

SMS Broker System
Software Integration

Annex. Getting Started With SMS API

(ver. from 8/31/2025)

Table of Contents.

| | |
|--|---|
| 1. Navigation property. | 1 |
| 2. How to create and fill a new customs entry document.(As a sample) | 2 |
| 3. Put to queue methods and their parameters. | 6 |
| 3.1 ContactManager. | 6 |
| 3.2 CustomsEntryManager. | 6 |
| 4. Entity Link. | 7 |
| 5. How to get a list of events.(As a sample) | 7 |
| 6. Error message processing method.(As a sample) | 8 |
| 7. How to get a validation result of asynchronous sent. | 8 |

1. Navigation property.

To understand how "navigation property" works in our system lets look at the following example. Suppose we get an ISF data from the server.

Each ISF document is represented by the record on the database. Its fields (Id, Importer, Carrier, TransportationMode, ClientRef and so on) fall into two groups – fields those contain some value by itself (`string` TransportationMode, `string` ClientRef, `DateTime` Date) and fields those contain Id-s of related records (`long?` Importer_Id, `long?` Carrier_Id). These id-s are Id numbers of corresponded records on Contacts and Carriers tables.

Table 1

| <Id> | Importer_Id <Id> | Carrier_Id <Id> | ClientRef <value> | TransportationMode <value> |
|------|---------------------|--------------------|----------------------|-------------------------------|
|------|---------------------|--------------------|----------------------|-------------------------------|

As a result of request on a client side we will have an `ImporterSecurityFiling` class object. Its structure in general repeats the structure of the database record but there is an important difference. It has navigation properties – properties those contain reference of objects are filled from corresponded tables. For example, `ImporterSecurityFiling` class has not only Importer_Id property (that is filled with Id from Contacts database record) but also Importer property (<reference>) that is a reference to object filled with the data from the Contacts table record with Id number = Importer_Id (compare Table 1 and Table 2).

Table 2

| <Id> | Importer_Id <Id> | Carrier_Id <Id> | ClientRef <value> | TransportationMode <value> | |
|------|-------------------------|------------------------|----------------------|-------------------------------|-----------------------------|
| | Importer <reference> | Carrier <reference> | | | Commodities <collection> |

What properties will be filled when we get the data, for example, by Get() method? To reduce a data exchange between client and server sides only Id-s and values (the first line of the Table 2) are returned by default. No navigation property (Carrier, Importer,...) are returned. All of them will contain `null`.

But if when we want to get the navigation properties filled we should use the special format of Get() method:

```
var isf = ISFmngr.Get(Id, "Importer,Carrier,Commodities"), where ISFmngr is an  
ImporterSecurityFilingManager.
```

The resulting ISF object will have Importer, Carrier and Commodities navigation properties filled (the second line of the Table 2).

As you can see at the Table 2 the resulting ISF object has also Commodities collection filled. Because ISF document record has one-to-many relation to Commodities there is no Commodities Id on ISF document record in a database. Commodities collection is filled by a data from records of the child table Commodities by ISF document Id foreign key.

While data saving (when we save a client object to database) the Id-s (the second line of the Table 2 example) have a more high priority then navigation properties (the first line of the Table 2 example) and exactly these id-s will replace id-s on the database records.

Other words if we get an ISF object, fill it with navigation properties (without Id-s) and then simply save it back, nothing will change in database.

Exclusion is only for collection properties because as mentioned above collection's parent record anyway doesn't contain reference Id-s to collection it owes (empty last field on the second line of the Table 2 example). And hence collection is always saved to database by its navigation property. Also, if you will save collection as null nothing will happen. If you will save collection as object with Count = 0 the collection will be deleted from a database.

2. How to create and fill a new customs entry document. (As a sample)

```
protected void btAddNewCE_Click(object sender, EventArgs e)
{
    // Initialize Client Context. It must be your connection data here:
    string url = @"https://localhost:8130/brokerservice";
    string username = "admin";
    string database = "sandbox";
    string password = "somepassword";
    SMS.Broker.ClientContext.InitializeContext(url, username + "@" + database, password);

    ICustomsEntryManager mngrCE =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Documents.ICustomsEntryManager>();
    ICustomsPortManager mngrCustomsPort =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.ICustomsPortManager>();
    IFIRMManager mngrFIRM = SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.IFIRMManager>();
    IContactManager mngrContact =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.IContactManager>();
    ICarrierManager mngrCarrier =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.ICarrierManager>();
    IShipmentManager mngrShipment =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Documents.IShipmentManager>();
    IForeignPortManager mngrForeignPort =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.IForeignPortManager>();
    IHarmonizedTariffManager mngrHarmonizedTariff =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.IHarmonizedTariffManager>();
    IStatementDailyManager mngrStatementDaily =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Documents.IStatementDailyManager>();
    ICustomerTariffManager mngrCustomerTariff =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.ICustomerTariffManager>();
    Broker.ServiceContracts.ISmsServiceManager mngrSmsService =
        SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.ISmsServiceManager>();

    #region prepare data to make new customs entry (Real data can be got from user interface)

    // ===== Header Data
    string EntryType = "01";
    string TransportationMode = "11";
    string LocationCode = "F430";
    string PortOfEntryCode = "4601";
    string PortOfUnladingCode = "4601";
    string ConveyanceName = "YM WIND";
    string Tripldentifier = "011E";
```

```

// Header Data, Bond Data
string BondType = "8";
string SuretyCode = "037";
decimal BondAmount = 5000m;
string BondNumber = "17C002XYZ";

// Header, Statement Data

string PaymentType = "7";

DateTime EstimatedEntryDate = DateTime.Today;
DateTime DateOfImport = DateTime.Today;
DateTime DateOfArrival = DateTime.Today;

// ===== Shipment Data
//Some user data (may be Id from external system)
string ClientRef = "CE005";

//SCAC code
string CarrierCode = "MAEU";

//Unique code of Contact in our database (Generated by our system on adding new contact)
//in this example used if contact already exists on the database
string ImporterCode = "FRENEW";

string ImporterName = "FREY NEWNAN";
string ImporterAddress1 = "10 RAYMOND HILL ROAD";
string ImporterAddress2 = "";
string ImporterAddress3 = "";
string ImporterCountry = "US";
string ImporterState = "GA";
string ImporterCity = "NEWNAN";
string ImporterZIP = "30265";
string ImporterIRS = "93-031218300";

string BillIssuer = "JAPT";
string BillNumber = DateTime.Now.ToString("MMddyyHHmmss");
string BillType = "House"; //Can be House or Regular

//Each party has a type Contact like Importer
//So can be processing a same manner
//For easy code will use same Contact as Importer
string ConsigneeCode = "FRENEW";
string SellerCode = "FRENEW";
string BuyerCode = "FRENEW";
string ShipToCode = "FRENEW";
string ConsolidatorCode = "FRENEW";

//===== Shipment Invoice data
string Description = "PCS OF BUZZER";
string InvoiceNumber = "123451234";

//===== Shipment article data
string ManufacturerCode = "WENTOP";
string CountryOfOrigin = "CN";
string CountryOfExport = "HK";
decimal GrossWeight = 393.00m;

//string Description = "PCS OF BUZZER";
string ForeignPortOfLadingCode = "58201";
decimal MerchandiseProcessingFee = 44.82m;
decimal HarborMaintenanceFee = 16.18m;

//===== Tariff data
string HarmonizedTariffCode = "8531909001";
decimal Quantity1 = 11000m;
string UnitOfMeasure1 = "NO";
//decimal Quantity2 =
//string UnitOfMeasure2 =
//decimal Quantity3 =
//string UnitOfMeasure3 =

decimal TariffValue = 12940m;
decimal DutyAmount = 38.82m;

// If it would be some textile
//string UserFeeAccountingClassCode = "056"; // Cotton Fee
//decimal UserFeeAmount = 25.30m;

#endregion

//Get new CE with filled Date and Number.
//System has a sequence of CE numbers.
//Start Number can be adjusted.
CustomsEntry newCE = mngrCE.New(null);

//Assigned next sequence number

```

```

var CENumber = newCE.Number;

newCE.ClientRef = ClientRef;

newCE.EstimatedEntryDate = EstimatedEntryDate;
newCE.DateOfImport = DateOfImport;
newCE.DateOfArrival = DateOfArrival;
newCE.BrokerReferenceNumber = newCE.EntryNumber;

newCE.BondType = BondType;
newCE.SuretyCode = SuretyCode;
newCE.BondAmount = BondAmount;
newCE.BondNumber = BondNumber;

//Some property has a list of possible values
//if Class has a static property with some name + "Values"
//we can get a list of possible values:
var ListOfEntryTypePossibleValues = CustomsEntry.EntryTypeValues;

// But here we use EntryType string value defined before:
//String fiels has an Attribute StringLength with lehght of field
//[StringLength(2)]
//public string EntryType { get; set; }
newCE.EntryType = EntryType;

newCE.TransportationMode = TransportationMode;

//Get FIRMS from database by FIRMS Code
SMS.Broker.DataContracts.Directories.FIRM Location = mngrFIRM.GetByCode(LocationCode, "");
newCE.Location_Id = Location.Id;

//Get domestic ports from database by Port Code
SMS.Broker.DataContracts.Directories.CustomsPort PortOfEntry = mngrCustomsPort.GetByCode(PortOfEntryCode, "");
newCE.PortOfEntry_Id = PortOfEntry.Id;
SMS.Broker.DataContracts.Directories.CustomsPort PortOfUnlading = mngrCustomsPort.GetByCode(PortOfUnladingCode, "");
newCE.PortOfUnlading_Id = PortOfUnlading.Id;

newCE.ConveyanceName = ConveyanceName;
newCE.TripIdentifier = TripIdentifier;

newCE.PaymentType = PaymentType;

//Get Carrier from database by Carrier Code
SMS.Broker.DataContracts.Directories.Carrier Carrier = mngrCarrier.GetByCode(CarrierCode, "");
newCE.Carrier_Id = Carrier.Id;

SMS.Broker.DataContracts.Directories.Contact Importer;
Importer = mngrContact.GetByCode(ImporterCode, "");
if (Importer == null) // new contact
{
    Importer = new SMS.Broker.DataContracts.Directories.Contact();

    Importer.Name = ImporterName;
    Importer.Address1 = ImporterAddress1;
    Importer.Address2 = ImporterAddress2;
    Importer.Address3 = ImporterAddress3;
    Importer.Country = ImporterCountry;
    Importer.State = ImporterState;
    Importer.City = ImporterCity;
    Importer.ZIP = ImporterZIP;
    Importer.IRSNumber = ImporterIRS;

    //Return Contact from dtabase (updated)
    //If it will be new - result will have a Id and Code generated on adding
    Importer = mngrContact.Save(Importer);
}

newCE.Importer_Id = Importer.Id;

#region Shipment
//Get new shipment with filled Date and Number.
//System has a sequence of Shipments.
//Start Number can be adjusted.
Shipment shipment = mngrShipment.New(null);

shipment.Carrier_Id = Carrier.Id;
shipment.Importer_Id = Importer.Id;
shipment.TransportationMode = TransportationMode;
if (BillType == "House")
{
    //For house bill #
    shipment.HouseBillIssuer_Id = mngrCarrier.GetByCode(BillIssuer, "").Id;
    shipment.HouseBillNumber = BillNumber;
}
else if (BillType == "Regular")
{
    //For Regular bill #
    shipment.MasterBillIssuer_Id = mngrCarrier.GetByCode(BillIssuer, "").Id;

```

```

    shipment.MasterBillNumber = BillNumber;
}
//Assing Entities
shipment.Consignee_Id = mngrContact.GetByCode(ConsigneeCode, "").Id;
shipment.Seller_Id = mngrContact.GetByCode(SellerCode, "").Id;
shipment.Buyer_Id = mngrContact.GetByCode(BuyerCode, "").Id;
shipment.ShipTo_Id = mngrContact.GetByCode(ShipToCode, "").Id;
shipment.StuffingLocation_Id = mngrContact.GetByCode(LocationCode, "").Id;
shipment.Consolidator_Id = mngrContact.GetByCode(ConsolidatorCode, "").Id;

```

```

//Add bill to new CE
newCE.Shipments.Add(shipment);
#endregion Shipment

```

```

#region ShipmentInvoice

```

```

ShipmentInvoice invoice = new ShipmentInvoice();
invoice.CreateCollections(); // We create invoice not from New() method (it is absent in shipmentArticle class) so
// we have to create its collections running this method;
invoice.Description = Description;
invoice.InvoiceNumber = InvoiceNumber;
shipment.Invoices.Add(invoice);

```

```

#endregion ShipmentInvoice

```

```

#region ShipmentArticle

```

```

ShipmentArticle shipmentArticle = new ShipmentArticle();
shipmentArticle.CreateCollections(); // We create shipment article not from New() method (it is absent in shipmentArticle class) so
// we have to create its collections running this method;

```

```

shipmentArticle.Manufacturer_Id = mngrContact.GetByCode(ManufacturerCode, "").Id;
shipmentArticle.Manufacturer_Id = mngrContact.GetByCode(ConsigneeCode, "").Id;
shipmentArticle.CountryOfOrigin = CountryOfOrigin;
shipmentArticle.CountryOfExport = CountryOfExport;
shipmentArticle.GrossWeight = GrossWeight;
shipmentArticle.Description = Description;

```

```

//Get foreign port of lading from database by Port Code
SMS.Broker.DataContracts.Directories.ForeignPort ForeignPortOfLading = mngrForeignPort.GetByCode(ForeignPortOfLadingCode, "");
shipmentArticle.Manufacturer_Id = ForeignPortOfLading.Id;

```

```

shipmentArticle.MerchandiseProcessingFee = MerchandiseProcessingFee;
shipmentArticle.HarborMaintenanceFee = HarborMaintenanceFee;

```

```

invoice.Articles.Add(shipmentArticle);

```

```

#endregion ShipmentArticle

```

```

#region Tariff

```

```

ShipmentArticleTariff tariff = new ShipmentArticleTariff();

```

```

//Get HTS tariff from database by its code
SMS.Broker.DataContracts.Directories.HarmonizedTariff HarmonizedTariff = mngrHarmonizedTariff.GetByCode(HarmonizedTariffCode, "");
tariff.HarmonizedTariff_Id = HarmonizedTariff.Id;

```

```

tariff.Quantity1 = Quantity1;
tariff.UnitOfMeasure1 = UnitOfMeasure1;
//tariff.Quantity2 = Quantity2;
//tariff.UnitOfMeasure2 = UnitOfMeasure2;
//tariff.Quantity3 = Quantity3;
//tariff.UnitOfMeasure3 = UnitOfMeasure3;

```

```

tariff.TariffValue = TariffValue;
tariff.DutyAmount = DutyAmount;

```

```

// If it would be, for example, some textile
//tariff.UserFeeAccountingClassCode = "056"; // Cotton Fee
//tariff.UserFeeAmount = 25.30m;

```

```

// Find HarmonizedTariffCode = "8531909001" on customer tariff table (tariff numbers those are used by importer "FRENEW")
// If it is absent there we add this tariff to customer tariff table before use it on current tariff.

```

```

SMS.Broker.DataContracts.Directories.CustomerTariff CustomerTariff = mngrCustomerTariff.GetCustomerTariffByCode(Importer.Id, "8531909001");
if (CustomerTariff == null)
{
    CustomerTariff = new SMS.Broker.DataContracts.Directories.CustomerTariff();

    CustomerTariff.Code = "8531909001";
    CustomerTariff.Name = HarmonizedTariff.Name;
}

```

```

CustomerTariff.Contact_Id = Importer?.Id;

//Return CustomerTariff from database (updated)
//If it will be new - result will have a Id and Code generated on adding
CustomerTariff = mngrCustomerTariff.Save(CustomerTariff);
}

tariff.Tariff_Id = CustomerTariff?.Id;

shipmentArticle.Tariffs.Add(tariff);

#endregion Tariff

DateTime preliminaryStatementPrintDate;
DateTime? preliminaryStatementMonthDate;

mngrSmsService.GetPreliminaryStatementDates(newCE.DateOfArrival, newCE.Date, newCE.PaymentType, out preliminaryStatementPrintDate,
out preliminaryStatementMonthDate);

newCE.PreliminaryStatementPrintDate = preliminaryStatementPrintDate;
newCE.PreliminaryStatementMonthDate = preliminaryStatementMonthDate;
//save a new CE to database with Shipments(Bills) and all other details
newCE = mngrCE.Save(newCE);

//Get Id of new CE as it saved to database
var NewCEId = newCE.Id;
}

```

3. Put to queue methods and their parameters.

3.1 ContactManager.

Synchronous method:

```
void PutToQUEUE(long id, Contact.ActionQUEUE action);
```

Asynchronous method:

```
void PutToQUEUEOneWay(long id, Contact.ActionQUEUE action);
```

long id – identifier Contact class object.
Contact.ActionQUEUE action - enumerator of different actions.

Actions:

```

Contact.ActionQUEUE.AddManufacturerNameAddress,
Contact.ActionQUEUE.AddManufacturerNameAddressIfNeed,
Contact.ActionQUEUE.ImporterBondQuery,
Contact.ActionQUEUE.ImporterBondQueryNameAndAddress,
Contact.ActionQUEUE.ImporterConsigneeCreate,
Contact.ActionQUEUE.ImporterConsigneeUpdate,
Contact.ActionQUEUE.ApplyForCBPAssignedNumber,
Contact.ActionQUEUE.GlobalBusinessIdentifierAdd,
Contact.ActionQUEUE.GlobalBusinessIdentifierUpdate,
Contact.ActionQUEUE.GlobalBusinessIdentifierDelete.

```

Examples:

```
IContactManager mngrContact = ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Directories.IContactManager>();
```

```
mngrContact.PutToQUEUE(ImporterId, Contact.ActionQUEUE.AddManufacturerNameAddress); // Add Manufacturer Name & Address sent.
```

```
mngrContact.PutToQUEUEOneWay(ImporterId, Contact.ActionQUEUE.AddManufacturerNameAddress); // Add Manufacturer Name & Address sent.
```

3.2 CustomsEntryManager.

Only asynchronous method:

```
void PutToQUEUEOneWay(long id, CustomsEntry.ActionQUE action);
```

`long id` – identifier CustomsEntry class object.
`CustomsEntry.ActionQUE action` - enumerator of different actions.

Actions:

```
CustomsEntry.ActionQUE.AddEntrySummary,  
CustomsEntry.ActionQUE.ReplaceEntrySummary,  
CustomsEntry.ActionQUE.DeleteEntrySummary,  
CustomsEntry.ActionQUE.AddCargoRelease,  
CustomsEntry.ActionQUE.ReplaceCargoRelease,  
CustomsEntry.ActionQUE.CancelCargoRelease,  
CustomsEntry.ActionQUE.BOLUpdate,  
CustomsEntry.ActionQUE.AddEntrySummaryAndCargoRelease,  
CustomsEntry.ActionQUE.ReplaceEntrySummaryAndCargoRelease,  
CustomsEntry.ActionQUE.ACHUpdate,  
CustomsEntry.ActionQUE.EntrySummaryQuery,  
CustomsEntry.ActionQUE.TemporaryImportationBondExtend,  
CustomsEntry.ActionQUE.TemporaryImportationBondClose.
```

Example:

```
ICustomsEntryManager mgrCE = ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Documents.ICustomsEntryManager>();
```

```
mgrCE.PutToQUEUEOneWay(ImporterId, CustomsEntry.ActionQUE.AddEntrySummary); // CBPF 7501 sent.
```

4. Entity Link.

`EntityLink` is a special class that identifies one data record. It's object contains an info about the type of Document/Directory object it is linked to and also about object's identifier. Entity link object has a string representation (Link property). It consists of a type (class) name, delimiter "=" and identifier of related record.

For example, CustomsEntry class object was saved to database. It means that in database CustomsEntries table the record with some Id has appeared. Suppose its Id = 25. Then, if we get this CustomsEntry object from database its EntityLink.Link property is equal to "CustomsEntry=25".

Until new Document or Directory class object was saved its Id = 0 and its EntityLink.Link contains GUID instead of Id. For example "CustomsEntry=e661a76f-a63f-4b99-994c-5e32db2c911f".

Detailed description of EntityLink class you can see in *05. Base, common usage classes and interfaces.pdf*, 5. EntityLink.

5. How to get a list of events. (As a sample)

```
public static List<EntityEvent> GetISFEvents(long isf_Id)  
{  
    // Initialize Client Context. It must be your connection data here:  
    string url = @"https://localhost:8130/brokerservice";  
    string username = "admin";  
    string database = "sandbox";  
    string password = "somepassword";
```

```

SMS.Broker.ClientContext.InitializeContext(url, username + "@" + database, password);

// Prepare manager:
SMS.Broker.ServiceContracts.Common.IEntityEventManager mngrEntityEvent =
SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Common.IEntityEventManager>();

// ISF document link looks like "ImporterSecurityFiling=24" (Id = 24, for example)
string ISFlink = "ImporterSecurityFiling=" + isf_Id.ToString().Trim();

// When we know a link we can use it to find related events.
// Here we read Customs ABI response from SMS-server by prepared link.
List<EntityEvent> ISFABIEvents = mngrEntityEvent.GetABIEvents(ISFlink, true); // true means that SMS-system returns
// also a "user-oriented" text in Text
// property of event objects.

return ISFABIEvents;
}

```

6. Error message processing method. (As a sample)

It returns a text message based on caught exception object getting it as parameter.

```

private static string ErrorMessage(System.ServiceModel.FaultException<SMS.Broker.Faults.EntryValidationFault> ex)
{
    string errorMessage = "";
    foreach (var entity_with_errors in ex.Detail.EntryValidationErrors)
    {
        var entityRepresentation = entity_with_errors.Representation; // entity string representation
        errorMessage += "\n" + entityRepresentation;

        foreach (var property_with_error in entity_with_errors.Details) // loop through all properties with error
        {
            var property_name = property_with_error.Property;
            var property_value = property_with_error.ValueAsString;
            var property_error_explanation = property_with_error.ErrorMessage; // error string representation

            errorMessage += "\n" + property_name + ": " + property_value + " - " + property_error_explanation;
        }
    }
    return errorMessage;
}

```

7. How to get a validation result of asynchronous sent.

```

private static FaultException<EntryValidationFault> GetPutToQUEFault(long CEntryId)
{
    // Initialize Client Context. It must be your connection data here:
    string url = @"https://localhost:8130/brokerservice";
    string username = "admin";
    string database = "sandbox";
    string password = "somepassword";
    SMS.Broker.ClientContext.InitializeContext(url, username + "@" + database, password);

    // Prepare manager:
    ICustomsEntryManager mngrCustomsEntry =
    SMS.Broker.ClientContext.ServicesFactory.GetManager<SMS.Broker.ServiceContracts.Documents.ICustomsEntryManager>();

    // Get customs entry object with its last statuses collection:
    CustomsEntry CE = mngrCustomsEntry.Get(CEntryId, "LastStatuses");
    if (CE == null)
    {
        return null;
    }

    // Return the last status object of exactly "que operation":
    EntityLastStatus status = CE.LastStatuses.Status(EntityStatusType.ABIAppeQUE);

    // If such last status object exists and result is an error we deserialize status.Text property and get a validation object.
    FaultException<EntryValidationFault> fault = null;
    if (status?.Value == SMS.Broker.DataContracts.Common.EntityStatusValue.ABIAppePutToQUEError)
    {

```



```
{  
    fault = SMS.Broker.Common.Serializator.ReadObjectDataContract<FaultException<EntryValidationFault>>(status.Text);  
}  
  
return fault;  
}
```