



Microsoft Dynamics® AX 2012

Implementing the address book framework for Microsoft Dynamics AX 2012 applications

White Paper

The address book framework in Microsoft Dynamics AX 2012 allows information to be shared across Microsoft Dynamics AX companies and entities through a central repository of users and organizations. It has been enhanced to facilitate easier sharing.

<http://microsoft.com/dynamics/ax>

Date: October 2011

Authors:

Marvs Adriano, Developer, CRM/Address Book
Chris Mossell, Developer, CRM/Address Book
Gaurav Gargate, Developer, CRM/Address Book

Table of Contents

| | |
|--|-----------|
| Overview | 4 |
| Terminology..... | 4 |
| Document purpose..... | 4 |
| Sections to read if you are developing new code..... | 5 |
| Sections to read if you are performing code and data upgrade | 5 |
| Patterns | 5 |
| Party..... | 5 |
| Cross-company sharing | 5 |
| Normalization..... | 6 |
| Date-effective name | 6 |
| Extensible data security | 6 |
| Postal address and contact information | 7 |
| Cross-company sharing | 7 |
| Normalization..... | 7 |
| Effective-date record..... | 7 |
| Using XDS to enforce location privacy | 8 |
| Changes to the party data model and extended data types | 8 |
| Updating code related to party | 8 |
| Party as a reference field that uses a lookup | 8 |
| Creating a new party role | 9 |
| Advanced scenarios | 11 |
| Lookups | 11 |
| Checking for duplicate names..... | 12 |
| Address book control | 13 |
| DirPartyLookup..... | 13 |
| Changes to the postal address data model and extended data types..... | 15 |
| Updating the code related to postal address | 15 |
| Updating the code for a single address | 15 |
| Prerequisites for updating the code for multiple addresses | 17 |
| Updating the code for multiple addresses | 17 |
| Updating code related to transactions..... | 19 |
| Changes to make to the PurchTable table | 19 |
| Changes to make to the PurchTable form | 20 |
| Other changes to make | 20 |
| The DirPartyPostalAddressFormHandler class | 20 |
| The LogisticsLocationEntity class | 21 |
| LogisticsLocationSelectForm..... | 21 |
| Helper classes and methods | 21 |
| DirUtility..... | 21 |
| DirParty | 21 |

| | |
|---|-----------|
| Views created for Microsoft Dynamics AX 2012..... | 23 |
| Enterprise Portal controls..... | 25 |
| Services..... | 26 |
| Data upgrade | 27 |
| Environments..... | 27 |
| Patterns | 27 |
| Transactions in which the address is associated with a party | 27 |
| Transactions in which the address is <i>not</i> associated with a party | 27 |
| Microsoft Dynamics AX 4.0 entity to party upgrade | 27 |
| Microsoft Dynamics AX 4.0 party foreign key | 28 |
| Microsoft Dynamics AX 2009 party upgrade | 28 |
| Microsoft Dynamics AX 2009 party RecId as foreign key | 28 |
| Helper classes | 29 |
| LogisticsElectronicAddressHelper | 29 |
| DirUpgrade | 29 |
| Conclusion..... | 29 |
| Appendix | 30 |
| Table mapping | 30 |
| Data model diagrams | 30 |
| Updates since initial publication..... | 45 |

Overview

The address book framework in Microsoft Dynamics AX® 2012 has been enhanced to include additional sharing of party records and related tables.

Microsoft Dynamics AX 2009 introduced the address book framework, which allowed information to be shared across Microsoft Dynamics AX companies and entities through a central repository of users and organizations. Each person or organization in the address book is referred to as a *party*. Records stored in the repository about parties are called *party records*. Party records include name, address, contact information, and person/organization data.

In Microsoft Dynamics AX 2012, roles that are associated with party records are referred to as *party roles*. Party roles include customer, vendor, prospect (formerly known as business relations), contact, worker, applicant, competitor, and Human Resources (HR) organization units. An individual party can be associated with one or more party roles in a Microsoft Dynamics AX company. For example, the organization party of A. Datum Corporation can be associated with a customer, prospect, and vendor in Microsoft Dynamics AX company CEE and can be associated with a vendor in Microsoft Dynamics AX company CEU. Benefits of this shared data include:

- Demonstrating how people and organizations have relationships with other areas of the enterprise. The relationship and communication between two organizations changes when one organization has more than one role such as a vendor and customer. There might be special agreements that can be negotiated to encourage a closer partnership with the other organization.
- Ease of setup and maintenance. For example, when a change is made to an address, the update only needs to be made in one place; all of the other associated records are updated automatically.

Note: Changes have been made to this paper after it was initially published. For details, see [Updates since initial publication](#).

Terminology

Terms used in this paper include:

| Term | Definition |
|--------------------------------|--|
| Party | A person or organization. A party can be internal or external to an organization. |
| Address book | Group of parties |
| Party roles | Entities that refer to customers, vendors, competitor, worker, applicant, and prospect. |
| Non-party entities | Refers to inventory, bank group, bank accounts, etc. |
| Location | Refers to either a postal address or electronic address (contact information like phone, fax, URL, email, telex) |
| Extensible data security (XDS) | Provides enhanced filtering for party and location records, depending on the user's access. |
| Postal address | Refers to a physical address location. |
| Electronic address | Refers to electronic contact information. |
| Electronic address method type | Consists of phone, fax, URL, email, and telex. |

Document purpose

This document highlights the new patterns used to consume or uptake address book controls, including party name/details and postal and electronic addresses. When detailing the new patterns, the document

also describes what existing pattern is being replaced and how developers should approach updating their legacy code.

Sections to read if you are developing new code

Developers developing new code for Microsoft Dynamics AX 2012 that references party name and postal and electronic addresses should read the following sections of this document:

- [Patterns](#)
- [Data model diagrams](#)
- [Enterprise Portal controls](#)

Sections to read if you are performing code and data upgrade

Developers needing to do a code upgrade for existing applications should first attempt to identify all references to the defined code patterns and then follow the instructions in the related sections to upgrade their code. The code upgrade can be done in any sequence, but in the end the following steps are required:

- Identify the pattern your code uses today.
- Add new fields in data model to represent new foreign keys to party.
- Delete the old foreign key fields.
- Create a data upgrade script to populate the new fields from the old. There is a specific data upgrade section at the end of this document.
- Update the user interface to use the new control appropriate for the pattern defined. The new controls will leverage the new foreign keys that you added to your data model.
- Update the references and business logic in your X++ classes and table methods to use the new code patterns defined in the defined pattern's section of this document.
- Update existing reports to leverage the new data model, including the specific views created for reporting.

Patterns

This section describes the changes to the sections of Microsoft Dynamics AX that support the address book framework that may force changes to existing application code.

The physical model for the tables can be found in the [Data model diagrams](#) section of this document.

Party

The method of sharing data across companies and the normalization of party-related data has changed significantly in Microsoft Dynamics AX 2012. In addition, support has been added for date-effective names.

Cross-company sharing

In the previous version, the party tables, including DirPartyTable were striped by dataArea, and were shared across companies through the use of virtual tables.

In Microsoft Dynamics AX 2012, we have moved away from the use of dataArea. We do not save party data per company (SaveDataPerCompany = No). Party tables are now shared, which means that the need for setting up virtual tables is eliminated.

Normalization

In the previous version, party-related attributes for party entities were stored in the tables for each entity, and were synchronized to the DirParty Table. These attributes included name, language, organization/person details for vendor, customer, worker, contact person, prospect (business relation), applicant, competitor, company (legal entities), department, and other internal organizations.

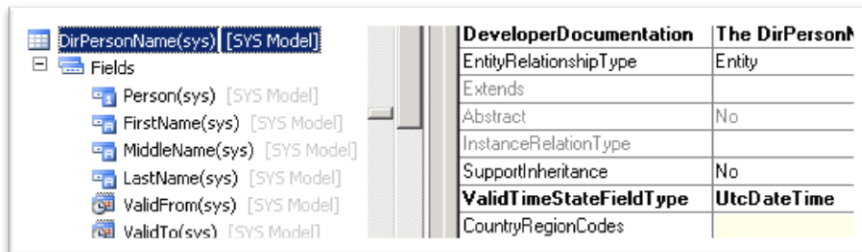
In Microsoft Dynamics AX 2012, these attributes are now stored in a central table, which can be shared across entities. Because of the support for super-type and sub-type in Microsoft Dynamics AX 2012, data is more normalized and organized in a hierarchical structure. There is less (if not no) duplication of fields across tables. Foreign key substitution makes it easier to point to the party reference these entities belong or associated to.

Date-effective name

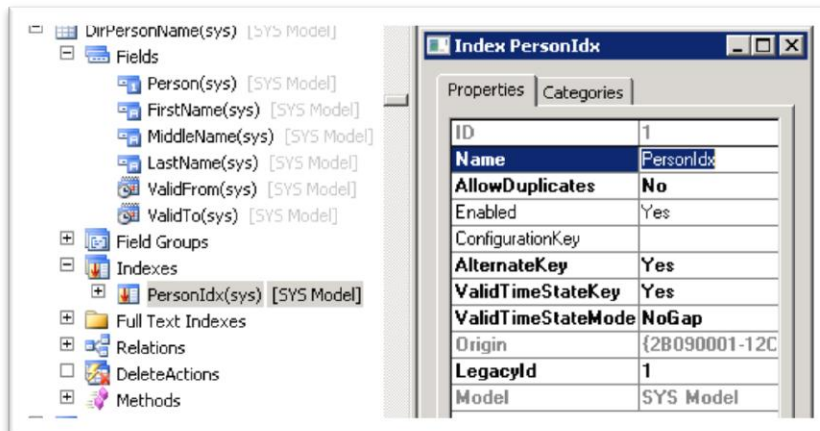
In Microsoft Dynamics AX 2012, names for organization and person are now date-effective. Date-effective names are supported by the use of the ValidTimeStateFieldType property of the table.

The affected tables include:

- DirOrganizationName
- DirPersonName



| DeveloperDocumentation | The DirPersonName |
|--------------------------------|--------------------|
| EntityRelationshipType | Entity |
| Extends | |
| Abstract | No |
| InstanceRelationshipType | |
| SupportInheritance | No |
| ValidTimeStateFieldType | UtcDateTime |
| CountryRegionCodes | |



| Index PersonIdx | |
|--------------------|---------------|
| Properties | Categories |
| ID | 1 |
| Name | PersonIdx |
| AllowDuplicates | No |
| Enabled | Yes |
| ConfigurationKey | |
| AlternateKey | Yes |
| ValidTimeStateKey | Yes |
| ValidTimeStateMode | NoGap |
| Origin | {2B090001-12C |
| LegacyId | 1 |
| Model | SYS Model |

Extensible data security

Extensible data security can be used to restrict the parties that can be viewed or accessed within the system. You can use extensible data security to restrict parties by legal entity or by address book.

Using XDS to restrict parties by legal entity

When parties are restricted by legal entity, a Microsoft Dynamics AX user has access only to parties that are related to legal entities that the user has access to. The restriction is put in place by creating policies for the party entities (customer, vendor, worker, prospect, and so on). There are two policies in this group, which must be used together to meet the full requirement of this data restriction:

- DirRestrictViewPartyInLegalEntity
- DirRestrictViewPartyTableInLegalEntity

Using XDS to restrict parties by address book

When parties are restricted by address book, a Microsoft Dynamics AX user has access only to parties in the address books that the user has access to because of the teams that the user belongs to. There are four policies in this group, which must be used together to meet the full requirement of this data restriction:

- DirRestrictViewPartyInAddressBook
- DirRestrictViewPartyInAddressBook_Cust
- DirRestrictViewAddressBook
- DirRestrictPartyTableInAddressBook

Postal address and contact information

In Microsoft Dynamics AX 2012, we introduced the concept of *location* to refer to both a physical location (postal address) and electronic address (contact information). Postal address and contact information cannot exist without a location.

The addition of support for location affects cross-company sharing, and normalization. In addition, support has been added for date-effective postal addresses and contact information.

Cross-company sharing

In the previous version, postal address and contact information were stored for each entity table, which were dataArea striped. Additional addresses were saved in the Address table and were also stored per company.

In Microsoft Dynamics AX 2012, the Address table has been deprecated. It has been replaced by the shared table LogisticsPostalAddress. All address fields have been removed from party and non-party related tables. Information from these tables has been moved to the LogisticsPostalAddress table. Contact information fields have been removed from party related tables and moved to LogisticsElectronicAddress shared table. Non-party related contact information fields remain unchanged.

Normalization

In the previous version, primary address and contact information for party entities were stored in the table for each entity (vendor, customer, worker, contact person, prospect (business relation), applicant, competitor, external and internal organizations).

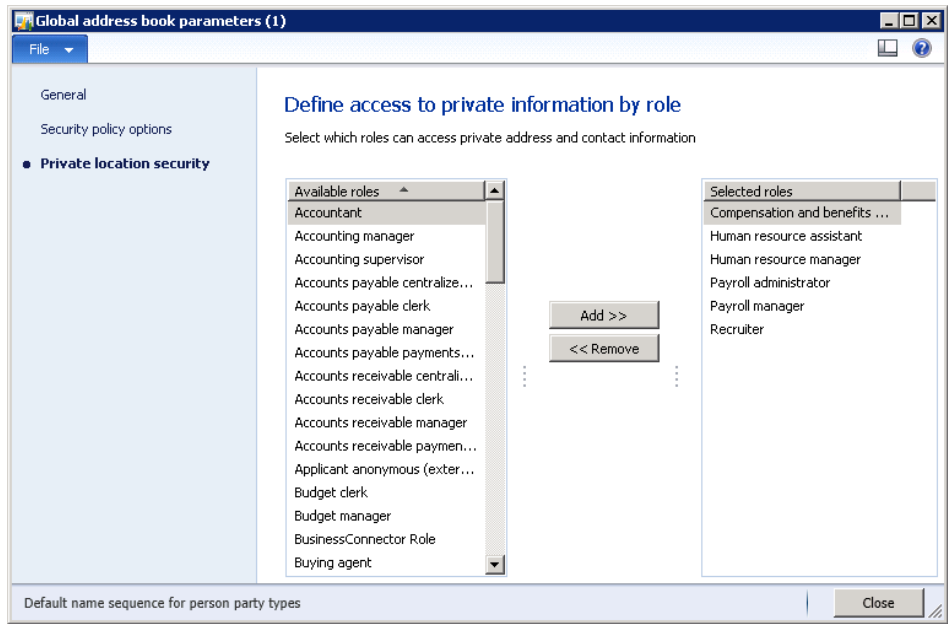
In Microsoft Dynamics AX 2012, the primary address and contact information is now stored in the shared tables LogisticsPostalAddress and LogisticsElectronicAddress. Unlimited numbers of records can be created for postal address and contact information using a relationship table. For example, DirPartyTable uses DirPartyLocation to associate party record to locations.

Effective-date record

In Microsoft Dynamics AX 2012, postal address and contact information are effective-date-enabled. This allows creation of future effective records and keeps track of changes to these records.

Using XDS to enforce location privacy

The LogisticsLocationPrivacy XDS policy is used to restrict the locations that a Microsoft Dynamics AX user can access. This policy is enabled out of the box, because it is integrated into the address book infrastructure through parameters.



Changes to the party data model and extended data types

The following table lists the previous and new extended data types (EDTs) and fields related to parties.

| | Previous | Microsoft Dynamics AX 2012 |
|--------|------------|--|
| EDTs | DirPartyId | DirPartyRecID DirPersonRecID |
| Fields | PartyId | Party. Party is a foreign key to the DirPartyTable. Person. Person is a foreign key to the DirPerson table. |

Updating code related to party

This section describes patterns that you can use when updating code related to parties.

Party as a reference field that uses a lookup

This section describes the changes that need to be made to use party as a reference to find other parties that are in the system.

Note: It is a best practice and beneficial for performance to use the concrete type table (DirPerson, CompanyInfo, OMOperatingUnit, etc.) when possible, and to avoid using the super type DirPartyTable.

Replacing the DirPartyId extended data type with DirPartyRecId (or DirPersonRecId) requires the following changes throughout the AOT.

1. If the DirPartyRecId EDT does not fit the needs of your application, nor does an EDT that extends DirPartyRecId, create an EDT based on DirPartyRecId and customize the label and help text. Use DirPersonRecId for person type records.
2. Add a new foreign key relationship on your table to one of the DirPartyTable concrete tables, such as DirPerson, DirOrganization, or CompanyInfo, which will add a field and index. Validate the relationship properties in the AOT and set according to your applications need.

Determine whether the auto-created index is needed on your table. If it is not necessary, it may be removed.

3. Go to the new party-related field that was added when the relationship was created and complete these tasks:
 - a. Change the value in the Name property to whatever name you'd like and set its ExtendedDataType property to the EDT created/chosen in step 1.
 - b. Verify that the **Label**, **HelpText**, **Mandatory**, **AllowEditOnCreate**, **AllowEdit**, and any other properties set on the field being replaced are correctly set by checking their values against those on the old DirPartyId-based field.
 - c. If the Mandatory property on the DirPartyId-based field was set to **Mandatory=Yes**, you should change the old field's Mandatory property to **No**.
 - d. Determine whether the index created when adding the relationship is required and, if not, remove the index.

1. Add the new field to the desired field groups, removing the old DirPartyId-based field.

Note If the table existed in Microsoft Dynamics AX 2009, the old DirPartyId-based field should receive the DEL_ prefix rather than being removed from the table.

2. If the old DirPartyId-based field in your table is contained in an index for your table, you should add the new DirPartyRecId-based field to the index as well.
3. Find all of the UX locations where the old field was used and drag out the new replacement DirPartyRecId-based field (or re-pull the field group) in order for the surrogate field substitution to occur.
4. Optional. Change the **ReplacementFieldGroup** property from the AutoIdentification field group with the field group you desire.
5. Optional. On these forms, implement the **lookupReference** method on the data source for the new field as discussed in the [Party lookup form](#) section of this document.
6. Upgrade data as described in the [Data upgrade](#) section of this document.
7. Add DEL_ prefix to old, DirPartyId-based fields and set the ConfigurationKey to "SysDeletedObjects60".

Creating a new party role

This section describes the changes that need to be made to create a new party in a form without going to the address book.

In this example, we add the student party role and the person party type.

Modifying the table

After you have modeled your student table, add a DirPerson foreign key to it. Take note of the fields that may already be in the DirPerson table, so that these are not duplicated in your new table.

1. Right-click the Relations node of the student table, and then click **New Relation**.
2. Set the following properties.

| Property | Value |
|-------------------------|-------------|
| Table | DirPerson |
| RelatedTableCardinality | ZeroOne |
| Cardinality | ZeroMore |
| RelationshipType | Association |

3. Right-click the newly created relation. Select **New**, select **ForeignKey**, and then click **PrimaryKey based**. This creates a new field in your student table.
4. Go to the newly created field (by default, it is called **DirPerson**), remove the prefix Dir, and change the EDT to **DirPersonRecId**.

Modifying the form

Add the fields from the person-related tables to your form.

1. On the data source node, set **ChangeGroupMode** to **ImplicitInnerOuter**.
2. Add the student table as the main data source.
3. Add the DirPerson table. Set **JoinSource** to the student table and **LinkType** to **InnerJoin**.
4. Add the DirPersonName table. Set **JoinSource** to the DirPerson table and **LinkType** to **InnerJoin**.
5. You are now ready to add the fields you need to your form design. We recommend that you drag from the data source to your design to easily get the ReferenceGroup functionality, and to set the necessary data source and field automatically.

Note: For organization-type roles, use DirOrganization and DirOrganizationName tables instead of DirPerson and DirPersonName tables, respectively.

Advanced scenarios

The following examples describe advanced scenarios for updating code related to parties.

Lookups

The screenshot shows a Dynamics AX lookup dialog for a person. The left pane contains input fields for 'First name', 'Middle name', 'Last name', 'Personal suffix', 'Search name', 'Customer group', and 'Classification group'. Below these is a 'Show more filters' button. A list of search results is displayed, with 'Anna Drozdiewicz' (Party 1177) selected. The right pane shows details for the selected person, including 'Telephone: 45 45 22 43 45', 'E-mail: test@test.test', and 'Primary address: Street name DK-2950 Vedbaek'. The bottom of the dialog has 'General' and 'Roles' tabs, and 'Select' and 'Cancel' buttons.

Person name components do not get auto-lookups. To enable lookup for these fields, do the following:

1. Override the field's **lookup** method on the DirPersonName data source.

```
public void lookup(FormControl _formControl, str _filterStr)
{
    DirUtility::personNameLookup(_formControl, dirPersonName, DirSubNameSequenceType::FirstName,
    dirPerson, element, false);
}
```

2. Repeat step 1 for FirstName, MiddleName, and LastName, changing the DirSubNameSequenceType parameter.

3. Override the **modified** methods for DirPersonName FirstName, MiddleName, and LastName.

```
if (dirPersonName.Person && dirPerson.RecId != dirPersonName.Person)
{
    person = DirPerson::find(dirPersonName.Person);
    dirPerson.data(person);
    dirPerson.selectForUpdate(true);
    dirPerson_ds.setCurrent();
}
student.Person = dirPersonName.Person;
```

Checking for duplicate names

This example detects whether the name being entered already exists, based on an address book parameter flag. The user can either select from an existing record or create a new one. To achieve this, do the following:

1. Create an instance of `DirPartyFormHandler_Entity`, and set the necessary parameters.

```
partyForm = new DirPartyFormHandler_Entity(element);
partyForm.setDatasourceCaller(student_ds);
partyForm.setDatasourceParty(dirPerson_ds);
partyForm.setDatasourcePartyName(dirPersonName_ds);
partyForm.setDatasourcePartySubType(dirPerson_ds);
partyForm.parmPartyType(DirPartyType::Person);
partyForm.manageFields();
```

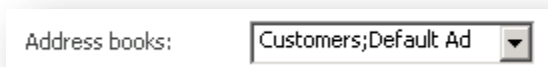
2. Override the **element.selectControl** in your form.

```
public boolean selectControl(FormControl _control)
{
    boolean ret;
    ret = partyForm.selectControl(_control, dirPersonName);
    if (!ret)
        return false;
    ret = super(_control);
    return ret;
}
```

3. Override the `DirPersonName` `FirstName`, `MiddleName` and `LastName` modified methods in your form.

```
if (dirPersonName.Person && dirPerson.RecId != dirPersonName.Person)
{
    person = DirPerson::find(dirPersonName.Person);
    dirPerson.data(person);
    dirPerson.selectForUpdate(true);
    dirPerson_ds.setCurrent();
}
student.Person = dirPersonName.Person;
```

Address book control



If you need to group parties by address book, you must expose the address book control.

1. Add a string control to the form, setting the **AutoDeclaration** property to **Yes**. Name it **addressBooks**.

2. Add the following code to the form's init method after the **super** method.

```
addressBooksCtrl = SysLookupMultiSelectCtrl::construct(element, addressBooks,
queryStr(DirAddressBookLookup));
partyForm = new DirPartyFormHandler_Entity(element);
partyForm.setDatasourceParty(dirPerson_ds);
partyForm.parmAddressBookCtrl(addressBooksCtrl);
```

3. Override the control's **modified** method. Set the forceWrite of the main data source to **true**. This marks the data source as dirty, so that changes can be tracked and saved later on.

```
public boolean modified()
{
    boolean ret;
    ret = super();
    if (ret)
    {
        contactPerson_ds.forceWrite(true);
    }
    return ret;
}
```

4. Add the following code to the **written** method of the main data source.

```
partyForm.writeAddressBookParty();
```

DirPartyLookup

Both the DirPartyRecId or DirPersonRecId extended datatypes automatically inherit the lookup for party or person without any filter.

If there is a need for the lookup to be filtered, use the DirUtility::nameLookup or DirUtility::personNameLookup methods to do so. These methods use the **DirPartyLookupEntitiesFilter** class to filter by the party role. For more information, see the code sample below.

```
public Common lookupReference(FormReferenceControl _formReferenceControl)
{
    Common ret;
    dirPartyLookupEntitiesFilter dirPartyLookupEntitiesFilter;
    dirPartyLookupEntitiesFilter =
dirPartyLookupEntitiesFilter::generateDirPartyLookupEntitiesFilter(
(auditPolicyListParty.AddressBookRole == DirPartyRoleView::All ? NoYes::Yes : NoYes::No),
(auditPolicyListParty.AddressBookRole == DirPartyRoleView::CustomersOnly ? FilterStatus::Set :
FilterStatus::NotSet),
(auditPolicyListParty.AddressBookRole == DirPartyRoleView::VendorsOnly ? FilterStatus::Set :
FilterStatus::NotSet),
```

```

(auditPolicyListParty.AddressBookRole == DirPartyRoleView::BusinessRelationsOnly ? FilterStatus::Set
: FilterStatus::NotSet),
(auditPolicyListParty.AddressBookRole == DirPartyRoleView::CompetitorsOnly ? FilterStatus::Set :
FilterStatus::NotSet),
(auditPolicyListParty.AddressBookRole == DirPartyRoleView::ContactsOnly ? FilterStatus::Set :
FilterStatus::NotSet),
(auditPolicyListParty.AddressBookRole == DirPartyRoleView::EmployeesOnly ? FilterStatus::Set :
FilterStatus::NotSet));
    ret = DirUtility::namelookup(_formReferenceControl, auditPolicyListParty,
auditPolicyListParty.AddressBookRole, element, false, dirPartyLookupEntitiesFilter);
    return ret;
}

```

You can also achieve this behavior without overriding the **lookupReference** method. Add a form method called **getEntityFilter** that returns a **DirPartyLookupEntitiesFilter** type. The DirPartyLookup automatically picks it up and applies the filtering for auto-lookups. See the **DirPartyEntityAssociationUpdate** form, for example.

DirPartyLookupEntitiesFilter class

DirPartyLookup uses this class to filter records that are being shown in the lookup. Normally, you would call the **generateDirPartyLookupEntitiesFilter** method to do this. Below is the list of parameters with their definitions.

| Name | Type | Default |
|------------------------------------|--------------|----------------------------|
| <i>disableAllEntityTypeFilters</i> | NoYes | Yes |
| <i>filterCustomers</i> | FilterStatus | Undefined |
| <i>filterVendors</i> | FilterStatus | Undefined |
| <i>filterBusinessRelations</i> | FilterStatus | Undefined |
| <i>filterCompetitors</i> | FilterStatus | Undefined |
| <i>filterContacts</i> | FilterStatus | Undefined |
| <i>filterEmployees</i> | FilterStatus | Undefined |
| <i>filterProspectiveVendors</i> | FilterStatus | Undefined |
| <i>filterApplicants</i> | FilterStatus | Undefined |
| <i>filterDataAreaId</i> | dataAreaId | SysQuery::valueUnlimited() |

FilterStatus

Set means filter needs to be applied with a flag of true.

NotSet means filter needs to be applied with a flag of false.

Undefined means range should not be added.

Changes to the postal address data model and extended data types

The following table lists the previous and new EDTs and fields related to postal address.

| | Previous | Microsoft Dynamics AX 2012 |
|------------|--------------------------------------|---|
| Table Text | AddressRefTableId AddressRefRecId | LogisticsLocationRecId LogisticsPostalAddressRecId |
| Table Text | AddrRefTableId AddrRefRecId | Location, DeliveryLocation. RecId is a foreign key to the LogisticsLocation table. Postal Address, DeliveryPostalAddress. RecId is a foreign key to LogisticsPostalAddress. |

Updating the code related to postal address

Because creating postal addresses involves large amounts of logic, including validation, formatting, and field references; we recommend that you use the form LogisticsPostalAddress to manage addresses.

Updating the code for a single address

1. Use or extend the LogisticsPostalAddressFormHandler class to implement single address on a form.
2. Create menu items for New, Edit, Clear and Map buttons with the following properties:
 - a. Label = @SYS2055 (New), @SYS2475 (Edit), @SYS2079 (Clear), @SYS136333 (Map)
 - b. ObjectType = Class
 - c. Object = LogisticsPostalAddressFormHandler
 - d. EnumType Parameter = LogisticsLocationAddressActionButtons
 - e. EnumParameter = New, Edit, Clear or Map
 - f. OpenMode = New or Edit (do not set on Clear and Map)
 - g. NormalImage = 11045 (New), 7696 (Edit), 10563 (Clear)
 - h. ImageLocation = EmbeddedResource
3. On the form where the address part will be added do the following:
 - a. Add the form handler class in the class declaration.

```
public class FormRun extends ObjectRun
{
    LogisticsPostalAddressFormHandler addressController; // or the new derived class
}
```

- b. Add a form method called **getAddressController**.

```
public LogisticsPostalAddressFormHandler getAddressController()
{
    return addressController;
}
```

- c. Override the main **datasource active** method.

```
public int active()
{
    int ret;
```

```

    ret = super();
    if (ret)
    {
        addressController.callerActive();
        addressController.callerUpdateButtons(newAddress, editAddress, clearAddress,
mapButton);
    }
    return ret;
}

```

- d. Add LogisticsPostalAddressTable as a datasource and override the **validateWrite** and **write** methods.

```

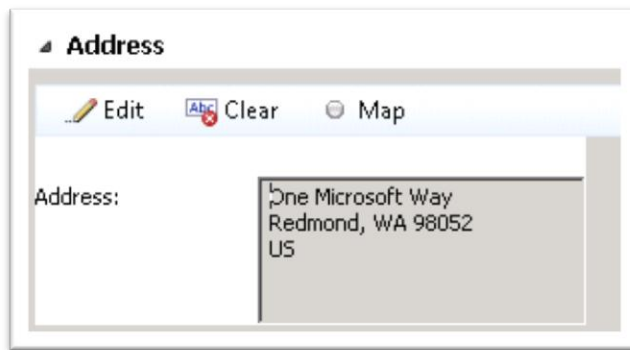
public boolean validateWrite()
{
    return true;
}
public void write()
{
    //super();
}

```

Example forms

The following forms have been updated to use single address logic.

- BankAccountTable
- BankGroup
- IntrastatParameters
- CustBankAccounts
- VendBankAccounts
- HcmWorkerBankAccount



Note: We recommend that you use the Duplicate feature in AOT as well as the drag and drop functionality; this also works for methods.

Prerequisites for updating the code for multiple addresses

1. Create a 1:n association table that links your entity and the postal address. Add a foreign key to the LogisticsLocation table. The following are some examples:
 - DirPartyLocation
 - InventLocationLogisticsLocation
 - InventSiteLogisticsLocation
 - HcmApplicationBasketLocation
2. Add the table association to the LogisticsEntityLocationMap table and set the field mapping. Update the **getEntity*** methods and **entityType2TableId** method.
3. Add the table to the LogisticsEntityLocationUnion query. When you save, it will synchronize the views **LogisticsEntityLocationView** and **LogisticsEntityPostalAddressView**.

Updating the code for multiple addresses

1. Use or extend the **LogisticsEntityPostalAddressFormHandler** class to implement unlimited address association.
2. Create menu items for Add, Edit, Deactivate, Remove, Advanced and Map buttons with the following properties:
 - a. Label = @SYS2318 (Add), @SYS2475 (Edit), @SYS23466 (Advanced), @SYS134283 (Deactivate), @SYS311855 (Remove), @SYS136333 (Map)
 - b. ObjectType = Class
 - c. Object = LogisticsEntityPostalAddressFormHandler
 - d. EnumTypeParameter = LogisticsLocationAddressActionButtons
 - e. EnumParameter = 0 – New, 1 – Edit, 3 – Map, 4 – Delete, 5 – Advanced, 6 – Deactivate
 - f. LinkedPermissionType = Form
 - g. LinkedPermissionObject = LogisticsPostalAddress
 - h. NormalImage = 11421 (Add), 10040 (Edit), 10106 (Advanced), 11438 (Delete), 10041 (Map), 10039 (Deactivate)
 - i. ImageLocation = EmbeddedResource
 - j. OpenMode = New, Edit, Auto (for Advanced, Map, Delete and Deactivate)
3. Add these menu items on an ActionPane in your form.

4. Create a Grid using LogisticsEntityPostalAddressView as the datasource, set as Passive linkType, AutoSearch=No. Do the following:

- a. Add the placeholder variable for the selected/active postal address in the classDeclaration.

```
public class FormRun extends ObjectRun
{
    LogisticsLocationRecId      postalAddressLocation;
}
```

- b. Add a form method called **parmPostalAddressLocation**.

```
public LogisticsLocationRecId parmPostalAddressLocation(LogisticsLocationRecId
_postalAddressLocation = dirPartyPostalAddressView.Location)
{
    postalAddressLocation = _postalAddressLocation;
    return postalAddressLocation;
}
```

- c. Override the main datasource active method.

```
public int active()
{
    int ret;
    ret = super();
    if (ret)
    {
        LogisticsEntityPostalAddressView_ds.executeQuery();
    }
    return ret;
}
```

- d. Add LogisticsEntityPostalAddressView as a datasource and override the **executeQuery** method.

```
public void executeQuery()
{
    LogisticsEntityPostalAddressView::updateQuery(this.query().dataSourceName(this.name()),
    custTable.Party);
    super();
    this.updateButtons();
    if (postalAddressLocation != 0 && postalAddressLocation !=
    logisticsEntityPostalAddressView.Location)
    {
        this.findValue(fieldnum(LogisticsEntityPostalAddressView, Location), SysQuery::value(postal
        AddressLocation));
    }
}
```

Note: For party-based entities, use DirPartyPostalAddressView instead of LogisticsEntityPostalAddressView.

Example forms

The following forms have been updated to use multiple address logic:

- CustTable (Party-based)
- tutorial_Form_GABPrimitive (Party-based)
- HcmApplicantBasket (Party- and non-party based)
- InventLocation (Non-party based)
- VendTable (Party-based)

- HcmWorker (Party-based)

| Name or description | Address | Purpose | Primary |
|---------------------|---|-------------------|---------|
| Primary address. | Vikingelvej 16 DK-3600 Frederiksund | Delivery; Invoice | Yes |
| DIR_000506 | Vikingelvej 16 DK-3600 Frederikssund | Delivery | No |
| DIR_000570 | | Delivery | No |

Updating code related to transactions

Because of the work done to normalize address book entities, the name properties for parties and locations (postal and electronic addresses) can now be centrally managed and referenced. For transactions and journals, you do not need to store copies of an address in the transaction record being created. All that is needed is a reference to the postal address or location in the table that represents the transaction.

The following example describes how to update the address book infrastructure for a purchase order transaction entity (PurchTable) that holds a delivery address.

Changes to make to the PurchTable table

1. Remove the address component fields for the delivery address in the PurchTable by prefixing them with DEL_.
2. Add a **DeliveryPostalAddress** field to the PurchTable to hold the RecId of the LogisticsPostalAddress for the delivery address record of the company (LE) that the purchase order is created for.
3. Upgrade scripts to upgrade from the deleted address fields to the **DeliveryPostalAddress** field must be implemented.
4. Consider adding the method **deliveryLocation** to the PurchTable to return the LogisticsLocation record for the address related to the **DeliveryPostalAddress** field.

```
public LogisticsLocationRecId deliveryLocation()
{
    return LogisticsPostalAddress::getLocation(this.DeliveryPostalAddress);
}
```

Changes to make to the PurchTable form

1. Add the following public method to the **PurchTable** form. This is used by the **DirPartyPostalAddressFormHandler** class provided by the global address book foundation to view and edit the selected location for the purchase order that is selected in the form.

```
public LogisticsLocationRecId parmPostalAddressLocation(LogisticsLocationRecId
_postalAddressLocation = purchTableTable.deliveryLocation())
{
    postalAddressLocation = _postalAddressLocation;
    return postalAddressLocation;
}
```

2. Remove old address data sources, and add referenceDataSource LogisticsPostalAddress under the PurchTable data source.
3. Implement the address lookup on the **DeliveryPostalAddress** field by overriding the **LookupReference** method for the **PurchTable.DeliveryPostalAddress** data source field in the form and calling the **lookupAddress** method in LogisticsLocation.

```
public Common lookupReference(FormReferenceControl _formReferenceControl)
{
    ret = LogisticsLocationSelectionLookup::lookupAddress(.....);
    return ret;
}
```

4. Override the **Modified** method on the **DeliveryPostalAddress** field in the **PurchTable** form data source.

```
public void modified()
{
    super();
    LogisticsLocationEntity::showHideEffectiveAddressMessageBar(element,
    PurchTable.DeliveryPostalAddress);
}
```

5. Override the **rereadReferenceDataSources** method on the **DeliveryPostalAddress** field in the PurchTable data source.

```
public void rereadReferenceDataSources()
{
    super();
    LogisticsLocationEntity::showHideEffectiveAddressMessageBar(element,
    PurchTable.DeliveryPostalAddress);
}
```

6. Override the **ExecuteQuery** method on the **PurchTable** form data source. This allows the form to show addresses, even after they are no longer date effective.

```
public void executeQuery()
{
    logisticsPostalAddress_ds.validTimeStateUpdate(ValidTimeStateUpdate::Correction);
    logisticsPostalAddress_ds.query().validTimeStateDateTimeRange(DateTimeUtil::minValue(),
    DateTimeUtil::maxValue());
    super();
}
```

Other changes to make

Make the following additional changes to incorporate the address book infrastructure to support PurchTable.

The DirPartyPostalAddressFormHandler class

Add cases to the **getTransactionEntity** and **isTransactionCommon** methods as needed.

The LogisticsLocationEntity class

Add the following to the class:

- transactionPostalAddressFieldId
- transactionNewAddressDefaultRole
- relatedLocationRole
- postalAddressTableList
- locationTableList

LogisticsLocationSelectForm

Perform the following actions:

- Create a new class inheriting from LogisticsLocationSelectForm.
- Update the method **LogisticsLocationSelectForm::construct** to create instances of your new class.

Helper classes and methods

This section describes the helper classes and methods that are useful for creating, retrieving, and updating party-related and location-related records.

DirUtility

The methods in the **DirUtility** class include:

- **personNameLookup** – A static method that opens the DirPartyLookup and returns a DirPersonName record. The following is a sample call.

```
public void lookup(FormControl _formControl, str _filterStr)
{
    DirUtility::personNameLookup(_formControl, dirPersonName, DirSubNameSequenceType::FirstName,
    dirPartyTable_DirPerson, element);
}
```

- **nameLookup** – A static method that opens the DirPartyLookup and returns a DirPartyTable record. The following is a sample call.

```
public void lookup(FormControl _formControl, str _filterStr)
{
    DirUtility::namelookup(_formControl, dirPartyTable, DirPartyType::Organization, element);
}
```

DirParty

This class handles most of the operations related to a party record. You can also use the concrete classes for certain roles, such as CustomerEntity, VendorEntity, and ContactPersonEntity. The methods in this class include:

- **constructFromCommon** – A static method that instantiates the **DirParty** class and initializes pertinent information about a party record. This method takes a common table buffer that represents either a party role or the party record itself.
- **constructFromPartyRecId** – Similar to **constructFromCommon**, except that it takes only the DirPartyRecId long value.
- **constructFromPartyNumber** – Similar to **constructFromCommon**, except that it takes only the DirPartyNumber string value.
- **getPrimaryPostalAddressLocation** – Retrieves the primary postal address of a particular party record. This method returns a **LogisticsLocationEntity** class instance.

- **getPrimaryElectronicAddressLocation** – Retrieves the primary electronic address of a particular party record for a particular method type (**Phone, Fax, Telex, E-mail, or URL**). This method returns a **LogisticsLocationEntity** class instance.
- **createOrUpdatePostalAddress** – Creates or updates a postal address record for a party. This method takes a **DirPartyPostalAddressView** parameter.
- **createOrUpdateContactInfo** – Creates or updates an electronic address record for a party. This method takes a **DirPartyContactInfoView** parameter.
- **parm* methods** – These are attributes of party information that can either be retrieved or updated.
- **write** – Commits any changes to the party record information.

Static methods

- **addLocation** – A static method that creates a relationship between a party and a location. This method also creates the roles relationship table.


```
DirParty::addLocation(CustTable::find(ProjFundingSource::findCustAccount(projInvoiceTable.ProjInvoiceProjId).CustAccount).Party,
logisticsLocationHeader.RecId, true, false, false,
[LogisticsLocationRole::findBytype(LogisticsLocationRoleType::Invoice).RecId])
```
- **findPostalAddressByRole** – A static method that finds the contact information of a party of a certain purpose. If no primary record is found, it returns the first record, based on the date of creation. This method returns a **LogisticsPostalAddress** record.
- **findElectronicAddressByRole** – A static method that finds the contact information of a party of a certain type and purpose. If no primary record is found, it returns the first record, based on the date of creation. This method returns a **LogisticsElectronicAddress** record.
- **primaryPostalAddress** – A static method that gets the primary postal address of a party. This method returns a **LogisticsPostalAddress** record.
- **primaryElectronicAddress** – A static method that gets the primary electronic address of a party, based on the method type. This method returns a **LogisticsElectronicAddress** record.

The following code samples demonstrate the use of the **ContactPersonEntity** concrete class.

```
contactPersonEntity contactPersonEntity;
contactPerson contactPerson;
DirPartyTable partyTable = DirPartyTable::findByName('Contoso', DirPartyType::Organization);
DirPartyPostalAddressView postalAddress;
```

```
contactPersonEntity = ContactPersonEntity::construct(contactPerson);
contactPersonEntity.parmFirstName('FirstName');
contactPersonEntity.parmMiddleName('MiddleName');
contactPersonEntity.parmLastName('LastName');
contactPersonEntity.parmAssistantName('AssitantName');
contactPersonEntity.parmBillingInformation('Billing info');
contactPersonEntity.parmCharacter('Character description');
contactPersonEntity.parmComputerNetworkName('Computer network name');
contactPersonEntity.parmContactForParty(partyTable.RecId);
contactPersonEntity.parmContactMemo('Memo');
contactPersonEntity.parmContactPersonId('CP1');
contactPersonEntity.parmLoyalty('Loyalty');
contactPersonEntity.parmMileage('Mileage');
contactPersonEntity.parmOfficeLocation('Office location');
contactPersonEntity.parmOutlookCategories('Outlook categories');
contactPersonEntity.parmProfession('Profession');
contactPersonEntity.parmSensitivity(smmSensitivity::Personal);
```

```

contactPersonEntity.parmSpouse('Spouse');
contactPersonEntity.parmTimeAvailableFrom(1000);
contactPersonEntity.parmTimeAvailableTo(2000);
contactPersonEntity.write();

// Set the address fields
postalAddress.Street = '1 Microsoft Way';
postalAddress.City = 'Redmond';
postalAddress.State = 'WA';
postalAddress.ZipCode = '98052';
postalAddress.CountryRegionId = 'USA';
// Write the address to the appropriate tables.
contactPersonEntity.createOrUpdatePostalAddress(postalAddress);

```

Views created for Microsoft Dynamics AX 2012

Because the table structure has been normalized further, we created views to help developers more easily work with address book data model.

DirPartyNameView: This view contains name and other important fields from DirPartyTable and DirPersonName.

| Field | Source | Comment |
|------------|---------------|---|
| Name | DirPartyTable | |
| Firstname | DirPersonName | |
| MiddleName | DirPersonName | |
| Lastname | DirPersonName | |
| Type | DirPartyTable | Based on InstanceRelationType 1 – Person 2 – Organization 3 – Legal Entity 4 – Team 5 – Operating Unit |

DirPartyListPageView: This view is used for the address book list page to filter by the address book role.

| Field | Source | Comment |
|--------------------|---------------|-------------|
| Name | DirPartyTable | |
| NameAlias | DirPartyTable | |
| KnownAs | DirPartyTable | |
| CompetitorCount | DirPartyView | Aggregation |
| CustCount | DirPartyView | Aggregation |
| VendCount | DirPartyView | Aggregation |
| BusRelCount | DirPartyView | Aggregation |
| ContactPersonCount | DirPartyView | Aggregation |

| | | |
|------------------|-----------------|-------------|
| EmplCount | DirPartyView | Aggregation |
| IsCompetitor | Computed column | |
| IsCustomer | Computed column | |
| IsVendor | Computed column | |
| IsBusRel | Computed column | |
| IsContactPerson | Computed column | |
| IsWorker | Computed column | |
| IsApplicant | Computed column | |
| IsLegalEntity | Computed column | |
| IsOperatingUnit | Computed column | |
| IsCostCenter | Computed column | |
| IsUser | Computed column | |
| IsProspectVendor | Computed column | |
| IsDepartment | Computed column | |

DirPartyPostalAddressView: This view contains party related postal address records.

| Field | Source | Comment |
|--------------------|------------------------|--------------|
| LocationName | LogisticsLocation | |
| Address | LogisticsPostalAddress | |
| StreetNumber | LogisticsPostalAddress | |
| Street | LogisticsPostalAddress | |
| City | LogisticsPostalAddress | |
| DistrictName | LogisticsPostalAddress | |
| ZipCode | LogisticsPostalAddress | |
| State | LogisticsPostalAddress | |
| County | LogisticsPostalAddress | |
| CountryRegionId | LogisticsPostalAddress | |
| PostBox | LogisticsPostalAddress | |
| BuildingCompliment | LogisticsPostalAddress | |
| TimeZone | LogisticsPostalAddress | |
| Longitude | LogisticsPostalAddress | |
| Latitude | LogisticsPostalAddress | |
| ValidFrom | LogisticsPostalAddress | |
| ValidTo | LogisticsPostalAddress | |
| PartyLocation | DirPartyLocation | FK reference |
| Location | LogisticsLocation | FK reference |

| | | |
|---------------------|------------------------|--|
| CountryCurrencyCode | LogisticsPostalAddress | |
| IsPrimary | DirPartyLocation | |
| IsPrivate | DirPartyLocation | |
| IsLocationOwner | DirPartyLocation | |

DirPartyContactInfoView: This view contains party related contact information records.

| Field | Source | Comment |
|-------------------|----------------------------|-------------------------------|
| Name | DirPartyTable | |
| Firstname | DirPersonName | |
| MiddleName | DirPersonName | |
| Lastname | DirPersonName | |
| Type | DirPartyTable | Based on InstanceRelationType |
| LocationName | LogisticsLocation | |
| Locator | LogisticsElectronicAddress | |
| LocatorExtension | LogisticsElectronicAddress | |
| CountryRegionCode | LogisticsElectronicAddress | |
| ValidFrom | LogisticsElectronicAddress | |
| ValidTo | LogisticsElectronicAddress | |
| PartyLocation | DirPartyLocation | FK reference |
| Location | LogisticsLocation | FK reference |

LogisticsPostalAddressView – This view contains all postal address records from LogisticsLocation, LogisticsPostalAddress and LogisticsAddressCountryRegion tables.

LogisticsContactInfoView – This view contains all contact information records from LogisticsLocation and LogisticsElectronicAddress tables.

LogisticsEntityLocationView – This view contains all location records for a party (DirPartyLocation), application basket (HcmApplicationBasketLocation), site (InventSiteLogisticsLocation), and warehouse (InventLocationLogisticsLocation). This can be extended to other 1:n location models.

LogisticsEntityPostalAddressView – This view contains all postal address records related to the entities included in the LogisticsEntityLocationView. This can be extended to other 1:n postal address models.

LogisticsEntityContactInfoView – This view contains all contact information records related to the entities included in the LogisticsEntityLocationView. This can be extended to other 1:n contact information models.

Enterprise Portal controls

Microsoft Dynamics AX 2012 has many reusable controls to facilitate the party and address book operations.

DirPartyNameFields – In the previous version, each form had to individually handle the party operations.

With Microsoft Dynamics AX 2012, this control helps in Create, Read, Update, Delete (CRUD) operations of the party for any party-based entities. Enterprise Portal entities (such as Customer) do not need to handle

party operations and can focus on business logic. This control allows user to quickly add the party creation and modification functionality to any party-based entity form.

DirNameChange – In the previous version, the party would be handled on every entity form.

In Microsoft Dynamics AX 2012, this control is used to change the party name from any entity. This common form can be used from any party-based entity to update the party name. This common place helps update the party name across Microsoft Dynamics AX.

LogisticsAddressGrid – In the previous version, this was done on a separate page.

In Microsoft Dynamics AX 2012, this control handles the postal addresses for a party in a single place. Each party-based entity can use this control to display the list of addresses for the party. This control allows selecting the addresses to edit and delete.

LogisticsManageAddress – In the previous version, this was done on a separate page.

In Microsoft Dynamics AX 2012, this control handles the actual creation and updating of the addresses for a party. This control helps handle the address details and roles assigned to an address.

ContactInfo – In the previous version, this was done on a separate page.

In Microsoft Dynamics AX 2012, this control handles the contact information for a party in a single place. Each party-based entity can use this control to display the list of electronic addresses for the party. This control allows you to select the electronic addresses and edit them in place. This control helps in performing the CRUD operations on the electronic addresses.

Services

This section describes the new functionality supported for Services, including the new XML formats for representing accounts and dimensions.

Microsoft Dynamics AX 2012 has new events and handling mechanism for incoming XML requests that help in handling and managing the relations between the elements efficiently.

The following APIs handle the CRUD operations for party, postal addresses, and electronic addresses.

DirPartyServiceOp – This class helps in handling the part operations for an entity. The public APIs will create and update the party details and link them to respective entities. Such an API abstracts the entire complexity of party-based operation from the party-based entity.

In the previous version, there was no single place to handle the party operations for a service.

DirPartyLocationServiceOp – This class helps in handling the location, postal address and electronic address operations for a party-based entity. The public APIs will create, update, and link the addresses and electronic addresses to a party of an entity. This API abstracts the complexity of the postal address and electronic address CRUD operation and smoothly assigns them to the respective entity.

In the previous version, there was no single place to handle the address and electronic address functionality.

DirTrxLocationServiceOp – This class helps in handling the location and address operations for a transaction entity. The public API handles the CRUD operations for addresses to be assigned to any transaction and line entities.

In the previous version, there was no single place to handle the transaction addresses.

Data upgrade

This section describes the environments, patterns, and helper classes used in a data upgrade.

Environments

Upgrading data can take place in two different locations, the Source environment (Microsoft Dynamics AX 4.0/Microsoft Dynamics AX 2009) or the Target environment (Microsoft Dynamics AX 2012). Common patterns are provided for your use in the Source environment. Please refer to the paper, "How to write data upgrade scripts for Microsoft Dynamics AX 2012," for information regarding the new features in the upgrade.

Patterns

This section describes the patterns associated with upgrading address book data.

Transactions in which the address is associated with a party

Examples of this pattern include the SalesTable and InventTransfer tables.

The preprocessing upgrade code is contained in the classes **ReleaseUpdateTransformDB40_CRM** and **ReleaseUpdateTransformDB50_CRM**.

For each table to be upgraded, we create a shadow table typically prefixed with "Shadow_". This is used to track which records have already been upgraded.

The upgrade script for each table performs a few primary actions. In delta scripts, shadow records that no longer match the original record are deleted. We then process the main table records one at a time. First, we find the party corresponding to the transaction. Then the address is copied into a new LogisticsPostalAddress record by calling `DirUpgrade::addressMapToLogisticsPostalAddress` or by explicitly copying each part of the address. If the address contains data, the address CountryRegionId is validated. If validation passes, `DirDataPopulation_AX6::createPostalAddress` is called to add the address to the party. The information from the address is then saved in the shadow table.

Transactions in which the address is *not* associated with a party

Examples of this pattern include the BankGroup and TaxAuthorityAddress tables.

The preprocessing upgrade code is contained in the classes **ReleaseUpdateTransformDB40_GAB** and **ReleaseUpdateTransformDB50_GAB**.

For each table to be upgraded, we create a shadow table typically prefixed with "Shadow". This is used to track which records have already been upgraded.

For each transaction, the address is copied into a new LogisticsPostalAddress record by calling `DirUpgrade::addressMapToLogisticsPostalAddress` or by explicitly copying each part of the address. The address records are saved by calling `DirUpgrade::createNonPartyAddress`.

Microsoft Dynamics AX 4.0 entity to party upgrade

Examples of this pattern include the CustTable and smmBusRelTable tables.

The preprocessing upgrade code is contained in class **ReleaseUpdateTransformDB40_Basic**.

Parties can have a flexible or fixed type. For example, we know that parties created from EmplTable will always be created with party type Person, but parties from CustTable may be either Person or Organization. To handle this situation, the PartyUpgrade form is displayed after preprocessing, so that users can select the appropriate party type.

Flexible party type upgrade example

For this example, consider the CustTable upgrade. In the live script, we call **`updateParty_CustTableSingleRecord(CustTable)`** for each record in CustTable. This class creates

associated records in DirPartyTable and TmpPartyUpgrade. The address and contact info for the customer are also upgraded. Finally, a shadow table record is created, with the flag PartySubtypesCreated set to false.

After the live scripts, the Upgrade Checklist item **Global address book party upgrade** is required. This **Party Upgrade** form is where the user will specify if the upgraded customer is a person or an organization and the correct name for the entity. (When the user selects person, the name is split into first, middle, and last names.)

Because the **Global address book party upgrade** checklist item is run after the live scripts, the delta scripts must upgrade CustTable records where the PartySubtypesCreated is false and where records have been changed. In addition to the upgrade steps performed by the live script, the delta script calls DirUpgrade::writePartyWithHierarchy to create the Party hierarchy records. If this is a new customer that wasn't processed in the live script and no existing TmpPartyUpgrade record is found, the scripts will call updateParty_CustTableSingleRecord(CustTable) and then throw an error alerting the user that he or she needs to revisit the "Global address book party upgrade" form.

This example is specific to the CustTable upgrade, but the ideas apply equally to all the parties where the party type is chosen by the user.

Fixed party type upgrade example

For this example, consider the EmplTable upgrade. The only significant difference from the flexible party type upgrade is that the party hierarchy can be created during the live script. Only changed records will be updated during the delta scripts.

There are validation scripts used for this upgrade. In Microsoft Dynamics AX 2012, we have added the requirement that all parties must have a name. The **validateEmplTableRecordsWithoutName** script checks all employee records for a name.

The ideas presented for this upgrade script work equally well in all party upgrades where the type is known.

Microsoft Dynamics AX 4.0 party foreign key

Examples of this pattern include the smmTMCallListTable and smmBusRelSectorTable tables. The preprocessing upgrade code is contained in ReleaseUpdateTransformDB40_Basic. For each table to be upgraded, we create a shadow table typically prefixed with "Shadow_". This is used to track which records have already been upgraded.

Microsoft Dynamics AX 2009 party upgrade

Examples of this pattern include the CustTable and smmBusRelTable tables. The preprocessing upgrade code is contained in ReleaseUpdateTransformDB50_Basic.

For each table to be upgraded, we create a shadow table typically prefixed with "Shadow_". This is used to track which records have already been upgraded.

The script looks up the party corresponding with the record, and stores the RecId of the party in the shadow table.

Microsoft Dynamics AX 2009 party RecId as foreign key

Examples of this pattern include the smmMailings and HRPLimitTableRelationship tables. The preprocessing upgrade code is contained in ReleaseUpdateTransformDB50_Basic.

For each table to be upgraded, we create a shadow table typically prefixed with "Shadow_". This is used to track which records have already been upgraded.

The script looks up the party corresponding with the record, and stores the RecId of the party in the shadow table.

Helper classes

This section describes the helper classes that are used in data upgrade.

LogisticsElectronicAddressHelper

Used to easily create contact info for a party. Create a new instance of the class, then set each of the parm* methods appropriately. Then call **createElectronicAddress**, passing in the contact information, to create the contact information for the party.

DirUpgrade

The methods in the **DirUpgrade** class include:

- **addressMapToLogisticsPostalAddress(AddressMap, LogisticsPostalAddress)** – A static void method that copies the address data from the AddressMap to the LogisticsPostalAddress.
- **checkAndInsertWithoutCountryRegion** – A static Boolean method that, if the LogisticsPostalAddress does not have a CountryRegionId, looks for a corresponding record in DEL_AddressWithoutCountryRegion. If the record is found, the original LogisticsPostalAddress is updated with the new CountryRegionId; otherwise, a DEL_AddressWithoutCountryRegion record is created.
- **createNonPartyAddress** – A static void method that creates LogisticsPostalAddress and LogisticsLocation records.
- **DirPartyLocationRole createPartyOwnedAddress** – A static method that creates LogisticsPostalAddress and LogisticsLocation records, and then associates these with a specified party.
- **LogisticsLocationId getNewLocationId** – A static method that returns a new LocationId from the number sequence.
- **DirPartyNumber getNewPartyNumber** – A static method that returns a new PartyNumber from the number sequence.
- **normalizeAddress** – A static void method that updates the LogisticsPostalAddress with the new address data created for the upgraded system.
- **DirPartyTable writePartyWithHierarchy** – A static method that updates or creates a DirPartyTable record, and creates the Person or Organization party hierarchy records based on the values in the TmpPartyUpgrade record.

Conclusion

The address book framework allows information to be shared across Microsoft Dynamics AX companies and entities through a central repository of users and organizations demonstrating how people and organizations have relationships with other areas of the enterprise.

Note: Changes have been made to this paper after it was initially published. For details, see [Updates since initial publication](#).

Appendix

Table mapping

| Microsoft Dynamics AX 2009 Table | Microsoft Dynamics AX 2012 Table |
|--|--|
| DirPartyTable DirOrganizationDetail | DirPartyTable <ul style="list-style-type: none">DirOrganizationBase<ul style="list-style-type: none">DirOrganizationOMInternalOrganization (see data model diagrams for the full hierarchy)DirPerson |
| DirPersonPartyDetail DirPartyInternalOrganizationTable | DirOrganizationName DirPersonName |
| DirPersonSalutationTable DirPersonGenerationalSuffixTable | DirNameAffix |
| DirPrivacyGroupTable | Deprecated. New XDS policy framework will replace privacy implementation. |
| DirPartyAddressRelationship DirPartyAddressRelationshipMapping DirECommunicationRelationship DirECommunicationRelationshipMapping | DirPartyLocation |
| DirECommunicationAddress DirECommunicationTypeTable DirECommunicationTypeTxt | LogisticsElectronicAddress |
| Address | LogisticsPostalAddress |
| AddressCountryRegion AddressCounty AddressFormatHeading AddressFormatLines AddressState AddressZipCode | LogisticsAddressCountryRegion LogisticsAddressCounty LogisticsAddressFormatHeading LogisticsAddressFormatLines LogisticsAddressState LogisticsAddressZipCode |

Data model diagrams

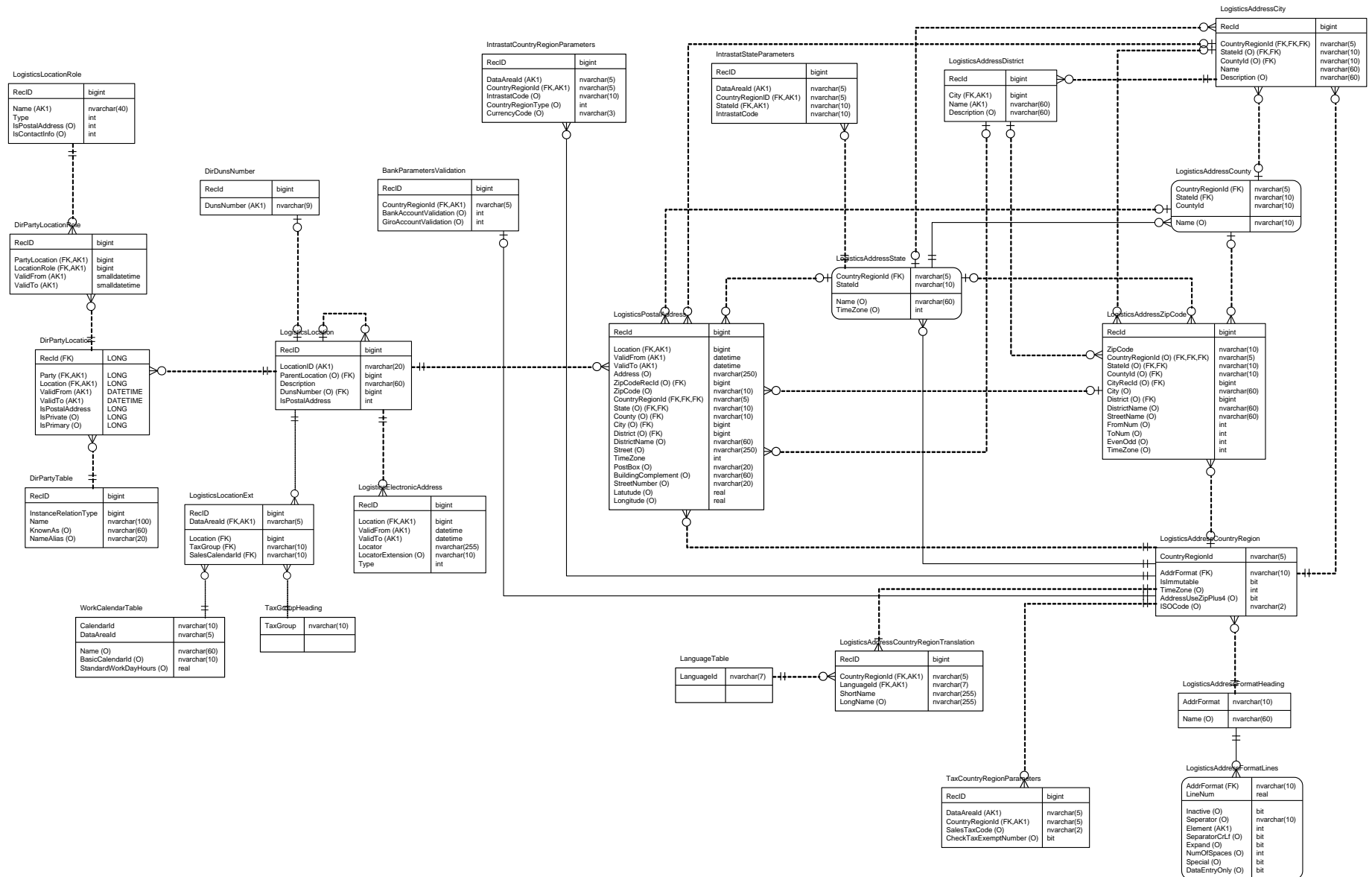
This section includes the following data model diagrams:

- Core (party and address book)
- Location (postal and electronic address)
- Transactions
- Location object model

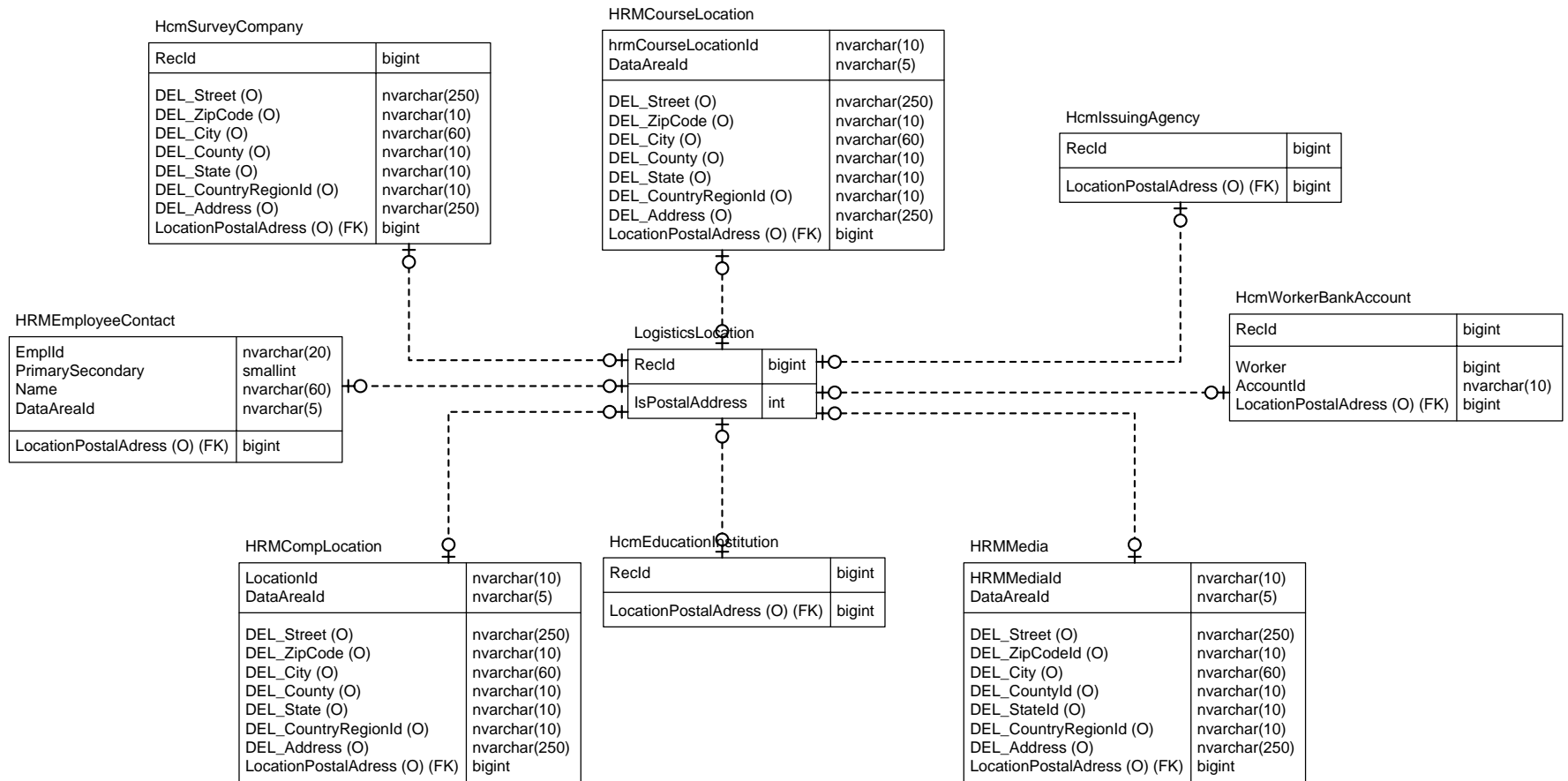
© 2015 Pearson Education, Inc. or its affiliate(s). All rights reserved. Pearson Education, Inc., publishing as Pearson Benjamin Cummings, 101 University Avenue, New York, NY 10017-2423.



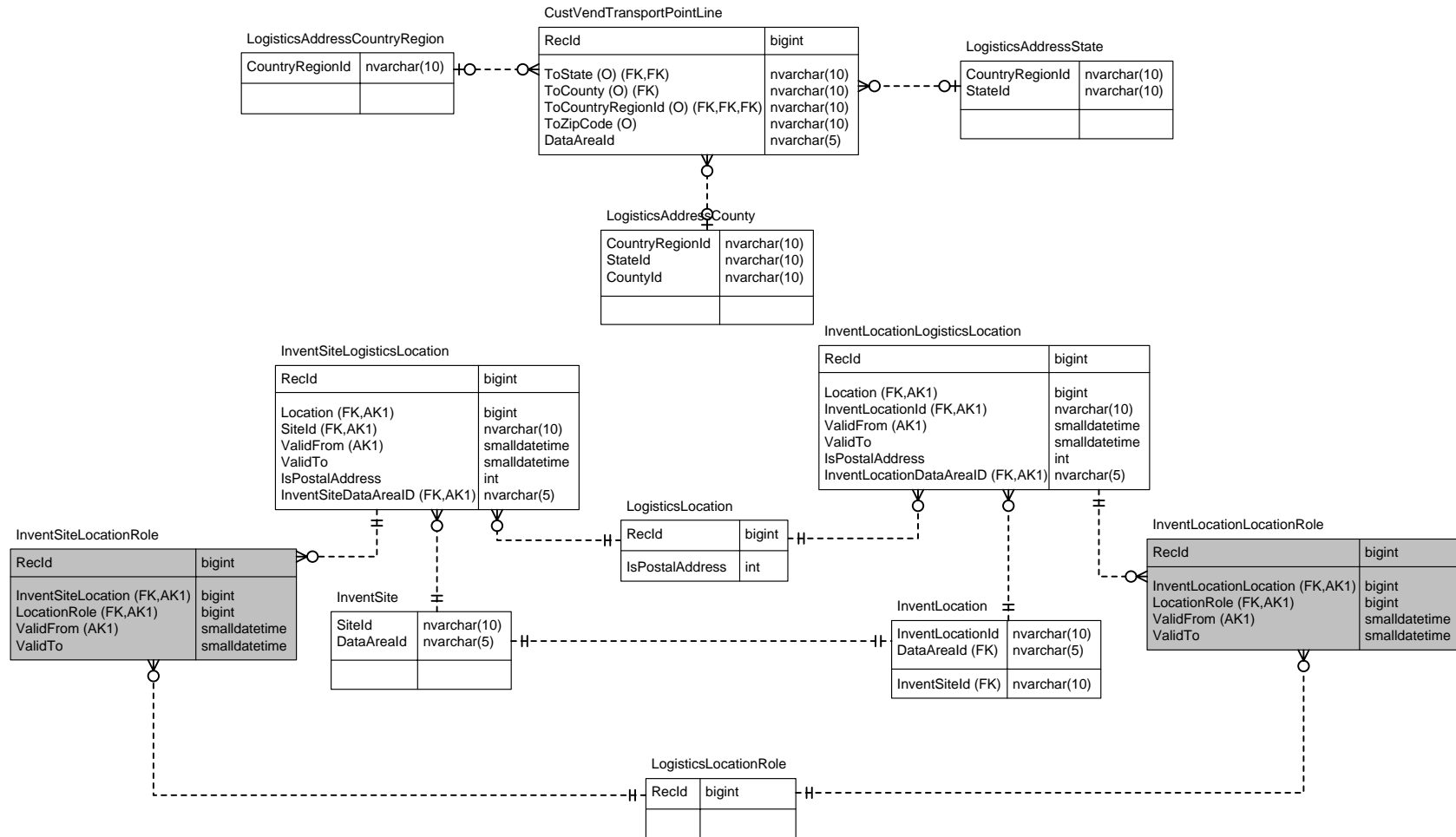
Location (postal and electronic address)



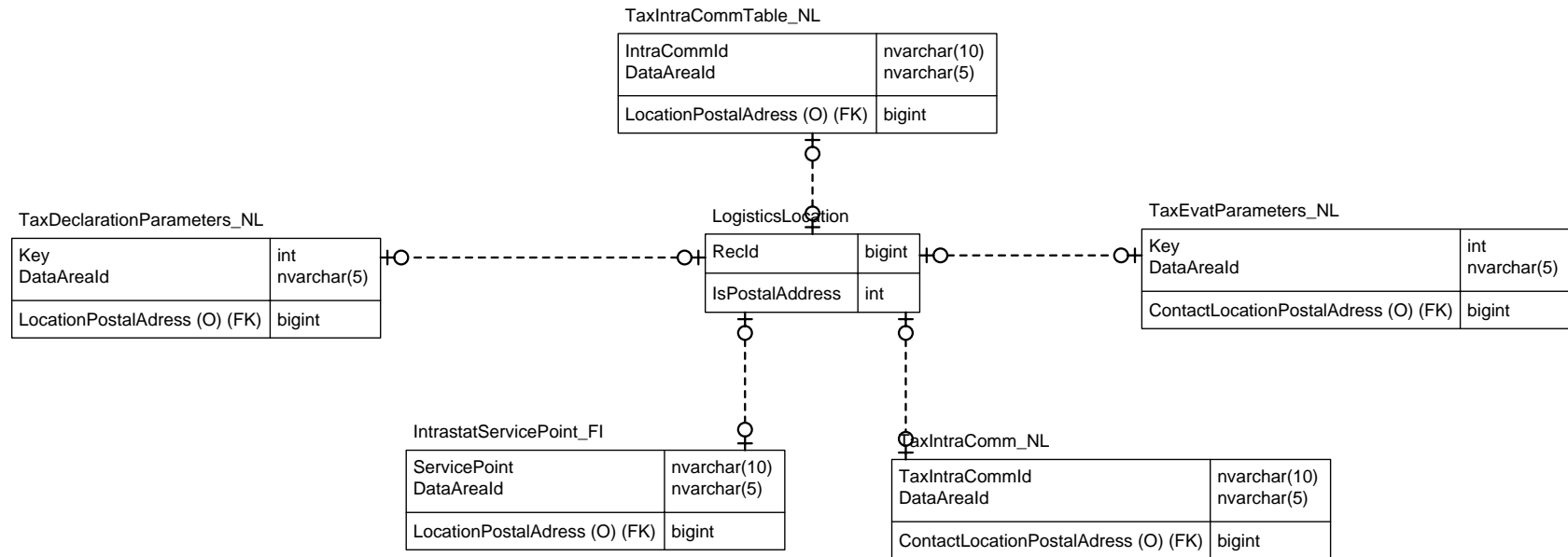
Human resource management



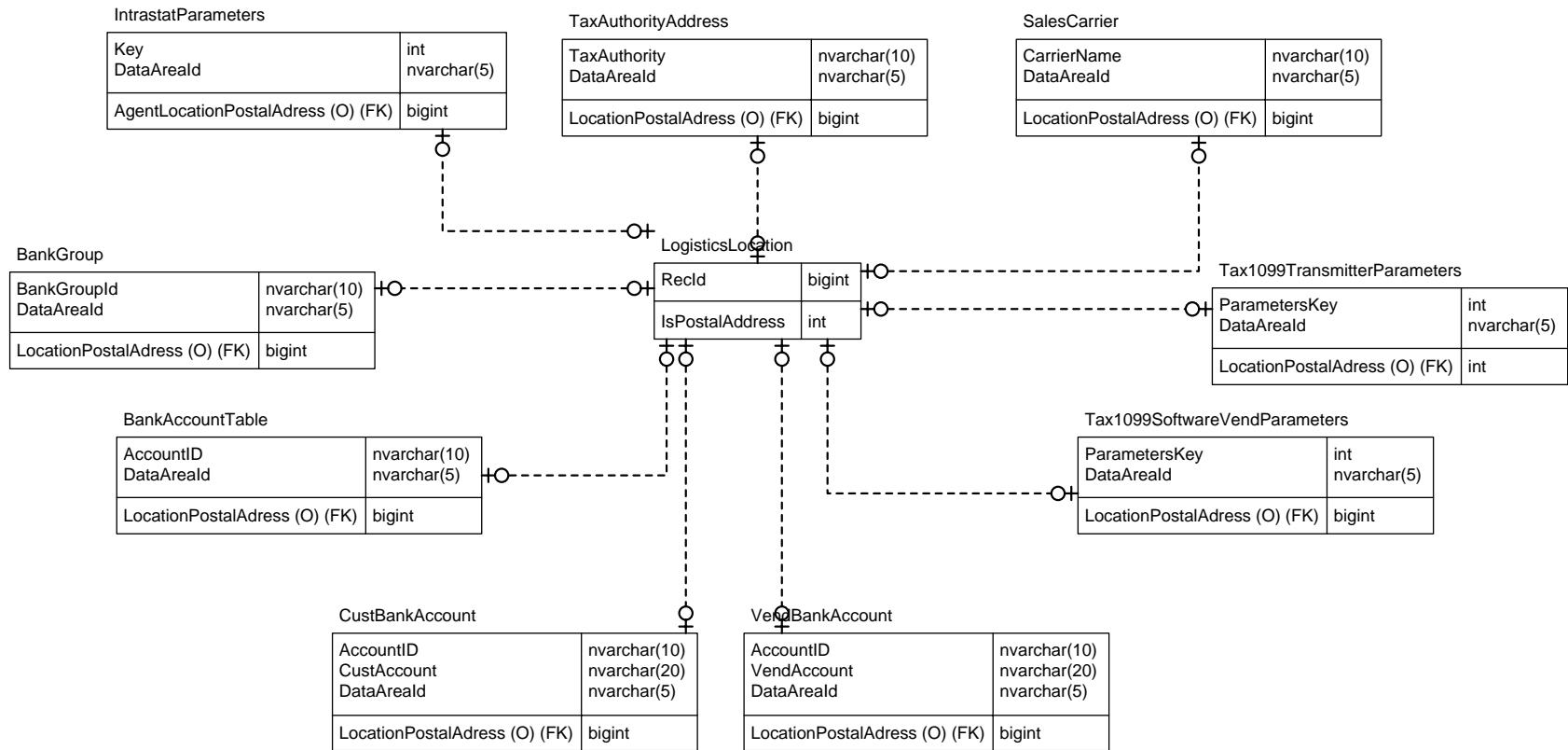
Inventory



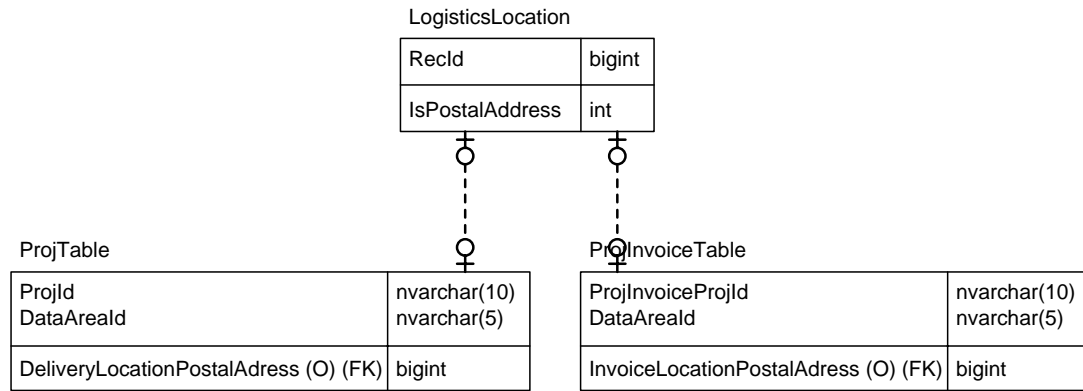
GDL



Financials

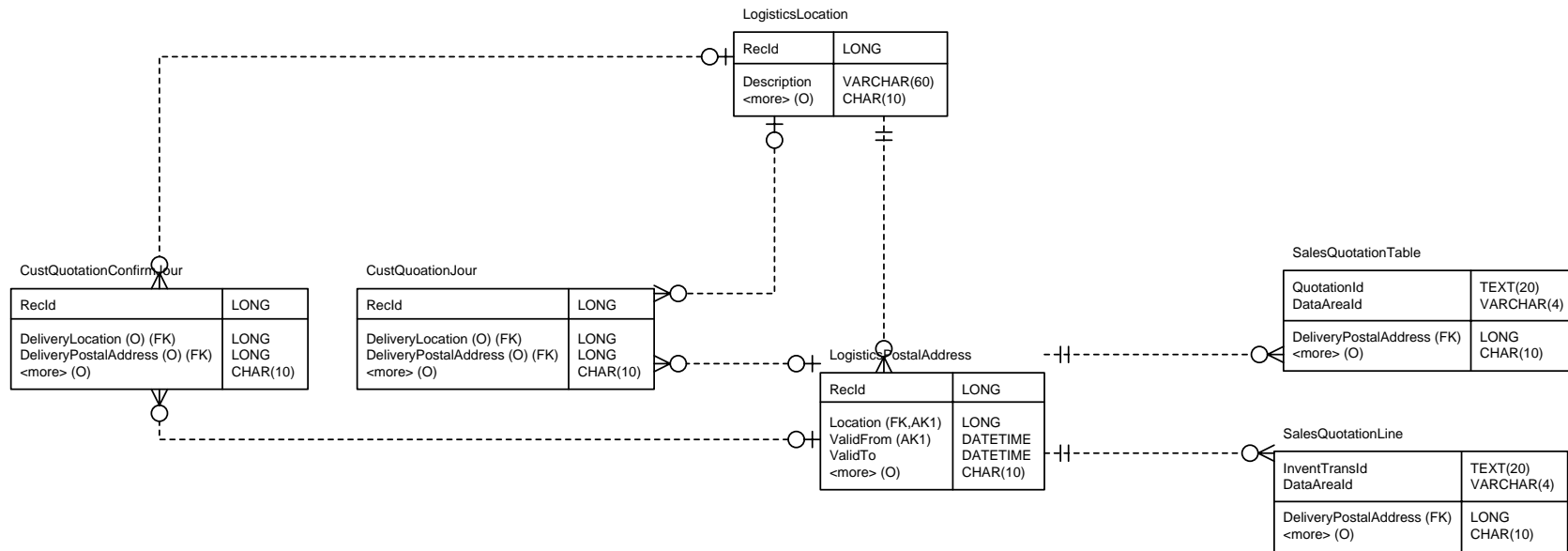


Project accounting



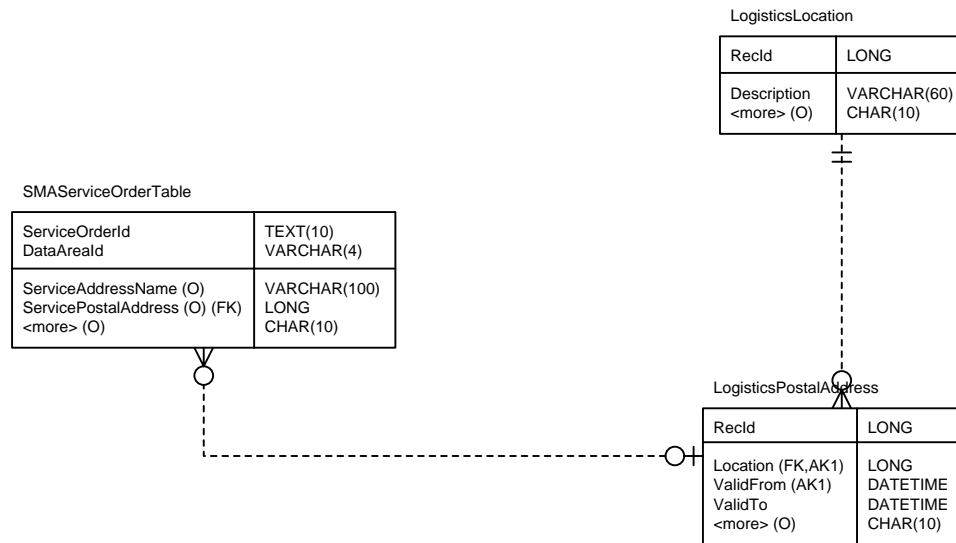
Transactions

Quotation

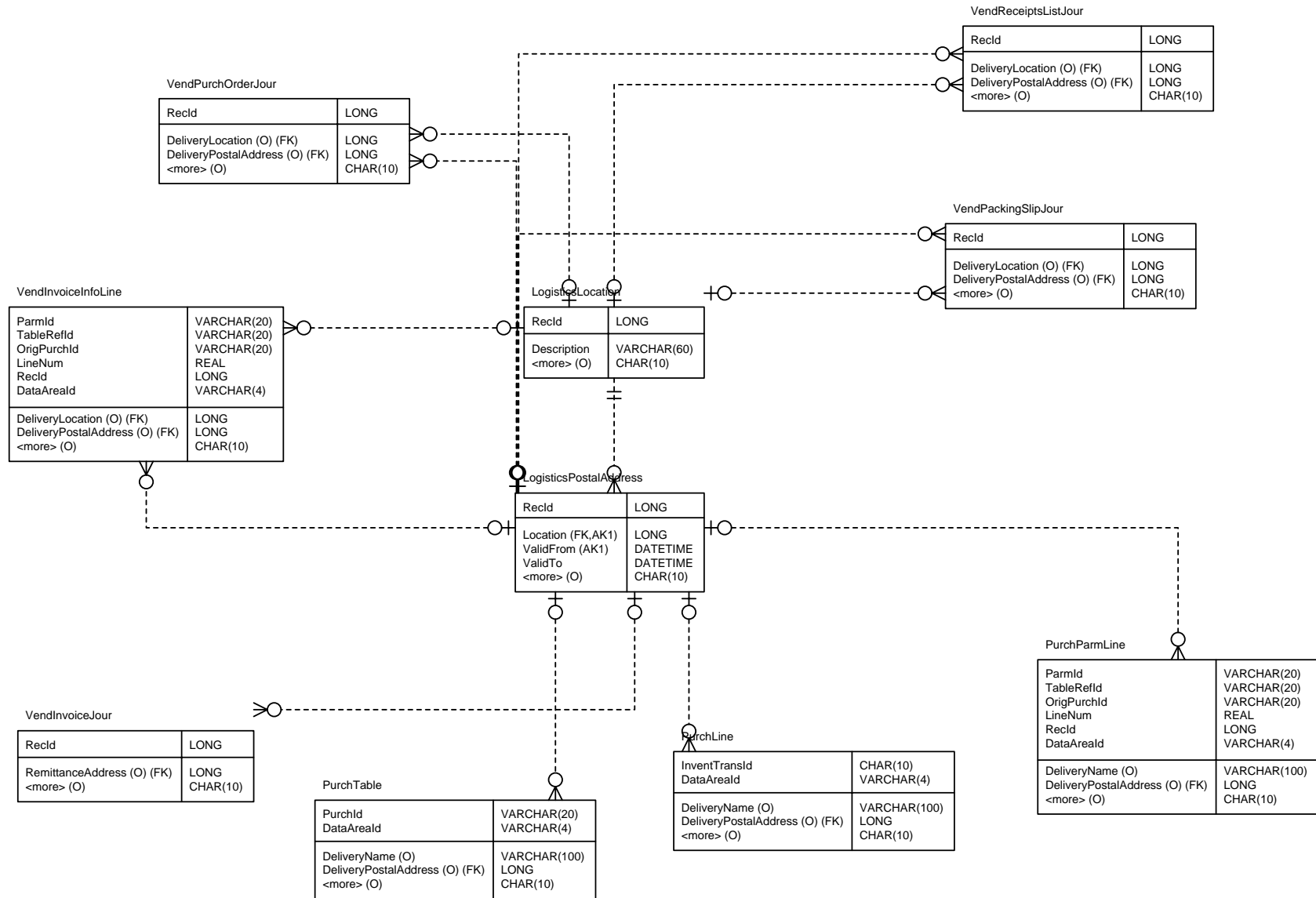




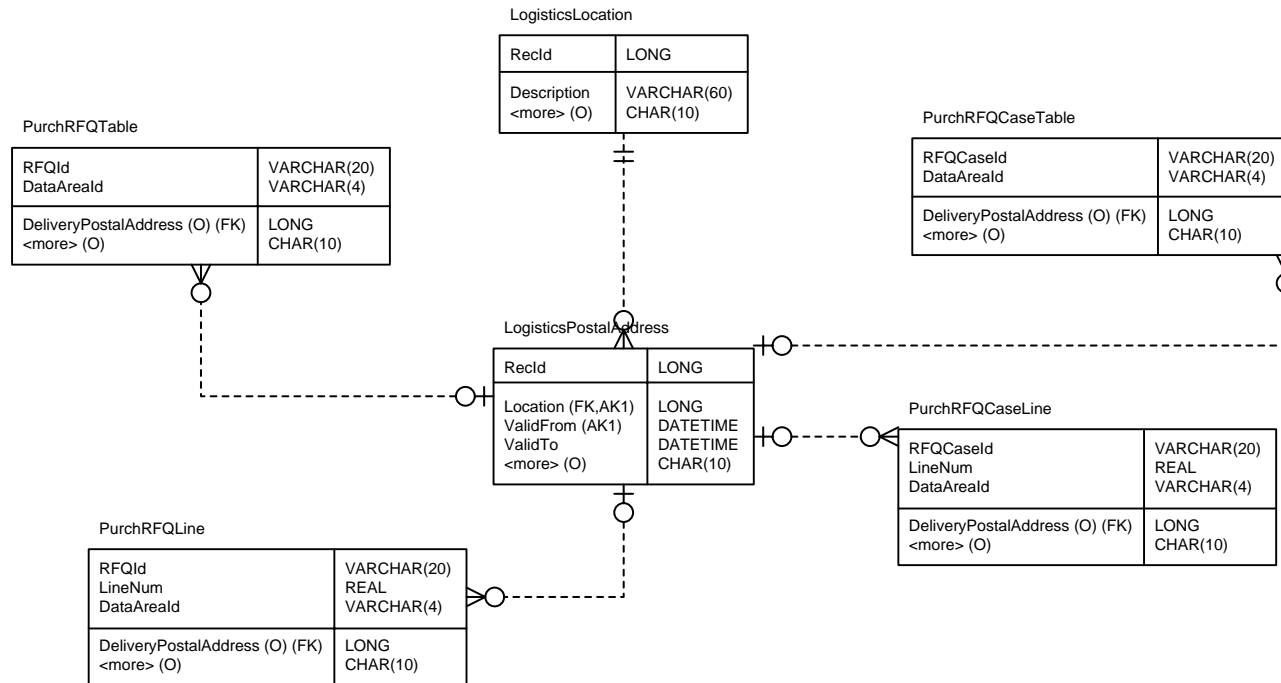
Service order



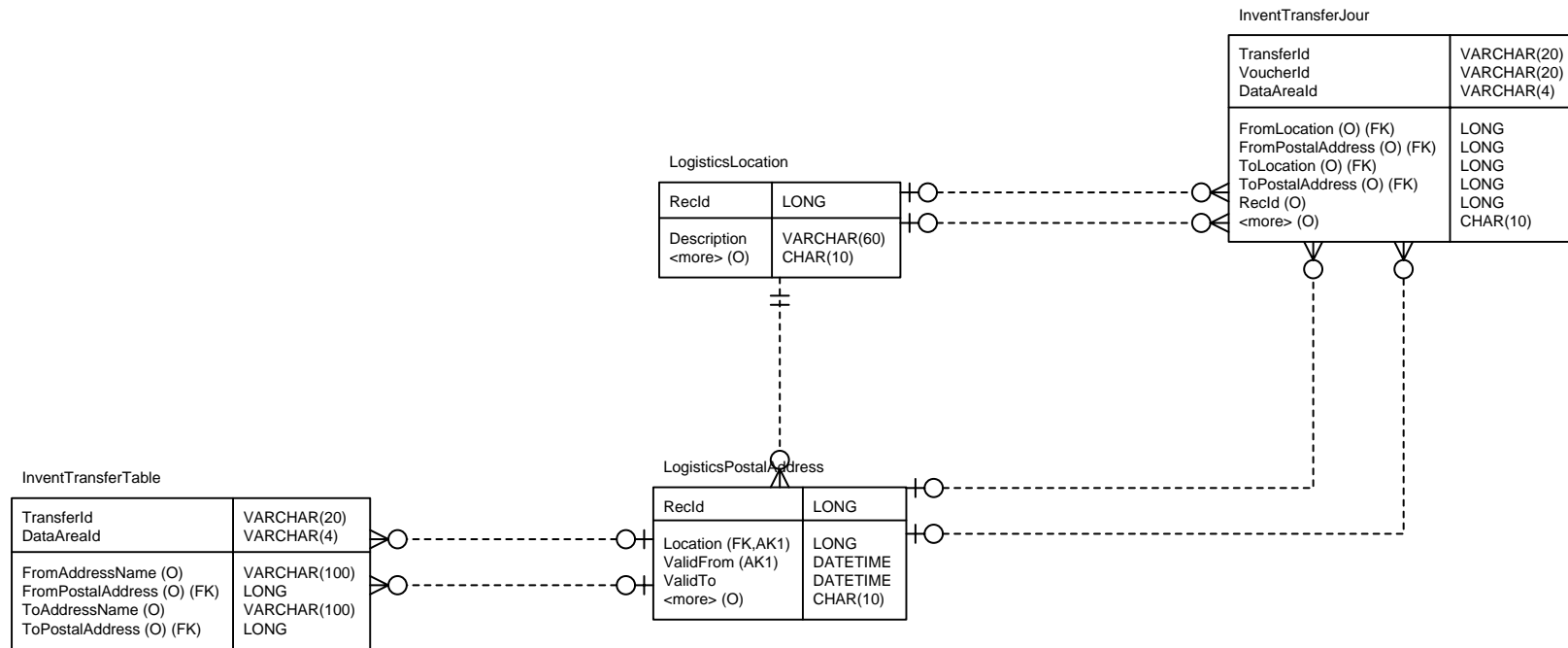
Purchase order



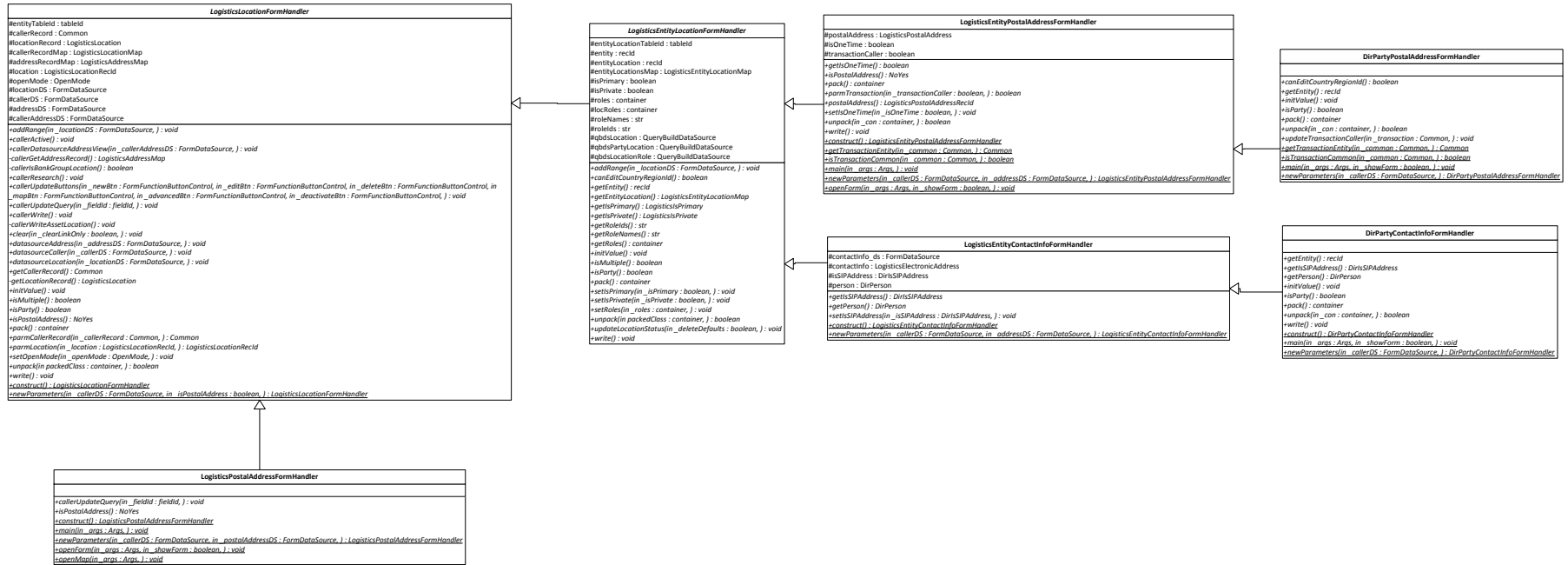
RFQ and reply



Transfer order



Location object model



Updates since initial publication

The following table lists changes made to this document after it was initially published.

| Date | Change |
|--------------|---|
| October 2011 | <p>Added sections that describe the following:</p> <ul style="list-style-type: none">• How to use the extensible data security (XDS) framework to restrict access to parties that can viewed or accessed. For details, see Extensible data security and Using XDS to enforce location privacy.• How to create a new party role. For details, see Updating code related to party.• How to update code related to transactions. For details, see Updating code related to transactions.• Code samples for the ContactPersonEntity class. |
| January 2011 | Initial publication |

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

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, Microsoft Dynamics, and the Microsoft Dynamics logo are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

