Microsoft Dynamics® AX 2012

Adapting your code to table normalization for country/region fields in Microsoft Dynamics AX 2012 R2

White Paper

This document provides common patterns for how to use country-specific/region-specific tables.

November 2012

www.microsoft.com/dynamics/ax

Rekha Nanda, Mark Skunberg, and Jon Bye

Send suggestions and comments about this document to adocs@microsoft.com. Please include the title with your feedback.

Table of Contents

Overview

Normalization of country/region fields that have been added to the core transaction tables in Microsoft Dynamics AX provides performance improvements in the product. Based on analysis of core transaction tables, Microsoft has moved several country-specific or region-specific fields into country-specific/region-specific child tables. Moving these fields to separate tables narrows the width of heavily used transaction tables, resulting in performance improvements. Country-specific/region-specific fields are good candidates to be moved, because the performance cost associated with the retrieval of the country-specific/region-specific fields should only be incurred by the countries and regions requiring those fields. Patterns and guidelines have been established for data model normalization and access to country-specific/region-specific fields. These guidelines apply to commonly used patterns and do not necessarily cover all application design scenarios. They will help save on development time by enabling access to these country-specific/region-specific tables and fields anywhere in code where we have access to the parent table, without requiring that we refactor complex class hierarchies and without bloating method signatures with country-specific/region-specific table buffer parameters. These patterns include form patterns and logic patterns.

Document purpose

This document provides common patterns for how to use country-specific/region-specific tables. The following topics are discussed:

- Moving fields out of existing tables for normalization
- Table definition changes
- Code refactoring patterns

Audience

This white paper is intended for developers who integrate their code with Microsoft Dynamics AX. It is assumed that the reader of this document is familiar with the Microsoft Dynamics AX development environment and X++.

Terms

- **Parent table** – The parent table contains the core information. For example, TaxTrans contains the **Voucher** field, which is used in all country/region context scenarios.
- **Child table** – The child table contains country/region context information. For example, TaxTrans_W contains the **VATNum_PL** field, which is only used in Polish country/region context scenarios.

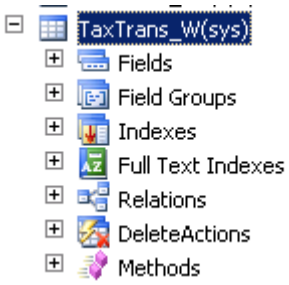
Moving fields out of core tables

Naming convention for child tables

Country/region fields have been moved out of core transaction tables into child tables. Child tables have been named as follows:

- A new child table that has fields for a single country/region will be named <ParentTable>_<CountryRegionCode>. For example, TaxTrans_RU contains the Russian tax transaction fields.
- A new child table that has fields for multiple countries/regions will be named <ParentTable>_W – for example, TaxTrans_W.

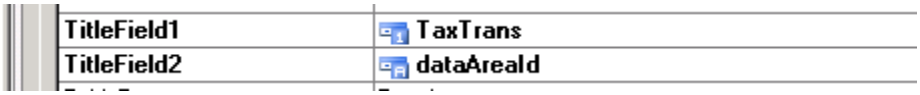
Create a child table



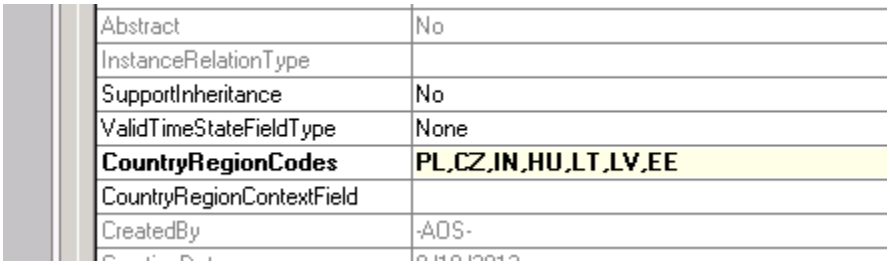
Set the table metadata

The child table metadata should have the same property settings as the parent table metadata.

1. Adjust the child table label appropriately.
2. Adjust the child table developer documentation appropriately.
3. Set **TitleField1** to the reference to the primary table.
4. Set **TitleField2** to the dataAreaId for company-specific tables. Company-specific tables are defined when the table metadata property **SaveDataPerCompany** is set to **Yes**.

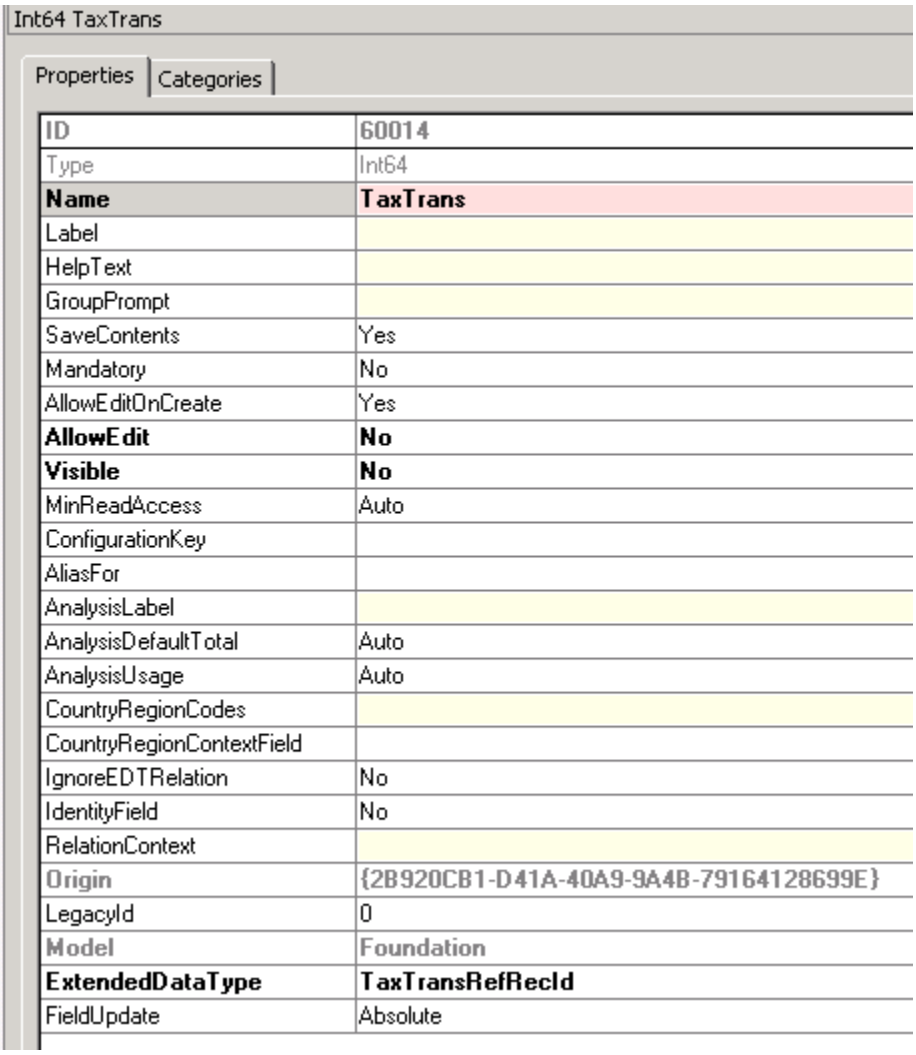


5. Add the necessary country/region codes, separated by commas.

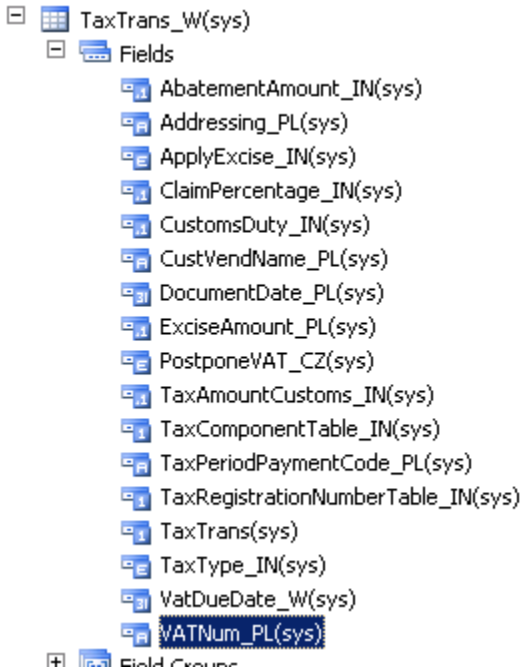


Add fields

1. Add a reference field to the primary table. Give it the same name as the primary table.

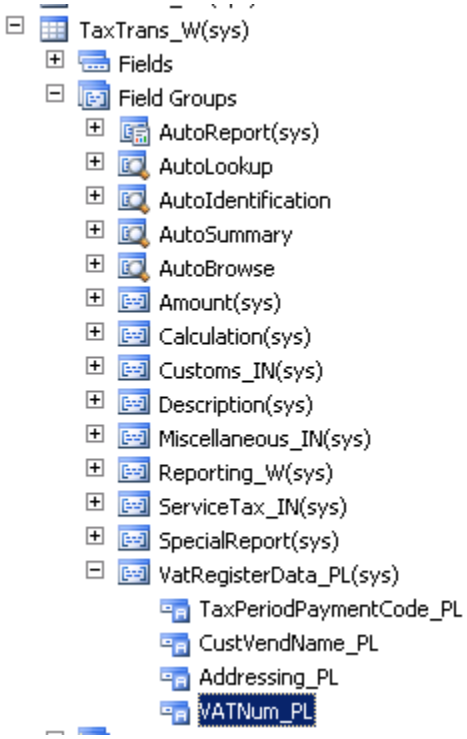


2. Drag country/region context fields from the parent table to the child table. The fields will now exist in both the parent table and the child table.



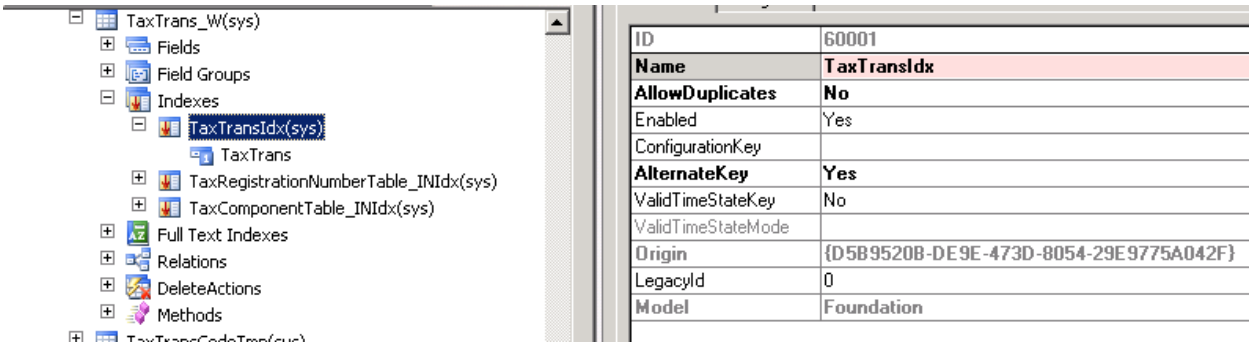
Create field groups

- Refer to the parent table field groups, and copy those that contain the country/region context fields.

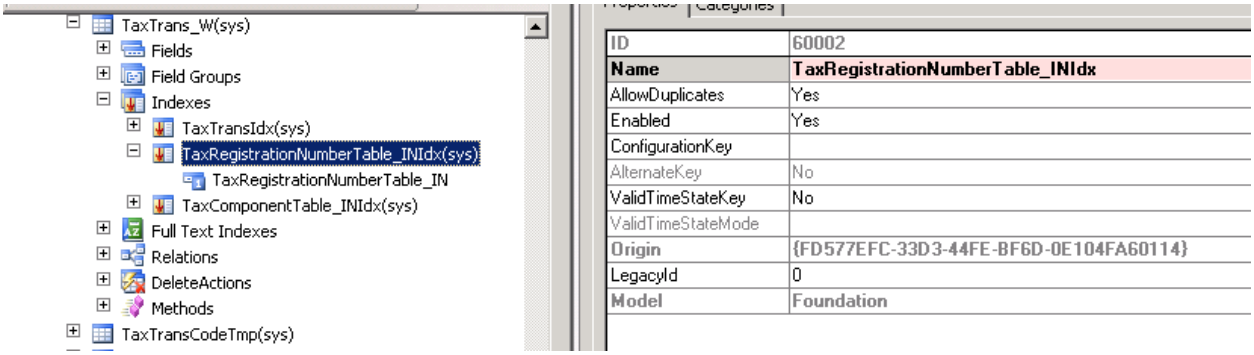


Create table indexes

1. Create a unique index for the reference field that points back to the primary table.

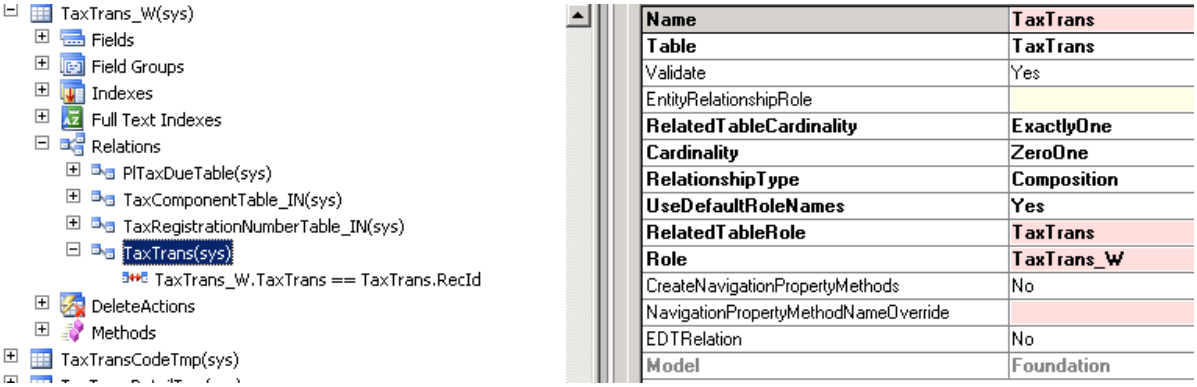


2. Refer to the parent table indexes, and copy those that contain the country/region context fields. After refactoring the application logic, reconsider whether the index is actually necessary.

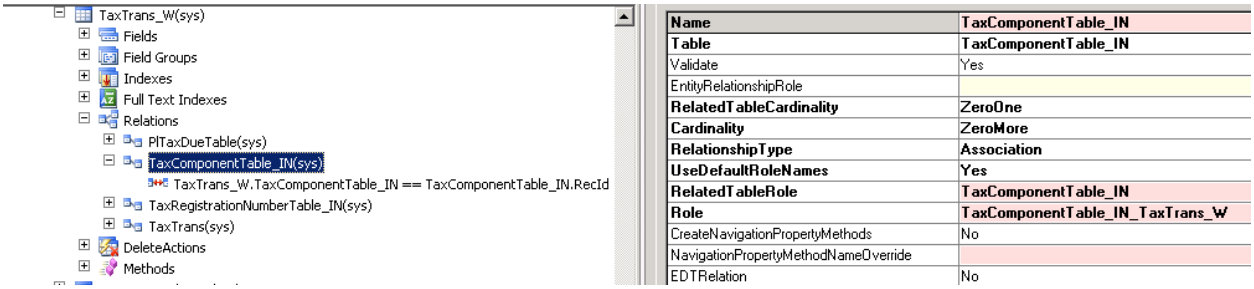


Create relations

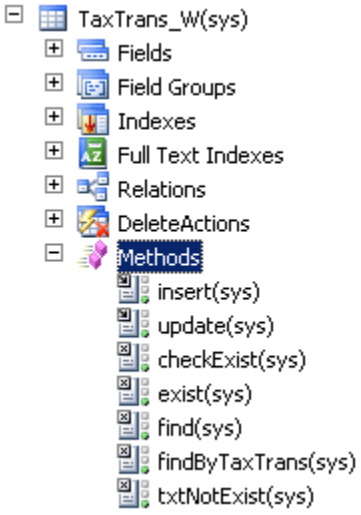
1. Create a relation for the child table to the parent table.



2. Refer to the parent table relations, and copy those that contain the country/region context fields.



Create methods



- **insert**

  The forms engine will try to save the child before the parent is saved. This code enables the framework to trigger the insert of the child after the parent is saved.

```
public void insert()
{
    if (this.TaxTrans != 0)
    {
        // only insert if the FK is valid
        super();
    }
}
```

- **update**

  The forms engine will try to save the child before the parent is saved. This code enables the framework to trigger the update of the child after the parent is saved.

```
public void update()
{
    if (this.TaxTrans != 0)
    {
        // only update if the FK is valid
        super();
    }
}
```

- **find**

```
public static TaxTrans_W find(
    recId _recId,
    boolean _forUpdate = false,
    ConcurrencyModel _concurrencyModel = ConcurrencyModel::Auto)
{
    TaxTrans_W taxTrans_W;

    taxTrans_W.selectForUpdate(_forUpdate);
    if (_forUpdate && _concurrencyModel != ConcurrencyModel::Auto)
    {
        taxTrans_W.concurrencyModel(_concurrencyModel);
    }

    select firstonly taxTrans_W where taxTrans_W.RecId == _recId;

    return taxTrans_W;
}
```

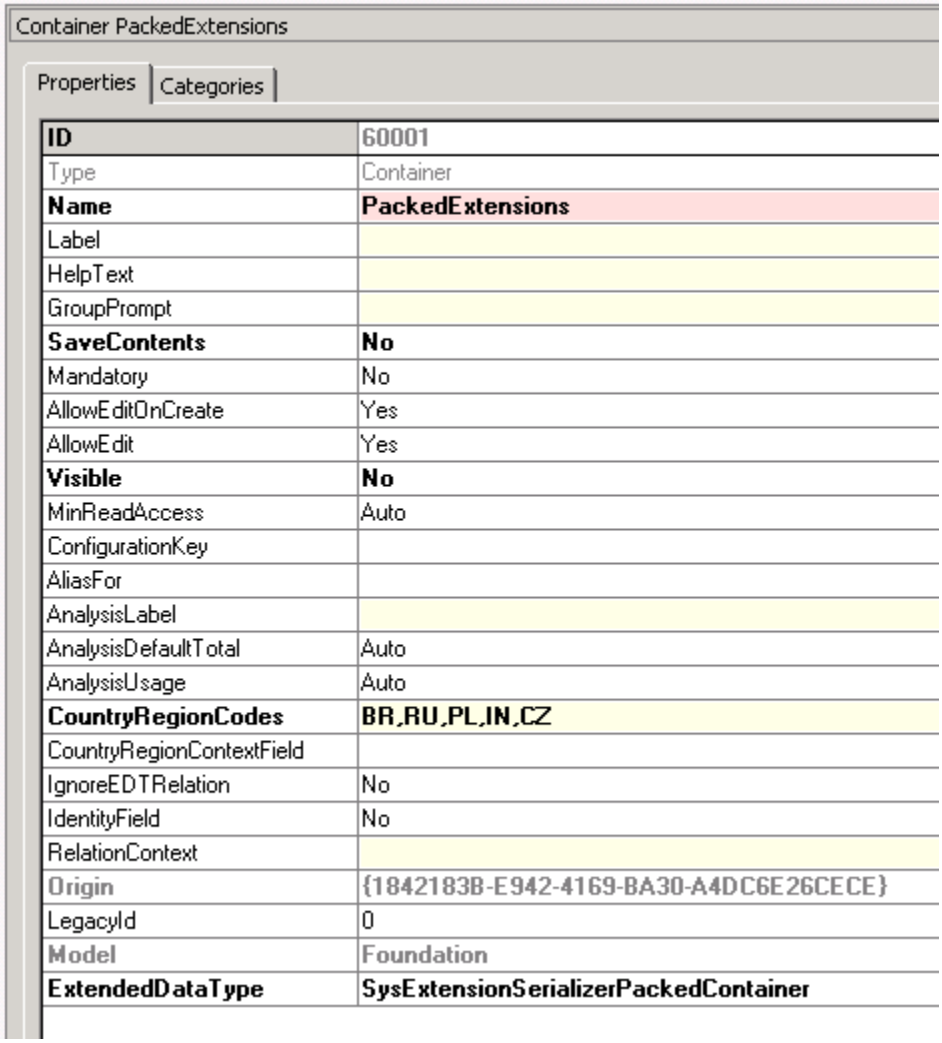
- **findByParent**

```
public static TaxTrans_W findByTaxTrans(RecId _taxTransRecId, boolean _forUpdate = false)
{
    TaxTrans_W taxTrans_W;

    taxTrans_W.selectForUpdate(_forUpdate);

    if (_taxTransRecId != 0)
    {
        select firstonly * from taxTrans_W
            where taxTrans_W.TaxTrans == _taxTransRecId;
    }

    return taxTrans_W;
}
```

Update the parent table

Add fields

- Add the **PackedExtensions** field.

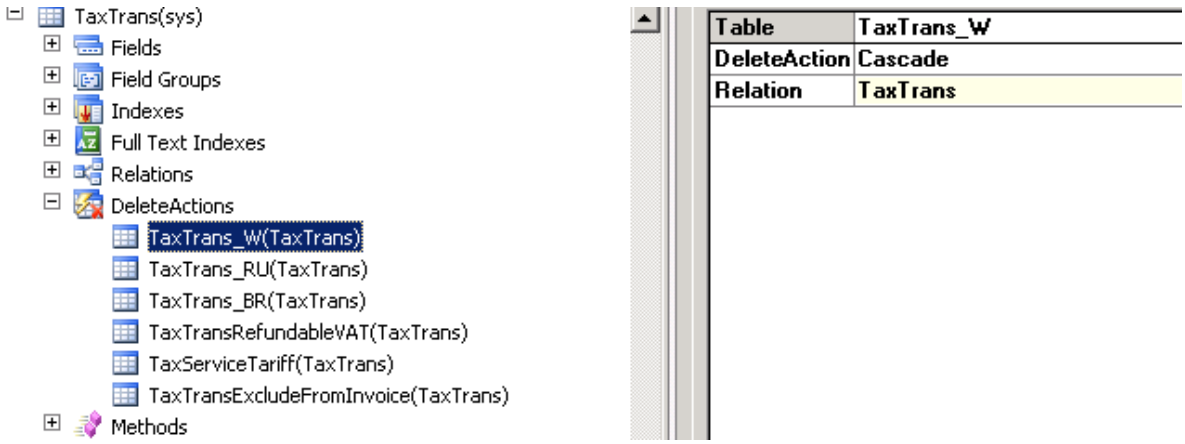


ID	60001
Type	Container
Name	**PackedExtensions**
Label	
HelpText	
GroupPrompt	
SaveContents	**No**
Mandatory	No
AllowEditOnCreate	Yes
AllowEdit	Yes
Visible	**No**
MinReadAccess	Auto
ConfigurationKey	
AliasFor	
AnalysisLabel	
AnalysisDefaultTotal	Auto
AnalysisUsage	Auto
CountryRegionCodes	**BR,RU,PL,IN,CZ**
CountryRegionContextField	
IgnoreEDTRelation	No
IdentityField	No
RelationContext	
Origin	{1842183B-E942-4169-BA30-A4DC6E26CECE}
LegacyId	0
Model	**Foundation**
ExtendedDataType	**SysExtensionSerializerPackedContainer**

Add delete actions

- Add a delete action from the parent table to the child table.

  The framework lets developers add the delete action on either the parent table or the child table. Therefore, it is possible to add the delete action only to the child table.



Add or update methods

1. Add a method to get the child table.

```
public TaxTrans_W taxTrans_W()
{
    return this.SysExtensionSerializerMap::getExtensionTable(tableNum(TaxTrans_W));
}
```

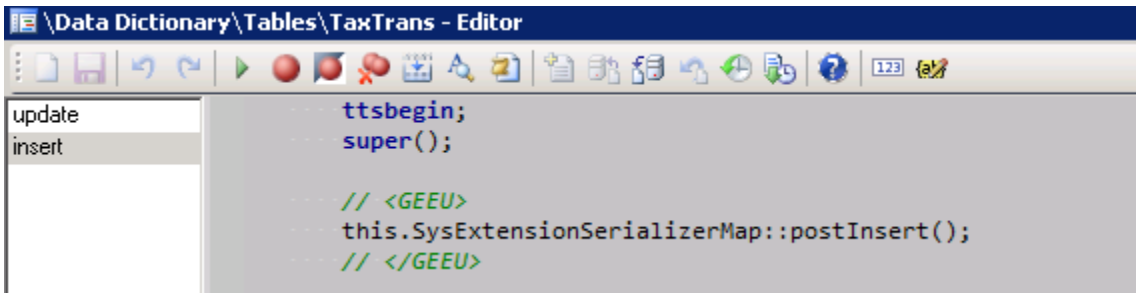
2. Add a method to pack the child table on the parent table packed extensions.

```
public void packTaxTrans_W(TaxTrans_W _taxTrans_W)
{
    _taxTrans_W.TaxTrans = this.RecId;
    this.SysExtensionSerializerMap::packExtensionTable(_taxTrans_W);
}
```

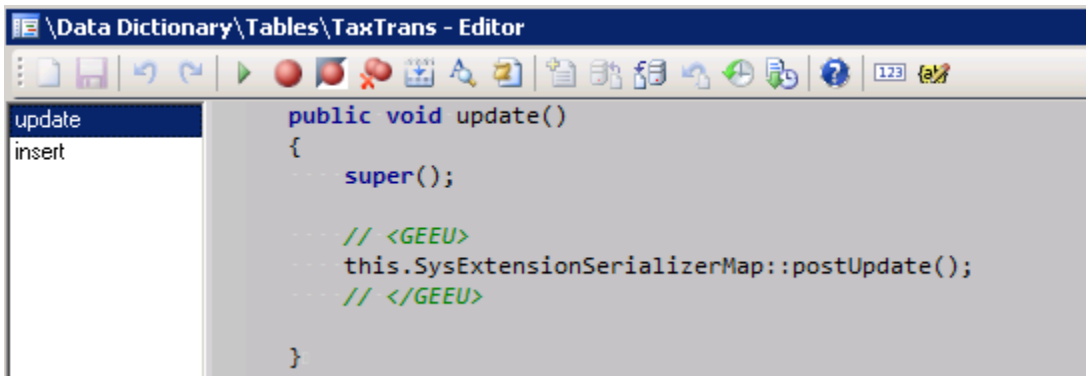
3. Add a method to add a child table data source.

```
public static QueryBuildDataSource addDataSource(QueryBuildDataSource _parentDataSource, TableId _tableId, JoinMode _joinMode = JoinMode::InnerJoin)
{
    QueryBuildDataSource ds;

    if (_parentDataSource == null || _parentDataSource.table() != tableNum(TaxTrans) || (_tableId != tableNum(TaxTrans_W) && _tableId != tableNum(TaxTrans_RU)))
    {
        throw error(Error::wrongUseOfFunction(funcName()));
    }

    switch (_tableId)
    {
        case tableNum(TaxTrans_W):
            ds = _parentDataSource.addDataSource(tableNum(TaxTrans_W));
            ds.addLink(fieldNum(TaxTrans, RecId), fieldNum(TaxTrans_W, TaxTrans));

            ds.fetchMode(QueryFetchMode::One2One);
            ds.joinMode(_joinMode);
            break;
        case tableNum(TaxTrans_RU):
            ds = _parentDataSource.addDataSource(tableNum(TaxTrans_RU));
            ds.addLink(fieldNum(TaxTrans, RecId), fieldNum(TaxTrans_RU, TaxTrans));

            ds.fetchMode(QueryFetchMode::One2One);
            ds.joinMode(_joinMode);
            break;
    }


    return ds;
}
```

4. Update the **insert** method to invoke **postInsert** after the super call.



5. Update the **update** method to invoke **postUpdate** after the super call.



Integrate with the table extensions framework

Update the SysExtensionSerializerMap mapping for the parent table

Update the SysExtensionSerializerExtensionMap mappings for the child table

Remove country/region context information from the parent table

1. Rename the country/region context fields, using the **DEL_** prefix.
2. Compile the parent table, and fix any errors for field groups, indexes, relations, and delete actions.

Code refactoring patterns

Forms

Forms that expose these country/region context fields need to be updated, because we have moved the country/region context fields to a new table. The child table only applies when the application is in the country/region context. We want to introduce a new data source for the child table, and we want to have a passive outer join. We want it passive, because in most execution contexts, the child table does not apply and we don't need to perform the outer join. We use an outer join, because these fields are assumed to be optionally available even in the valid country/region context.

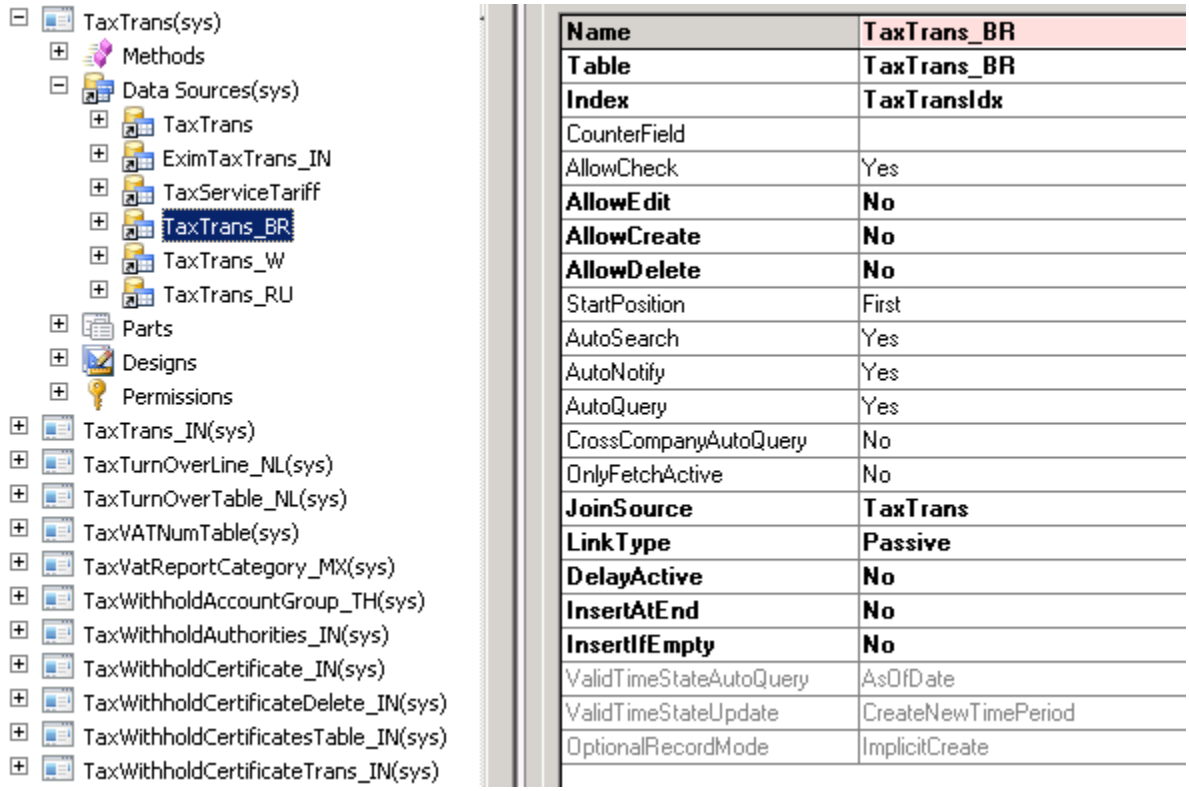
In the form init method, before the super call, determine whether the child table is in the context

```
// <GBR>
isBR = SysCountryRegionCode::isLegalEntityInCountryRegion([#isoBR]);
// </GBR>
```

Add a new data source for the child table

1. Define the basic metadata, just as for the parent table data source.
2. Define the index to the parent table.

3. Define the link type as passive.



Name	TaxTrans_BR
Table	**TaxTrans_BR**
Index	**TaxTransIdx**
CounterField	
AllowCheck	Yes
AllowEdit	**No**
AllowCreate	**No**
AllowDelete	**No**
StartPosition	First
AutoSearch	Yes
AutoNotify	Yes
AutoQuery	Yes
CrossCompanyAutoQuery	No
OnlyFetchActive	No
JoinSource	**TaxTrans**
LinkType	**Passive**
DelayActive	**No**
InsertAtEnd	**No**
InsertIfEmpty	**No**
ValidTimeStateAutoQuery	AsOfDate
ValidTimeStateUpdate	CreateNewTimePeriod
OptionalRecordMode	ImplicitCreate

4. In the child data source, add the **init** method to dynamically change the link type to outer join when it is in the country/region context.

```
public void init()
{
    if (isBR)
    {
        this.linkType(FormLinkType::OuterJoin);
    }

    super();
}
```

5. For each affected control, change the data source to point to the new child data source.
6. Some controls will point to display methods on the parent form data source or the parent table. We recommend that you just change the method implementation to access the child table fields.

```
public display PlExciseAmount displayExciseAmount_PL(TaxTrans _taxTrans)
{
    changeDisplaySign = this.doChangeDisplaySign(_taxTrans);

    return _taxTrans.taxTrans_W().ExciseAmount_PL * this.taxChangeDisplaySign(_taxTrans.TaxDirection, _taxTrans.ReverseChargeApplies_UK);
}
```

Code

We only want to access the child table in the country/region context. An **if** statement that ensures that the country/region context is valid should exist around all child table interaction code.

```
if (SysCountryRegionCode::isLegalEntityInCountryRegion([ #isoPL]) && TaxTable::find(this.TaxCode).TaxType_W == TaxType_W::Excise)
{
    taxTrans_W.ExciseAmount_PL = Currency::amount(TaxData::find(this.TaxCode,
                                                                this.TransDate,
                                                                this.TaxBaseAmount).ExciseDuty_PL / 100 * this.TaxBaseAmount);
    this.packTaxTrans_W(taxTrans_W);
}
else
{
    taxTrans_W.ExciseAmount_PL = 0;
}
```

The parent table, with the help the table extension framework, keeps a copy of the child table in memory. When we want to get values from the child table, we ask the parent table for the child table. When we want to update values on the child table, we ask the parent table for the child table, set the values, and then give the child table back to the parent table. When the parent table is created or updated, the extension framework will invoke the appropriate methods on the child table. Here is how the application code will get a child table and update fields on a child table.

Get the child table

To get the child table, we invoke a method on the parent table that includes the name of the child table. The framework will pull the table from memory or retrieve it from the database, and keep it in memory.

```
taxTrans_W = this.taxTrans_W();
```

Set fields on the child table

To update values on the child table, we need to get the table from the parent, set fields on the child table, and finally pack the child table onto the parent table. It is important to ensure that all set values are packed onto the parent table before any methods are invoked in which the parent buffer is passed.

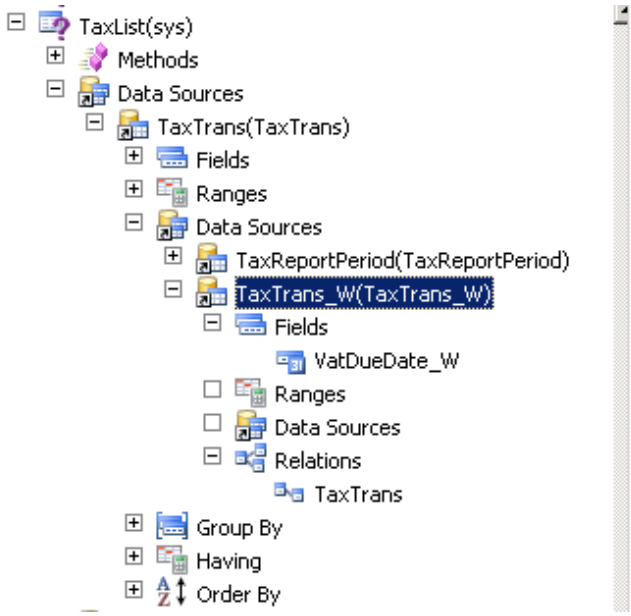
```
taxTrans_W = this.taxTrans_W();
taxTrans_W.TaxPeriodPaymentCode_PL          = _tmpTaxWorkTrans.TaxPeriodPaymentCode_PL;
taxTrans_W.CustVendName_PL                  = _tmpTaxWorkTrans.CustVendName_PL;
taxTrans_W.Addressing_PL                    = _tmpTaxWorkTrans.Addressing_PL;
taxTrans_W.vatNum_PL                        = _tmpTaxWorkTrans.vatNum_PL;
taxTrans_W.DocumentDate_PL                  = _tmpTaxWorkTrans.DocumentDate_PL;
this.packTaxTrans_W(taxTrans_W);
```

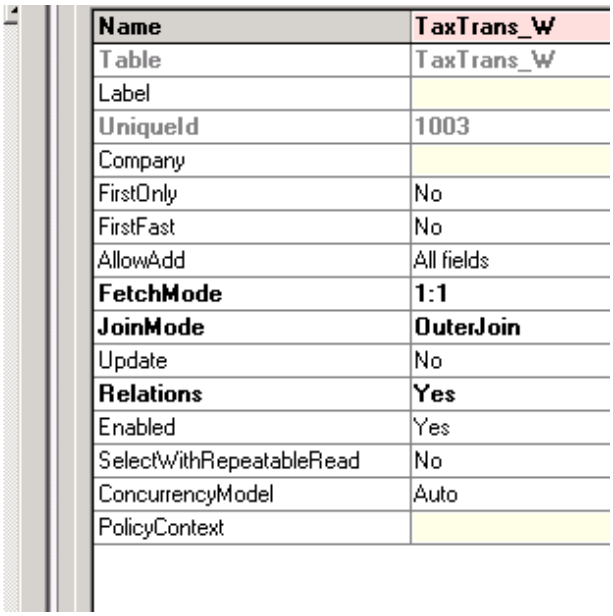
Modeled query definitions

We have modeled queries in the application that expose these country/region context fields.

Data source

1. Add a child data source for the child table, and join it with an outer join to the parent table.
2. Copy ranges for the country/region context fields from the parent data source to the child data source.





Name	TaxTrans_W
Table	TaxTrans_W
Label	
UniqueId	1003
Company	
FirstOnly	No
FirstFast	No
AllowAdd	All fields
FetchMode	**1:1**
JoinMode	**OuterJoin**
Update	No
Relations	**Yes**
Enabled	Yes
SelectWithRepeatableRead	No
ConcurrencyModel	Auto
PolicyContext	

Query execution

In code, we will often create or adjust queries, and execute them.

Join types

Understanding how to join to the child table is important.

- Inner join

  We use this join when we need fields from the child table, and the child has restrictions.

```
select taxTrans
     where taxTrans.RecId            == taxTransloc.InvoiceRefRecID
     join TaxType_IN, TaxTrans, TaxComponentTable_IN from taxTrans_W
        where taxTrans_W.TaxTrans == taxTrans.RecId
        && taxTrans_W.TaxComponentTable_IN  == custVendTrans_W.TaxComponentTable_IN
        && taxTrans_W.TaxType_IN       == TaxType_IN::ServiceTax;
```

- Exists join

  We use this join when we don't need any fields from the child table, but the child table has restrictions.

```
LedgerTransVoucherLink ledgerTransVoucherLinkRel;
LedgerTransVoucherLink ledgerTransVoucherLink;
TaxTrans               taxTransPaym;
TaxTrans_W             taxTransPaym_W;
TaxItemGroupHeading    taxItemGroupHeadingPayment;
TaxParameters          taxParameters = TaxParameters::find();
Set                    paymentTimeBasis = new Set(Types::String);

select firstonly RecId, Voucher, TransDate, VoucherGroupId from ledgerTransVoucherLink
    where ledgerTransVoucherLink.Voucher         == _paymentVoucher
        && ledgerTransVoucherLink.TransDate      == _paymentDate
        && ledgerTransVoucherLink.VoucherGroupId == ledgerTransVoucherLink.RecId
    exists join ledgerTransVoucherLinkRel
        where ledgerTransVoucherLinkRel.VoucherGroupId  == ledgerTransVoucherLink.RecId
            && ledgerTransVoucherLinkRel.VoucherGroupId != ledgerTransVoucherLinkRel.RecId
    exists join taxTransPaym
        where taxTransPaym.TransDate   == ledgerTransVoucherLinkRel.TransDate
            && taxTransPaym.Voucher    == ledgerTransVoucherLinkRel.Voucher
        exists join taxTransPaym_W
            where taxTransPaym_W.TaxTrans == taxTransPaym.RecId
            && taxTransPaym_W.TaxType_IN == TaxType_IN::ServiceTax
        exists join taxItemGroupHeadingPayment
            where taxItemGroupHeadingPayment.TaxItemGroup == taxTransPaym.TaxItemGroup
                && (taxItemGroupHeadingPayment.ServiceTaxBasis_IN    == ServiceTaxBasis_IN::CashBasis
                    || taxItemGroupHeadingPayment.ServiceTaxBasis_IN == ServiceTaxBasis_IN::POTBasis);
```

- Outer join

  We use this join when we need fields from the child table, but the information in the child table may or may not exist.

```
while select RecId from exportSalesInvoiceDomesticSalesTaxVoucher
        where exportSalesInvoiceDomesticSalesTaxVoucher.ExportSalesInvoice == exportSalesInvoice.RecId &&
            exportSalesInvoiceDomesticSalesTaxVoucher.ReferenceDomesticSalesTaxVoucher == 0 &&
            exportSalesInvoiceDomesticSalesTaxVoucher.IsPosted == NoYes::Yes
    join TaxCode, SourceBaseAmountCur, SourceTaxAmountCur from taxTrans
        where taxTrans.Voucher == exportSalesInvoiceDomesticSalesTaxVoucher.Voucher &&
            taxTrans.TransDate == exportSalesInvoiceDomesticSalesTaxVoucher.PostingDate
    outer join CustVendName_PL, VATNum_PL, Addressing_PL from taxTrans_W
        where taxTrans_W.TaxTrans == taxTrans.RecId
```

Query ranges

Query range values on child table fields are often populated in code. We delegate the creation of the child data source to the parent table method **addDataSource**. We can then easily add ranges and range values.

```
protected void setUpTaxTransQueryDateRange()
{
    QueryBuildDataSource qbds;

    if (fromDate || toDate)
    {
        qbds = TaxTrans::addDataSource(taxTransGeneralJournalAcctEntryQuery.dataSourceTable(tableNum(TaxTrans)), tableNum(TaxTrans_W));
        SysQuery::findOrCreateRange(qbds, fieldnum(TaxTrans_W, VatDueDate_W)).value(SysQuery::range(fromDate, toDate));

        qbds = TaxTrans::addDataSource(taxTransAcctDistributionQuery.dataSourceTable(tableNum(TaxTrans)), tableNum(TaxTrans_W));
        SysQuery::findOrCreateRange(qbds, fieldnum(TaxTrans_W, VatDueDate_W)).value(SysQuery::range(fromDate,toDate));
    }
}
```

Considerations

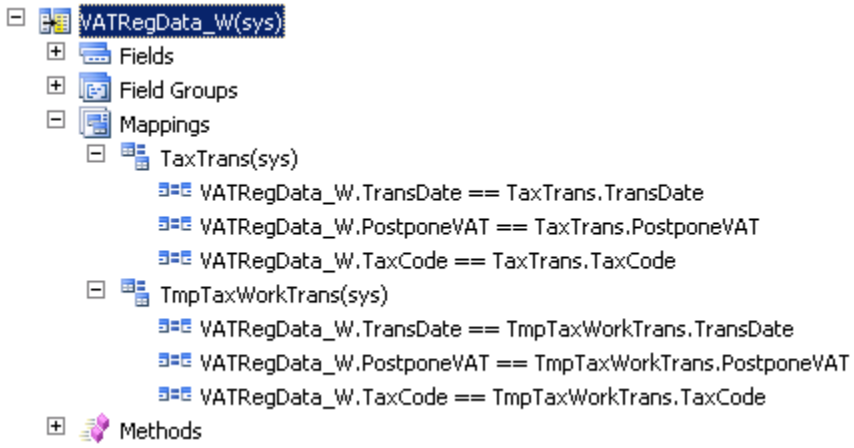
- Query results should return only those fields that are necessary. These partially populated table buffers should only be used in the current method or private methods inside the current class. The parent or child table buffers should never be passed to other methods.
- It is faster to include the child table in the complex query execution, because all the appropriate rows are retrieved from the database.

Maps

When the parent table is contained in a map, the country/region context fields are sometimes included in that mapping. However, because we have moved these country/region context fields to the child table, the map is invalid.

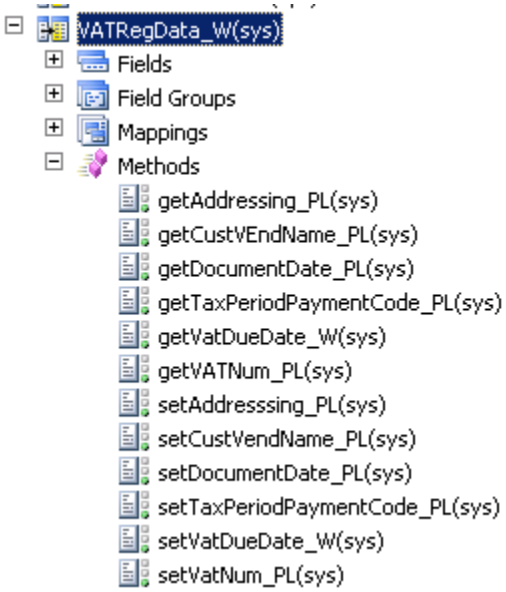
Create methods on the map to encapsulate the getting and setting of mapped fields

1. Remove country/region context fields from the mapping.



2. Add **get** and **set** methods.

   These methods have to explicitly code for each mapped physical table.



Get example

```
public VATNum getVATNum_PL()
{
    VATNum vatNum;
    TaxTrans taxTrans;
    TaxTrans_W taxTrans_W;
    TmpTaxWorkTrans tmpTaxWorkTrans;

    switch (this.TableId)
    {
        case tableNum(TaxTrans):
            taxTrans = this;
            taxTrans_W = taxTrans.taxTrans_W();
            vatNum = taxTrans_W.VATNum_PL;
            break;
        case tableNum(TmpTaxWorkTrans):
            tmpTaxWorkTrans = this;
            vatNum = tmpTaxWorkTrans.VATNum_PL;
            break;
        default:
            break;
    }

    return vatNum;
}
```

Set example

```
public void setVatNum_PL(VATNum _vatNUM)
{
    TaxTrans taxTrans;
    TaxTrans_W taxTrans_W;
    TmpTaxWorkTrans tmpTaxWorkTrans;

    switch (this.TableId)
    {
        case tableNum(TaxTrans):
            taxTrans = this;
            taxTrans_W = taxTrans.taxTrans_W();
            taxTrans_W.VATNum_PL = _vatNUM;
            taxTrans.packTaxTrans_W(taxTrans_W);
            break;
        case tableNum(TmpTaxWorkTrans):
            tmpTaxWorkTrans = this;
            tmpTaxWorkTrans.VATNum_PL = _vatNUM;
            break;
        default:
            break;
    }
}
```

3. Change map references to point to the map methods.

```
_taxTrans.VATRegData_W::setAddresssing_PL(addressing_PL);
_taxTrans.VATRegData_W::setDocumentDate_PL(documentDate_PL);
_taxTrans.VATRegData_W::setCustVendName_PL(custVendName_PL);
_taxTrans.VATRegData_W::setVatNum_PL(vatNum_PL);
_taxTrans.VATRegData_W::setTaxPeriodPaymentCode_PL(taxPeriodPaymentCode_PL);
_taxTrans.VATRegData_W::setVatDueDate_W(vatDueDate_W);
```

Create a new map for the child fields

Instead of moving the mapped fields to methods on the parent map, we can create another map that contains the child tables.

Examples: Maps\CustVendTrans and Maps\CustVendTrans_W

Views

Views are based on queries. The guidelines are similar to those that you use to refactor modeled queries.

Document services

The parent table may be exposed in one or more Application Integration Framework (AIF) document services. We need to find those services, change the query definition, and regenerate the service artifacts.

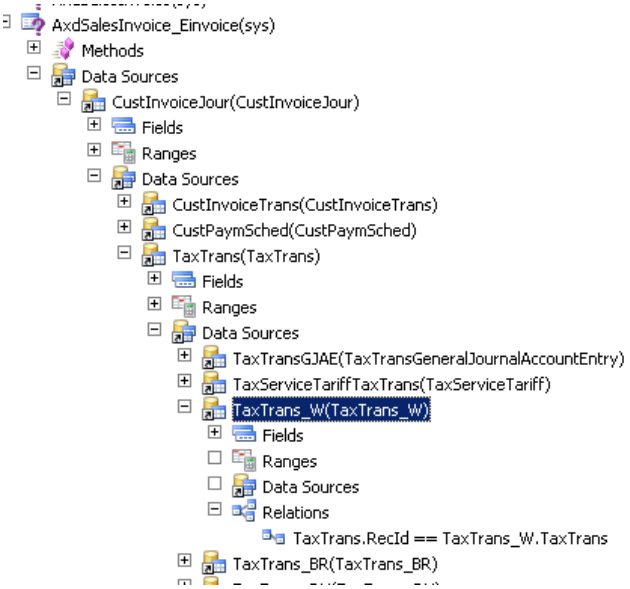
Find the services

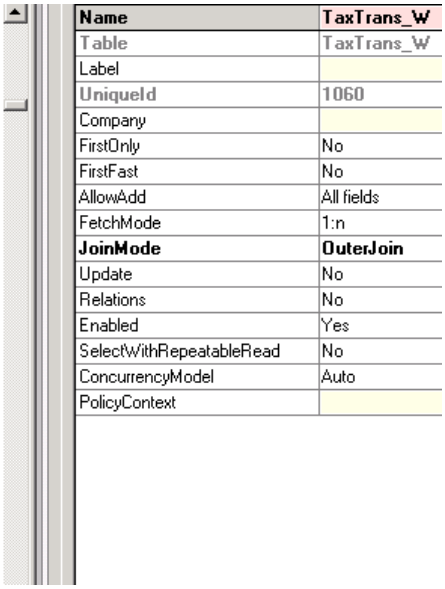
This job will find the services for a given table and field.

```
static void findServices(Args _args)
{
List                    list;
ListEnumerator          listEnumerator;
SysDictClass            sysDictClass;
SysDictClass            serviceDictClass;
AxdWizardParameters     axdWizardParameters;
Query                   axdQuery;
QueryBuildDataSource    qbds;
QueryBuildFieldList     qbfl;
int                     fieldIndex, fieldCount;
TableId                 tableId;
FieldId                 fieldId;

tableId = tableNum(TaxTrans);
fieldId = fieldNum(TaxTrans, VatDueDate_W);

if (tableId != 0)
{
    sysDictClass = new SysDictClass(classnum(AifDocumentService));
    list = sysDictClass.extendedBy();
    listEnumerator = list.getEnumerator();
    while (listEnumerator.moveNext())
    {
        serviceDictClass = new SysDictClass(listEnumerator.current());
        if (serviceDictClass.id() != classNum(AifGenericDocumentService))
        {
            axdWizardParameters =
AifServiceClassGenerator::getServiceParameters(serviceDictClass.name());

            if (axdWizardParameters != null)
            {
                axdQuery = new Query(axdWizardParameters.parmQueryName());
                qbds = axdQuery.dataSourceTable(tableId);
                if (qbds != null)
                {
                    if (fieldId != 0)
                    {
                        qbfl = qbds.fields();
                        fieldCount = qbfl.fieldCount();
                        for (fieldIndex = 1; fieldIndex <= fieldCount; fieldIndex++)
                        {
                            if (qbfl.field(fieldIndex) == fieldId)
                            {
```

```
                                info(strFmt("Service class: %1, query: %2, data source:%3",
serviceDictClass.name(), axdQuery.name(), qbds.name()));
                            }
                        }
                    }
                    else
                    {
                        info(strFmt("Service class: %1, query: %2, data source:%3",
serviceDictClass.name(), axdQuery.name(), qbds.name()));
                    }
                }
            }
        }
    }
}
}
```

Change the query definition to include the child table



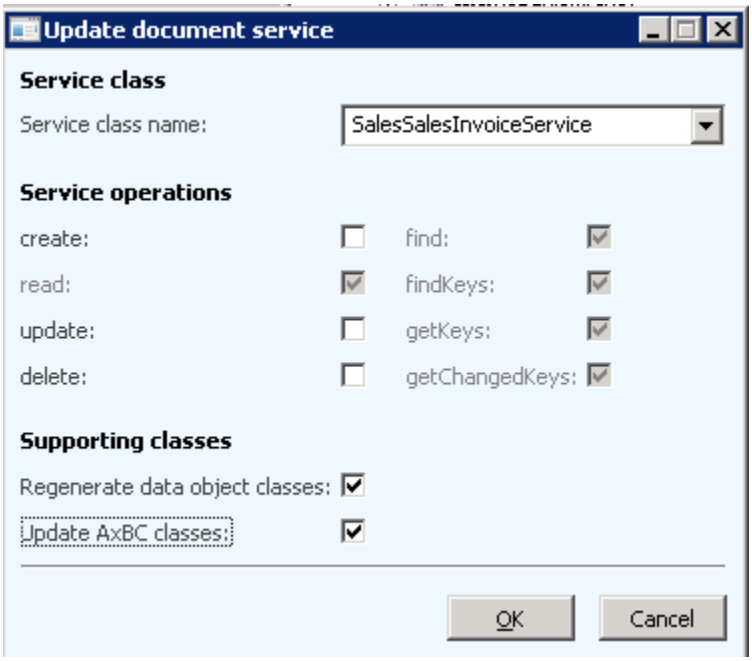


Name	TaxTrans_W
Table	TaxTrans_W
Label	
UniqueId	1060
Company	
FirstOnly	No
FirstFast	No
AllowAdd	All fields
FetchMode	1:n
JoinMode	**OuterJoin**
Update	No
Relations	No
Enabled	Yes
SelectWithRepeatableRead	No
ConcurrencyModel	Auto
PolicyContext	

Regenerate the service artifacts

1. Select the service class name.
2. Select the **Regenerate data object classes** check box.

3. Select the **Update AxBC classes** check box.



Reports

The reports can be classified as two general types.

Remote data processor pattern

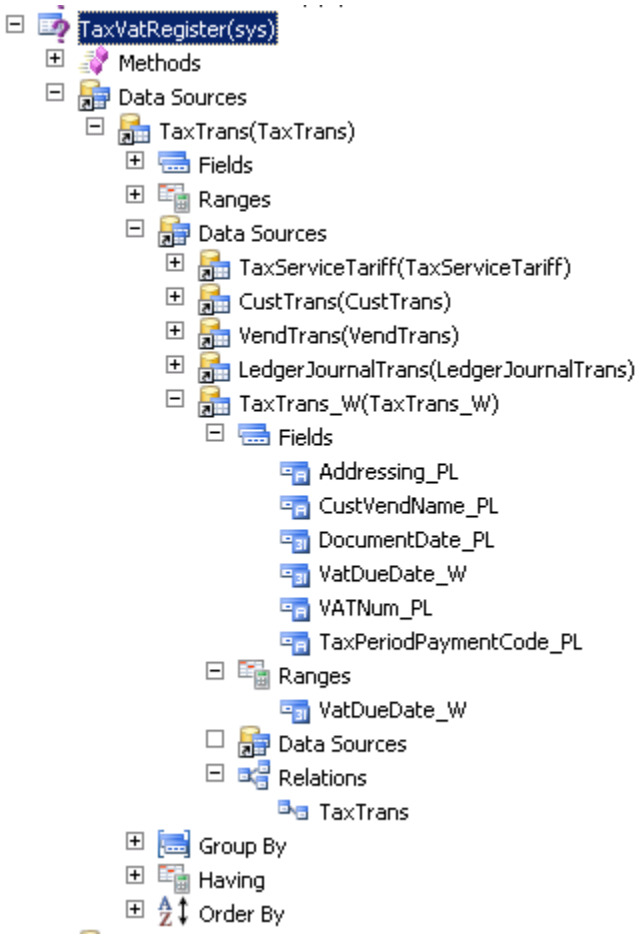
The remote data processor (RDP) is a class that is invoked to gather the data for reports. Typically, the data for these reports is gathered from many different tables and put into a single temporary table. This temporary table is used by a report to surface the information. Generally, the RDP class just needs to be modified to get the information into the temporary table.

Example: Classes\TaxListDP.processReport

Query pattern

1. Update the query.

   The report data source defines a query and fields that are executed. The query needs to be modified to include the child table, appropriate context fields, ranges, and relations.

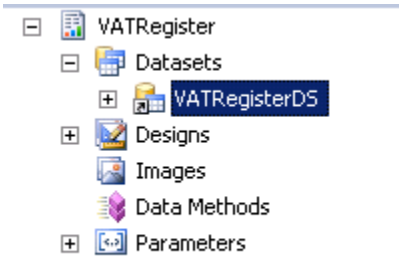


2. Update the report.

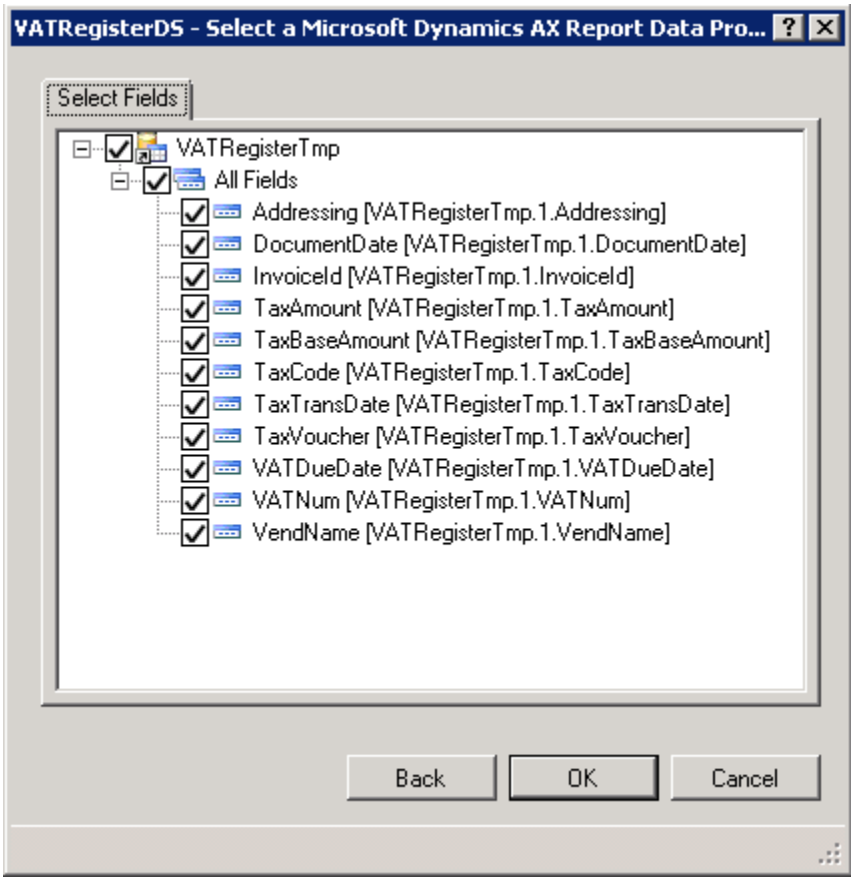
   Because the report controls point to a flattened data set table, as long as the fields retain the same names between the parent table and the child table, the actual report controls and expressions should not need to be changed. We only need to fix the report data set.

3. Refresh the data set.

   Right-click the data set, and then click **Refresh**.



4. Select the fields from the child table.



Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics