Microsoft Dynamics® AX 2012

Implementing the Account and Financial Dimensions Framework for Microsoft Dynamics AX 2012 Applications

White Paper

This document highlights new patterns used to represent accounts and financial dimensions, and describes how to convert the existing patterns to the new Microsoft Dynamics AX 2012 patterns. This white paper has been updated for Microsoft Dynamics AX 2012 R2 and Microsoft Dynamics AX 2012 R3.

February 2014

http://microsoft.com/dynamics/ax

Bill Frandsen, Senior Developer
Jason Dinham, Senior Development Lead

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

Table of Contents

Overview **3**
Audience 3
Terminology 4

Implementing or upgrading code **5**
Process for performing code and data upgrades 5

Changes to the data model **5**
Default account 6
Main account 6
Ledger account 6
Multi-type account 6
Default dimension 7
Dimension attribute set 8

Revising your data patterns **8**
Default account 8
Main account 12
Ledger account 13
Multi-type account 17
Default dimension 21
Dimension attribute set 24

Setting an entity to be dimensionable **27**

X++ code patterns **28**
Default account pattern 28
Main account pattern 28
Ledger account pattern 29
Multi-type account pattern 29
Default dimension pattern 29
Dimension attribute set pattern 29

Web services **29**
DimensionService class 30
ChartOfAccountsService class 31
DimensionValueService class 33
FinancialDimensionValidationService class 35
FinancialDimensionBalanceService class 40

Data upgrade **40**
Conversion patterns 41
Environments 43
DimensionConversionHelper API 45
Set-based upgrade 45

Updates since initial publication **46**

Appendix **47**

Overview

In Microsoft Dynamics AX 2012, the account and financial dimensions framework has been enhanced to provide substantially more functionality. To support this new functionality, the data model has also been completely redesigned. Therefore, developers will need to update every reference to ledger accounts and financial dimensions in existing applications to reference the new data model.

Each of the account and financial dimensions patterns defined in this white paper has a corresponding Microsoft Dynamics AX form control. Form controls provide a consistent implementation of the functionality across the application and a simplified programming model for the framework. Account form controls combine the Segmented Entry control and a controller class. The Segmented Entry control is a general-purpose Microsoft Dynamics AX client control that is being introduced in Microsoft Dynamics AX 2012. The controller class is an X++ class that handles the events raised by the Segmented Entry control. The combination of a control and an underlying class allows the user to see a dynamic number of segments, based on the values the user provides. The controls described in this white paper handle lookup, "view details" functionality, and validation for each segment in the account number. This design allows for a straightforward and consistent implementation of these patterns in Microsoft Dynamics AX forms.

In Microsoft Dynamics 2009, a ledger account was represented by a single string value and was considered to be separate from the financial dimensions. Ledger accounts were stored in the LedgerTable table. When developers needed to hold a foreign key to a ledger account, they stored the string "LedgerTable.AccountNum" for that account in their table. Financial dimensions were referenced as an array of up to 10 array elements that each held the string value of a foreign key to the Dimensions table. All financial dimension values were stored in the Dimensions table. You could not use data stored in other Microsoft Dynamics AX tables as financial dimensions.

In Microsoft Dynamics AX 2012, no part of the legacy framework still exists. However, many of the underlying concepts persist. Microsoft Dynamics AX 2012 still has ledger accounts and financial dimensions, but the definition of these terms has changed slightly to account for the new functionality.

This document does not discuss all of the new functionality within ledger accounts and financial dimensions. Instead, it highlights the new patterns that are used to represent ledger accounts and financial dimensions and describes how to convert existing patterns to the new Microsoft Dynamics AX 2012 patterns. Note that financial dimension pattern enhancements do not affect inventory dimension patterns in any way.

Note: *Changes have been made to this paper after it was initially published.*
For details, see Updates since initial publication.

Audience

This white paper is intended for developers who are building new applications for Microsoft Dynamics AX 2012 and developers who are updating their existing application code and data.

Terminology

Microsoft Dynamics AX 2012 terms:

Term	Definition
Advanced rule	A rule in an accounting system that controls the additional valid financial dimension value combinations when the requirements of the account structure condition are met.
Account structure	A configuration of the main account financial dimension and other financial dimensions.
Financial dimension	A financial data classifier created from the parties, locations, products, and activities in an organization and used for management reporting.
Financial dimension value	A data element in the domain of a financial dimension.
Ledger account	A classifier created from the combination of the main account value listed in a chart of accounts and other financial dimension values; used to classify the financial consequences of economic activity.
Ledger account alias	A shortcut that is used to retrieve a complete or partial ledger account.
Main account	A classifier of economic resource value based on the claims that parties make on the economic resources owned by a legal entity; used to classify debit and credit entries in an accounting system.

Additional notes about terminology use in Microsoft Dynamics AX 2012:

- **Ledger account**: In previous versions of Microsoft Dynamics AX, the term *ledger account* was used to specify what is now referred to as the *main account*. This terminology was used mainly because the application never merged the ledger account with financial dimensions to create ledger account combinations. Financial dimension fields were always tracked separately. In Microsoft Dynamics AX 2012, the main account is merged with financial dimensions to create a more integrated view of the chart of accounts (COA) and financial activity.
- **Financial dimension:** The meaning of this term across the application has not changed. However, the functionality and implementation are very different. With the new support for unlimited financial dimensions in Microsoft Dynamics AX 2012 and the ability to use existing application data to define financial dimensions, the functionality has been greatly enhanced.

Implementing or upgrading code

The information in this white paper is intended for developers who need to perform the following tasks for ledger accounts and financial dimensions:

- Code upgrade
- Data upgrade
- Setting an entity to be dimensionable
- Web service integration
- Microsoft Dynamics AX 2012 form development
- Microsoft Dynamics AX 2012 X++ code development

Process for performing code and data upgrades

Developers who need to perform 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 relevant sections of this white paper to upgrade their code. You can perform a code upgrade in any sequence, but all of the following steps are required:

- Identify the pattern or patterns that your code currently uses.
- Add new fields in the data model to represent the new foreign keys to accounts and financial dimensions.
- Assign the DEL_ prefix to the field and set the ConfigurationKey to SysDeletedObjects60 to delete the old foreign key fields.
- Update the user interface to use the new control that is appropriate for the pattern defined. The new controls make use of 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 X++ code patterns section of this white paper.
- Update existing reports to make use of the new data model, including the specific views created for reporting.

After the code upgrade is complete, developers should refer to the Data upgrade section of this white paper to code their data upgrade scripts.

Changes to the data model

The first step in upgrading your code for use in the new framework for accounts and financial dimensions is to identify where in the data model you were using these patterns. The following sections explain the new patterns that you will use to upgrade your code and how the new patterns relate to the old account and financial dimension patterns. The new patterns are as follows:

- Default account
- Main account
- Ledger account
- Multi-type account
- Default dimension
- Dimension attribute set

Default account

Previous version

A default account in Microsoft Dynamics AX 2009 was a string field that held the ledger account and was a foreign key to the LedgerTable table. The field was used primarily on posting profiles to determine which ledger account should be used when posting to the general ledger.

Microsoft Dynamics AX 2012

The default account pattern consists of a single segment, main account. The default account is a separate pattern from the main account because of the additional business logic built into its underlying control. This business logic handles data restrictions in lookups and other critical functionality that would have to be developed separately in each uptake scenario without the control. A foreign key representing a default account is a 64-bit integer field that contains the data from the corresponding RecId field in the DimensionAttributeValueCombination (or LedgerDimension) table. Foreign key fields for default accounts are named LedgerDimension because that is the alias used for the DimensionAttributeValueCombination table.

For a data model diagram of the DimensionAttributeValueCombination subsystem, see the Appendix.

Main account

Previous version

In Microsoft Dynamics AX 2009, the main account pattern was most closely represented by ledger accounts in the LedgerTable table. There was no difference between default account patterns and main account patterns in the previous version of the framework.

Microsoft Dynamics AX 2012

The main account pattern represents just one of the possible segments in a ledger account. A main account is required in a ledger account, but it does not have to be the first segment. When referencing a main account as a foreign key, the developer should set up a reference to the RecId field in the MainAccount table.

For a data model diagram of the MainAccount table, see the Appendix.

Ledger account

Previous version

In Microsoft Dynamics AX 2009, the ledger account pattern was the same as the main account and default account patterns.

Microsoft Dynamics AX 2012

A ledger account contains the main account, account structure, and the financial dimension values that are needed to populate the related account structure and advanced rule structures. A foreign key representing a ledger account is a 64-bit integer field that contains the data from the corresponding RecId field in the DimensionAttributeValueCombination table (also called the LedgerDimension table). Foreign key fields for ledger accounts are named LedgerDimension because that is the alias used for the DimensionAttributeValueCombination table.

For a data model diagram of the DimensionAttributeValueCombination subsystem, see the Appendix.

Multi-type account

Previous version

In Microsoft Dynamics AX 2009, all financial modules used a common structure for their account numbers, which was a **String10**. This pattern allowed a single string field to hold the account number, regardless of which account type was selected. Additional business logic was needed to determine which account type the developer was dealing with. This pattern could produce discrepancies, such as when a user decided to change the account number format for one module

(for example, changing "bank" to a **String30**) but not for the related modules that shared the account number field.

Microsoft Dynamics AX 2012

The multi-type account pattern consists of an account type field and its related ledger account or default account. In this pattern, a ledger account or default account can store accounts other than ledger accounts. When the account type field for the related account is set to "ledger," this pattern becomes the default account or ledger account pattern (depending on the extended data type used for the field). If the account type field is not set to "ledger," the pattern stores a system-generated account structure (used to specify which account number should be stored in the field) and an account number for the related account type. For example, if the account type field is set to "customer," the related account number field will contain a customer number with a related account structure that indicates that the account number field should contain a single customer value.

This pattern is primarily used for the setup and entry of financial journals. In this case, a multi-type account will store one of six types of accounts, based on the related account type field (asset, bank, customer, vendor, project, or ledger). A foreign key representing a multi-type account is a 64-bit integer field that contains the data from the corresponding RecId field of the DimensionAttributeValueCombination (LedgerDimension) table. Foreign key fields for multi-type accounts are named LedgerDimension because that is the alias used for the DimensionAttributeValueCombination table.

For a data model diagram of the DimensionAttributeValueCombination subsystem, see the Appendix.

Default dimension

Previous version

The default dimension pattern represented a set of financial dimensions and their values. A customer typically set the values for given financial dimensions on master data records for defaulting purposes or directly on some transactions.

This pattern was represented in the Microsoft Dynamics AX 2009 data model with a financial dimensions array field. In the Application Object Tree (AOT), this field was represented as a single array field. In the database, the array was represented as having a "one field per-array element." The dimension array could have from 3 to 10 elements.

Microsoft Dynamics AX 2012

The default dimension pattern still represents a set of financial dimensions and their related values. A customer will still set the values for given financial dimensions on master data records for defaulting purposes or directly on some transactions. The primary difference is in how this data is stored. In the new data model, this pattern is represented by a single field. The single field is a foreign key reference to the RecId field in the DimensionAttributeValueSet table. This new value references the correct record in the DimensionAttributeValueSet (or Default Dimension) table. Whereas the previous model limited storage to 10 financial dimensions, the new model allows you to store an unlimited number of financial dimensions and their values.

For a data model diagram of the DimensionAttributeValueSet subsystem, see the Appendix.

Dimension attribute set

The dimension attribute set pattern represents a set of financial dimensions and their related enumeration values. This pattern is used in advanced general ledger processing where different financial dimensions can be handled differently, based on these stored enumeration values.

Previous version

The dimension attribute set pattern was represented in the Microsoft Dynamics AX 2009 data model by a financial dimension array of enumeration fields. In the AOT, the dimension attribute set pattern was represented as a single array field. In the database, the array was represented as having a "one field per-array element." The dimension array could have from 3 to 10 elements.

Microsoft Dynamics AX 2012

The dimension attribute set pattern is still used to store a set of financial dimensions and related data, such as enumeration values. This pattern is used in places where it is necessary to store additional information about dimensions, such as whether they are optional or required. For an example of this implementation, see the **VendPaymMode** form under **Forms** in the AOT. A foreign key representing a dimension attribute set is a 64-bit integer field that contains the data from the corresponding RecId field of the DimensionAttributeSet table. Whereas the previous model limited storage to 10 financial dimensions, the new model allows you to store an unlimited number of financial dimensions.

For a data model diagram of the DimensionAttributeSet subsystem, see the Appendix.

Revising your data patterns

This section provides detailed information about how to map your existing account and financial dimension patterns to the new patterns. Each pattern subsection has a table that shows the old extended data types and field names that were used to represent the pattern in Microsoft Dynamics AX 2009. The table also shows the new extended data types and the field names that you must use to represent the pattern in Microsoft Dynamics AX 2012. Each pattern subsection explains how the Microsoft Dynamics AX form control for the pattern must be set up and how to query the data for inquiry and reporting needs.

Default account

In Microsoft Dynamics AX 2009, account numbers from the LedgerTable table that were used to post financial activity to the general ledger were considered to represent default accounts.

To upgrade your code for Microsoft Dynamics AX 2012, you must replace your data model with the new field names and extended data types specified in the following table. You need to update the forms where these fields are displayed to use the control defined in this section. The X++ code that references these fields must use the new fields and code patterns defined for default accounts.

Data model and extended data types

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

	Previous	Microsoft Dynamics AX 2012
EDTs	AccountNum LedgerAccount	LedgerDimensionDefaultAccount
Fields	AccountNum LedgerAccount Account	LedgerDimension LedgerDimension is a foreign key to the DimensionAttributeValueCombination table.

Microsoft Dynamics AX form control

The Default Account control is a combination of the Segmented Entry control and the **LedgerDimensionDefaultAccountController** class. The **LedgerDimensionDefaultAccountController** class handles the events raised by the Segmented Entry control.

Changes needed on the form

In simple scenarios, the changes needed on the form are as follows:

1. Verify that the table that will hold the foreign key to the DimensionAttributeValueCombination table is a data source on the form.
2. Drag the LedgerDimension field from the data source to the desired location on the form design. This should add a Segmented Entry control at this location on the form with appropriate **DataSource** and **ReferenceField** property values. Alternatively, you can complete this step by adding a Segmented Entry control to the design and manually setting the **DataSource** and **ReferenceField** properties.
3. Override the following methods on the form.

```
public class FormRun extends ObjectRun
{
    LedgerDimensionDefaultAccountController ledgerDimensionDefaultAccountController;
}

public void init()
{
    super();
    ledgerDimensionDefaultAccountController =
        LedgerDimensionDefaultAccountController::construct(myTable_ds,
            fieldstr(MyTable, LedgerDimension));
}
```

4. Override the following methods on the Segmented Entry control instance in the form design.

```
public void jumpRef()
{
    ledgerDimensionDefaultAccountController.jumpRef();
}

public void loadAutoCompleteData(LoadAutoCompleteDataEventArgs _e)
{
    super(_e);
    ledgerDimensionDefaultAccountController.loadAutoCompleteData(_e);
}

public void segmentValueChanged(SegmentValueChangedEventArgs _e)
{
```

```
    super(_e);
    ledgerDimensionDefaultAccountController.segmentValueChanged(_e);
}

public void loadSegments()
{
    super();
    // (Optional parm*() specification should go here, see the Control options section.)
    ledgerDimensionDefaultAccountController.parmControl(this);
    ledgerDimensionDefaultAccountController.loadSegments();
}

public boolean validate()
{
    boolean isValid;

    isValid = super();
    isValid = ledgerDimensionDefaultAccountController.validate() && isValid;

    return isValid;
}
```

5. Override the following methods on the data source field that backs the Segmented Entry control.

```
public Common resolveReference(FormReferenceControl _formReferenceControl)
{
    return ledgerDimensionDefaultAccountController.resolveReference();
}
```

Control options

There are four parameter methods available for specifying items that the control can link to and validate. These parameter methods should be called in the **loadSegments** method. These methods are called every time that the control receives focus. Always declaring the parameters in one method ensures that developers can easily verify whether all parameters have been correctly set.

- **parmCurrentLedgerCOA**: Specifies a particular chart of accounts from which to derive the list of main accounts. The default value is the current chart of accounts.
- **parmFilterLedgerPostingType**: Specific main accounts can be excluded from the lookup based on the posting type, but the values are allowed to be entered manually. The default value is None, meaning all accounts are valid.

  Note The parameter name (*FilterLedgerPostingType*) differs from the one for the Ledger Account control because, in this case, it only restricts what is shown in the lookup when the filter is selected. For account entry, it prevents entry of values entirely.
- **parmIncludeFinancialAccounts** and **parmIncludeTotalAccounts**: Specific main accounts can be excluded from the lookup and prevented from being entered manually. The *IncludeFinancialAccounts* and *IncludeTotalAccounts* parameters are used for validation and lookup to restrict valid values. The default value for each parameter is **false**.

Reporting and querying

The default account is a single-level account combination that always encompasses a MainAccount table record. The reporting issues are the same as those described for the main account in the Main account section, except that you must first query the DimensionAttributeValueCombination table to get the RecId of the MainAccount table.

The following are query examples:

- Display the value of the default account.

```
select
    DISPLAYVALUE
from
    DIMENSIONATTRIBUTEVALUECOMBINATION
where
    RECID = 1
```

- Join to the MainAccount table to display additional fields.

```
select
    MA.MMAINACCOUNTID, MA.NAME
from
    DIMENSIONATTRIBUTEVALUECOMBINATION DAVC
    join MAINACCOUNT MA on (DAVC.MAINACCOUNT = MA.RECID)
where
    DAVC.RECID = 1
```

Main account

In Microsoft Dynamics AX 2012, account numbers from the LedgerTable table that are used to specify the account as a generic reference are considered to be main accounts. All references used to build ledger accounts should use the default account pattern instead.

When the main account pattern is required, the data model must be replaced with the new field names and extended data types specified in the following table. The forms where these fields are displayed need to be updated to use the reference group controls used for the refRecId pattern. The X++ code that references these fields must use the new fields that you add to your data model.

Data model and extended data type

In Microsoft Dynamics AX 2009, the main account and the default account were similar patterns that used the same EDT names and field names. When performing updates, you need to determine which pattern the new fields should use, based on how the data was being used in the old pattern.

	Previous	Microsoft Dynamics AX 2012
EDTs	AccountNum LedgerAccount	MainAccountNum
Fields	AccountNum LedgerAccount Account	MainAccount MainAccount is a foreign key to the MainAccount table.

Microsoft Dynamics AX form control

Tables with foreign key references to main account should be implemented as surrogate keys. When displayed on the form, the ReferenceGroup control should be used to provide surrogate key replacement, which allows the user the view and edit the natural key of the main account, yet persist it as a surrogate foreign key.

See the section on Reporting and querying for an explanation of this usage.

Reporting and querying

A central issue when performing reporting or general querying of MainAccount data is that most of the fields are shared through the chart of accounts specified in the MainAccount table. Each MainAccount record must have a valid foreign key to the chart of accounts. Another new concept in Microsoft Dynamics AX 2012 is that only one chart of accounts can be specified for a given ledger. Each legal entity must specify which ledger it will use. The ledger specifies the chart of accounts, account structures, financial calendar, accounting currency, reporting currency, and other critical ledger setup information. When querying the MainAccount table from a table with a foreign key, you are assured of getting the proper record because a unique RecId is used as the foreign key. If you have only the MainAccount.MainAcountId**,** you must know the legal entity, ledger, or chart of accounts to query for your data. Because a legal entity only has a single ledger, and because a ledger only has a single chart of accounts, you can find the current chart of accounts with any of these three pieces of data.

Within the MainAccount data model there is also a set of fields that is specific to a legal entity. These fields (such as default tax codes) can have different values for each legal entity that uses this chart of accounts. To access these fields, you need to know the legal entity for the main account that you are querying. For an illustration of the MainAccount data model, see the Appendix.

The following are query examples:

- Find all MainAccount records for a specific chart of accounts, given the name of that chart.

```
select
    MainAccountId
from
    MAINACCOUNT
```

```
    join LEDGERCHARTOFACCOUNTS
on (MAINACCOUNT.LEDGERCHARTOFACCOUNTS = LEDGERCHARTOFACCOUNTS.RECID)
where
    LEDGERCHARTOFACCOUNTS.NAME = 'EXT'
```

- Find all MainAccount records for a specific chart of accounts, given the legal entity name.

```
select
    MAINACCOUNT.MAINACCOUNTID
from
    MAINACCOUNT
    join LEDGERCHARTOFACCOUNTS
ON (MAINACCOUNT.LEDGERCHARTOFACCOUNTS = LEDGERCHARTOFACCOUNTS.RECID)
    join LEDGER ON (LEDGER.CHARTOFACCOUNTS = LEDGERCHARTOFACCOUNTS.RECID)
    join COMPANYINFO ON (LEDGER.PRIMARYFORLEGALENTITY = COMPANYINFO.RECID)
where
COMPANYINFO.DATAAREA = 'DMO'
```

Ledger account

In Microsoft Dynamics AX 2009, an account number from the LedgerTable table was considered to represent a ledger account. The account number was combined with the financial dimensions array to link financial activity to a ledger account. An example of this was the LedgerTrans table, in which the AccountNum field represented the ledger account and the Dimension field represented the financial dimensions.

In Microsoft Dynamics AX 2012, these fields are deleted and replaced with a single field that holds the main account and all financial dimensions related to financial activity. To upgrade your code for Microsoft Dynamics AX 2012, your data model must be replaced with the new field names and extended data types specified in the following table. You need to update forms in which these fields are displayed to use the control defined in this section. The X++ code that references these fields must use the new fields and code patterns defined for default accounts. Often, the most difficult upgrade task is determining whether to convert AccountNum and Dimension fields in a single table to the ledger account pattern, or to convert the fields to use the default account and default dimensions patterns.

Data model and extended data types

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

	Previous	Microsoft Dynamics AX 2012
EDTs	AccountNum LedgerAccount	LedgerDimensionAccount
Fields	AccountNum LedgerAccount Account	LedgerDimension LedgerDimension is a foreign key to the DimensionAttributeValueCombination table.

Microsoft Dynamics AX form control

The Ledger Account control uses a combination of the **LedgerDimensionAccountController** class and the Segmented Entry control. This control handles the entry and display of ledger accounts in Microsoft Dynamics AX forms.

Changes needed on the form

In simple scenarios, the changes needed on a Microsoft Dynamics AX form are as follows:

1. Verify that the table holding the foreign key to the DimensionAttributeValueCombination table is a data source on the form.
2. Drag the LedgerDimension field from the data source to the desired location on the form design. This creates a Segmented Entry control with the appropriate **DataSource** and **ReferenceField** property values. Alternatively, you can complete this step by adding a Segmented Entry control to the design and manually setting the **DataSource** and **ReferenceField** properties.
3. Override the following methods on the form. If the methods already exist, just add the code to the methods. Be sure to indicate where the **super()** call is for the **init** method.

   class declaration:

   ```
   public class FormRun extends ObjectRun
   {
       LedgerDimensionAccountController ledgerDimensionAccountController;
   }
   ```

   init (for the form):

   ```
   public void init()
   {
       super();

       ledgerDimensionAccountController =
   LedgerDimensionAccountController::construct({BackingDataSource_ds},
   fieldstr({BackingTable},LedgerDimension));
   }
   ```

4. Override the following methods on the Segmented Entry control instance in the form design:

   ```
   public void jumpRef()
   {
       ledgerDimensionAccountController.jumpRef();
   }

   public boolean validate()
   {
       boolean isValid;

       isValid = super();
       isValid = ledgerDimensionAccountController.validate() && isValid;
   ```

```
    return isValid;
}

public void segmentValueChanged(SegmentValueChangedEventArgs _e)
{
    super(_e);
    ledgerDimensionAccountController.segmentValueChanged(_e);
}

public void loadSegments()
{
    super();

    ledgerDimensionAccountController.parmControl(this);
    // The value of this parameter varies depending on the type of form.
    // See the Control options section below for more detail.

    ledgerDimensionAccountController.loadSegments();
}

public void loadAutoCompleteData(LoadAutoCompleteDataEventArgs _e)
{
    super(_e);
    ledgerDimensionAccountController.loadAutoCompleteData(_e);
}
```

5. Override the following methods on the data source field that backs the Segmented Entry control:

```
public Common resolveReference(FormReferenceControl _formReferenceControl)
{
    return ledgerDimensionAccountController.resolveReference();
}
```

Control options

Several parameters affect the validation, lookup, and saving of ledger accounts performed by the Segmented Entry control. The following are the methods used to set these parameters:

- **parmCurrency**: Specifies the currency code associated with the control being managed. The *Currency* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the currency.
- **parmDataAreaId**: Specifies the legal entity associated with the control being managed. The *DataAreaId* parameter is used to restrict valid values for validation and lookup. The default value is the current legal entity. If this parameter is provided by a field that the user can manipulate on the form, we recommend that you call the corresponding **parm** method from the **modified** method of the control for the DataAreaId field.
- **parmDate**: Specifies the date of the transaction associated with the control being managed. The *Date* parameter is used for validation. The default value is **empty** and no validation is done against the date.
- **parmDialogField**: Specifies the dialog field that references the ledger account when using the **LedgerDimensionAccountController** on a **RunBase** dialog.
- **parmDimensionAccountStorageUsage**: Specifies how to use account storage for the control being managed. The *DimensionAccountStorageUsage* parameter is used for validation and for saving combinations. The default value is **setup**. Use the **DimensionAccountStorageUsage** enumeration to pass in the type.

- **parmDimensionAutocompleteFilter**: Specifies custom filtering of the autocomplete data. Use the **DimensionAutocompleteFilterable** interface to specify the filtering code.
- **parmDisableMRU**: Specifies whether the lookup for the most recently used data should be prevented from appearing. The default value is false. Note: the full lookup showing all or just valid values will still be accessible.
- **parmJournalName**: Specifies the journal name associated with the control being managed. The *JournalName* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the journal name.
- **parmLockMainAccountSegment**: Specifies whether the segment containing the main account is non-editable. The default value is false.
- **parmPostingType**: Specifies the posting type associated with the control being managed. The *PostingType* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the posting type.
- **parmSkipSuspendedAndActiveDateValidation**: Specifies whether suspended and inactive values can be entered without displaying a validation error. The default value is false.
- **parmTaxCode**: Specifies the tax code associated with the control being managed. The *TaxCode* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the tax code.
- **parmUser**: Specifies the user associated with the control being managed. The *User* parameter is used for validation and lookup to restrict valid values. The default value is the current user.
- **parmValidateBlockedForManualEntry**: Specifies whether validation should be enforced for dimension values marked as not allowing manual entry. The default value is false.

These parameters should be specified in the **loadSegments** method. They are called every time the control receives focus. Always declaring the parameters in one method ensures that developers can easily verify whether all parameters have been correctly set.

Special scenarios

When multiple fields have an account number control, each field should have its own **LedgerDimensionAccountController** instance. Each instance must have a unique name that ends with "LedgerDimensionAccountController" and can begin with a functional description of how the backing field is used. If the variable name becomes too long, the "LedgerDimension" prefix in the variable name can be dropped.

Code that is similar to the example shown earlier in this section should be added for each controller instance. When one field needs to be edited in multiple places on the same form, there should be a single **LedgerDimensionAccountController** instance for the field. When two controls share the same controller, the **loadSegments** method for each control should always contain a call to **parmControl(this)** before the call to the **loadSegments** method for the controller.

Reporting and querying

A ledger account contains a default account and a collection of financial dimensions. To make it easier to implement common reporting scenarios, such as viewing the full combination, Microsoft Dynamics AX 2012 has added a DisplayValue field in the DimensionAttributeValueCombination table. This string shows all ledger account segments, separated by the defined segment separator.

The following are query examples:

- Display the ledger account value.

```
select
    DISPLAYVALUE
from
    DIMENSIONATTRIBUTEVALUECOMBINATION
where
    RECID = 1
```

- Display only the MainAccount.MainAccountId and MainAccount.Name values.

```
select
    MA.MAIMACCOUNTID, MA.NAME
from
    DIMENSIONATTRIBUTEVALUECOMBINATION DAVC
    join MAINACCOUNT MA on (DAVC.MAINACCOUNT = MA.RECID)
where
    DAVC.RECID = 1
```

- Display each segment of the ledger account with data related to the segments.

  Note We have created a view in the AOT to simplify querying the database in this common scenario. In this example, we return the display value separately for each segment in a ledger account.

```
select
    DALVV.DISPLAYVALUE
from
    DIMENSIONATTRIBUTELEVELVALUEVIEW DALVV
where
    DALVV.VALUECOMBINATIONRECID = 1
```

Multi-type account

In Microsoft Dynamics AX 2009, there were scenarios in which the value of an account number was dependent on a related field (often called the account type field). Depending on the account type that was selected, a specific type of account number was stored. For example, if the "customer" account type was selected, an account number representing a customer was expected in the account number field. The most common scenario for this pattern involved the financial journals, where the user could select from six different account types.

In Microsoft Dynamics AX 2012, this pattern still exists. The pattern has been built into the account and financial dimension framework to address the requirements of financial journals. The pattern is more complicated in Microsoft Dynamics AX 2012 because the ledger account is very different from other accounts that can be entered on a financial journal. Ledger accounts now allow the user to enter many segments.

Data model and extended data types

The following table lists the previous and new extended data types (EDTs) and fields related to multi-type accounts.

	Previous	Microsoft Dynamics AX 2012
EDTs	AccountNum LedgerAccount	DimensionDynamicAccount
Fields	AccountNum LedgerAccount Account	LedgerDimension LedgerDimension is a foreign key to the DimensionAttributeValueCombination table.

Microsoft Dynamics AX form control

The Multi-Type Account control represents a combination of the Segmented Entry control and the DimensionDynamicAccountController class. The Segmented Entry control is a general-purpose control that has been introduced in Microsoft Dynamics AX 2012. The **DimensionDynamicAccountController** class handles the events raised by the Segmented Entry control. This combination allows the control to handle accounts of several different types.

Changes needed on the form

In simple scenarios, the changes needed on the form are as follows:

1. Verify that the table that will hold the foreign key to the DimensionAttributeValueCombination table is a data source on the form.
2. Drag the LedgerDimension field from the data source to the desired location on the form design. This action should add a Segmented Entry control at this location on the form with appropriate **DataSource** and **ReferenceField** property values. Alternatively, you can complete this step by adding a Segmented Entry control to the design and manually setting the **DataSource** and **ReferenceField** properties.
3. Override the following methods on the form.

   class declaration:

   ```
   public class FormRun extends ObjectRun
   {
       DimensionDynamicAccountController dimAccountController;
   }
   ```

   init (for the form):

   ```
   public void init()
   {
       super();
       dimAccountController = DimensionDynamicAccountController::construct(
           ledgerJournalTrans_ds,
           fieldstr(LedgerJournalTrans, LedgerDimension),
           fieldstr(LedgerJournalTrans, AccountType));
   }
   ```

4. Override the following methods on the Segmented Entry control instance in the form design.

```
public void jumpRef()
{
        dimAccountController.jumpRef();
}

public void loadAutoCompleteData(LoadAutoCompleteDataEventArgs _e)
{
        super(_e);
        dimAccountController.loadAutoCompleteData(_e);
}

public void segmentValueChanged(SegmentValueChangedEventArgs _e)
{
        super(_e);
        dimAccountController.segmentValueChanged(_e);
}

public void loadSegments()
{
        super();
        // (Optional parm*() specification should go here, see the Control options
section.)
        dimAccountController.parmControl(this);
        dimAccountController.loadSegments();
}

public boolean validate()
{
        boolean isValid;

        isValid = super();
        isValid = dimAccountController.validate() && isValid;

        return isValid;
}
```

5. Override the following methods on the data source field that backs the Segmented Entry control.

```
public Common resolveReference(FormReferenceControl _formReferenceControl)
{
        return dimAccountController.resolveReference();
}
```

Control options

Several parameter methods can be used to manage the information that the control links to and validates:

- **parmCurrency**: Specifies the currency code associated with the control that is being managed. The *Currency* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the currency.
- **parmDataAreaId**: Specifies the legal entity associated with the control being managed. The *DataAreaId* parameter is used for validation and lookup to restrict valid values. The default value is the current legal entity. If this is provided by a field that the user can manipulate on the form, we recommend that you call the corresponding **parm** method from the modified method of the control for the DataAreaId field.
- **parmDate**: Specifies the date of the transaction associated with the control being managed. The *Date* parameter is used for validation. The default value is **empty** and no validation is done against the date.
- **parmDialogField**: Specifies the dialog field that references the ledger account when using the **LedgerDimensionAccountController** on a **RunBase** dialog.
- **parmDimensionAccountStorageUsage**: Specifies how the control being managed is used. The *DimensionAccountStorageUsage* parameter is used for validation and the saving of combinations. The default value is **setup**.
- **parmDimensionAutocompleteFilter**: Specifies custom filtering of the autocomplete data. Use the **DimensionAutocompleteFilterable** interface to specify the filtering code.
- **parmDisableMRU**: Specifies whether the lookup for the most recently used data should be prevented from appearing. The default value is false.

  Note The full lookup showing all or just valid values will still be accessible.
  parmFilterLedgerPostingType: Specific main accounts can be excluded from the lookup based on the posting type, but the values are allowed to be entered manually. The default value is None, meaning all accounts are valid. See also **parmPostingType**.
- **parmIsDefaultAccount**: Specifies whether the control presents a default account or a full account/dimension combination pattern.
- **parmJournalName**: Specifies the journal name associated with the control being managed. The *JournalName* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the journal name.
- **parmPostingType**: Specifies the posting type associated with the control being managed. The *PostingType* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the posting type.
- **parmSkipSuspendedAndActiveDateValidation**: Specifies whether suspended and inactive values can be entered without displaying a validation error. The default value is false.
- **parmTaxCode**: Specifies the tax code associated with the control being managed. The *TaxCode* parameter is used for validation and lookup to restrict valid values. The default value is **empty** and no restriction or validation is done against the tax code.
- **parmUser**: Specifies the user associated with the control being managed. This parameter is used for validation and lookup to restrict valid values. The default value is the current user.
- **parmValidateBlockedForManualEntry**: Specifies whether validation should be enforced for dimension values marked as not allowing manual entry. The default value is false.
- **parmIncludeTotalAccounts**: Specifies whether main accounts marked as "total" can be used. The *IncludeTotalAccounts* parameter is used for validation and lookup to restrict valid values. The default value is **false**.

Parameters should be specified in the **loadSegments** method and are called every time the control receives focus. Always declaring the parameters in one method ensures that developers can easily verify whether all parameters have been correctly set.

Reporting and querying

Multi-type accounts are stored in the same table as default accounts and ledger accounts. Multi-type accounts always contain a single account segment that holds an account number for an account other than the main account. For example, in financial journals, the user has a choice of entering accounts types: bank, asset, customer, vendor, project, or ledger. In all cases except the ledger account, a multi-type account is stored. Because multi-type accounts use the same data model as ledger accounts, the same query helpers exist.

The following are query examples:

- Display the account value.

```
select
    DISPLAYVALUE
from
    DIMENSIONATTRIBUTEVALUECOMBINATION
where
    RECID = 1
```

- Display the account with data related to the segment.

  Note We have created a view in the AOT to simplify querying the database in this common scenario. The DimensionAttributeLevelValue.DisplayValue, DimensionAttribute.RecId**,** and DimensionAttributeValue.RecId are returned for each segment in the multi-type account.

  The RecIds that are returned can be useful because the DimensionAttribute table defines which entity the account number is related to. You can use this additional information to determine whether the DisplayValue is displaying a vendor account or a customer account.

```
select
    DALVV.DISPLAYVALUE,
    DALVV.DIMENSIONATTRIBUTE,
    DALVV.ATTRIBUTEVALUERECID
from
    DIMENSIONATTRIBUTELEVELVALUEVIEW DALVV
where
    DALVV.VALUECOMBINATIONRECID = 1
```

Default dimension

In Microsoft Dynamics AX 2009, financial dimensions were stored in an array of string values on every table that needed to reference a set of financial dimension values. Master data tables, such as Customer, Vendor, and Bank Account, held these sets of financial dimension defaults for use during subledger posting to the general ledger. The posting code merged multiple sets of financial dimensions to determine which financial dimension values to update during the general ledger posting.

To upgrade your code for Microsoft Dynamics AX 2012, your data model must be replaced with the new field names and extended data types specified in the following table. Forms that display these fields need to be updated to use the control defined in this section. The X++ code that references these fields must use the new fields and code patterns defined for default dimensions.

Data model and extended data type

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

	Previous	Microsoft Dynamics AX 2012
EDTs	Dimension SysDimension DimensionLedgerJournal	DimensionDefault
Fields	Dimension	DefaultDimension DefaultDimension is a foreign key to the DimensionAttributeValueSet table.

Microsoft Dynamics AX form control

The Default Dimension control is backed by the **DimensionDefaultingController** class. The **DimensionDefaultingController** class handles drawing of individual field controls for each financial dimension for the current ledger's account structures, and field entry/validation, lookup, and "view details" functionality. Drawing the fields is a dynamic process because it is possible to add, change, or remove financial dimensions after installation of Microsoft Dynamics AX 2012. Validation, lookup, and "view details" functionality is encapsulated into the control for ease of implementation. Parameters can be set on the control to change the functionality of validation and the lookup.

Changes needed on the form

In simple scenarios, the changes needed on the form are as follows:

1. Verify that the table that will hold the foreign key to the DimensionAttributeValueSet table is a data source on the form.
2. Create a tab page that will contain the financial dimensions control. This control is often the only data shown on the tab because the number of financial dimensions can be large.
    1. Set the **Name** metadata of the tab to **TabFinancialDimensions**.
    2. Set the **AutoDeclaration** metadata of the tab to **Yes**.
    3. Set the **Caption** metadata of the tab to **@SYS101181 (Financial dimensions)**.
    4. Set the **NeedPermission** metadata of the tab to **Manual**.
    5. Set the **HideIfEmpty** metadata of the tab to **No**.
3. Override the **pageActivated** method on the new tab.

```
public void pageActivated()
{
    dimDefaultingController.pageActivated();

    super();
}
```

4. Override the following methods on the form.

class declaration:

```
public class FormRun extends ObjectRun
{
    DimensionDefaultingController dimDefaultingController;
}
```

init (for the form):

```
public void init()
{
    super();
    dimDefaultingController=DimensionDefaultingController::constructInTabWithValues(
            true,
            true,
            true,
            0,
            this,
            tabFinancialDimensions,
            "@SYS138487");

    dimDefaultingController.parmAttributeValueSetDataSource(myTable_ds,
        fieldstr(myTable, DefaultingDimension));
}
```

5. Override the following methods on the form data source that contains the foreign key to the DimensionAttributeValueSet table.

```
public int active()
{
    int ret;
    ret = super();
    dimDefaultingController.activated();
    return ret;
}
public void write()
{
    dimDefaultingController.writing();
    super();
}
public void delete()
{

    super();
    dimDefaultingController.deleted();
}
```

Control options

Two overloads of the constructor will build the Default Dimension control with different characteristics:

- **DimensionDefaultingController::constructInGroupWithValues**: Constructs an instance of the **DimensionDefaultingController** class with a StringEdit control associated with each dimension attribute that is added to the specified **FormGroupControl** object.
- **DimensionDefaultingController::constructInTabWithValues**: Constructs an instance of the **DimensionDefaultingController** class with a StringEdit control associated with each dimension attribute that is added to the specified **FormTabPageControl** object.

Reporting and querying

The default dimension pattern is stored in a data model that starts with the DimensionAttributeValueSet table. All foreign keys to this pattern hold a RecId to this table. We have provided a view in the AOT to simplify querying the segments defined in the pattern.

The following query example displays the name of the dimension segment and the related value stored for that segment.

```
select
    NAME,
    DISPLAYVALUE
from
    DEFAULTDIMENSIONVIEW
where
    DEFAULTDIMENSION = 5637144603
```

Dimension attribute set

A *dimension attribute set* is a list of one or more dimension attributes that are being tracked for a particular purpose. Each dimension attribute has a related enumeration value that needs to be tracked. For example, suppose you need to specify a list of dimension attributes and specify the actions that can be performed for each dimension attribute. Assume that there are three dimension attributes in the system: **Department**, **Cost Center**, and **Purpose**. Also, assume that there is a **Required** field that specifies an enumeration with the following order of values: **Optional**, **Blank**, and **Required**. You would like to specify **Required** for **Department**, **Blank** for **Cost Center**, and **Optional** for **Purpose**. In this scenario, the pattern you would use would be a dimension attribute set. This pattern also supports variations, including a simple checkbox instead of an enumeration field.

Data model and extended data types

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

	Previous	Microsoft Dynamics AX 2012
EDTs	DimensionAllocation DimensionKeepFromTransaction	DimensionEnumeration
Fields	DimensionAllocation KeepTransactionDimension	DimensionSelectionCriteria DimensionKeepTransaction DimensionSelectionCriteria and DimensionKeepTransaction are foreign keys to the DimensionAttributeSet table.

Microsoft Dynamics AX form control

The Dimension Attribute Set control is similar to the Default Dimension control, but you are required to add an enumeration to the control instead of the dimension attribute values. The Dimension Attribute Set control is supported by the **DimensionDefaultingController** class. The **DimensionDefaultingController** class handles the drawing of individual field controls for each

financial dimension in the ledger, and the field entry/validation, lookup, and "view details" functionality. Drawing the fields is a dynamic process because it is possible to add, change, or remove financial dimensions after the installation of Microsoft Dynamics AX 2012. Validation, lookup, and "view details" functionality are encapsulated into the control for ease of implementation. You can set parameters on the control to change the functionality of validation and the lookup.

Changes needed on the form

In simple scenarios, the changes needed on the form are as follows:

1. Verify that the table that will hold the foreign key to the DimensionAttributeSet table is a data source on the form.
2. Create a tab page that will contain the financial dimensions control. This control is often the only data shown on the tab because the number of financial dimensions can be large.
    1. Set the **Name** metadata for the tab to **TabFinancialDimensions**.
    2. Set the **AutoDeclaration** metadata for the tab to **Yes**.
    3. Set the **Caption** metadata for the tab to **@SYS101181 (Financial dimensions)**.
    4. Set the **NeedPermission** metadata for the tab to **Manual**.
    5. Set the **HideIfEmpty** metadata for the tab to **No**.
3. Override the **pageActivated** method on the new tab.

```
public void pageActivated()
{
    dimensionDefaultingController.pageActivated();

    super();
}
```

4. Override the following methods on the form.

class declaration:

```
public class FormRun extends ObjectRun
{
    DimensionDefaultingController dimDefaultingController;
}
```

init (for the form):

```
public void init()
{
    super();
    dimDefaultingController=DimensionDefaultingController::
constructInGroupWithChecks(
            true,
0,
this,
tabFinancialDimensions,
"@SYS138487",
"",
"");
    dimDefaultingController = parmAttributeValueSetDataSource(myTable_ds,
        fieldstr(myTable, DefaultingDimension));
}
```

5. Override the following methods on the form data source that contains the foreign key to the DimensionAttributeSet table.

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

```
        ret = super();
        dimDefaultingController.activated();
        return ret;
    }

    public void write()
    {
        dimDefaultingController.writing();
        super();
    }

    public void delete()
    {
        dimDefaultingController.deleted();
        super();
    }
```

Control options

Several overloads of the constructor will build the Dimension Attribute Set control with different characteristics:

- **DimensionDefaultingController::constructInGroupWithChecks**: Constructs an instance of the **DimensionDefaultingController** class that has a check box control associated with each dimension attribute that is added to the specified **FormGroupControl** object.
- **DimensionDefaultingController::constructInGroupWithChecksAndValues**: Constructs an instance of the **DimensionDefaultingController** class with a check box and a StringEdit control associated with each dimension attribute that is added to the specified **FormGroupControl** object.
- **DimensionDefaultingController::constructInGroupWithCombosAndValues**: Constructs an instance of the **DimensionDefaultingController** class with a combo box and a StringEdit control associated with each dimension attribute that is added to the specified **FormGroupContro**l object.
- **DimensionDefaultingController::constructInTabWithChecks**: Constructs an instance of the **DimensionDefaultingController** class with a check box control associated with each dimension attribute that is added to the specified **FormTabPageControl** object.
- **DimensionDefaultingController::constructInTabWithChecksAndValues**: Constructs an instance of the **DimensionDefaultingController** class with a check box and a StringEdit control associated with each dimension attribute that is added to the specified **FormTabPageControl** object.
- **DimensionDefaultingController::constructInTabWithCombos**: Constructs an instance of the **DimensionDefaultingController** class with a combo box control associated with each dimension attribute that is added to the specified **FormTabPageControl** object.
- **DimensionDefaultingController::constructInTabWithCombosAndValues**: Constructs an instance of the **DimensionDefaultingController** class with a combo box and a StringEdit control associated with each dimension attribute that is added to the specified **FormTabPageControl** object.

Reporting and querying

The dimension attribute set pattern is stored in a data model that starts with the DimensionAttributeSet table. All foreign keys to this pattern hold a RecId to this table. We have provided a view in the AOT to simplify querying the segments (dimension attributes) defined in the pattern.

The following query example displays the name of the dimension attribute (such as department, cost center, and purpose), the integer that defines the AOT enumeration, and the integer enumeration value.

```
select
    DA.NAME,
    DAS.BASEENUMTYPE,
    DASI.ENUMERATIONVALUE
from
    DIMENSIONATTRIBUTESET DAS
    join DIMENSIONATTRIBUTESETITEM DASI on (DASI.DIMENSIONATTRIBUTESET = DAS.RECID)
    join DIMENSIONATTRIBUTE DA on (DA.RECID = DASI.DIMENSIONATTRIBUTE)
where
    DAS.RECID = 1
```

Setting an entity to be dimensionable

To set an entity to be dimensionable, create a view as directed below. The entity will automatically appear as an available backing entity type.

Also, to integrate with the dimensions framework when deleting or renaming the natural key of the backing entity, you must write custom code on the backing table's **delete** method, and also on either the **update** or **renamePrimaryKey** method. See CustTable for an example of the pattern these methods must follow.

1. The view name must be DimAttribute[*yourentityname*]. For example, DimAttributeCustTable.
2. The view must contain a root data source named BackingEntity that points to your backing table to identify a surrogate key and natural key.
3. The view may optionally contain additional related data sources to handle inheritance or relational associations to provide additional fields, such as a name from the DirPartyTable.
4. The view must contain the following fields named exactly as follows:
    - Key - Must point to the backing entity's SK field. For example, an int64 RecId field.
    - Value - Must point to the backing entity's NK field. For example, a str30 AccountNum field.
    - Name - Must point to the source of an additional description for the entity. For example, a str60 Description field.
5. The view must have a SingularLabel assigned that is different from its Label property.
6. The view must have the same ConfigKey as the backing table.
7. The view must be included in the DimensionEssentials Security Privilege.

Because the list of dimensionable entities are cached on both the client and server, the creation of a new dimensionable entity will not appear in the list of existing entities until a call to clear the caches is performed, or until both the client and the server are restarted. In order to clear the caches and have the new backing entity appear immediately, you must execute the following line of code within a job:

```
DimensionCache::clearAllScopes();
```

If a new Organization Model OMOperatingUnitType enumeration is added, the steps to make it dimensionable are similar but can be made shorter as follows:

1. Copy one of the existing DimAttributeOM[*entityname*] views, rename it appropriately and adjust all associated labels and help text.
2. Expand the Datasource\BackingEntity (OMOperatingUnit)\Ranges node on the copied view and change the value property on the range to the new OMOperatingUnitType enumeration value that was just added.

Because the OMOperatingUnitType is backed by the OMOperatingUnit table, generic code already exists to handle the **delete**, **update** and **renamePrimaryKey** methods. Therefore, in this case, you do not need to update these methods.

X++ code patterns

This section describes the common APIs, which are used to access and modify account and financial dimensions data. These APIs are all class methods.

Note Calls to the DimensionDefaultingService will not share the same database transaction context as the caller, even if the caller has a transaction currently running. As with all X++ service calls, these calls are asynchronous.

Default account pattern

Use the following methods for the default account pattern.

- **DimensionStorage::getDefaultAccount**: Gets the default account that represents the specified main account RecId.
- **DimensionStorage::getDefaultAccountForMainAccountNum**: Gets the default account that represents the specified main account number.
- **DimensionStorage::getLedgerDefaultAccountFromLedgerDim**: Gets the default account that represents the main account from the specified ledger dimension.

Main account pattern

Use the following methods for the main account pattern.

- **DimensionStorage::getMainAccountFromLedgerDimension:** Gets the main account table buffer that corresponds to the specified ledger dimension.
- **DimensionStorage::getMainAccountIdFromLedgerDimension:** Gets the RecId of the main account that corresponds to the specified ledger dimension.
- **DimensionStorage::getMainAccountNumFromLedgerDimension:** Gets the account number of the main account that corresponds to the specified ledger dimension.

Ledger account pattern

Use the following methods for the ledger account pattern.

- **DimensionDefaultingEngine::getLedgerDimensionFromAccountAndDim**: Gets the ledger dimension that represents the combination for the specified main account and default dimension.
- **DimensionDefaultingService::serviceApplyFixedDimensions**: Gets the ledger dimension that represents the combination formed by applying any fixed dimensions specified on the main account on the specified ledger dimension.
- **DimensionDefaultingService::serviceCreateLedgerDimension**: Gets the ledger dimension that represents the combination for the specified ledger dimension and default dimensions.
- **DimensionDefaultingService::serviceMergeLedgerDimensions**: Gets the ledger dimension that represents the combination formed by merging the specified ledger dimensions together.
- **DimensionStorage::getAccountStructureFromLedgerDimension**: Gets the RecId of the account structure for the specified ledger dimension.

Multi-type account pattern

Use the following methods for the multi-type account pattern.

- **DimensionStorage::getDynamicAccount**: Gets the ledger dimension that represents the dynamic account for the specified account and account type.
- **DimensionStorage::ledgerDimension2AccountNum:** Gets the string value of the account for the specified dynamic account.

Default dimension pattern

Use the following methods for the default dimension pattern.

- **DimensionDefaultingEngine::overrideDefaultDimension:** Updates a default dimension with the specified dimension values.
- **DimensionDefaultingService::serviceMergeDefaultDimensions:** Merges the specified default dimensions into a single default dimension.
- **DimensionDefaultingService::serviceReplaceAttributeValue:** Replaces the value of the specified dimension attribute in the target default dimension with the value from the source default dimension.
- **DimensionStorage::getDefaultDimensionFromLedgerDimension**: Gets the default dimension that represents all dimension values from the specified ledger account other than the main account.
- **DimensionAttributeValueSetStorage**: This class provides an API for creating and editing default dimensions.

Dimension attribute set pattern

Use the following method for the dimension attribute set pattern.

- **DimensionAttributeSetStorage:** This class manages the storage of the <c>DimensionAttributeSet</c> table and the <c>DimensionAttributeSetItem</c> table.

Web services

Web service classes are provided to more easily integrate with the major subsystems of the account/dimension framework. The classes use only natural key information instead of surrogate key record IDs. This section describes the service APIs.

DimensionService class

The **DimensionService** class provides standard primitive operations for retrieving a list of all financial dimensions available or for a specific account structure.

getDimensions method

Returns an X++ list type that contains elements of class type **DimensionContract** for all dimensions associated with the specified account structure. The **AccountStructureContract** class is used to specify the name of the account structure to retrieve the list of dimensions for.

Example

[X++]

```
accountStructureContract = new AccountStructureContract();
accountStructureContract.parmName('Expense');

dimensionService = new DimensionService();
dimensionList = dimensionService.getDimensions(accountStructureContract);

dimIterator = new ListIterator(dimensionList);
dimensionContract = dimIterator.value();
name = dimensionContract.parmDimensionName();
dimIterator.next();
…
```

[C#]

```
AccountStructureContract accountStructureContract =
                         new AccountStructureContract { parmName = accountStructureName };
DimensionServiceClient proxy = new DimensionServiceClient();

try
{
   DimensionContract[] dimensionList = proxy.getDimensions(null, accountStructureContract);
}
finally
{
   proxy.Close();
}
```

getDimensionsAll method

Returns an X++ list type that contains elements of class type **DimensionContract** for all dimensions in the system.

Example

[X++]

```
dimensionService = new DimensionService();
dimensionList = dimensionService.getDimensionsAll();

dimIterator = new ListIterator(dimensionList);
dimensionContract = dimIterator.value();
name = dimensionContract.parmDimensionName();
dimIterator.next();
…
```

[C#]

```
DimensionServiceClient proxy = new DimensionServiceClient();

try
{
    DimensionContract[] dimensionList = proxy.getDimensionsAll(null);
}
finally
{
    proxy.Close();
}
```

ChartOfAccountsService class

The **ChartOfAccountsService** class provides standard primitive operations for retrieving a list of all ledgers, ledger chart of accounts, main accounts for a specific ledger chart of accounts or creation of main accounts.

createMainAccount method

Creates a new main account. The class **MainAccountContract** is used to specify the name and other fields that describe the main account.

Example

[X++]

```
service = new ChartOfAccountsService();
mainAccountContract = new MainAccountContract();

mainAccountContract.parmMainAccountId('999');
mainAccountContract.parmName('Service Test Account');
mainAccountContract.parmLedgerChartOfAccounts(curext());

result = service.createMainAccount(mainAccountContract);
```

[C#]

```
MainAccountContract mainAccountContract = new MainAccountContract();
mainAccountContract.parmMainAccountId = "99999";
mainAccountContract.parmName = "Test Main Account";
mainAccountContract.parmLedgerChartOfAccounts = this.CompanyName;

ChartOfAccountsServiceClient proxy = new ChartOfAccountsServiceClient();

try
```

```
{
    bool created = proxy.createMainAccount(null, mainAccountContract);
}
finally
{
   proxy.Close();
}
```

getLedgerChartOfAccounts method

Returns an X++ list type that contains elements of class type **LedgerChartOfAccountsContract** for all ledger charts of accounts in the system.

Example

[X++]

```
service = new ChartOfAccountsService();
list = service.getLedgerChartOfAccounts();
enumerator = list.getEnumerator();
while (enumerator.moveNext())
{
    ledgerCOAcontract = enumerator.current();
    name = ledgerCOAcontract.parmName();
}
```

[C#]

```
ChartOfAccountsServiceClient proxy = new ChartOfAccountsServiceClient();

try
{
    LedgerChartOfAccountContract[] ledgerChartOfAccounts =
proxy.getLedgerChartOfAccounts(null);
    this.VerifyLedgerChartOfAccounts(ledgerChartOfAccounts);
}
finally
{
    proxy.Close();
}
```

getLedgers method

Returns an X++ list type that contains elements of class type **MainAccountContract** for all main accounts for a specific ledger chart of accounts. The class **LedgerChartOfAccountContract** is used to specify the name of the ledger chart of account to retrieve the list of main accounts for.

Example

[X++]

```
service = new ChartOfAccountsService();
ledgerCOAContract = new LedgerChartOfAccountContract();
ledgerCOAContract.parmName(curext());

list = service.getMainAcounts(ledgerCOAContract);
enumerator = list.getEnumerator();
while (enumerator.moveNext())
{
    mainAccountContract = enumerator.current();
    mainAccountId = mainAccountContract.parmMainAccountId();
}
```

[C#]

```
ChartOfAccountsServiceClient proxy = new ChartOfAccountsServiceClient();

try
{
   LedgerContract[] ledgers = proxy.getLedgers(null);
}
finally
{
   proxy.Close();
}
```

DimensionValueService class

The **DimensionValueService** class provides standard primitive operations for creating a user-defined dimension value for a specific financial dimension and retrieving a list of all values for a specific financial dimension.

createDimensionValue method

Creates a new user-defined financial dimension value for the specified financial dimension. The class **DimensionValueContract** is used to specify the dimension attribute and value as well as additional properties of the value.

Example

[X++]

```
service = new DimensionValueService();
dimensionValueContract =  new DimensionValueContract();

dimensionValueContract.parmDescription('Sales department');
dimensionValueContract.parmValue('Dept001');
dimensionValueContract.parmDimensionAttribute('Department');

service.createDimensionValue(dimensionValueContract);
```

[C#]

```
DimensionValueContract dimensionValueContract =
               new DimensionValueContract { parmDimensionAttribute = dimensionAttributeName,
                                            parmValue = "Fund_09",
```

```
                                           parmDescription = "Fund 09",
                                           parmActiveFrom = DateTime.Today,
                                           parmActiveTo = DateTime.Today,
                                           parmIsSuspended = Reference.NoYes.Yes,
                                           parmIsBlockedForManualEntry = Reference.NoYes.Yes,
                                           parmBackingEntityType = 656,
                                           parmGroupDimension = string.Empty,
                                           parmPersonnelNumber = "AMO" };

DimensionValueServiceClient proxy = new DimensionValueServiceClient();

try
{
   bool isSuccessful = proxy.createDimensionValue(null, dimensionValueContract);
}
finally
{
   proxy.Close();
}
```

getDimensionsValues method

Returns an X++ list type that contains elements of class type **DimensionValueContract** for all values for a specified financial dimension. The class **DimensionContract** is used to specify the dimension name.

Example

[X++]

```
service = new DimensionValueService();
dimensionContract =  new DimensionContract();

dimensionContract.parmDimensionName('Department');

dimensionValueContractList = service.getDimensionValues(dimensionContract);
contractIterator = new ListIterator(dimensionValueContractList);
dimensionValueContract = contractIterator.value();
dimensionValue = dimensionValueContract.parmValue();
contractIterator.next();
…
```

[C#]

```
DimensionContract dimensionContract =
                     new DimensionContract { parmDimensionName = dimensionAttributeName };
DimensionValueServiceClient proxy = new DimensionValueServiceClient();

try
{
    DimensionValueContract[] dimensionValueList = proxy.getDimensionValues(null,
dimensionContract);
}
finally
{
    proxy.Close();
}
```

FinancialDimensionValidationService class

The **FinancialDimensionValidationService** class provides standard primitive operations for validating a budget, budget planning, or ledger account.

getStatusForBudgetAccount method

Returns an instance of the **DimensionValidationStatusContract** class that contains the validation status for a budget account. You use the **BudgetAccountContract** class to specify the current structure, constraints, and budget-enabled dimensions for the account.

Example

[X++]

```
service = new FinancialDimensionValidationService();
budgetAccountValidationContract = new BudgetAccountValidationContract();
budgetAccountContract = new BudgetAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();
validationStatus = new DimensionValidationStatusContract();

budgetAccountContract.parmAccountStructure('Expense');
budgetAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
budgetAccountContract.parmValues().addEnd(dimensionAttributeValueContract);

budgetAccountValidationContract.parmBudgetAccountContract(budgetAccountContract);
validationStatus = service.getStatusForBudgetAccount(budgetAccountValidationContract);
```

getStatusForBudgetAccountList method

Returns an instance of the **DimensionValidationStatusListContract** class that contains the validation status for a collection of budget account contracts. You use instances of the **BudgetAccountContract** class to specify the current structure, constraints, and budget-enabled dimensions for each account in the collection.

Example

[X++]

```
service = new FinancialDimensionValidationService();
validationContracts = new List(Types::Class);
validationStatusList = new DimensionValidationStatusListContract();
budgetAccountListValidationContract = new BudgetAccountListValidationContract();
```

```
budgetAccountValidationContract = new BudgetAccountValidationContract();
budgetAccountContract = new BudgetAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

budgetAccountContract.parmAccountStructure('Expense');
budgetAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
budgetAccountContract.parmValues().addEnd(dimensionAttributeValueContract);
budgetAccountValidationContract.parmBudgetAccountContract(budgetAccountContract);
validationContracts.addEnd(budgetAccountValidationContract);

budgetAccountValidationContract = new BudgetAccountValidationContract();
budgetAccountContract = new BudgetAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

budgetAccountContract.parmAccountStructure('Expense');
budgetAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('20');
budgetAccountContract.parmValues().addEnd(dimensionAttributeValueContract);
budgetAccountValidationContract.parmBudgetAccountContract(budgetAccountContract);
validationContracts.addEnd(budgetAccountValidationContract);

budgetAccountListValidationContract.parmValidationContracts(validationContracts);

validationStatusList = service.getStatusForBudgetAccountList(
        budgetAccountListValidationContract);
```

getStatusForBudgetPlanning method

Returns an instance of the **DimensionValidationStatusContract** class that contains the validation status for a budget planning account. You use the **BudgetPlanningContract** class to specify the current structure, constraints, advanced rules, budget planning rules, and organization model relationships for the account.

Example

[X++]

```
service = new FinancialDimensionValidationService();
budgetPlanningValidationContract = new BudgetPlanningValidationContract();
budgetPlanningContract = new BudgetPlanningContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();
validationStatus = new DimensionValidationStatusContract();

budgetPlanningContract.parmAccountStructure('Expense');
budgetPlanningContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
budgetPlanningContract.parmValues().addEnd(dimensionAttributeValueContract);

budgetPlanningValidationContract.parmBudgetPlanningContract(budgetPlanningContract);
validationStatus = service.getStatusForBudgetPlanning(budgetPlanningValidationContract);
```

getStatusForBudgetPlanAccountList method

Returns an instance of the **DimensionValidationStatusListContract** class that contains the validation status for a collection of budget plan accounts. You use instances of the **BudgetPlanningContract** class to specify the current structure, constraints, advanced rules, budget planning rules, and organization model relationships for each budget plan contract in the collection.

Example

[X++]

```
service = new FinancialDimensionValidationService();
validationContracts = new List(Types::Class);
validationStatusList = new DimensionValidationStatusListContract();
budgetPlanAccountListValidationContract = new BudgetPlanAccountListValidationContract ();

budgetPlanningValidationContract = new BudgetPlanningValidationContract();
budgetPlanningContract = new BudgetPlanningContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

budgetPlanningContract.parmAccountStructure('Expense');
budgetPlanningContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
budgetPlanningContract.parmValues().addEnd(dimensionAttributeValueContract);

budgetPlanningValidationContract.parmBudgetPlanningContract(budgetPlanningContract);
validationContracts.addEnd(budgetPlanningValidationContract);

budgetPlanningValidationContract = new BudgetPlanningValidationContract();
budgetPlanningContract = new BudgetPlanningContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

budgetPlanningContract.parmAccountStructure('Expense');
budgetPlanningContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('20');
budgetPlanningContract.parmValues().addEnd(dimensionAttributeValueContract);

budgetPlanningValidationContract.parmBudgetPlanningContract(budgetPlanningContract);
validationContracts.addEnd(budgetPlanningValidationContract);

budgetPlanAccountListValidationContract.parmValidationContracts(validationContracts);

validationStatusList = service.getStatusForBudgetPlanAccountList(
    budgetPlanAccountListValidationContract);
```

getStatusForLedgerAccount method

Returns an instance of the **DimensionValidationStatusContract** class that contains the validation status for a ledger account. You use the **LedgerAccountContract** class to specify the current structure, constraints, advanced rules, and organization model relationships for the account.

Example

[X++]

```
service = new FinancialDimensionValidationService();
```

```
ledgerAccountValidationContract = new LedgerAccountValidationContract();
ledgerAccountContract = new LedgerAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

ledgerAccountContract.parmMainAccount('65000');
ledgerAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
ledgerAccountContract.parmValues().addEnd(dimensionAttributeValueContract);

ledgerAccountValidationContract.parmLedgerAccount(ledgerAccountContract);
validationStatus = service.getStatusForLedgerAccount(ledgerAccountValidationContract);
```

getStatusForLedgerAccountList method

Returns an instance of the **DimensionValidationStatusListContract** class that contains the validation status for a collection of ledger accounts. You use instances of the **LedgerAccountContract** to specify the current structure, constraints, advanced rules, and organization model relationships for each account in the collection.

Example

[X++]

```
service = new FinancialDimensionValidationService();
validationContracts = new List(Types::Class);
validationStatusList = new DimensionValidationStatusListContract();
ledgerAccountListValidationContract = new LedgerAccountListValidationContract ();

ledgerAccountValidationContract = new LedgerAccountValidationContract();
ledgerAccountContract = new LedgerAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

ledgerAccountContract.parmMainAccount('59000');
ledgerAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
ledgerAccountContract.parmValues().addEnd(dimensionAttributeValueContract);

ledgerAccountValidationContract.parmLedgerAccount(ledgerAccountContract);
validationContracts.addEnd(ledgerAccountValidationContract);

ledgerAccountValidationContract = new LedgerAccountValidationContract();
ledgerAccountContract = new LedgerAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

ledgerAccountContract.parmMainAccount('65000');
ledgerAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('20');
ledgerAccountContract.parmValues().addEnd(dimensionAttributeValueContract);

ledgerAccountValidationContract.parmLedgerAccount(ledgerAccountContract);
validationContracts.addEnd(ledgerAccountValidationContract);

ledgerAccountListValidationContract.parmValidationContracts(validationContracts);
```

```
validationStatusList = service.getStatusForLedgerAccountList(
    ledgerAccountListValidationContract);
```

validateBudgetAccount method

Validates that the combination specified in the **BudgetAccountContract** class is valid per the current structure, constraints, and budget-enabled dimensions.

Example

[X++]

```
service = new FinancialDimensionValidationService();
budgetAccountValidationContract = new BudgetAccountValidationContract();
budgetAccountContract = new BudgetAccountContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

budgetAccountContract.parmAccountStructure('Expense');
budgetAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
budgetAccountContract.parmValues().addEnd(dimensionAttributeValueContract);

budgetAccountValidationContract.parmBudgetAccountContract(budgetAccountContract);
isValid = service.validateBudgetAccount(budgetAccountValidationContract);
```

validateBudgetPlanningAccount method

Validates that the combination specified in the **BudgetPlanningContract** class is valid per the current structure, constraints, advanced rules, budget planning rules, and organization model relationships.

Example

[X++]

```
service = new FinancialDimensionValidationService();
budgetPlanningValidationContract = new BudgetPlanningValidationContract();
budgetPlanningContract = new BudgetPlanningContract();
dimensionAttributeValueContract = new DimensionAttributeValueContract();

budgetPlanningContract.parmAccountStructure('Expense');
budgetPlanningContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
budgetPlanningContract.parmValues().addEnd(dimensionAttributeValueContract);

budgetPlanningValidationContract.parmBudgetPlanningContract(budgetPlanningContract);
isValid = service.validateBudgetPlanningAccount(budgetPlanningValidationContract);
```

validateLedgerAccount method

Validates that the combination specified in the **LedgerAccountContract** class is valid per the current structure, constraints, advanced rules, and organization model relationships.

Example

[X++]

```
service = new FinancialDimensionValidationService();
ledgerAccountValidationContract = new LedgerAccountValidationContract();
ledgerAccountContract = new LedgerAccountContract();
```

```
dimensionAttributeValueContract = new DimensionAttributeValueContract();

ledgerAccountContract.parmMainAccount('65000');
ledgerAccountContract.parmValues(new List(Types::Class));

dimensionAttributeValueContract.parmName('Department');
dimensionAttributeValueContract.parmValue('10');
ledgerAccountContract.parmValues().addEnd(dimensionAttributeValueContract);

ledgerAccountValidationContract.parmLedgerAccount(ledgerAccountContract);
isValid = service.validateLedgerAccount(ledgerAccountValidationContract));
```

FinancialDimensionBalanceService class

The **FinancialDimensionBalanceService** class provides standard primitive operations for retreiving the balance for a specific dimension set.

getBalance method

Returns the balance in the class **DimensionSetBalanceContract** for the specified dimension set. The class **DimensionSetContract** is used to specify the dimension attribute and value as well as additional properties of the value.

Example

[X++]

```
service = new FinancialDimensionBalanceService();
attributeValueContractMA = new DimensionAttributeValueContract();

contract = new DimensionSetContract();
contract.parmIncludeOperatingFiscalPeriod(true);
contract.parmPostingLayer(OperationsTax::Current);

contract.parmCombination(new DimensionSetCombinationContract());
contract.parmCombination().parmDimensionSetName('Main Account Focus');
contract.parmCombination().parmValues(new List(Types::Class));
attributeValueContractMA.parmName('MainAccount');
attributeValueContractMA.parmValue('10025');
contract.parmCombination().parmValues().addEnd(attributeValueContractMA);

resultContract = service.getBalance(contract);
amount = resultContract.parmAccountCurrencyAmount();
```

Data upgrade

This section provides information that you should know before you attempt to upgrade account and dimension data in Microsoft Dynamics AX 2012. Included is information about the new fields or values, the methods used to convert to them, the APIs where these methods are found, and examples of how to use them.

Important: You must use the official APIs for data creation or imports that use hash values. This includes data creation for the DimensionAttributeValueCombination, DimensionAttributeValueGroup, DimensionAttributeSet, DimensionAttributeValueSet and related child tables. The hash generation process is protected framework functionality and may be subject to change. Therefore, all creation of combination and dimension set data must flow through the provided APIs, which go through DimensionStorage, DimensionAttributeSetStorage or DimensionAttributeValueSetStorageTest.

Warning: Direct import or modification of data in these tables and/or altering hash values and linked records WILL result in corruption of the account/dimension combination data and be very difficult to trace and fix.

Conversion patterns

The following patterns demonstrate how to convert the legacy account and financial dimension data to the new Microsoft Dynamics AX 2012 values. The patterns described in this section are the same patterns defined earlier in the Revising your data patterns section.

Default dimension conversion pattern

Converts Dimension to DefaultDimension.

The default dimension pattern converts legacy financial dimension values from the legacy financial dimension array field to a DefaultDimension value. The new value will reference the correct record in the DimensionAttributeValueSet (Default Dimension) table. The conversion is handled by the **DimensionConversionHelper** class.

Example

```
assetBook.DefaultDimension = DimensionConversionHelper::getNativeDefaultDimension(
assetBook.Dimension);
```

Default account conversion pattern

Converts AccountNum to LedgerDimension

The default account pattern converts an existing ledger account from the AccountNum field to a LedgerDimension value. The new LedgerDimension value will contain a reference to a main account and an account structure. This new value will reference a record in the DimensionAttributeValueCombination (Ledger Dimension) table. The conversion is handled by the **DimensionConversionHelper** class.

Example

```
bankAccountTable.LedgerDimension =
DimensionConversionHelper::getNativeDefaultAccount(bankAccountTable.LedgerAccount);
```

Ledger account conversion pattern

Converts AccountNum + Dimension to LedgerDimension

The ledger account pattern converts an existing ledger account and related dimension array values to a LedgerDimension value. The new ledger account pattern stores the related main account, the account structure, and all Dimension values. Just as in the default account pattern, the converted information is stored in the DimensionAttributeValueCombination (LedgerDimension) table.

Example

```
bankAccountTrans.LedgerDimension = DimensionConversionHelper::getNativeLedgerDimension(
bankAccountTrans.LedgerAccountNum, bankAccountTrans.Dimension);
```

Dimension attribute set conversion pattern

Converts DimUse to DimensionAttributeSet

The dimension attribute set pattern converts an array of enumeration values (mapped to be the same as the dimension array) to a DimensionAttributeSet value. The new value will reference the correct record in the DimensionAttributeSet table. This pattern works with any enumeration type value. This pattern is not used often (because it does not often apply), but because it is different from the others, guidance is provided about how to convert the data.

Example

```
CustPaymModeTable                  custPaymModeTable;
DimensionAttributeSetStorage       dimensionAttributeSetStorage;
HashKey                            dimAttrHashKeys[];
RecId                              dimAttrIds[];
SysDictEnum                        sysDictEnum = new SysDictEnum(enumNum(SysDimension));
DimensionAttribute                 attribute;
str                                name;
int                                i;
int                                dimSize;

// Get the size of the dimension.
dimSize = sysDictEnum.values();

// Retrieve the attribute ids for the dimension.
for (i = 1; i<=dimSize; i++)
{
name = sysDictEnum.index2Symbol(i-1);
attribute = DimensionAttribute::findByName(name);
dimAttrIds[i] = attribute.RecId;
dimAttrHashKeys[i] = attribute.HashKey;
}

// Updating the table.
while select forupdate custPaymModeTable
{
dimensionAttributeSetStorage = new DimensionAttributeSetStorage(enumnum(NoYes));

for (i = 1; i<=dimSize; i++)
{
        dimensionAttributeSetStorage.addItem(dimAttrIds[i], dimAttrHashKeys[i],
custPaymModeTable.DimUse[i]);
}

custPaymModeTable.DimensionAttributeSet = dimensionAttributeSetStorage.save();
custPaymModeTable.doUpdate();
}
```

Multi-type account conversion pattern

Converts AccountNum to LedgerDimension

The multi-type account pattern uses an AccountType field (for example, LedgerJournalACType) to determine the correct method to call. If the related account type has a value of "ledger," either the default account pattern or the ledger dimension pattern is used, because these will be ledger accounts.

Otherwise, the AccountNum field contains a subledger account (customer, vendor, project, asset, or bank) and the **DimensionConversionHelper::getNativeNonLedgerAccount** method is used. The subledger account value is converted to a LedgerDimension value. This pattern is always stored in the DimensionAttributeValueCombination (LedgerDimension) table, regardless of the account type (ledger, asset, bank, customer, vendor, or project) that it is representing.

Example

```
if (ledgerJournalTrans.AccountType == LedgerJournalACType::Ledger)
{
    shadow_LedgerJournalTrans.LedgerDimension =
DimensionConversionHelper::getNativeLedgerDimension(ledgerJournalTrans.AccountNum,
ledgerJournalTrans.Dimension);
}
else
{
    shadow_LedgerJournalTrans.DefaultDimension =
DimensionConversionHelper::getNativeDefaultDimension(ledgerJournalTrans.Dimension);
    shadow_LedgerJournalTrans.LedgerDimension =
DimensionConversionHelper::getNativeNonLedgerAccount(ledgerJournalTrans.AccountNum,
ledgerJournalTrans.AccountType);
}
```

Main account conversion pattern

Converts AccountNum to MainAccount

The main account pattern converts a legacy ledger account value to a reference to the corresponding MainAccount table record. Use the **MainAccount::findByMainAccountId** method for this pattern.

Example

```
ledgerAllocation.FromMainAccount = MainAccount::findByMainAccountId(
ledgerAllocation.del_FromAccount).RecId;
```

Dimension ledger account type conversion pattern

Converts LedgerAccountType to DimensionLedgerAccountType

The dimension ledger account type pattern converts a **LedgerAccountType** enumeration value to a **DimensionLedgerAccountType** enumeration value. Use the **DimensionConversionHelper::ledgerAccountType2DimLedgerAccountType** method for this pattern.

Example

```
vendDefaultAccounts.DimensionLedgerAccountType =
DimensionConversionHelper::ledgerAccountType2DimLedgerAccountType(
vendDefaultAccounts.DEL_LedgerAccountType);
```

Environments

You can upgrade data in either the source environment (Microsoft Dynamics AX 4.0 or Microsoft Dynamics AX 2009) or the target environment (Microsoft Dynamics AX 2012). The **DimensionConversionHelper** API is provided for use in both environments.

Microsoft Dynamics AX 2012

When you upgrade data in the Microsoft Dynamics AX 2012 environment, a table attribute must be added for every table referenced by the upgrade script. The following is a list of table attributes that must be added to your script when you upgrade one of the account dimension fields.

```
UpgradeScriptTableAttribute(tableStr(DimensionAttribute),false,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeDirCategory),false,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeLevelValue),true,true,false,true),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValue),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueCombination),true,true,false,true)
,
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueCombinationStatus),false,true,fals
e,true),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueGroup),true,true,false,true),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueGroupCombination),true,true,false,
true),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueGroupStatus),false,true,false,true
),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueSet),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeValueSetItem),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeSet),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionAttributeSetItem),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionFinancialTag),false,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionHierarchy),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionHierarchyLevel),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionLedgerAccount),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionLedgerAccountValue),true,true,false,false),
UpgradeScriptTableAttribute(tableStr(DimensionValueGroupJournalControlStatus),false,true,false
,true),
UpgradeScriptTableAttribute(tableStr(FinancialTagCategory),false,true,false,false),
UpgradeScriptTableAttribute(tableStr(LedgerParameters),false,true,false,false),
UpgradeScriptTableAttribute(tableStr(MainAccount),false,true,false,false),
UpgradeScriptTableAttribute(tableStr(Ledger), false, true, false, false),
```

Microsoft Dynamics AX 4.0 and Microsoft Dynamics AX 2009

When you upgrade a table in one of the source environments, you must ensure that the necessary account and dimension upgrade scripts have been run before you run your table script.

You can do this by adding a transform dependency to the transform that was created for the LedgerTable table, which is named transformation_LedgerTable.

There are two **ReleaseUpdateTransformDB** methods provided for this purpose: **addTransformDependency** and **addTransformDependencyByTable**.

If your transform resides in the **ReleaseUpdateTransformDB50_Ledger** or the **ReleaseUpdateTransformDB40_Ledger** class, use the following syntax:

```
ReleaseUpdateTransformDB::addTransformDependency(transformation_LedgerTable.getTransformationI
d(), ledgerJournalTableTransform.getTransformationId());
```

Otherwise, use the following syntax:

```
ReleaseUpdateTransformDB::addTransformDependencyByTable(tablenum(LedgerTable),
custCollectiontransform.getTransformationId());
```

DimensionConversionHelper API

The **DimensionConversionHelper** class is provided in both the source and target environments. All patterns described in the Conversion patterns section earlier in this white paper use the **DimensionConversionHelper** class, with the exception of the main account and dimension attribute set conversion patterns. The implementation in each environment is the same; you will see no difference in the conversion, regardless of the environment you work in.

The following methods are available for use in the **DimensionConversionHelper** class:

- **getNativeDefaultAccount**
- **getNativeDefaultDimension**
- **getNativeLedgerDimension**
- **getNativeNonLedgerAccount**
- **ledgerAccountType2DimLedgerAccountType**

These methods work as follows:

- **public static RecId getNativeDefaultAccount(LedgerAccount _ledgerAccount)**

  Passes in a LedgerAccount value; a RecId is returned. Sets the LedgerDimension field with the RecId.

- **public static RecId getNativeDefaultDimension(Dimension _dimension)**

  Passes in a Dimension value; a RecId is returned. Sets the DefaultDimension field with the RecId.

- **public static RecId getNativeLedgerDimension(LedgerAccount _ledgerAccount, Dimension _dimension, RecId _hierarchyId = DimensionConversionHelper::getAccountStructureHierarchyId())**

  Passes in a LedgerAccount and a Dimension value; a RecId is returned. Sets the LedgerDimension field with the RecId.

- **public static RecId getNativeNonLedgerAccount(LedgerJournalAC _account, int _accountType, enumId _enumType = enumnum(LedgerJournalACType), ModuleInventCustVend _custVend = ModuleInventCustVend::Cust)**

  Passes in the account and account type values; a RecId is returned. Sets the LedgerDimension field with the RecId.

- **public static DimensionLedgerAccountType ledgerAccountType2DimLedgerAccountType(LedgerAccountType _ledgerAccountType)**

  Passes in the LedgerAccountType value; a DimensionLedgerAccountType value is returned.

Set-based upgrade

Although not designed with set-based operations in mind, modifications have been made to the DimensionAttributeValueCombination table to assist in set-based operations when you are working with an extremely large table.

The DEL_AccountNum and DEL_Dimension fields have been added to the DimensionAttributeValueCombination table and will be populated as combinations during the upgrade. These fields have been created to give you old table fields to join to, which enables you to map your new data to the old data, if it exists. Combinations still need to be processed individually because it is not feasible to implement the hashing algorithm used in the creation of the combination in a set-based manner.

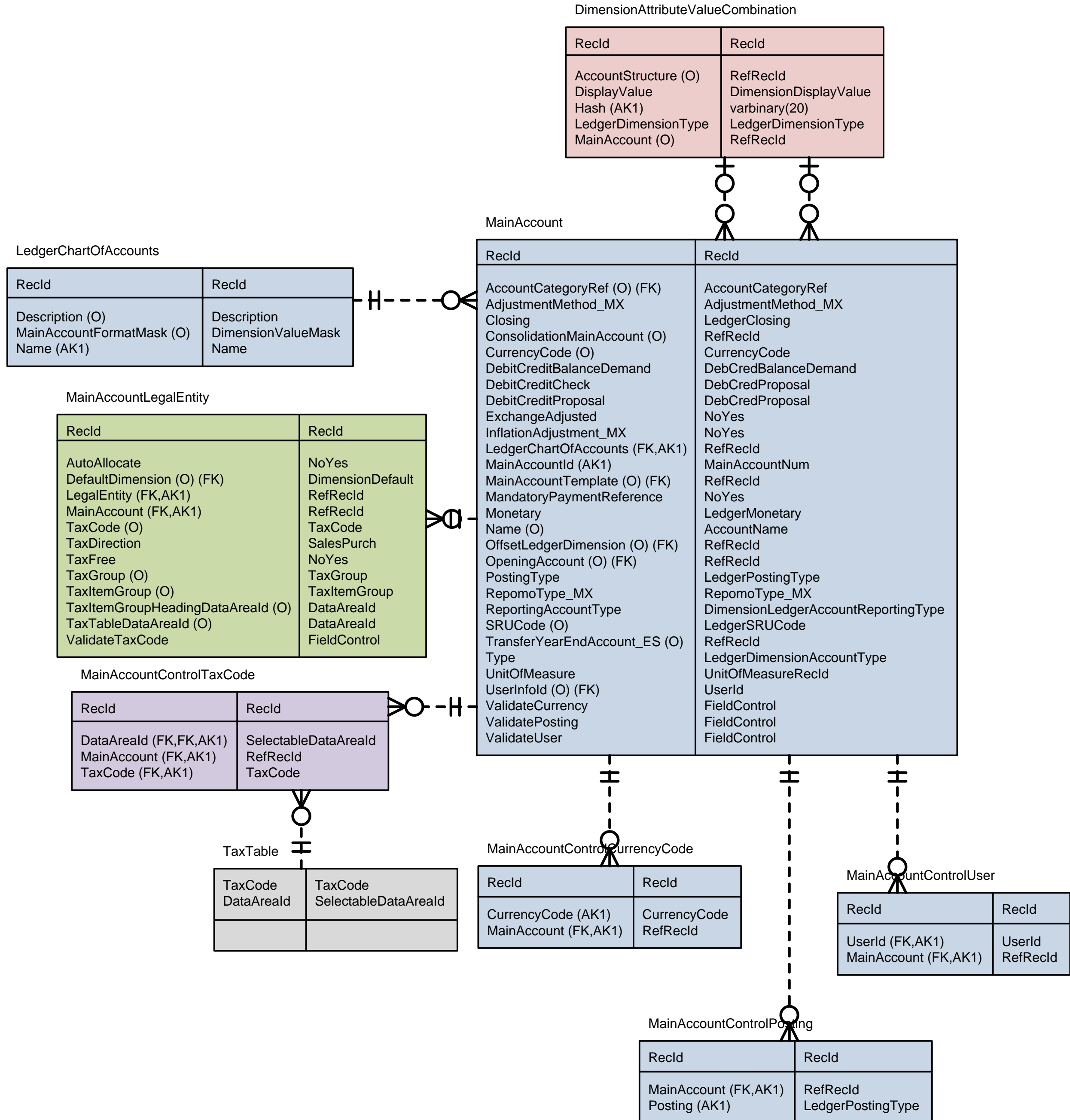
After the combinations have been created, you need to perform a join to the DimensionAttributeValueCombination table and the table that you want to populate.

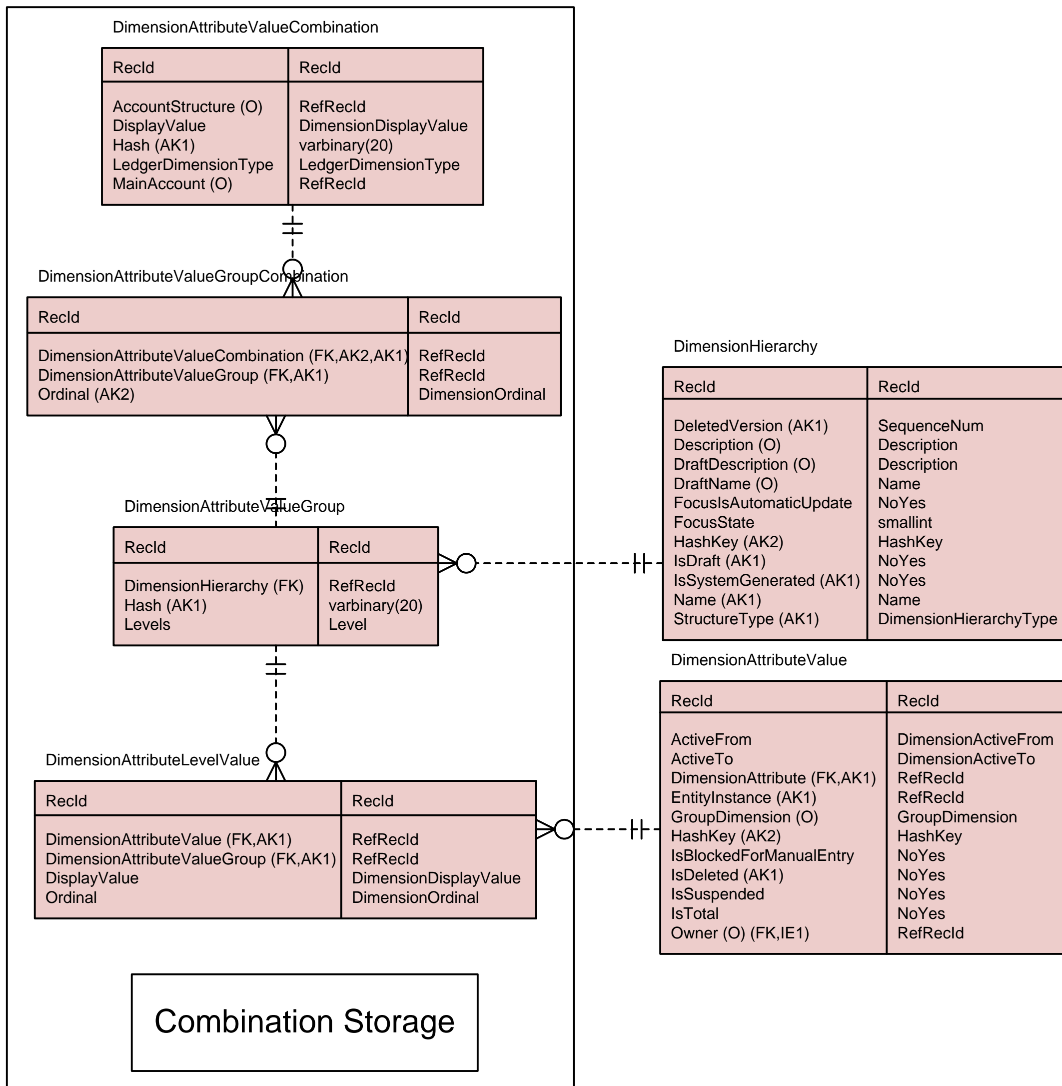
Updates since initial publication

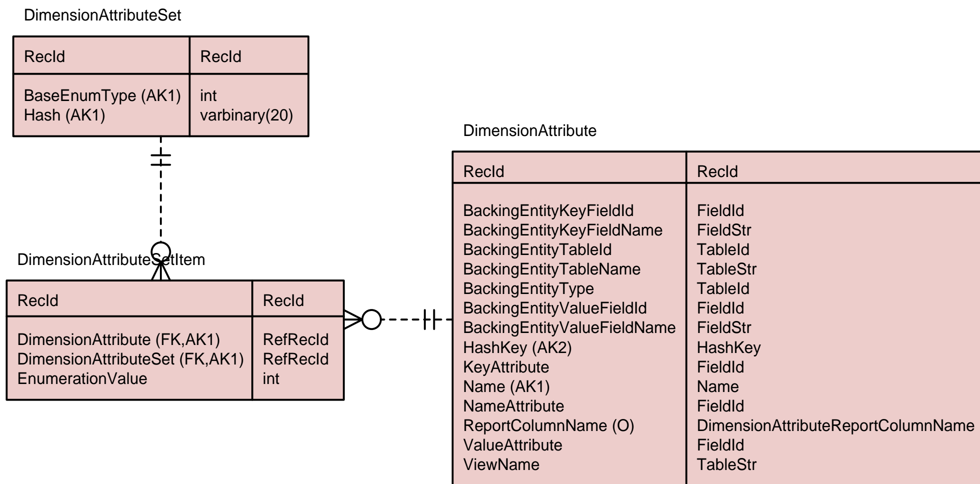
The following table lists changes made to this document after it was initially published.

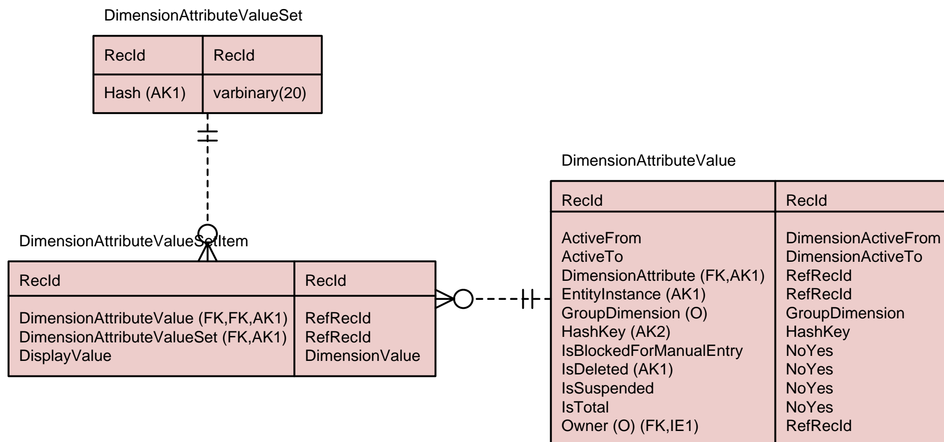
Date	Change
February 2014	Updated to address Microsoft Dynamics AX 2012 R3 and updated some earlier content for accuracy: • Added information about the budget planning validation that is part of the FinancialDimensionValidationService service. • Described the new FinancialDimensionValidationService functionality for all account types.
June 2013	Updated to address Microsoft Dynamics AX 2012 R2, and updated some earlier content for accuracy: • Added more information about how to make an entity dimensionable. • Corrected some misinformation regarding default accounts (structures were not associated with them). • Updated the description of MainAccountList for Microsoft Dynamics AX 2012 R2. Sharing of structures now occurs at the Ledger not Chart of Accounts level. • Described the DimensionDefaultingService call for fixed dimensions functionality.
August 2011	Initial publication

Appendix




DimensionAttributeValueCombination
RecId
RecId
AccountStructure (O)
DisplayValue
Hash (AK1)
LedgerDimensionType
MainAccount (O)
RefRecId
DimensionDisplayValue
varbinary(20)
LedgerDimensionType
RefRecId
DimensionAttributeValueGroupCombination
RecId
RecId
DimensionAttributeValueCombination (FK,AK2,AK1)
DimensionAttributeValueGroup (FK,AK1)
Ordinal (AK2)
RefRecId
RefRecId
DimensionOrdinal
DimensionAttributeValueGroup
RecId
RecId
DimensionHierarchy (FK)
Hash (AK1)
Levels
RefRecId
varbinary(20)
Level
DimensionAttributeLevelValue
RecId
RecId
DimensionAttributeValue (FK,AK1)
DimensionAttributeValueGroup (FK,AK1)
DisplayValue
Ordinal
RefRecId
RefRecId
DimensionDisplayValue
DimensionOrdinal
Combination Storage
DimensionHierarchy
RecId
RecId
DeletedVersion (AK1)
Description (O)
DraftDescription (O)
DraftName (O)
FocusIsAutomaticUpdate
FocusState
HashKey (AK2)
IsDraft (AK1)
IsSystemGenerated (AK1)
Name (AK1)
StructureType (AK1)
SequenceNum
Description
Description
Name
NoYes
smallint
HashKey
NoYes
NoYes
Name
DimensionHierarchyType
DimensionAttributeValue
RecId
RecId
ActiveFrom
ActiveTo
DimensionAttribute (FK,AK1)
EntityInstance (AK1)
GroupDimension (O)
HashKey (AK2)
IsBlockedForManualEntry
IsDeleted (AK1)
IsSuspended
IsTotal
Owner (O) (FK,IE1)
DimensionActiveFrom
DimensionActiveTo
RefRecId
RefRecId
GroupDimension
HashKey
NoYes
NoYes
NoYes
NoYes
RefRecId


DimensionAttributeSet
RecId
RecId
BaseEnumType (AK1)
Hash (AK1)
int
varbinary(20)
DimensionAttribute
RecId
RecId
BackingEntityKeyFieldId
BackingEntityKeyFieldName
BackingEntityTableId
BackingEntityTableName
BackingEntityType
BackingEntityValueFieldId
BackingEntityValueFieldName
HashKey (AK2)
KeyAttribute
Name (AK1)
NameAttribute
ReportColumnName (O)
ValueAttribute
ViewName
FieldId
FieldStr
TableId
TableStr
TableId
FieldId
FieldStr
HashKey
FieldId
Name
FieldId
DimensionAttributeReportColumnName
FieldId
TableStr
DimensionAttributeSetItem
RecId
RecId
DimensionAttribute (FK,AK1)
DimensionAttributeSet (FK,AK1)
EnumerationValue
RefRecId
RefRecId
int


DimensionAttributeValueSet
RecId
RecId
Hash (AK1)
varbinary(20)
DimensionAttributeValue
RecId
RecId
ActiveFrom
ActiveTo
DimensionAttribute (FK,AK1)
EntityInstance (AK1)
GroupDimension (O)
HashKey (AK2)
IsBlockedForManualEntry
IsDeleted (AK1)
IsSuspended
IsTotal
Owner (O) (FK,IE1)
DimensionActiveFrom
DimensionActiveTo
RefRecId
RefRecId
GroupDimension
HashKey
NoYes
NoYes
NoYes
NoYes
RefRecId
DimensionAttributeValueSetItem
RecId
RecId
DimensionAttributeValue (FK,FK,AK1)
DimensionAttributeValueSet (FK,AK1)
DisplayValue
RefRecId
RefRecId
DimensionValue

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.

© 2014 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.

Microsoft