attempt to fetch the data into an instance of Datastream.EWS.GridData, exposed through the GridData property, and will populate the associated controls with the appropriate data.

```

GridDisplayControl.Session = Globals.Session
GridDisplayControl.LoadData()

```

Always make the call to LoadData in a Try/Catch block, since any problems encountered by the component will be raised as exceptions.

The EWSSGridDisplayControl component automatically handles the pertinent events for its constituent controls. An example of this is if any of the "Apply" buttons are clicked, they will attempt to gather the appropriate information and reload the data with that new information. Another example is if more records remain cached and the DataGrid is scrolled to the bottom, it will attempt to load more "Grid" data.

## User interface

The user interface of the application is composed of five forms. The first form that is displayed is the "MainForm". It allows the selection of the method used to display the Grid data of work orders. The "BrowseWithManagerForm" and the "BrowseWithControlForm" use different Datastream .NET Library components to provide virtually identical information. The "LoginForm" is used to gather the User ID and Password, and the "AboutForm", accessed by pressing F1, displays the application title and the copyright information.

The "BrowseWithManagerForm" uses the EWSSGridDisplayManager component to manage various controls on the form to display the data from the work orders grid (WSJOBS).

To use this component, drag it from the Visual Studio Toolbox to a form in the designer. The component will appear in the tray at the bottom of the designer. After selecting the component in the tray, the properties will appear in the property editor. The GridName and FunctionName properties are required to get the data. See "Using the EWSSGridDisplayManager Component" earlier in this chapter for details on how the component is used in the sample.

EWSSGridDisplayManager contains many additional properties for determining the appearance and behavior of the controls. See "EWSSGridDisplayManager" in Chapter 3 Datastream .NET Library for a list of all the properties, methods, and events that are available.

The "BrowseWithControlForm" uses the EWSSGridDisplayControl to display the data from the work orders grid (WSJOBS).

To use this component, drag it from the Visual Studio Toolbox to a form in the designer. Like other standard controls, it will appear on the form and can be sized and moved as desired. After selecting the control on the form, the properties will appear in the property editor. The GridName and FunctionName properties are required to get the data. See "Using the EWSSGridDisplayControl Component" earlier in this chapter for details on how the component is used in the sample.

EWSSGridDisplayControl contains many additional properties for determining the appearance and behavior of the controls. See "EWSSGridDisplayControl" in Chapter 3 Datastream .NET Library for a list of all the properties, methods, and events that are available.

The "Record View" tab, which is found on both the "BrowseWithManagerForm" and the "BrowseWithControlForm", displays the details of the currently selected work order. The next and previous buttons, found above the tabs, will display the next and previous work order details, respectively.

The "Record View" tab only displays the Work Order details that are included in the grid results. Another web service call would be required if additional information is desired.


## AddWorkOrderUsingLookups sample

The AddWorkOrderUsingLookups sample application demonstrates how to use components from the Datastream .NET Library to present dialogs with lookup information, such as codes and dates, during data entry. The two primary components that are demonstrated are Datastream.WinForms.Dialogs.LookupDlg and Datastream.WinForms.Dialogs.CalendarDlg. See "LookupDlg" and "CalendarDlg" in Chapter 3 Datastream .NET Library.

Unlike some of the previous sample applications, adding a Web Reference to the appropriate WSDL files is not necessary. The components already include the appropriate references and expose their functionality through .NET components.

Since the sample uses components from the Datastream .NET Library, a reference to the library file must be added to the project. This reference will point to the Datastream.EWS.dll file which is the Datastream .NET Library. For details on how to do this, please search the Visual Studio help for "Adding and Removing References."

The sample's primary form (AddWorkOrderForm) contains input fields to gather some information in order to add a work order.

All of the input fields provide lookup capability, with the exception of the work order description. For example, if you click on the lookup icon (  ), the system will display a dialog that will assist in entering appropriate data for the corresponding field.

The input fields with the associated lookup icon are actually instances of the Datastream.WinForms.EWSFilterInputBox control from the Datastream .NET Library. This control provides the visual representation we were looking for by simply setting its Style property to "EWSFilterInputBoxStyle.TextBoxWithLookup". See Chapter 3 Datastream .NET Library for a detailed description of this control.

The lookup dialogs are displayed in response to the "LookupButtonClick" event for the EWSFilterInputBox controls. This event is triggered when the lookup icon is clicked. The handlers for these events have all been consolidated into one method, "LookupButtonClickHandler".

The "LookupButtonClickHandler" uses the "Tag" property of the control to determine the type of lookup to provide. The following lookup types are available:

- A calendar to select a date. This is shown by both the "Sched. Start Date" and "Sched. End Date" fields. A control is marked to use this style by placing "date" in its Tag property.
- A calendar with hour and minutes fields to select a date and enter a time. This is shown by the "Date Reported" field. A control is marked to use this style by placing "datetime" in its Tag property.
- A list of codes from which to select. This is shown by the "WO Organization", "Department Code", "Equipment", "Status Code", "Location", "Type Code", "Cost Code", "Class", "Reported By" and "Assigned To" fields. A control is marked to use this style by placing a lookup description string in its Tag property.
- The description string starts with the name of the lookup grid. This is followed by a semi-colon and a comma-separated list of parameter names and values, if any are required by the lookup.
- Finally the "gridtagname=" identifies the value of the field that will be returned in the Selection property of the dialog.

- Below is an example of the string used for the "Class" field:  
LVCLAS;parameter.rentity=EVNT,gridtagname=class
- The following sections will elaborate further on how to call the dialogs that provide each style of lookup functionality as well as what happens when the "Submit" button is clicked.

## Date and DateTime lookups

When the "LookupButtonClickHandler" finds "date" or "datetime" in the Tag property of the control that triggered the event, it attempts to display an instance of the `Datastream.WinForms.Dialogs.CalendarDlg`.

The following steps outline the procedure for a date and datetime lookup.

- Create a `CalendarDlg` variable, named `dlg`, to hold the new instance of `CalendarDlg`.
- The `IncludeTime` property will be set to `false` if the Tag property was "date" and set to `true` if the property was "datetime". Note that the value has been copied into the `LOVDescription` variable.

```
Dim dlg As New Datastream.WinForms.Dialogs.CalendarDlg
```

```
dlg.IncludeTime = (LOVDescription.ToLower() = "datetime")
```

A call to the `PositionAsDropDown` method of the dialog is made in order to have the dialog act as if it were a dropdown. When the dialog is displayed, it will appear below the control that was specified in the call.

```
dlg.PositionAsDropDown(inputBox)
```

- The current value held by the control, if any, is passed to the dialog.
- The dialog is displayed, and if the user did not "Cancel" the dialog the selection is assigned to the control.

```
If inputBox.TextLength > 0 Then dlg.Selection = inputBox.Text
```

```
If dlg.ShowDialog(Me) = DialogResult.OK  
Then inputBox.Text = dlg.Selection
```

```
End If
```

## Code List lookups

When the "LookupButtonClickHandler" finds "date" or "datetime" in the Tag property of the control that triggered the event, it attempts to display an instance of the `Datastream.WinForms.Dialogs.LookupDlg`.

The following steps outline the procedure for a code list lookup.

- Create a `LookupDlg` variable, named `dlg`, to hold the new instance of `LookupDlg`.

```
Dim dlg As New Datastream.WinForms.Dialogs.LookupDlg( _  
    Globals.Session, "WSJOBS",  
    LOVDescription)
```



The LookupDlg constructor requires a Session instance, a function name ("WSJOBS"), and a lookup description string. The function names, like WSJOBS or SMPART will vary depending on the lookup being used. See the *Infor EAM User's Guide* to find the appropriate function name to use for the desired lookup. The lookup description string is described earlier in the list of available lookup types.

The sample uses a global instance of the Datastream.EWS.Session class. This instance is accessed through the "Session" property of the "Globals" module that can be found in the Globals.vb file.

- The dialog is displayed, and if the user did not "Cancel" the dialog the selection is assigned to the control.

```
    If dlg.ShowDialog(Me) = DialogResult.OK Then inputBox.Text =
        dlg.Selection
```

- The rest of the code determines which control made the lookup request and determines if additional values from the selected row need to be stored. All the available values for the selected row are returned in the SelectionList property of the dialog. This property is an instance of the NameValuePairList class. The additional values are stored in variables to be used once the "Submit" button is clicked to create the new work order.

The first case, in the following code, is notable in that in addition to storing off the equipment organization, it also retrieves the description of the selected equipment and assigns it to a separate control – the EquipmentDescTextBox control.

```
With dlg.SelectionList
    Select Case True
        Case inputBox Is Me.EquipmentNoInputBox
            If .Contains("equiporganization", True) Then
                m_EquipmentOrg
                =.Item("equiporganization",True).Value
            End If
            If.Contains("equipmentdesc", True) Then
                Me.EquipmentDescTextBox.Text
                =.Item("equipmentdesc", True).Value
            End If
        Case inputBox Is Me.ClassInputBox
            If.Contains("classorganization", True) Then
                m_ClassOrg =.Item("classorganization", True).Value
            End If
        Case inputBox Is Me.LocationInputBox
            If.Contains("locorganization", True) Then
                m_LocationOrg =.Item("locorganization",
                True).Value
            End If
        Case inputBox Is Me.CostCodeInputBox
```

```
                If.Contains("costorganization", True) Then
                    m_CostCodeOrg = .Item("costorganization", True) .
                        Value
                End If
            End Select
        End With
```

## Add a work order

The "Submit" button's click event handler calls the "CallAddWorkOrderService" routine, which will set up and call the "AddWorkOrderService" web service. The routine behaves basically the same as the previous AddAndGetWorkOrder sample with the following differences:

- Like in the AddAndGetWorkOrder sample, we call the "CopyValuesFromControls" method of the "ObjFieldLinkProvider" to copy the values from the controls to the "serviceData.WorkOrder" object.
- Next, we need to set the values that do not have a corresponding control. These are the organizations that were stored in the "LookupButtonClickHandler".

```
    ' Set the values that do not map to a control
    Datastream.Reflection.FieldAccess.SetValue(serviceData.WorkOrder,
        _ "EQUIPMENTID.ORGANIZATIONID.ORGANIZATIONCODE",
        m_EquipmentOrg)
    If (ClassInputBox.TextLength > 0) Then

        Datastream.Reflection.FieldAccess.SetValue(serviceData.WorkOrder,
            _ "CLASSID.ORGANIZATIONID.ORGANIZATIONCODE",
            m_ClassOrg)
        End
        If

        If (LocationInputBox.TextLength > 0) Then

            Datastream.Reflection.FieldAccess.SetValue(serviceData.WorkOrder,
                _ "LOCATIONID.ORGANIZATIONID.ORGANIZATIONCODE",
                m_LocationOrg)
            End If
            If (CostCodeInputBox.TextLength > 0) Then

                Datastream.Reflection.FieldAccess.SetValue(serviceData.WorkOrder,
                    _ "COSTCODEID.ORGANIZATIONID.ORGANIZATIONCODE",
                    m_CostCodeOrg)
                End If
```

- We are using the "DataStream.Reflection.FieldAccess.SetValue" function which automatically instantiates objects before assigning the value. This allows us to replace the following code:

```

If serviceData.WorkOrder.EQUIPMENTID Is Nothing
    Then serviceData.WorkOrder.EQUIPMENTID = New
    WebServices.MP0023.EQUIPMENTID_Type
End
If
If serviceData.WorkOrder.EQUIPMENTID.ORGANIZATIONID Is Nothing
    Then serviceData.WorkOrder.EQUIPMENTID.ORGANIZATIONID = _
        New WebServices.MP0023.ORGANIZATIONID_Type
End If
serviceData.WorkOrder.EQUIPMENTID.ORGANIZATIONID.ORGANIZATIONCODE
=
m_EquipmentOrg

```

with this:

```

DataStream.Reflection.FieldAccess.SetValue(serviceData.WorkOrder,
    _ "EQUIPMENTID.ORGANIZATIONID.ORGANIZATIONCODE",
    m_EquipmentOrg)

```

- We also need to set the date/time values, which require special processing. The data entered in the control is first validated to make sure it is actually a date/time.

```

' Set the date fields
Dim dt As System.DateTime
If (DateReportedInputBox.TextLength > 0) Then
    Try
        dt =
        DateTime.Parse(DateReportedInputBox.Text) Catch
        ex As Exception
            MessageBox.Show("Date Reported does not contain a valid
date/time.", _
                "Validation Error",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
            DateReportedInputBox.Focus
            () Exit Sub
    End Try
    serviceData.WorkOrder.REPORTED = CreateWSDateTime(dt,
True) End If


```

- Finally the "Reported" property of the "serviceData.WorkOrder" is set to the result of passing the validated date/time to CreateWSDateTime. This function splits out the values that make up a date/time, such as Month, Day, and Year, and assigns them to the appropriate properties used by the web service to define a data/time.

- The result of the call to the web service is passed to the "CompleteServiceRequest" method of the Session object. This method extracts the session ID returned by the web service call and stores it in the "SessionId" property of the Session object for later use.

## User interface

The user interface of the application is composed of three forms. The main form "AddWorkOrderForm" provides for the entry of the fields required to create a work order, as well as a few optional fields. The "LoginForm" is used to gather the User ID and Password and "AboutForm", accessed by pressing the F1 key, displays the application title and the copyright information.

When the AddWorkOrderForm displays, all of the fields are read-only until an organization is selected. The organization is selected by clicking on the lookup icon (  ) and selecting the desired organization from the list presented in the lookup dialog that opens.

Once the organization is selected, all the read-only fields will be switched to allow editing. Enter information into all of the required fields and any desired optional fields. Any field that is followed by a lookup icon can be filled in by clicking on the icon and selecting a value from the dialog that opens.

The "Submit" button triggers the AddWorkOrder web service.

See "AddWorkOrder Sample" for conventions that are used in the user interface.

This chapter describes the .NET Framework library provided by with the Infor EAM Web Services Toolkit.

In order to use this library, a reference to the assembly "Datastream.EWS.dll" must be added from a .NET project. Additionally, some of the components in the library are designed to work with the Visual Studio Form Designer and should be added to the Visual Studio Toolbox. Once added, they can be dragged and dropped onto a Windows Form and have their properties set through the Property Editor.

The library components are designed to assist using the Infor EAM Web Services. Some classes will be more useful than others depending on the usage scenario. Below are some common scenarios.

- 1 When importing one or more WSDLs, you will mainly be working with the classes that are generated by the import process. Aside from the Session class used in all scenarios, some of the utility classes, such as SoapExceptionHandler to help parse soap exception messages and ObjectFieldLinkProvider to map object fields and controls, may be useful.
- 2 When retrieving data from grid web services, the lowest level class is the Grid class. It is responsible for retrieving data using the grid web services. The data is returned in an instance of the GridData class. The GridData in turn contains an instance of GridToolbar and various pieces of information in instances of NameValuePairList.
- 3 When retrieving data from grid web services to be displayed using standard controls, the EWSSGridDisplayManager component should be used. This component provides the most flexibility for control placement and appearance. It handles retrieving the data and displaying it in controls that have been associated with it. The component uses the Grid class internally and so it is useful to have familiarity with the same classes mentioned in scenario 2.
- 4 When retrieving data from grid web services to be displayed WITHOUT using standard controls, the EWSSGridDisplayControl should be used. This component is the fastest method to set up due to the controls being provided. However, it is limited when it comes to control placement and appearance. It handles retrieving the data and displaying it in controls that are a built in to EWSSGridDisplayControl. The control uses the Grid class internally and so it is useful to have familiarity with the same classes mentioned in scenario 2.

All scenarios should use the Session class, which is required by all web service calls to set security, session, and web service URL details.

## Datastream namespace

This namespace provides various utility classes and modules that may be used by the other Datastream namespaces.

### Configuration

This class provides an alternative to using the `System.Configuration.ConfigurationSettings.AppSettings` collection. Instead of raising an exception for an undefined key, it will, depending on which overload is called, return a user-specified default value or an empty string.

### Methods

`ReadAppSetting`—Read the configured value for the specified key. Four overloads are provided to allow specifying the default value to return if the key is undefined as well as choosing whether or not to ignore the case of the key when searching for its value.

### NameValuePair structure

This structure defines the properties for a name-and-value pair relation.

### Properties

- `Name`—The identifying name of the associated value.
- `Value`—The value of associated with the specified name.

### NameValuePairList

This class is an implementation of an `ArrayList` of `NameValuePair` objects. It is used to store a list of keyed values.

### Properties

- `Item`—Retrieves a stored `NameValuePair` through the specified index.

## Methods

- Supports all methods of an `ArrayList` with added overloads to support the `NameValuePair` type.

## Datastream.EWS namespace

This namespace provides various web service utility classes and modules.

### EWSException

This class is an implementation of an `ApplicationException`. It is thrown by a `Datastream .NET Library` method, which encounters a non-fatal application error.

### EWSSoapException

This class is an implementation of a `SoapException`. It is thrown when an XML Web service method is called over SOAP by a `Datastream .NET Library` method and an exception occurs.

## Grid

This abstract (`MustInherit`) class provides the base properties and methods used to call the grid web services. It should be inherited for each specific grid request type.

## Constructors

Since this class is abstract (`MustInherit`), the constructors may only be called by inherited classes.

- `New(session, gridName, functionName, gridType)`—Sets the properties required to make a grid request.
- `New(session, gridName, functionName, gridType, parameters)`—Adds the ability to set the parameters to be used by the grid request.
- `New(session, gridName, functionName, gridType, rowsPerPage)`—Adds the ability to set the `RowsPerPage` property to something other than the default.
- `New(session, gridName, functionName, gridType, rowsPerPage, parameters)`—Adds the ability to set the `RowsPerPage` property to something other than the default and to set the parameters to be used by the grid request.

- `New(session, gridName, functionName, gridType, rowsPerPage, localizeResults)`—Adds the ability to set the `RowsPerPage` and `LocalizeResults` properties to something other than the default.
- `New(session, gridName, functionName, gridType, rowsPerPage, localizeResults, parameters)`—Adds the ability to set the `RowsPerPage` and `LocalizeResults` properties to something other than the default and to set the parameters to be used by the grid request.

## Properties

- `FunctionName`—Gets the name of the function that will be called.
- `GridName`—Gets the name of the grid that will be called.
- `GridType`—Gets the type of grid that will be requested.
- `LocalizeResults`—Determines if the data returned by grid is localized.
- `Parameters`—Gets or sets the `GridParameterList` of parameters used by the grid request.
- `RowsPerPage`—Determines the number of rows that are returned from the cache for each call. Defaults to `DEFAULT_ROWS_PER_PAGE` and must stay between `MIN_ROWS_PER_PAGE` and `MAX_ROWS_PER_PAGE`.
- `Session`—Gets the `Datastream.EWS.Session` object that will be used make the grid requests.
- `WebServiceTimeout`—Gets or sets the time, in milliseconds, to wait for a synchronous web service to complete.

## Methods

All the methods return a `GridData` object with the results of the web service call.

- `FetchAllRecords`—Performs a `FetchDataWithHeader` and then loops while `HasMoreRows` until all of the records have been retrieved.
- `FetchData(gridData, filterField, filterOp, filterValue)`—Calls MP0116 grid web service with the supplied filter.
- `FetchData(gridData, filterItems())`—Calls MP0116 grid web service with the supplied filter list. This is an abstract (`MustOverride`) method.
- `FetchDataWithHeader`—Calls the MP0118 grid web service.
- `FetchDataWithHeader(dataspyId)`—Calls MP0118 grid web service with the specified `DataspY`.
- `FetchDataWithHeader(dataspyId, filterItems())`—Calls MP0118 grid web service with the specified `DataspY` and filter list.
- `FetchDataWithHeader(filterItems())`—Calls MP0118 grid web service with the specified filter list.
- `FetchDataWithHeader(filterField, filterOp, filterValue)`—Calls MP0118 grid web service with the specified filter.
- `FetchMoreData(gridData)`—Calls MP0117 grid web service for more cached data. This is an abstract (`MustOverride`) method.
- `GetTotalRowCount()`—Calls MP0116 grid web service to get the total potential rows for the `Session` defined grid.



- `GetTotalRowCount(dataspyId)`—Calls MP0116 grid web service to get the total potential rows for the Session defined grid with the specified Dataspy.
- `GetTotalRowCount(gridData)`—Calls MP0116 grid web service to get the total potential rows for the Session defined grid with the dataspy and filter list specified in the specified gridData.
- `GetTotalRowCount(dataspyId, filterItems)`—Calls MP0116 grid web service to get the total potential rows for the Session defined grid with the specified Dataspy and Filter Items.

## GridData

This class is the repository for the results of a call to a grid web service.

It maintains the resulting data and metadata as well as the details on what information was used to make the grid web service request.

## Properties

- `Dataspy`—Get the details of the active Dataspy when the data was fetched
- `MetaDataList`—Get the list of the metadata for the grid web service result
- `Fields`—Get the list of metadata for the fields included in the grid web service result
- `Data`—Get the DataTable containing the records from the grid web service result
- `FilterItems`—Get the active filter details when the data was fetched
- `Toolbar`—Get the active toolbar details when the data was fetched
- `Parameters`—Get the list of parameters used by the grid request.
- `RequestInfo`—Get a GridRequestInfo structure that defines the fields used to make a call to the grid web service.
- `CursorPosition`—Get the index into the cached data for the data that has been retrieved thus far.
- `Records`—The number of records that have been cached. May or may not be the total number of records.
- `HasMoreRows`—Indicates whether more data is available in the cache.
- `GridLabel`—Get the text that describes the grid.

## GridFilterItem

This structure is used to store one line of a filter definition.

## Properties

- `SequenceNumber`—Controls the order of the filter lines if more than one is defined.
- `LeftParen`—Determines whether a left parenthesis "(" is defined.

- **Field**—The field on which to filter. This is actually the field name followed by a colon ":" and the datatype.
- **Operator**—The operator used to compare the Field and Value.
- **Value**—The value that is compared to the field.
- **RightParen**—Determines whether a right parenthesis ")" is defined.
- **Joiner**—A "GridFilterItemJoinCondition" enumeration that determines if this filter line is joined to the next using an "And" or an "Or".
- **FieldName**—The name of the field extracted from the Field property.
- **DataType**—The datatype of the field extracted from the Field property.

## GridList

This is a specialization of the Grid class for grid requests of type LIST.

## Constructors

- **New(session, gridName, functionName)**—Sets the properties required to make a grid request.
- **New(session, gridName, functionName, parameters)**—Add the ability to set the parameters to be used by the grid request.
- **New(session, gridName, functionName, rowsPerPage)**—Adds the ability to set the RowsPerPage property to something other than the default.
- **New(session, gridName, functionName, rowsPerPage, parameters)**—Adds the ability to set the RowsPerPage property to something other than the default and to set the parameters to be used by the grid request.
- **New(session, gridName, functionName, rowsPerPage, localizeResults)**—Adds the ability to set the RowsPerPage and LocalizeResults properties to something other than the default.
- **New(session, gridName, functionName, gridType, rowsPerPage, localizeResults, parameters)**—Adds the ability to set the RowsPerPage and LocalizeResults properties to something other than the default and to set the parameters to be used by the grid request.

## Methods

- **FetchData(gridData)**—Calls MP0116 grid web service.
- **FetchData(gridData, filterItems())**—Calls MP0116 grid web service with the supplied filter list. Overrides the base abstract (MustOverride) method.
- **FetchMoreData(gridData)**—Calls MP0117 grid web service for more cached data. Overrides the base abstract (MustOverride) method.

## GridLOV

This is a specialization of the Grid class for grid requests of type LOV.

### Constructors

- `New(session, gridName, functionName)`—Sets the properties required to make a grid request.
- `New(session, gridName, functionName, parameters)`—Add the ability to set the parameters to be used by the grid request.
- `New(session, gridName, functionName, rowsPerPage)`—Adds the ability to set the `RowsPerPage` property to something other than the default.
- `New(session, gridName, functionName, rowsPerPage, parameters)`—Adds the ability to set the `RowsPerPage` property to something other than the default and to set the parameters to be used by the grid request.
- `New(session, gridName, functionName, rowsPerPage, localizeResults)`—Adds the ability to set the `RowsPerPage` and `LocalizeResults` properties to something other than the default.
- `New(session, gridName, functionName, gridType, rowsPerPage, localizeResults, parameters)`—Adds the ability to set the `RowsPerPage` and `LocalizeResults` properties to something other than the default and to set the parameters to be used by the grid request.

### Methods

- `FetchData(gridData)`—Calls MP0116 grid web service.
- `FetchData(gridData, filterItems())`—Calls MP0116 grid web service with the supplied filter list. Overrides the base abstract (`MustOverride`) method.
- `FetchMoreData(gridData)`—Calls MP0117 grid web service for more cached data. Overrides the base abstract (`MustOverride`) method.

## GridParameter structure

This structure defines the properties for a grid parameter.

### Properties

- `Name`—The identifying name of the associated value.
- `Value`—The value associated with the specified name.
- `Type`—A `GridParameterType` enumeration that defines the data type of the specified value.

## GridParameterList

This class is an implementation of an `ArrayList` of `NameValuePair` objects. It is used to store a list of keyed values.

### Properties

- `Item`—Retrieves a stored `GridParameter` through the specified index

### Methods

- Supports all methods of an `ArrayList` with added overloads to support the `GridParameter` structure.

## GridParameterType

An enumeration of available data types for a grid parameter value. The data types defined by this enumeration coincide with the enumeration values defined by the `FIELD_TYPE` element in the `GridRequest` schema (`GridRequest.xsd`).

## GridToolbar

This class maintains the toolbar metadata returned by call to the MP0118 grid web service.

### Properties

- `DataspYDisplay`—Gets or sets the currently selected `DataspY` on the toolbar.
- `DataspYList`—Gets or sets a `NameValuePairList` of available `DataspY`s.
- `FilterDisplay`—Gets or sets the currently selected filter on the toolbar.
- `FilterList`—Gets or sets a `NameValuePairList` of available filters.
- `Operators`—Gets a `NameValuePairList` of all defined operators for use in filtering.

### Methods

- `GetFilterOperators(datatype)`—Return a `NameValuePairList` of filter operators based upon the specified data type.

## Session

Stores properties used by all Infor web service calls. The properties can then be set on any instance of a web service class.

Of particular interest is the `SessionId`. This value is created when a web service is called with the `EstablishSession` property set to "True". Future calls to `Session.PrepareServiceRequest` will use this `SessionId` in place of the `Username` and `Password`.

## Properties

- `EstablishSession`—Determine whether or not a session should be created when the web service is called.
- `Organization`—Organization to use during the web service call.
- `Password`—Password of the user to log in during the web service call.
- `SessionId`—The ID of a session that was created and returned by a web service call with the `EstablishSession` set to `True`.
- `Tenant`—Database tenant to use during the web service call.
- `Url`—Location of the web services.
- `Username`—Name/ID of the user to log in during the web service call.

## Methods

- `PrepareServiceRequest`—Sets all the appropriate properties of the specified service object in preparation for the web service being executed.
- `CompleteServiceRequest`—Performs any cleanup on the specified service object. Currently stores the `Session ID` if one was created.
- `TerminateSession`—Closes the session on the server. This method should be called if previous web services requests have been made with 'EstablishSession' set to true.
- `ProcessServiceRequest`—Provides a mechanism to automatically retry a web service request when the initial request fails. By default this method will retry a failed request only when the failure is due to a session timeout. The caller can modify this behavior and control whether the failed request continues to be retried by registering a listener for the `TryAgain` event (see below).

## Events

- `TryAgain`—The purpose of this event is to let the caller control whether the `ProcessServiceRequest` method continues to retry a failed web service request. To use this event, register a listener. When a request fails, the listener will be passed a `SessionFailureEventArgs` parameter, which contains the session, information about the exception that occurred, and a `Cancel` property. If the listener wishes to retry the request, it can set `Cancel` to `False` (by default `Cancel` is set to `True`).

## Datastream.EWS.Util namespace

This namespace contains various utility classes and modules to help when working with the web services.

### SoapExceptionHelper

The "SoapExceptionHelper" class provides methods to parse and format the information contained in a SoapException thrown from a web service call. The class takes an instance of a SoapException in its constructor.

#### Methods

- **GetExceptionInfoList**—Parses all the exceptions into an ArrayList of "SoapExceptionInfo" objects.
- **ToString**—Parses exception information from the SoapException.Detail and constructs a display-ready formatted string.

### SoapExceptionInfo

"SoapExceptionInfo" is a class used to store the details (Type, ReasonCode, and Message) of an Infor web service exception parsed from a SoapException.

### SoapMessageHelper

The "SoapMessageHelper" module provides helper functions to populate commonly used properties of a web service object.

#### Methods

- **CreateUsernameTokenSecurityElement**—Given a username and password, creates an XElement for the "UsernameToken" used in the "Security" property of a web service.

## Datastream.Reflection namespace

This namespace contains various utility classes and modules to help when reflecting information from .NET assemblies, classes, types, etc.

### FieldAccess

This class allows access to fields. The namespace for this class is Datastream.Reflection. To use this class without full namespace qualification, add a call to `Import Datastream.Reflection` in your code.

All the methods in this class are Shared/Static, so the class does not need to be instantiated.

### Methods

- **GetValue**—Given an instantiated object and a fieldname, this method will return the value stored by that field as a string.

Example: Assume you have an instance of `WorkOrderData` named `WO`. In order to read the value of the status code, you would call `FieldAccess.GetValue(WO, "Codes.Status")`, which will return the value or `Nothing` if the value has not been defined.

- **SetValue**—Given an instantiated object, a fieldname, and a value, this method will attempt to set the field to the value given.

Example: Assume you have an instance of `WorkOrderData` named `WO`. In order to set the value of the equipment organization, you would call `FieldAccess.SetValue(WO, "Equipment.Org", "ORG1")`, which will set the value to `"ORG1"`.

- **GetFieldWithForcedInstantiation**





See "Diagram 1—Classes Used in Code Examples" in Appendix B for the classes used in the examples above.

## Datastream.WinForms namespace

This namespace contains components that deal with the user interface for Windows-based (non-browser) applications.

### EWSFilterInputBox

This user control provides an entry area for the values of filterable fields. It currently provides three styles of entry that are defined in the `EWSFilterInputBoxStyle` enumeration:

Style of Entry	Description	
EWSFilterInputBoxStyle.Checkbox	Useful for boolean fields	
EWSFilterInputBoxStyle.TextBox	Standard string entry	
EWSFilterInputBoxStyle.TextBoxWith Lookup	Useful for lookups	
EWSFilterInputBoxStyle.TextBoxWith CaseSensitiveLookup	Converts all keypresses to upper case	

## Properties

- **CheckBoxCheckAlign**—Determines the location of the check box inside the control.
- **CheckBoxText**—Gets or sets the text contained by the check box when displayed.
- **CheckBoxTextAlign**—The alignment of the text that will be displayed by the check box.
- **Checked**—Indicates whether the check box is checked or unchecked.
- **DataType**—This property controls the allowed keypresses during data entry based on the type that is set. Types of "number", "currency" and "decimal" restrict entry to only numeric characters, the decimal point, and a sign character.
- **DefaultChecked**—Determines the checked state of the check box when displayed.
- **DefaultText**—Determines the text contained by the text box when displayed.
- **FalseValue**—Gets or sets the value returned in Text when the style is CheckBox and it is NOT checked.
- **LookupButtonImage**—The image displayed in the button.
- **ResetOnStyleSwitch**—Revert value to default whenever the Style changes.
- **Style**—Controls the appearance of the input control.
- **Text**—The text contained in the control.
- **TriggerLookupFromKeyboard**—Allow Alt-Down to emulate clicking on the lookup button.
- **TrueValue**—Gets or sets the value returned in Text when the style is CheckBox and it is checked. Also includes most properties available for a standard TextBox control.

## Methods

- Includes most methods available for a standard TextBox control.

## Events

- **InputStyleChanged**—Triggered when the Style property is changed.
- **LookupButtonClick**—Triggered when the graphical lookup button is clicked.



---

## EWSSGridDisplayControl

This is a user control composed of multiple constituent controls that can be used to display various data items returned from a grid web service. This class internally uses the Grid class to retrieve the data from the grid web services.

### Properties

- **AllowCellSelection**—Determines whether the individual cells in the DataGrid get focus.
- **AutoApplyDataspYOnChanged**—Determines whether the Dataspy should be loaded as soon as it is selected in the ComboBox.
- **AutoLoadMoreData**—Determines whether more data, if available, is loaded automatically once the DataGrid has been scrolled to the bottom.
- **AutosizeComboDropDownWidth**—Determines whether or not to size the dropdown portion of the ComboBoxes to display the widest item.
- **BottomPanel**—Gets the Panel control that displays at the bottom.
- **BottomPanelBackColor**—Gets or sets the background color of the panel.
- **BottomPanelBorderStyle**—Gets or sets the type of border for the panel.
- **BottomPanelHeight**—Gets or sets the height of the panel.
- **BottomPanelVisible**—Determines whether or not the corresponding control is visible.
- **DataspYAlignment**—Gets or sets where to display the Dataspy controls.
- **DataspYApplyButton**—Gets the Button control that will apply the selected Dataspy.
- **DataspYApplyButtonText**—Gets or sets the Text to display by the corresponding control.
- **DataspYApplyButtonToolTip**—Gets or sets the ToolTip for the corresponding control.
- **DataspYApplyButtonVisible**—Determines whether or not the corresponding control is visible.
- **DataspYApplyButtonWidth**—Gets or sets the Width property of the corresponding control.
- **DataspYComboBox**—Gets the ComboBox control that will display the available Daspys.
- **DataspYComboBoxToolTip**—Gets or sets the ToolTip for the corresponding control.
- **DataspYComboBoxVisible**—Determines whether or not the corresponding control is visible.
- **DataspYComboBoxWidth**—Gets or sets the Width property of the corresponding control.
- **DataspYLabel**—Gets the label that will describe the DataspyComboBox.
- **DataspYLabelText**—Gets or sets the Text to display by the corresponding control.
- **DataspYLabelVisible**—Determines whether or not the corresponding control is visible.
- **FilterAlignment**—Gets or sets where to display the Filter controls.
- **FilterApplyButton**—Gets the Button control that will apply the specified filter.
- **FilterApplyButtonText**—Gets or sets the Text to display by the corresponding control.
- **FilterApplyButtonToolTip**—Gets or sets the ToolTip for the corresponding control.
- **FilterApplyButtonVisible**—Determines whether or not the corresponding control is visible.
- **FilterApplyButtonWidth**—Gets or sets the Width property of the corresponding control.

- **FilterComboBox**—Gets the ComboBox control that will display the available fields on which to filter.
- **FilterComboBoxToolTip**—Gets or sets the ToolTip for the corresponding control.
- **FilterComboBoxVisible**—Determines whether or not the corresponding control is visible
- **FilterComboBoxWidth**—Gets or sets the Width property of the corresponding control.
- **FilterLabel**—Gets the label that will describe the FilterComboBox.
- **FilterLabelText**—Gets or sets the Text to display by the corresponding control.
- **FilterLabelVisible**—Determines whether or not the corresponding control is visible
- **FilterLookupDateFormat**—Gets or sets the format for the Date returned from a lookup.
- **FilterLookupDateTimeFormat**—Gets or sets the format for the DateTime returned from a lookup.
- **FilterLookupFalseValue**—Gets or sets the value returned in Text when the style is CheckBox and it is NOT checked.
- **FilterLookupTrueValue**—Gets or sets the value returned in Text when the style is CheckBox and it is checked.
- **FilterValueInputBox**—Gets the EWSFilterInputBox control that will contain the value with which to filter.
- **FilterValueInputBoxToolTip**—Gets or sets the ToolTip for the corresponding control.
- **FilterValueInputBoxVisible**—Determines whether or not the corresponding control is visible
- **FilterValueInputBoxWidth**—Gets or sets the Width property of the corresponding control.
- **FilterValueLabel**—Gets the label that will describe the FilterValueInputBox.
- **FilterValueLabelText**—Gets or sets the Text to display by the corresponding control.
- **FilterValueLabelVisible**—Determines whether or not the corresponding control is visible.
- **FunctionName**—Gets or sets the name of the Function to call for the specified GridName.
- **GridAllowDelete**—Gets or sets whether deletes are allowed.
- **GridAllowNew**—Gets or sets whether new rows can be added.
- **GridCaptionBackColor**—Gets or sets the background color of the grid control's caption bar.
- **GridCaptionForeColor**—Gets or sets the foreground color of the grid control's caption bar.
- **GridCaptionVisible**—Shows or hides the grid caption area.
- **GridColumnValue(colIndex)**—Gets or sets the value in the index specified column from the current row.
- **GridColumnValue(colName)**—Gets or sets the value in the name specified column from the current row.
- **GridColumnValue(colIndex, version)**—Gets the value in the index specified column from the specified DataRowVersion of the current row.
- **GridColumnValue(colName, version)**—Gets the value in the name specified column from the specified DataRowVersion of the current row.
- **GridContextMenu**—Gets or sets the ContextMenu for the grid control.
- **GridControl**—Gets the DataGrid control that will display the Grid results.
- **GridControlBackColor**—Sets the background color of the grid.
- **GridControlPadding**—Gets or sets how much space to allow around the DataGrid control.

- 
- GridControlSelectionBackColor—Gets or sets the background color of any selected cells or rows.
  - GridControlSelectionForeColor—Gets or sets the color of the text in any selected cells or rows.
  - GridControlToolTip—Gets or sets the ToolTip for the corresponding control.
  - GridControlVisible—Determines whether or not the corresponding control is visible.
  - GridData—Gets the object that contains the data loaded by the component.
  - GridName—Gets or sets the name of the Grid web service to call.
  - GridRowCount—Gets the number of rows of data.
  - GridType—Gets or sets the type of grid request to make.
  - HandleTriggeredException—Determines how to respond to exceptions thrown by events triggered by associated controls.
  - InitialDataspYId—Gets or sets the ID number of the initial Dataspy to use for the current GridName.
  - LocalizeResults—Determines whether or not the returned data is localized.
  - MiddlePanel—Gets the Panel control that displays in the middle.
  - Parameters—Gets or sets the GridParameterList of parameters used by a grid request.
  - OperatorComboBox—Gets the ComboBox control that will display the operators associated with each filter field.
  - OperatorComboBoxToolTip—Gets or sets the ToolTip for the corresponding control.
  - OperatorComboBoxVisible—Determines whether or not the corresponding control is visible
  - OperatorComboBoxWidth—Gets or sets the Width property of the corresponding control.
  - OperatorLabel—Gets the label that will describe the OperatorComboBox.
  - OperatorLabelText—Gets or sets the Text to display by the corresponding control.
  - OperatorLabelVisible—Determines whether or not the corresponding control is visible
  - RowsPerPage—Gets or sets the number of rows to return per request.
  - SelectRowOnCellClick—Determines whether or not the entire row is selected when clicking on any cell.
  - Session—Gets or sets the instance of the Session class used to make the grid requests.
  - ToggleButtonsEnabledState—Determines whether the Enabled property of buttons should be toggled based on the currently selected values.
  - ToggleButtonsVisibleState—Determines whether the Visible property of buttons should be toggled based on the currently selected values.
  - TopPanel—Gets the Panel control that displays at the top.
  - TopPanelBackColor—Gets or sets the background color of the panel.
  - TopPanelBorderStyle—Gets or sets the type of border for the panel.
  - TopPanelHeight—Gets or sets the height of the panel.
  - TopPanelVisible—Determines whether or not the corresponding control is visible.
  - UseEwsTableStyle—Determines whether or not to generate and use the standard EWS table style.
  - WebServiceTimeout—Gets or sets the time, in milliseconds, to wait for a synchronous web service to complete.

## Methods

- **ApplyDatspy**—Attempt to apply the Datspy that is selected in the constituent DatspyCombo control.
- **ApplyFilter**—Attempt to apply the filter defined by the constituent FilterFieldsComboBox, OperatorComboBox, and FilterValueInputBox controls.
- **GetActiveDataRow**—Returns a DataRow object for the row index in the data source that matches the current row index of the grid control.
- **GetTotalRowCount**—Returns a string that contains the count of the total potential rows for the grid.
- **LoadData**—Attempt to load the data defined with the GridName, GridType, and FunctionName properties.
- **LoadData(session)**—Using the specified session object, attempt to load the data for the defined with the GridName, GridType, and FunctionName properties.
- **LoadMore**—Attempt to load more data from the cache if available.
- **ViewCell**—Open a dialog with the contents of the current cell. Will be editable if the cell is.

## Events

- **DataGridDoubleClick**—This event is deprecated. It will be removed in a future version of the library. Please use GridDoubleClick event instead.
- **Exception**—Triggered if the HandleTriggeredException flag property is set to HandleTriggeredExceptionMethod.RaiseExceptionEvent and an exception is thrown by an event that was in turn triggered by an associated control.
- **FilterInputStyleChanged**—Triggered when the Style property of the constituent EWSFilterInputBox control is changed.
- **FilterLookupButtonClicked**—Triggered when the lookup button on the filter value entry is clicked.
- **GridAddingColumn**—Triggered for each column that is added to the constituent DataGrid control.
- **GridClick**—Triggered by the X event of the constituent DataGrid control.
- **GridCurrentCellChanged**—Triggered by the CurrentCellChanged event of the constituent DataGrid control.
- **GridDoubleClick**—Triggered by the DoubleClick event of the constituent DataGrid control.
- **GridDragDrop**—Triggered by the DragDrop event of the constituent DataGrid control.
- **GridDragEnter**—Triggered by the DragEnter event of the constituent DataGrid control.
- **GridDragLeave**—Triggered by the DragLeave event of the constituent DataGrid control.
- **GridDragOver**—Triggered by the DragOver event of the constituent DataGrid control.
- **GridEnter**—Triggered by the Enter event of the constituent DataGrid control.
- **GridGotFocus**—Triggered by the GotFocus event of the constituent DataGrid control.
- **GridKeyDown**—Triggered by the KeyDown event of the constituent DataGrid control.
- **GridKeyPress**—Triggered by the KeyPress event of the constituent DataGrid control.
- **GridKeyUp**—Triggered by the KeyUp event of the constituent DataGrid control.

- GridLeave—Triggered by the Leave event of the constituent DataGrid control.
- GridLostFocus—Triggered by the LostFocus event of the constituent DataGrid control.
- GridMouseDown—Triggered by the MouseDown event of the constituent DataGrid control.
- GridMouseEnter—Triggered by the MouseEnter event of the constituent DataGrid control.
- GridMouseHover—Triggered by the MouseHover event of the constituent DataGrid control.
- GridMouseLeave—Triggered by the MouseLeave event of the constituent DataGrid control.
- GridMouseMove—Triggered by the MouseMove event of the constituent DataGrid control.
- GridMouseUp—Triggered by the MouseUp event of the constituent DataGrid control.
- GridMouseWheel—Triggered by the MouseWheel event of the constituent DataGrid control.
- GridResize—Triggered by the Resize event of the constituent DataGrid control.
- GridValidated—Triggered by the Validated event of the constituent DataGrid control.
- GridValidating—Triggered by the Validating event of the constituent DataGrid control.
- Loaded—Triggered whenever the data is done loading.
- Loading—Triggered just before the attempt to load the data is initiated. May cancel the load by setting the event argument e.Cancel to True.

## EWSSGridDisplayManager

This is a non-visual component that is designed to associate controls on a form to display various data items returned from a grid web service. The data items that may be displayed includes Dataspy, Filtering, and DataGrid. This class internally uses the Grid class to retrieve the data from the grid web services.

### Properties

- AllowCellSelection—Determines whether the individual cells in the DataGrid get focus.
- AutoApplyDatsapyOnChanged—Determines whether the Datsapy should be loaded as soon as it is selected in the ComboBox.
- AutoLoadMoreData—Determines whether more data, if available, is loaded automatically once the DataGrid has been scrolled to the bottom.
- AutosizeComboDropDownWidth—Determines whether or not the dropdown portion is sized to display the widest item.
- DatsapyApplyButton—Gets or sets the Button control that will apply the selected Datsapy.
- DatsapyComboBox—Gets or sets the ComboBox control that will display the available Datsapys.
- DateFormat—Gets or sets the format for the Date returned from a lookup.
- DateTimeFormat—Gets or sets the format for the DateTime returned from a lookup.
- FilterApplyButton—Gets or sets the Button control that will apply the specified filter.
- FilterFieldsComboBox—Gets or sets the ComboBox control that will display the available fields to filter on.

- **FilterValueInputBox**—Gets or sets the TextBox control that will contain the value with which to filter.
- **FunctionName**—Gets or sets the name of the Function to call for the specified GridName.
- **GridControl**—Gets or sets the DataGrid control that will display the Grid results.
- **GridControlSelectionBackColor**—Gets or sets the background color of any selected cells or rows.
- **GridControlSelectionForeColor**—Get or sets the color of the text in any selected cells or rows.
- **GridData**—Gets the object that contains the data loaded by the component
- **GridName**—Gets or sets the name of the Grid web service to call.
- **GridType**—Gets or sets the type of grid request to make.
- **HandleTriggeredException**—Determines how to respond to exceptions thrown by events triggered by associated controls.
- **InitialDatsapyId**—Gets or sets the Id number of the initial Datsapy to use for the current GridName.
- **LoadMoreButton**—Gets or sets the Button control that will attempt to load more data from cache.
- **LocalizeResults**—Determines whether or not the returned data is localized.
- **LookupButton**—Gets or sets the Button control that will provide the appropriate lookup for the selected filter.
- **Parameters**—Gets or sets the GridParameterList of parameters used by a grid request.
- **OperatorComboBox**—Gets or sets the ComboBox control that will display the operators associated with each filter field.
- **RowsPerPage**—Gets or sets the number of rows to return per request.
- **SelectRowOnCellClick**—Determines whether or not the entire row is selected when clicking on any cell.
- **Session**—Gets or sets the instance of the Session class used to make the grid requests.
- **ToggleButtonsEnabledState**—Determines whether the Enabled property of buttons should be toggled based on the currently selected values.
- **ToggleButtonsVisibleState**—Determines whether the Visible property of buttons should be toggled based on the currently selected values.
- **UseEwsTableStyle**—Determines whether or not to generate and use the standard EWS table style.
- **WebServiceTimeout**—Gets or sets the time, in milliseconds, to wait for a synchronous web service to complete.

## Methods

- **ApplyDatsapy**—Attempt to apply the Datsapy that is selected in the associated DatsapyCombo control.
- **ApplyDatsapy(datsapyId)**—Attempt to apply the Datsapy identified by the specified datsapyId.
- **ApplyFilter**—Attempt to apply the filter defined by the associated FilterFieldsComboBox, OperatorComboBox, and FilterValueInputBox controls.
- **ApplyFilter(filterItems())**—Attempt to apply the filter specified by the arguments.

- `ApplyFilter(field, operator, value)`—Attempt to apply the filter defined by the array of `GridFilterItem` objects.
- `GetActiveDataRow`—Returns a `DataRow` object for the row index in the data source that matches the current row index of the grid control.
- `GetTotalRowCount`—Returns a string that contains the count of the total potential rows for the grid.
- `LoadData`—Attempt to load the data for the defined with the `GridName`, `GridType`, and `FunctionName` properties.
- `LoadData(session)`—Using the specified session object, attempt to load the data defined with the `GridName`, `GridType`, and `FunctionName` properties.
- `LoadMore`—Attempt to load more data from the cache if available.
- `ViewCell`—Open a dialog with the contents of the current cell. Will be editable if the cell is.

## Events

- `Exception`—Triggered if the `HandleTriggeredException` flag property is set to `HandleTriggeredExceptionMethod.RaiseExceptionEvent` and an exception is thrown by an event that was in turn triggered by an associated control.
- `GridAddingColumn`—Triggered for each column that is added to the `DataGrid` control.
- `Loaded`—Triggered whenever the data is done loading.
- `Loading`—Triggered just before the attempt to load the data is initiated. May cancel the load by setting the event argument `e.Cancel` to `True`.

## FilterLine

This control allows entry of the filter field, operator, and value.

## Properties

- `AndOrText`—Gets the And/Or text displayed below the line.
- `AndOrTextVisible`—Determines whether or not to display the And/Or text below the line.
- `FilterFieldComboBox`—Gets the constituent control.
- `FilterFieldDataSource`—An `IList` or `IListSource` object that contains the list of filter fields to display in the combobox.
- `FilterFieldDisplayMember`—Gets or sets a string that specifies the property of the `FilterFieldDataSource` whose contents will be displayed.
- `FilterFieldSelectedValue`—Gets the value of the selected item in the filter field combo box.
- `FilterFieldText`—Gets the text that is displayed by the filter field combo box.
- `FilterFieldValueMember`—Gets or sets a string that specifies the property of the `FilterFieldDataSource` from which to draw the value.
- `FilterValueText`—Gets the text that is displayed by the filter value textbox.

- `FilterValueInputBox`—Gets the constituent control.
- `LeftParen`—Determines whether or not to display the left parenthesis.
- `OperatorSelectedValue`—Gets the value of the selected item in the operator combo box.
- `OperatorText`—Gets the text that is displayed by the operator combo box.
- `OperatorComboBox`—Gets the constituent control.
- `RightParen`—Determines whether or not to display the right parenthesis.

## Methods

- `AllFieldsAreEntered`—Determines if all of the input boxes contain information.
- `LoadFromXml(XmlTextReader)`—Loads the contents of the filter line from an `XmlTextReader` with contents saved from the `SaveToXml` function.
- `SaveToXml(XmlTextWriter)`—Writes the contents of the filter line to an `XmlTextWriter`.
- `SetFilterLine(field, operator, value)`—Sets the filter line to the specified field, operator and value.
- `ToggleAndOr`—Sets the join condition to 'AND' or 'OR'.
- `ToString`—Returns a string representation of the data from the filter line.

## Events

- `AndOrChanged`—Triggered when the And/Or text is clicked and toggled.
- `AndOrVisibleChanged`—Triggered when the `AndOrTextVisible` property is changed.
- `FilterFieldSelectedIndexChanged`—Triggered when a filter field is selected from the drop-down.
- `FilterInputStyleChanged`—Triggered when the `FilterValueInputBox`'s `Style` property is changed.
- `FilterLookupButtonClick`—Triggered when the lookup button of the `FilterValueInputBox` is clicked.
- `FilterValueTextChanged`—Triggered when text is entered into the value textbox.
- `LeftParenChanged`—Triggered when the left parenthesis is toggled on or off.
- `OperatorSelectedIndexChanged`—Triggered when an operator is selected from the drop-down.
- `RightParenChanged`—Triggered when the right parenthesis is toggled on or off.

## MultilineFilter

Control to display and manage multiple `FilterLine` controls.

## Properties

- `AddLineButtonControl`—Expose the AddLine button control.
- `ApplyButtonControl`—Expose the Apply button control.
- `CurrentFilterLine`—Gets the `FilterLine` that currently has focus.



- **FilterFieldDataSource**—An `IList` or `IListSource` object that contains the list of filter fields to display in the combobox.
- **FilterFieldDisplayMember**—Gets or sets a string that specifies the property of the `FilterFieldDataSource` whose contents will be displayed.
- **FilterFieldValueMember**—Gets or sets a string that specifies the property of the `FilterFieldDataSource` from which to draw the value.
- **FilterLabel**—Allows changing of or hiding the label shown next to the filter lines.
- **FilterLines**—Exposes the list of filter lines shown in the multi-line filter.
- **LeftParenButtonControl**—Expose the `LeftParen` button control.
- **RightParenButtonControl**—Expose the `RightParen` button control.
- **RowExpandLimit**—Determines the number of filter rows that will be displayed before a scrollbar is displayed and the user is forced to scroll the list of filter rows. The minimum is set to `MIN_ROW_EXPAND_LIMIT` and the maximum to `MAX_ROW_EXPAND_LIMIT`.

## Methods

- **AddFilterLine**—Allows programmatically adding a new filter line.
- **ClearLine**—Clears the entered/selected value from all of the controls on the current line.
- **ClearAllLines**—Clears the entered/selected value from all of the controls on the all of the lines.
- **LoadFromXML(XmlTextReader)**—Loads the contents of the multi-line filter from the `XmlTextReader` when in the format saved by the `SaveToXml` method.
- **RemoveAllLines**—Delete all the lines added by clicking `Add Line`. The first lines cannot be removed.
- **RemoveLastLine**—Deletes the last line added by clicking `Add Line`. The first lines cannot be removed.
- **SaveToXml(XmlTextWriter)**—Saves the contents of the multi-line filter to an `XmlTextWriter`.
- **ToGridFilterItems**—Returns an array of `Datastream.EWS.GridFilterItem` instances; each index represents one filter row.
- **ToString**—Returns a string representation of the data from the filter rows, separated with a space character.
- **ToString(lineDelim)**—Returns a string representation of the data from the filter rows, separated with the specified `"lineDelim"`.
- **ValidateBalancedParens**—Determines if the number of left parentheses match then number of right parentheses.

## Events

- **AddLineButtonClick**—Triggered when the `"Add Line"` button is clicked.
- **AndOrChanged**—Triggered when the `And/Or` text is clicked and toggled.
- **ApplyButtonClick**—Triggered when the `"Apply"` button is clicked. This event should contain the code to validate and apply the entered filter to the appropriate data set.

- `FilterInputStyleChanged`—Triggered when the `FilterValueInputBox`'s `Style` property is changed.
- `FilterLookupButtonClick`—Triggered when the lookup button of the `FilterValueInputBox` is clicked.
- `FilterSelectedIndexChanged`—Triggered when a filter field is selected from the drop-down.
- `FilterValueTextChanged`—Triggered when text is entered into the value textbox.
- `LeftParenButtonClick`—Triggered when the "(" button is clicked.
- `LeftParenChanged`—Triggered when the left parenthesis is toggled on or off.
- `OperatorSelectedIndexChanged`—Triggered when an operator is selected from the drop-down.
- `RightParenButtonClick`—Triggered when the ")" button is clicked.
- `RightParenChanged`—Triggered when the right parenthesis is toggled on or off.

## ObjectFieldLinkProvider

This component allows the developer to designate what object field the control should use to read and store the data entered into it. It provides extender properties to the controls by implementing the `IExtenderProvider` interface. See the `ErrorProvider` component that comes with Visual Studio for more details.

The properties provided are as follows:

- `ObjectFieldName`—Name of the field to which to link the value.  
**Note:** The field names are case-sensitive and must match the case of the object field exactly. Example: Assume you have a `TextBox` named `EquipmentTextBox` and one named `EquipOrgTextBox`, and you want to link their contents with that of an instance of `WorkOrderData` named `WOData`.
  - Drop an `ObjectFieldLinkProvider` component onto your form and name it `LinkProvider`.
  - Now set the `ObjectFieldName` property for the `EquipTextBox` to "Equipment.Item" and set `EquipOrgTextBox` to "Equipment.Org".
  - Call `LinkProvider.CopyValuesToControls(WOData)` to load the controls with the data from the object, or call `LinkProvider.CopyValuesFromControls` to store the data from the controls to the object.
- `DefaultFromControl`—Value to use if the control's "Text" property is empty.
- `DefaultToControl`—Value to use if the object's value is empty.
- `OverrideFromControl`—Value to send to object regardless of the control's "Text" property contents.
- `OverrideToControl`—Value to send to the control regardless of the object's value.

See "Diagram 1—Classes Used in Code Examples" in Appendix B for the classes used in the examples above.

In order to use this component from the Form designer, you will need to add it to the Visual Studio Toolbox by selecting the "Add/Remove Items..." menu item from the Toolbox's context menu. From the dialog, press the "Browse..." button, and then select the `Datastream.WinForms.dll` assembly.

Click **OK** and you should have an "ObjectFieldLinkProvider" at the bottom of the Toolbox; you may have to scroll down to see it

After adding the component to the Toolbox, you can simply drag and drop to the Form designer like you do with any other component. Once this is done, the controls on your form will have the five new properties listed above.

## Methods

- **CopyValuesFromControls**—Copies the value from the "Text" property of the controls that have defined the ObjectFieldName into the specified target object.  
The ObjectFieldName property should contain the name of the objects field to which to copy the value.
- **CopyValuesToControls**—Copies the values stored in the fields of the specified source object into the "Text" property of the controls that have defined the ObjectFieldName property.  
The ObjectFieldName property should contain the name of the object's field from which to copy the value.

## Datastream.WinForms.Dialogs namespace

This namespace contains various dialog forms designed to provide various methods to select or enter data.

### CalendarDlg

This dialog is inherited from a standard .NET Windows Form and provides a method to select a date or a date and time using a monthly calendar and spinner-like controls for entering hours and minutes.

The form is designed to be called using ShowDialog in order to display as a modal dialog. Clicking **OK** or double-clicking a date will set the Selection property to the date/datetime shown on the dialog and will set the DialogResult property to DialogResult.OK. Clicking **Cancel** or pressing Esc will set the DialogResult property to DialogResult.Cancel. Note that the value of the DialogResult property is what is returned from the call to ShowDialog.

### Properties

- **IncludeTime**—Determines whether or not the Hour and Minute entry is displayed.
- **Selection**—The selected date/datetime in the specified format.
- **DateFormat**—Get or Set the format that the selected date will be returned as. The default is set to DEFAULT\_DATE\_FORMAT.

- **DateTimeFormat**—Get or Set the format that the selected datetime will be returned as. The default is set to `DEFAULT_DATETIME_FORMAT`.

## Methods

- **PositionAsDropDown**—Attempts to position the dialog below the specified control as if it were a dropdown.

## LookupDlg

This dialog is inherited from a standard .NET Windows Form and provides a method to select an item from a list of values. It uses the `EWSGridDisplayManager` component to load the list of values (LOV).

The form is designed to be called using `ShowDialog` in order to display as a modal dialog. Clicking **OK** or double-clicking a line item will set the `Selection` property to the value for item that is shown selected in the `DataGrid` and will set the `DialogResult` property to `DialogResult.OK`. Clicking **Cancel** or pressing `Esc` will set the `DialogResult` property to `DialogResult.Cancel`. Note that the value of the `DialogResult` property is what is returned from the call to `ShowDialog`.

The section above the list of data provides filtering of the list to a more manageable size. To accomplish this, select the field from the first dropdown, select the appropriate operator from the second dropdown, and then enter a value to compare the field against in the third entry box. Click **Apply** to cause the list to be filtered.

For more complex filtering, additional filter lines may be added and the filter lines may be grouped using parentheses. To add more filter lines, click **Add Line** once for each line desired. For grouping, click on the line to/from which you want to add/remove a parenthesis, and then click on the desired left or right parenthesis button. Filter lines may also be logically compared using "AND" or "OR". This is accomplished by clicking on the "AND" or "OR" text between the filter lines to toggle the value.

## Constructors

- `New()`—Standard constructor
- `New(session, parentGridName, lookup)`—Sets the properties required to make a grid request.

## Properties

- `FilterPanelControl(Panel)`—Provide access to the lookup's filter panel.
- `GridControl(DataGrid)`—Provide access to the lookup's grid control.
- `InitialFilter(EWS.GridFilterItem)`—Allow setting an initial filter for the lookup.
- `Lookup`—The text value that determines what list of values to retrieve and display. This is normally found as part of a `GridData.Toolbar.FilterList` item `Value` property.
- `MultilineFilterControl(MultilineFilter)`—Provide access to the lookup's multiline filter control.

- **Parameters**—The read-only list of parameters required by the grid request. It is parsed from the `Lookup` property.
- **ParentGridName**—Required by the LOV grid request.
- **Selection**—The read-only value that was selected from the list.
- **Session**—An instance of the `Session` class used to make a grid request.

## Methods

- **PositionAsDropDown**—Attempts to position the dialog below the specified control as if it were a dropdown.

## RecordViewDlg

The Record View Dialog provides a quick way to display one row from a grid in record view format.

## Constructors

- **New()**—Standard constructor

## Properties

None

## Methods

- **ShowRecordEditor(DataGrid, DataTable, formTitle as String)**

### AddWorkOrder Java sample

The AddWorkOrder sample application demonstrates how to generate proxy classes from a WSDL, how to use the generated classes to call the web service, and how to read the results of the call. The web service used in the sample is AddWorkOrder (Function #MP0023). This web service is described in the MP0023\_AddWorkOrder\_001.wsdl file.

The AddWorkOrder application in its finished form, with source code, is included in the WS Toolkit. The following sections describe how to create that sample from scratch. Alternatively, the existing AddWorkOrder sample may be opened directly in Eclipse. It should compile successfully once the library paths are adjusted for the local environment (Properties → Java Build Path → Libraries).

The JAX-WS library is used to generate the proxy classes and to perform some runtime tasks. Since JAX-WS is integrated into Java 7, this approach minimizes external dependencies.

The Apache CXF application, which typically uses JAX-WS to do much of the work, may be used in place of direct reliance on JAX-WS; however, this approach is not discussed here.

### Prerequisites for creating the sample

- Download and install Java 7 (the JDK, not the JRE).
- Download and install Apache Ant version 1.8 or later
- Download and install the Eclipse IDE for Java

### Generating the proxy classes

Copy the entire ClientGeneration directory (<WSTOOLKIT>\Java\ClientGeneration) to a new location.

Edit build.properties

- Note: build.properties will be read by Apache Ant, so all file paths should use a forward slash (/) as the separator, even when running on Windows.
- base.dir: a convenience property used only within build.properties. It should point to the new copy of the ClientGeneration directory.
- wsimport.path: the wsimport utility is located in the JAVA\_HOME\bin directory of the JDK installed earlier.
- build.src.dir: directory to hold the generated source files.

- build.output.dir: directory to hold the compiled class files.
- logging.dir: holds the log output from the ant script.
- cust.bind.file: must point to custombindings.xml. The default location is ClientGeneration\config.  
To run the ant script:

Open a command shell

Ensure that ant\bin is on the classpath

- Execute this command:

```
ant -Dwsdl.path="<path to WSToolkit\wsdl\Work\MP0023_AddWorkOrder_001.wsdl"
```

## Building the AddWorkOrder project

- Create a new Java project in Eclipse
- Copy the generated source files to the src directory of the new project
- Copy the config and lib directories from the AddWorkOrder sample into the new project
- Copy the com directory from the AddWorkOrder sample (under src) into the src directory of the new project
- Copy addworkorder.properties from the sample to the new project
- Add lib\log4j.jar to the project classpath
- The project should now compile successfully

## Running the AddWorkOrder project

Edit config/log4j.xml: set the file path to a valid value

Edit addworkorder.properties

This file contains that data necessary to connect to the server and create the work order. All values in this file are required.

- wsdlpath: fully qualified path to the AddWorkOrder wsdl file.
- webserviceurl: set the host and port as appropriate; the path should be left unchanged.
- sysuserid/password: These credentials will be inserted into the SOAP header and serve to authenticate the request on the server. This user must have Connector privileges (User Setup screen).
- tenant: used in the SOAP header of the request.
- soaporganization: used in the SOAP header of the request. This value should normally be set to '\*'.
- workorderorg: the organization of the work order. The sample assumes multiorg is enabled.
- workordertype: the appropriate UCODE for the Work Order Type.
- Statuscode: the appropriate UCODE for the Work Order Event status

- equipcode: the name of an existing asset
- equiporg: the organization associated with the asset
- createdbyuserid: this user will be recorded as the creator of the work order.
- returnalerts: if true, the SOAP body of the response from the server will have two children: the work order response and an InformationAlert with a more verbose message. A SOAP body with more than one child may cause problems in some environments. To disable the information alerts in the response, set this value to false.
- wocount: the number of work orders to create.

## Notes on the application

**Creating multiple work orders:** creating multiple work orders is useful because it illustrates how to create a single session on the server to process multiple requests.

**SOAP Header Creation:** the sample creates SOAP headers directly by casting the WSBindingProvider to the internal JAX-WS implementation. Since this approach involves using several classes in the com.sun namespace, it is not portable to other JVMs or to other JAX-WS implementations. An alternative portable approach (not illustrated here) is to create the SOAP headers in a SOAPHandler.



---

## Appendix A Application Config file

The “Application Congif File” is generated from the app.config source file. It will have the same name as the executable with an “exe.config” extension – i.e. if the application executable is named AddWorkOrder.exe then the config file will be named AddWorkOrder.exe.config. Here are the contents:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <!-- WebServicesURL is the URL to call the web services against.
         NOTE: Will default to WSDL defined value if not set
         here. example:
    <add key="WebServicesURL"
value="http://localhost/axis/services/EWSConnector"
/>
    -->

    <!-- Work Order default input values -->
    <add key="DefaultOrg" value="" />
    <add key="DefaultDeptCode" value="" />
    <add key="DefaultEquipment" value="" />
    <add key="DefaultEquipmentOrg" value="" />
    <add key="DefaultTypeCode" value="" />
    <add key="DefaultStatusCode" value="" /prompted
```

```
<!-- Login default values
  UserID - The User Id to prefill on the login screen.
           Remove or comment out to force entry of the User ID
  RememberLogin - If the value is NOT "true", the user will be
                  prompted for his login credentials every time the Submit button is
                  pressed. This value defaults to "true" if not defined. -->
```

**NOTE:** Regardless of the value, the user will need to enter his login credentials at least once.

```
<add key="UserID" value="" />
      <add key="RememberLogin" value="true" />

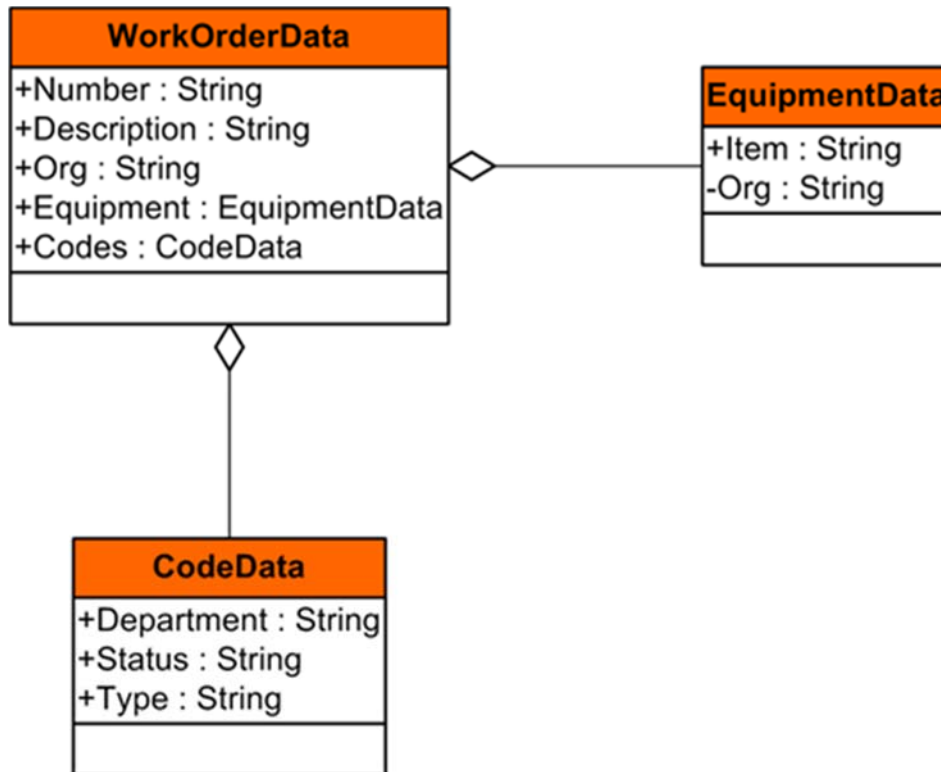
      </appSettings>
</configuration
```

---

## Appendix B Diagrams

This appendix contains diagrams referenced in the body of the guide.

Diagram 1—Classes for Examples



---

## Appendix C Known issues

This document summarizes changes to web services in the current release that could impact client programs written to work with a previous version of the server.

### LinearOverviewPreference\_001:

- The schema has been redesigned, with a number of xml elements added and other removed. Web services based on this schema are not backward compatible
- Affected web services:
  - MP3296\_GetLinearOverviewPreference
  - MP3297\_SyncLinearOverviewPreference

### MP3385\_AddNonConformityTypeEMRS\_001:

- The element MultipleNonConformityTypeEMRS has been replaced by the NonConformityTypeEMRS element, which has different semantics.