

Microsoft Dynamics® AX 2012

Implementing the Tax Framework for Microsoft Dynamics AX 2012

White Paper

This document provides technical information about the sales tax APIs and common usage patterns.

Date: June 2011

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

Author: Jon Bye, Senior Developer, Finance

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



Table of Contents

Overview.....	3
Document Purpose	3
Audience.....	3
Committed Sales Taxes	4
Posted sales taxes data model.....	4
The TaxTrans table	4
The Source Document Framework.....	4
The TaxTransGeneralJournalAccountEntry link table	6
Sales Tax Classes and Common APIs	7
TaxCalculation class	7
TaxPost class.....	9
Committing sales taxes	10
Querying for a sales tax amount using an instance of the Tax class	11
Querying for a sales tax amount for a transaction that supports TaxUncommitted	11
TaxCalculation class details	12
TaxPost class details	13
Sales tax ledger dimension API changes	13
Overview	13
Dimension uptake instructions.....	14
Uncommitted Sales Tax.....	14
Pattern in Microsoft Dynamics AX 2009.....	14
TaxUncommitted data model.....	15
Uptake patterns.....	16
TaxUncommitted methods that can be used to delete	16
When to calculate sales tax	16
Implementing the Source Document Framework with Tax.....	18
Split distributions	18
Accounts.....	19
AccountNum	19
OperationAccount	19
ChargeAccount	19
TaxOffsetAccountUseTax	19
Performance and Legacy sales tax considerations	19
Legacy Tax	19
Legacy sales tax	20
TaxUncommitted without the Source Document Framework	20
Source Document Framework support	20
Performance considerations.....	20

Overview

Sales tax functionality in Microsoft Dynamics AX supports collecting, reporting, and paying sales taxes. Transactions in Microsoft Dynamics AX 2009 such as free text invoice supported only one account per invoice line. In Microsoft Dynamics AX 2012, the Source Document Framework was introduced, which allows an invoice line to have multiple split distributions. The sales tax data model has been enhanced to provide a stronger relational link between the posted sales tax table and the general ledger for the sales tax amount, the use tax payable amount, and the transaction line amount.

The posted sales tax table allows for sales tax to be reported, collected, and paid. This table will relate to the Source Document Framework to discover the impact of split distributions on the sales tax amount and how it is posted. For example, if an Xbox® 360 is purchased for USD 330, including USD 30 tax, the cost may be distributed evenly between departments A and B. There will still be one posted sales tax record for \$30, but by using the Source Document Framework we can discover that USD 15 of the sales tax was charged to department A and USD 15 was charged to department B.

Taxes on documents that were not posted in Microsoft Dynamics AX 2009 were not stored. This caused taxes to be recalculated every time an action was performed that required a tax amount. If a setup change occurred in Tax, this could cause tax amounts on a document to change unexpectedly. In Microsoft Dynamics AX 2012, unposted sales taxes are stored. This increases performance and prevents accidental changes to sales taxes on documents. This paper discusses patterns for how to use this new table.

A new API exists for how to implement the tax engine. This paper discusses the new API.

Document Purpose

This document provides common patterns for how to use sales tax. The following topics are discussed:

- Changes to the committed sales tax data model
- Changes to the Tax API
- Patterns for uncommitted sales taxes
- Sales tax and the Source Document Framework
- Legacy sales tax considerations

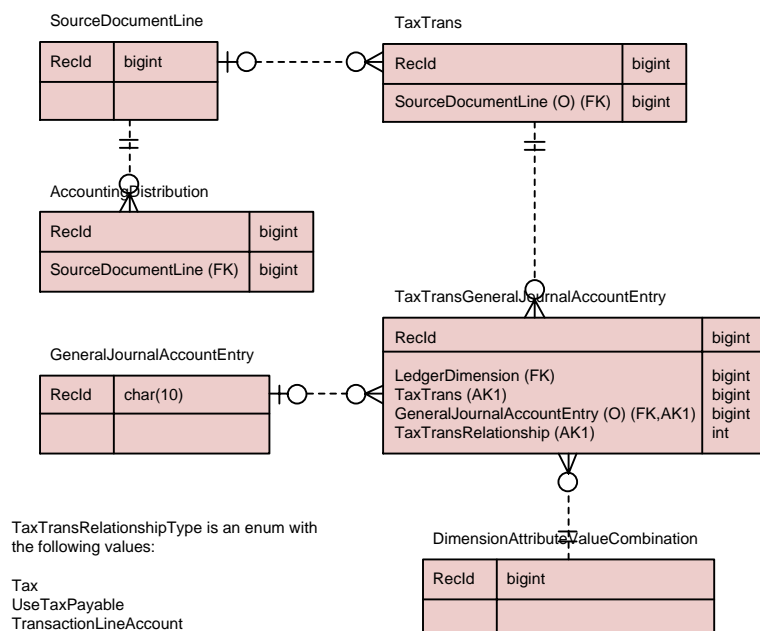
Audience

This white paper is intended for developers integrating their code with the tax framework in Microsoft Dynamics AX. It is assumed that the reader of this document is familiar with sales tax, the Source Document Framework, and the Ledger Dimensions functionality introduced in Microsoft Dynamics AX 2012.

Committed Sales Taxes

The following section describes the data model for posted sales taxes, the TaxTrans table, the Source Document Framework, and how to use the TaxTransGeneralJournalAccountEntry link table for transactions that do not support the Source Document Framework.

Posted sales taxes data model



The TaxTrans table

TaxTrans stores those sales taxes that are committed to a tax authority. When a transaction is posted, any sales taxes owed to a tax authority are posted to TaxTrans. TaxTrans is used heavily by tax reporting, transaction reporting, and the sales tax settlement process.

In Microsoft Dynamics AX 2009, TaxTrans stored several account numbers. Because of the Source Document Framework and multiple split distributions, these accounts no longer exist in TaxTrans but can be retrieved by joining to TaxTransGeneralJournalAccountEntry or to the AccountingDistribution table.

The Source Document Framework

The TaxTrans table foreign key to SourceDocumentLine is the same as the AccountingDistribution table foreign key to SourceDocumentLine for tax AccountingDistribution records. This allows for reporting scenarios that reflect how the sales tax amount was split across distributions and all ledger accounts. If the field SourceDocumentLine on TaxTrans has a value, it is preferable to use the Source Document Framework to gain access to accounts and amounts.

The following code retrieves the account and amount for the tax amount or the use tax expense:

```
while select LedgerDimension, TransactionCurrency, TransactionCurrencyAmount, SourceDocumentLine,
RecId from accountingDistribution
    where accountingDistribution.SourceDocumentLine == _taxTrans.SourceDocumentLine &&
           accountingDistribution.Type != AccountRole::Reversing &&
           accountingDistribution.MonetaryAmount == MonetaryAmount::Tax
{
    ...
}
```

The following code retrieves the account and amount for the nonrecoverable VAT amount:

```
while select LedgerDimension, TransactionCurrencyAmount from accountingDistribution
    where accountingDistribution.SourceDocumentLine == _taxTrans.SourceDocumentLine &&
           accountingDistribution.Type != AccountRole::Reversing &&
           accountingDistribution.MonetaryAmount == MonetaryAmount::TaxNonRecoverable
{
    ...
}
```

The following code retrieves the profit and loss ledger account by joining from the AccountingDistribution record for the TaxTrans record to the AccountingDistribution record for the transaction line that is the parent of the TaxTrans record:

```
while select * from accountingDistribution
    where accountingDistribution.SourceDocumentLine == _taxTrans.SourceDocumentLine &&
           accountingDistribution.Type != AccountRole::Reversing &&
           accountingDistribution.MonetaryAmount == MonetaryAmount::Tax
    join ledgerDimension from parentAccountingDistribution
        where parentAccountingDistribution.RecId == accountingDistribution.ParentDistribution
{
    ...
}
```

The following code retrieves the use tax payable account:

```
while select TaxCode, TaxDirection, SourceCurrencyCode, SourceDocumentLine from taxTrans
    where taxTrans.TaxDirection == TaxDirection::UseTax &&
           taxTrans.Voucher == _voucher &&
           taxTrans.TransDate == _transDate
    join RecId from accountingDistribution
        where accountingDistribution.SourceDocumentLine == TaxTrans.SourceDocumentLine &&
               accountingDistribution.Type != AccountRole::Reversing &&
               accountingDistribution.MonetaryAmount == MonetaryAmount::Tax
    join AccountingDistribution, SubledgerJournalAccountEntry from
subledgerJournalAccountEntryDistribution
    where subledgerJournalAccountEntryDistribution.AccountingDistribution ==
accountingDistribution.RecId
    join PostingType, RecId, LedgerDimension from subledgerJournalAccountEntry
        where subledgerJournalAccountEntry.RecId ==
subledgerJournalAccountEntryDistribution.SubledgerJournalAccountEntry &&
           subledgerJournalAccountEntry.PostingType == LedgerPostingType::Tax &&
           subledgerJournalAccountEntry.Side == DebitCredit::Credit
{
    ...
}
```

See the section called [Implementing the Source Document Framework with Tax](#) for details.

The TaxTransGeneralJournalAccountEntry link table

Not all transactions support the Source Document Framework. For transactions that do not support the Source Document Framework, the TaxTransGeneralJournalAccountEntry table provides information about which ledger accounts these transactions used.

The TaxTransGeneralJournalAccountEntry link table was introduced because in Microsoft Dynamics AX 2012, there is a many to many relationship between TaxTrans and GeneralJournalAccountEntry. This link table tracks multiple relationships between TaxTrans and GeneralJournalAccountEntry.

TaxTransGeneralJournalAccountEntry.TaxTransRelationship is used to identify the relationship. The relationships correspond to each of the accounts previously stored in TaxTrans. They are:

1. Tax. Specifies which GeneralJournalAccountEntry record contains the posted sales tax amount for the TaxTrans record. This replaces TaxTrans.AccountNum.
2. UseTaxPayable. Specifies which GeneralJournalAccountEntry record contains the posted use tax payable amount for the TaxTrans record. This replaces TaxTrans.TaxOffsetAccountUseTax.
3. TransactionLineAccount. For a TaxTrans record, specifies which GeneralJournalAccountEntry record contains the posted amount for the transaction line that is this tax line's parent. This replaces TaxTrans.OperationAccount and TaxTrans.TaxRefId.

Because the Source Document Framework allows splitting distributions for a single tax line, joining to TaxTransGeneralJournalAccountEntry may now return multiple records. Any existing reports, code, etc. that rely on a single account from TaxTrans will need to be refactored to handle multiple accounts properly.

Here is an example of how to join to the GeneralJournalAccountEntry to get the ledger accounts and amount:

```
TaxTrans taxTrans;
TaxTransGeneralJournalAccountEntry taxTransGeneralJournalAccountEntry;
GeneralJournalAccountEntry generalJournalAccountEntry;
DimensionAttributeValueCombination dimensionAttributeValueCombination;

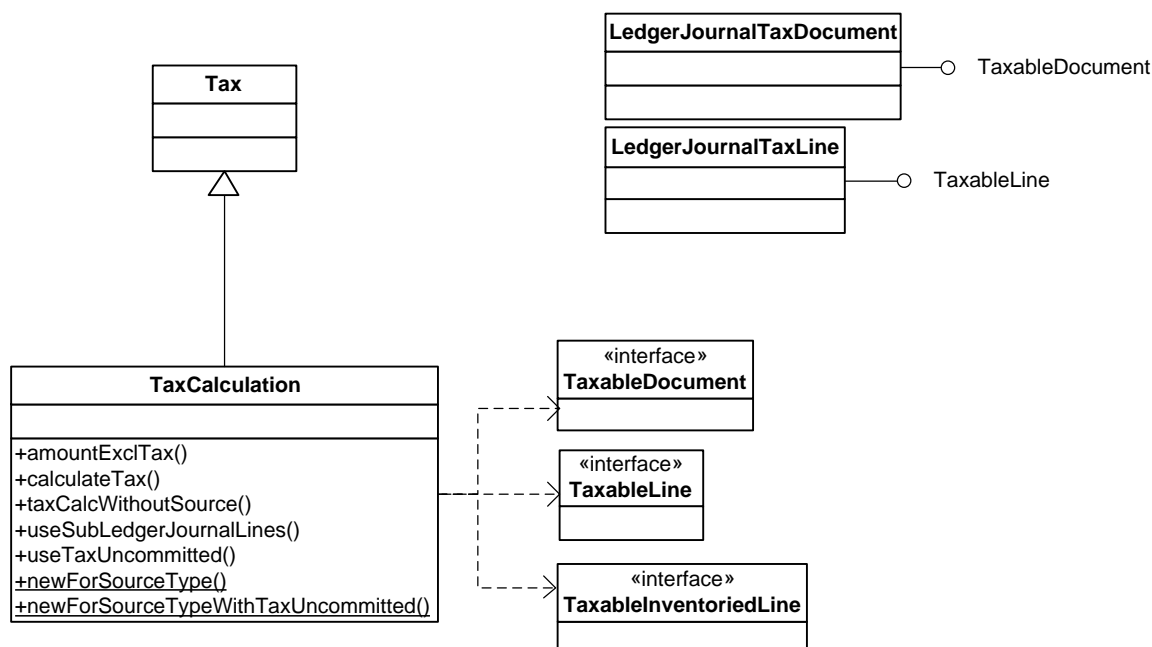
while select RecId from taxTrans
    where taxTrans.Voucher == _voucher &&
           taxTrans.TransDate == _transDate
    join RecId from taxTransGeneralJournalAccountEntry
    where taxTransGeneralJournalAccountEntry.TaxTrans == taxTrans.RecId &&
           taxTransGeneralJournalAccountEntry.TaxTransRelationship == TaxTransRelationshipType::Tax
    join TransactionCurrencyAmount, RecId from generalJournalAccountEntry
    where taxTransGeneralJournalAccountEntry.GeneralJournalAccountEntry ==
generalJournalAccountEntry.RecId
    join DisplayValue from dimensionAttributeValueCombination
    where dimensionAttributeValueCombination.RecId == generalJournalAccountEntry.LedgerDimension
{
    ...
}
```

Sales Tax Classes and Common APIs

The following section describes the TaxCalculation class, the TaxPost class, querying for a sales tax amount using an instance of the Tax class, querying for a sales tax amount for a transaction that supports TaxUncommitted, and provides details about the TaxCalculation and TaxPost classes.

TaxCalculation class

The class diagram depicted above represents the main classes required for calculating taxes that tax developers may need to use.



TaxCalculation is a new class that contains the main APIs used for sales tax calculations. Each transaction that integrates with Tax must implement the interfaces depicted in the diagram to use TaxCalculation. Only free text invoices, purchasing-based documents, and the journals support TaxCalculation. All other existing transactions use legacy methods of calculating taxes. We recommend that all new transactions use TaxCalculation.

Before a transaction can support TaxCalculation, it must implement the interfaces TaxableDocument, TaxableLine, and, optionally, TaxableInventoriedLine. The sales tax calculation engine will use these interfaces to request the required data from the transaction for tax calculation purposes. For Microsoft Dynamics AX 2012, only the journals have implemented the interfaces TaxableDocument and TaxableLine. All other transactions that support TaxCalculation use legacy means of communicating with Tax.

It is preferable for most customizations to be done in the classes that implement the Taxable interfaces. If this isn't sufficient, a derived class extending TaxCalculation can be used. For example, the journals have quite a bit of custom tax logic not supported by other transactions so the class TaxCalculationJournal was required.

The table below shows the derived classes by transaction. Transactions whose derived classes inherit from TaxCalculation should always declare an instance of TaxCalculation instead of the derived class.

Transaction	Transaction specific Tax Calculation Class	Inherits from Tax Calculation	Uses Taxable Interfaces
Bank	TaxBankAccountReconcile	No	No
Manufacturing	TaxBudget	No	No
Budget	TaxBudgetTransaction	No	No
Free Text Invoice	TaxFreeInvoice	Yes	No
PO Documents	TaxPurch	Yes	No
Customer Collection Letters	TaxCustCollectionLetter	No	No
Customer Interest Notes	TaxCustInterestNote	No	No
Journals	TaxCalculation	Yes	Yes
Payment Fees	TaxPaymManFee	No	No
Project	TaxProj	No	No
Sales Order Documents	TaxSales	No	No
Sales Basket	TaxSales_Basket	No	No
Sales Quotations	TaxSalesQuotations	No	No
Expense	TrvTaxExpense	Yes	No

Here is some example code used by free text invoice to get a tax amount. It is important to note that we only calculate taxes if they haven't been calculated yet:

```

ledgerVoucher = custInvoiceTable.custInvoiceJour().LedgerVoucher;

if (!TaxTrans::exist(ledgerVoucher, custInvoiceTable.InvoiceDate, this.transTransId()))
{
    loadTaxUncommitted =
    TaxUncommitted::existByDocumentId(tablenum(CustInvoiceTable), custInvoiceTable.RecId);

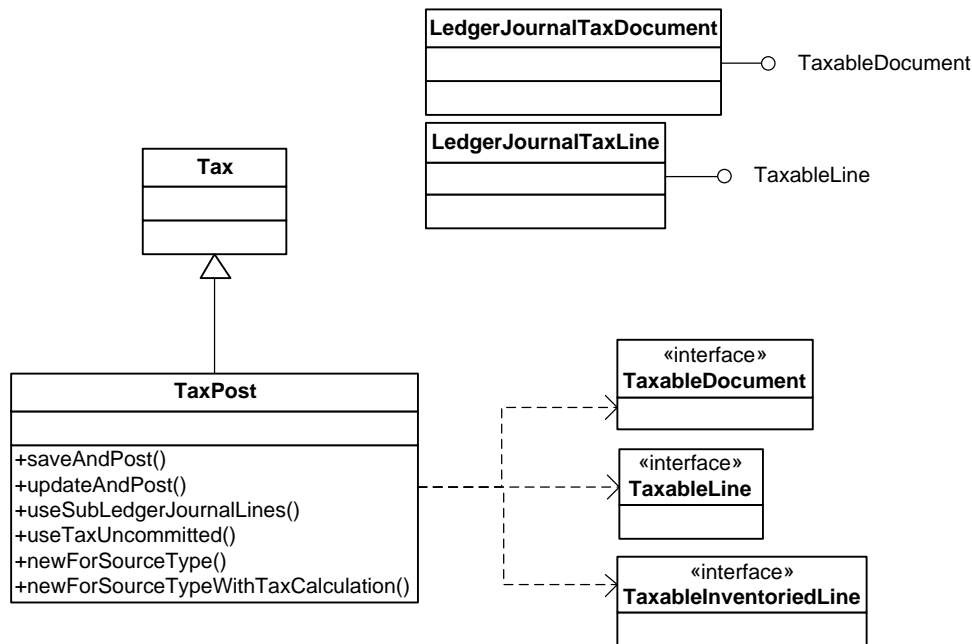
    taxCalculation =
    TaxCalculation::newForSourceTypeWithTaxUncommitted(TaxSourceType::FreeTextInvoice, this,
    loadTaxUncommitted, false);

    if (!loadTaxUncommitted)
    {
        amountCur = taxCalculation.calculateTax();
    }
    else
    {
        amountCur = taxCalculation.totalTaxAmount();
    }
}
else
{
    amountCur = Tax::taxTotal(ledgerVoucher, custInvoiceTable.InvoiceDate);
}

```


TaxPost class

The following diagram describes the TaxPost class.



TaxPost is an abstract class that provides functionality used to commit TaxUncommitted records and/or TmpTaxWorkTrans records to TaxTrans. For transactions that do not use the Source Document Framework, TaxPost also post tax amounts to the general ledger. Not all transactions can support TaxPost. See the table below for information concerning which class can be used to commit sales taxes.

Before a transaction can support TaxPost, it must implement the interfaces TaxableDocument, TaxableLine, and, optionally, TaxableInventoriedLine. TaxPost will use these interfaces to request the required data from the transaction for sales tax posting. Only the journals have implemented the Taxable interfaces. All other transactions that support TaxPost use legacy means of communicating with Tax.

Transaction	Transaction specific Tax Posting Class	Inherits from TaxPost
Bank	TaxBankAccountReconcile	No
Manufacturing*	N/A	N/A
Budget*	N/A	N/A
Free Text Invoice	TaxFreeInvoice_Invoice	Yes
Purchasing Documents	TaxPurchInvoice	Yes
Customer Collection Letters	TaxCustCollectionLetter	No
Customer Interest Notes	TaxCustInterestNote	No
Journals	TaxPostJournal	Yes

Payment Fees	TaxPaymManFee	No
Project	TaxProjInvoice	No
Sales Order Documents	TaxSalesInvoice	No
Sales Basket*	N/A	N/A
Sales Quotations*	N/A	N/A
Expense*	N/A	N/A

*Not all transactions commit sales taxes to TaxTrans.

Committing sales taxes

Sales tax posting assumes that the source document has already calculated taxes. Tax posting supports two posting scenarios:

1. Posting sales tax lines from TmpTaxWorkTrans. Some source documents may have an instance of the Tax class that already has the tax lines loaded into TmpTaxWorkTrans. This may be because the transaction calculated or recalculated taxes during posting, or had other functionality that required the tax lines to be in memory. In this case, it is faster to post using the memory table TmpTaxWorkTrans than to post from TaxUncommitted. The following code shows how to post sales taxes using an instance of the TaxCalculation class:

```
TaxPost taxPost = TaxPost::newForSourceTypeWithTaxCalculation(TaxSourceType::PurchaseOrder,
this, _post, taxCalculation);
taxPost.updateAndPost(LedgerPostingController::newForLedgerPostingJournal(_ledgerVoucher));
```

2. Posting sales tax lines from TaxUncommitted. If a transaction does not already have the tax lines loaded into TmpTaxWorkTrans, they can be posted using TaxUncommitted.

```
TaxPost taxPost;
LedgerJournalTaxDocument ledgerJournalTaxDocument;

ledgerJournalTaxDocument =
LedgerJournalTaxDocument::constructForPosting(ledgerJournalTable.JournalNum);
taxPost = TaxPost::newForSourceType(TaxSourceType::Journals, ledgerJournalTaxDocument, NoYes::Yes);
postedTaxAmount = taxPost.updateAndPost(_ledgerPostingController);
```

Querying for a sales tax amount using an instance of the Tax class

There are many existing methods on the Tax class that can be used to return a tax amount for various scenarios. The following table provides some recommended functions.

Tax Method	Description
totalTaxAmount	Returns the actual transactional total sales tax amount, excluding Use tax, for the transaction from TmpTaxWorkTrans. The TmpTaxWorkTrans records are created by loading them from TaxUncommitted or calculating sales taxes using the TaxCalculation class.
totalTaxAmountCalculated	Returns the calculated transactional total tax amount, including Use tax, from TmpTaxWorkTrans for a document.
totalTaxAmountSingleLine	Returns the actual total sales tax amount for a transaction line from TmpTaxWorkTrans. Use tax can optionally be included in the total. The caller can choose whether the amount will have the correct sign for display, or the correct sign for the database. This requires that sales taxes have been calculated or loaded from TaxUncommitted.
taxTotalVoucherDate	Static method that returns the actual sales tax total for the given voucher and date, excluding Use tax, from TaxTrans. Use this if a sales tax total is needed for a transaction that has been posted.

The term “actual” sales tax amount implies that tax adjustments have been included in the tax amount. The term “calculated” sales tax amount implies that tax adjustments have not been included in the tax amount.

Querying for a sales tax amount for a transaction that supports TaxUncommitted

Retrieving a sales tax amount for transactions that do not support TaxUncommitted can be an expensive process because calculating sales taxes is required. For Microsoft Dynamics AX 2012, those transactions that support TaxUncommitted can retrieve sales tax amounts via simple queries to TaxUncommitted. This can be considerably faster.

For some transactions, sales tax calculations may not have occurred yet, so the general rule is that if TaxUncommitted is empty, sales tax calculations will be required prior to retrieving a sales tax amount.

For example, the following code will get the sales tax amount for a single Journal line:

```
return TaxUncommitted::getActualTaxAmountForSourceLine(_ledgerJournalTrans.TableId,  
_ledgerJournalTrans.RecId, false, true);
```

The following example will get the sales tax amount for an entire voucher:

```
taxAmount = TaxUncommitted::getActualTaxAmountForVoucher(ledgerJournalTable.TableId,  
ledgerJournalTable.RecId, _ledgerJournalTrans.Voucher, false);
```

TaxCalculation class details

Tax Method	Description
calculateTax()	The main entry point for calculating taxes. If this instance was initialized using <code>TaxCalculation::newForSourceTypeWithTaxUncommitted()</code> , this will result in all <code>TaxUncommitted</code> records being deleted and replaced with a new set of <code>TaxUncommitted</code> records. The deletion process will ensure that all <code>SourceDocumentLine</code> records, <code>SourceDocumentDistributions</code> records, etc. are removed.
amountExclTax()	<p>This existing method determines what portion of a transaction amount does not include sales tax when a tax is included in the item price. This method cannot return a valid rounded amount in all scenarios. This method may be used to return an estimate. Any existing calls to this method that result in a rounded line amount excluding sales taxes that require an exact amount will need to be removed. It is preferable to calculate sales taxes and use the origin from <code>TmpTaxWorkTrans</code> or <code>TaxUncommitted</code>. The following code is the proper way to retrieve the line amount excluding tax from <code>TaxUncommitted</code>:</p> <pre>taxOrigin = TaxUncommitted::getOriginForSourceLine(this.TableId, this.RecId, true);</pre> <p>The following code is the proper way to retrieve the line amount excluding sales tax from an instance of <code>Tax</code>:</p> <pre>taxOrigin = _tax.getOriginForSourceLine(this.TableId, this.RecId, true);</pre>
taxCalcWithoutSource	This method will estimate the sales tax based on the passed in parameters. Since this method only considers a single transaction line, and not an entire transaction, it will not be correct in some scenarios such as tax based on invoice total. Therefore, this method should only be used in scenarios where an estimate is required.
useTaxUncommitted	Returns true if the calling transaction uses <code>TaxUncommitted</code> .
useSubLedgerJournalLines	Returns true if the calling transaction uses the Source Document Framework.
newForSourceType	Creates an instance of <code>TaxCalculation</code> that will not use <code>TaxUncommitted</code> . Transactions that support <code>TaxUncommitted</code> have reasons for wanting to calculate sales taxes without <code>TaxUncommitted</code> . Typically, this is to show a particular sales tax amount for a special scenario. For example, in purchase order, you can show document totals for the different quantities that the purchase order supports. This implies that sales taxes must be calculated using these different quantities. In this scenario, it is not desirable to create <code>TaxUncommitted</code> records for these different types of quantities.
newForSourceTypeWithTaxUncommitted()	<p>Creates an instance of <code>TaxCalculation</code> using <code>TaxUncommitted</code>. The transaction ultimately has control over whether it supports <code>TaxUncommitted</code>. If <code>useTaxUncommitted</code> returns false, the use of this method to construct an instance of <code>TaxUncommitted</code> will not result in <code>TaxUncommitted</code> records being generated for a transaction.</p> <p>Transactions that support <code>TaxUncommitted</code> sometimes have reasons for wanting to calculate taxes without <code>TaxUncommitted</code>. Typically, this is to show a particular sales tax amount for a special scenario. For example, in purchase orders, you can show document totals for the different quantities that a purchase order supports. This implies that sales taxes must be calculated using these different quantities. In this scenario, it is not desirable to create <code>TaxUncommitted</code> records for these different types of quantities.</p>

taxExists	Creates an instance of TaxCalculation using TaxUncommitted. The transaction ultimately has control over whether it supports TaxUncommitted. If useTaxUncommitted returns false, the use of this method to construct an instance of TaxUncommitted will not result in TaxUncommitted records being generated for a transaction.
-----------	--

TaxPost class details

Tax Method	Description
updateAndPost()	This is the main entry point for sales tax posting. This method will ensure that all sales taxes are posted to TaxTrans and will ensure that other transaction-specific logic occurs. For example, when TaxUncommitted records are posted to TaxTrans, an ownership change occurs. A TaxUncommitted record's parent may be PurchParmLine, but TaxTrans records must be owned by something posted – not a source document. So, in this example, this method changes the ownership so that TaxTrans records will be owned by VendInvoiceTrans.
saveAndPost()	This is a legacy method used by some transactions. This method will be deprecated. All transactions need to start using updateAndPost().
useTaxUncommitted	Returns true if the calling transaction uses TaxUncommitted.
useSubLedgerJournalLines	Returns true if the calling transaction uses SubLedgerJournal Lines.
newForSourceType	Creates an instance of TaxPost that will not use TaxUncommitted. If the transaction, during transaction posting, has an instance of Tax with all the tax lines loaded into memory (TmpTaxWorkTrans), it would be faster to use this constructor. This constructor will tell TaxPost to post directly from TmpTaxWorkTrans to TaxTrans. This is not a recommended scenario. See the performance section later in this document for details.
newForSourceTypeWithTaxUncommitted()	Creates an instance of TaxPost using TaxUncommitted. This will result in TaxTrans records being created directly from TaxUncommitted. If the transaction supports the Source Document Framework, the use of this constructor is required.

Sales tax ledger dimension API changes

The following section describes the changes to the API.

Overview

Because some of the APIs in Tax take financial dimensions and ledger accounts as parameters, these APIs have been modified as a part of the Microsoft Dynamics AX 2012 financial dimensions feature. In Microsoft Dynamics AX 2009 four account numbers and a dimension field were stored in TmpTaxWorkTrans and TaxUncommitted. These fields are being replaced with four LedgerDimension fields.

Field	Microsoft Dynamics AX 2009 Field	Microsoft Dynamics AX 2012 Field
Tax Account	AccountNum	LedgerDimension
Use Tax Payable Account	TaxOffsetAccountUseTax	TaxOffsetUseTaxLedgerDimension
Profit & Loss Account	OperationAccount	OperationLedgerDimension
Expense Account	ChargeAccount	OperationLedgerDimension

When the calculation engine calculates sales taxes, Tax takes a DefaultDimension and an OperationLedgerDimension. The OperationLedgerDimension is the LedgerDimension used on the transaction line that is the parent of the tax line. The DefaultDimension is the default dimension from the transaction line. Internally, the sales tax calculation engine looks up the LedgerDimensionDefaultAccount instances from sales tax setup (TaxLedgerAccountGroup table) for each tax line. These are the sales tax default accounts and, optionally, the Use Tax Payable default account. The passed in DefaultDimension is applied to these LedgerDimensionDefaultAccount instances to create the LedgerDimension and the TaxOffsetUseTaxLedgerDimension. Tax does not store the DefaultDimension.

The expense account in Microsoft Dynamics AX 2009 (ChargeAccount) should always hold the same values as the OperationAccount, so the field was removed in Microsoft Dynamics AX 2012.

Dimension uptake instructions

Every transaction that uptakes the new LedgerDimension types will need to modify the transaction-specific tax classes that apply to their transaction. See the table above for the list of classes. In Microsoft Dynamics AX 2012, all these classes have been modified.

Tax has two protected methods named Tax.insertIntersection() and Tax.insertLineInInternal(). Tax.insertIntersection() previously took the dimension as a parameter. This parameter has been removed. Tax.insertLineInInternal() has been modified to take the DefaultDimension and the OperationLedgerDimension. The transaction specific sales tax classes call into Tax.insertLineInInternal() and pass the DefaultDimension and the OperationLedgerDimension.

For more information, see the section [Implementing the Source Document Framework with Tax](#).

Uncommitted Sales Tax

Uncommitted sales taxes are those taxes which are not yet committed to a taxing authority. Typically, unposted transactions such as sales orders and free text invoices contain uncommitted sales taxes. The term “uncommitted” does not refer to whether the tax amounts are committed to the ledger. For encumbrance or budget purposes, uncommitted taxes may have ledger postings. When sales taxes have been committed to being reported to a taxing authority, the taxes are no longer considered uncommitted.

Pattern in Microsoft Dynamics AX 2009

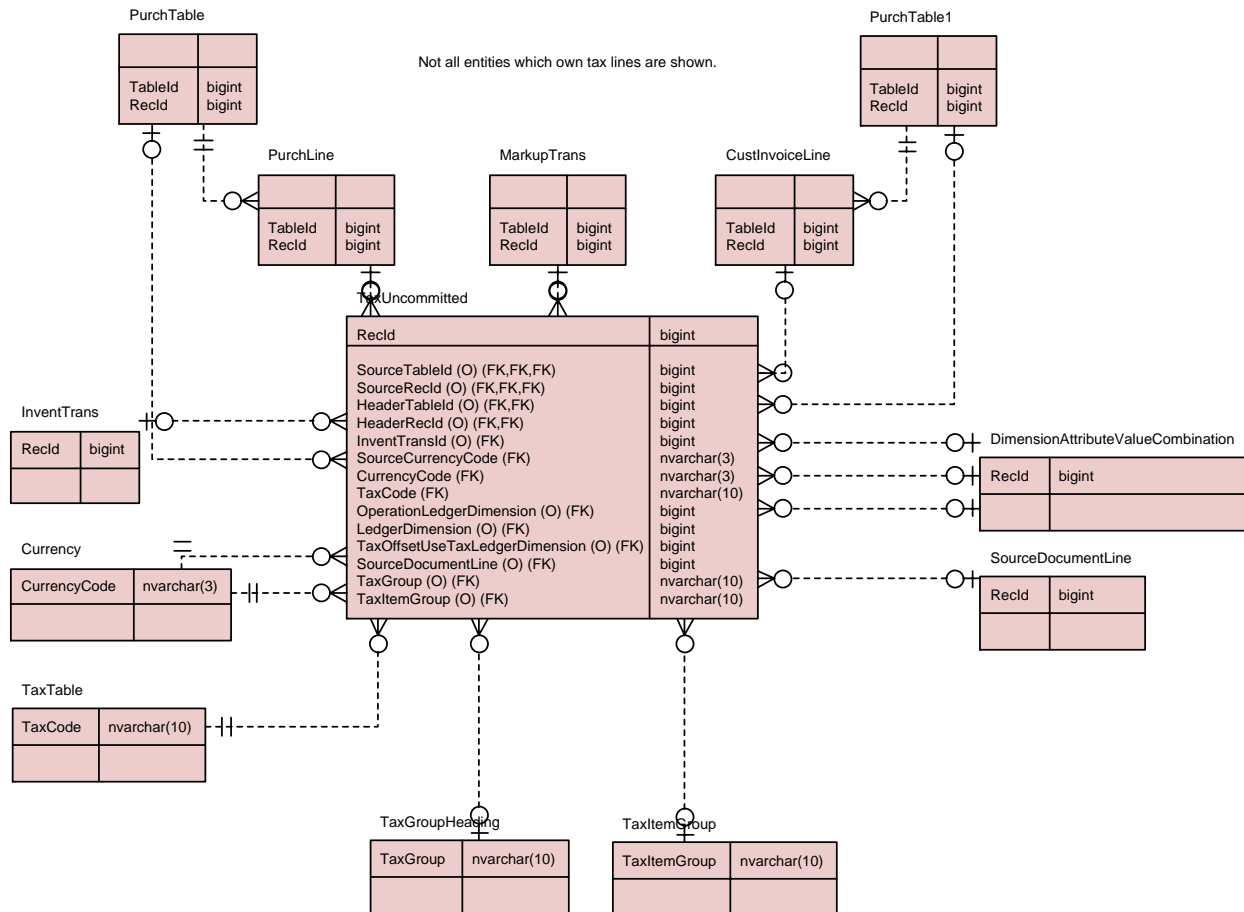
In Microsoft Dynamics AX 2009, uncommitted sales taxes are not persisted. These sales tax amounts are calculated as required by transactions to support the user interface, reporting, and posting needs. There are a number of problems with this approach.

Tax calculations are expensive and repeatedly recalculating sales taxes drains network, database, and server resources.

- The tax calculation process is dependent upon a number of sales tax and module settings. It is possible that at posting, a different sales tax amount could be calculated than what the user viewed on the transaction prior to posting. In some cases, if the user has already quoted a total to a customer or printed an invoice, this is not desirable.
- Calculating taxes during posting negatively affects posting performance.

TaxUncommitted data model

The following diagram describes the TaxUncommitted data model.



In an effort to address the issues with the pattern in Microsoft Dynamics AX 2009, we introduced the table TaxUncommitted in Microsoft Dynamics AX 2012. This table stores sales taxes that are not yet committed to a tax authority. The table has most of the same fields as TmpTaxWorkTrans and TaxTrans.

When a transaction that supports TaxUncommitted is posted, the TaxUncommitted records are deleted, and TaxTrans records are inserted. There are exceptions to this for purchase order/purchase order confirmations.

The use of TaxUncommitted for Microsoft Dynamics AX 2012 is considered optional. For those transactions that have not implemented TaxUncommitted, the tax calculation and posting processes work much as they do in Microsoft Dynamics AX 2009.

The tax lines in TaxUncommitted are persisted in detail. This implies that each transaction line owns 0 or more records in TaxUncommitted. There is one line for each tax code. TaxUncommitted contains the fields SourceRecId and SourceTableId, which together make up a foreign key to the transaction's line table that owns this tax. This is TaxUncommitted's

parent. This is typically used to discover what the total sales tax amount is for a transaction line.

TaxUncommitted also contains the fields HeadingRecId and HeadingTableId, which together make up a foreign key to the transaction's header table. This is TaxUncommitted's grandparent. This is typically used to discover the total sales tax for a given transaction.

TaxUncommitted also contains a foreign key to SourceDocumentLine. For details, see the section below titled [Implementing the Source Document Framework with Tax](#).

Uptake patterns

Transactions that uptake TaxUncommitted need to consistently follow the uptake pattern for TaxUncommitted. TaxUncommitted as it applies to a specific transaction that supports sales taxes has two valid states.

1. Sales taxes have been calculated and the TaxUncommitted records are correct.
2. TaxUncommitted records do not exist for the transaction. This can be due to one of three reasons:
 - a. Sales taxes have not yet been calculated.
 - b. TaxUncommitted records did exist but a document change has occurred, which resulted in the TaxUncommitted records being made obsolete and thus deleted.
 - c. Sales taxes have been calculated but no taxes apply.

Important: It is not valid for TaxUncommitted records to contain tax records that are out of date. It is the calling transaction's responsibility to delete TaxUncommitted records when a document change occurs that affects sales taxes. Do not recalculate sales taxes just because they have been deleted. Wait until a tax amount is needed before recalculating taxes.

TaxUncommitted methods that can be used to delete

There are several methods on the table TaxUncommitted that can be used to delete records.

Tax Method	Description
deleteForDocumentHeader()	Deletes all TaxUncommitted records for a given document.
deleteForDocumentLine()	This supports a specific scenario and should not be used. Deleting taxes for a single line will corrupt the tax information on the document in most scenarios.
deleteForInvoice()	Used for journal scenarios to delete taxes for a single invoice.

The timing of when to calculate taxes is also an important consideration. We do not calculate sales taxes if TaxUncommitted records exist. Instead, we use the TaxUncommitted records. If TaxUncommitted records do not exist and a tax amount is required, calculating sales taxes will be required.

When to calculate sales tax

Calculating sales taxes can be an expensive operation. For transactions that have a lot of transaction lines, we do not recommend calculating sales taxes prior to or at transaction line save. Instead, calculate sales taxes in a background process when the user is finished with the document. Other valid times to calculate sales taxes are when the user initiates an

action that requires a sales tax amount. For example, if the user selects to view document totals, sales taxes may need to be calculated.

The reason for not calculating sales taxes on line save is that taxes may get calculated based on invoice total. This tax amount is then distributed back to all the lines in the document that calculate that tax code. This implies that adding a fifth line to a document could change the tax calculated for the other four transaction lines. If we calculated taxes on line save, this results in too much server, database, and network bandwidth usage.

A transaction must perform the following steps to implement TaxUncommitted:

1. Implement the Taxable interfaces defined above.

2. Ensure that the transaction-specific tax posting class inherits from TaxPost.

The Tax base class contains a method named `useTaxUncommitted()`. The transaction-specific tax classes for both calculation and posting must override this method and return true to enable TaxUncommitted support.

3. In transaction calculation and posting code, declare instances of TaxCalculation and TaxPost instead of using instances of the legacy transaction-specific tax class.
4. The transaction has the ability to control when to persist tax information to TaxUncommitted. `TaxCalculation::newForSourceType()` is the constructor to use when the transaction does not want to persist taxes to TaxUncommitted. `TaxCalculation::newForSourceTypeWithTaxUncommitted()` is the constructor to use when the transaction does want taxes to persist to TaxUncommitted. Generally, the transaction will want to use the constructor that supports TaxUncommitted, but there are scenarios where a transaction that supports TaxUncommitted may not want the results of a tax calculation to be persisted.
5. When a document change occurs that affects taxes, delete the TaxUncommitted records. This is typically implemented using the table insert/update/delete events for the transaction line and transaction header. The following code deletes all TaxUncommitted records. The Source Document Framework is updated properly via this call:

```
TaxUncommitted::deleteForDocumentHeader (vendInvoiceInfoTable.TableId, vendInvoiceInfoTable.RecId, true, true);
```

6. Posting tax for transactions that use the Source Document Framework. See the section [Implementing the Source Document Framework with Tax](#) for details.

Implementing the Source Document Framework with Tax

To use the Source Document Framework for transactions, you must understand how Tax and the Source Document Framework interact. It is also helpful to understand the impact that Tax has on the Source Document Framework feature.

A transaction must first implement the TaxUncommitted. For details about this process, see the section [TaxUncommitted](#) earlier in this document.

TaxUncommitted has a foreign key to the SourceDocumentLine table. This allows access to the AccountingDistributions for the given TaxUncommitted record. It was stated earlier in this document that we do not recommend calculating sales taxes during transaction entry time unless the user initiates an action that requires a sales tax amount. If sales taxes do get calculated during data entry time, we do not recommend that the AccountingDistributions be created for tax during data entry time. Since taxes are deleted or recreated for the entire document as modifications to the document occur, trying to include the AccountingDistributions in this delete/recreate pattern results in too much database and network traffic.

The TaxableDocument interface has a method named useSourceDocumentFramework(). This must return true, which will enable various behavior differences in Tax that are specific to the Source Document Framework.

The first major behavior difference is that every TaxUncommitted record will result in the creation of a new Source Document Line record. The foreign key to that SourceDocumentLine is on TaxUncommitted, TmpTaxWorkTrans, and TaxTrans. When the transaction is posted, TaxTrans becomes the new owner of the SourceDocumentLine foreign key.

The next behavior difference is that Tax no longer posts tax amounts directly to the ledger. Instead, the Source Document Framework handles the posting to the ledger. Tax continues to post to TaxTrans as it has in the past.

Split distributions

The Source Document Framework allows for a transaction line to split an amount across multiple distributions with different account/dimension combinations. This feature has a big impact on tax.

It is important to understand that Tax does not calculate on these distributions. Instead, Tax calculates on the transaction line as it did in Microsoft Dynamics AX 2009. Thus, the records in TaxUncommitted and TaxTrans are not split the way the distributions are split. In fact, how the distributions are split does not affect TaxUncommitted or TaxTrans.

This feature does affect the distributions owned by TaxUncommitted or TaxTrans. These distributions will be split following the transaction lines distributions. Splitting distributions has the following impact on Tax:

1. US Rules On, Tax Direction Incoming. Tax distributions split the same way using the same main account and dimensions as the transaction lines.
2. All other scenarios. Tax distributions split the same way using a single tax account as the main account but the dimensions and split percentage come from how the transaction line is split.

Accounts

In Microsoft Dynamics AX 2009, for the tax account, the relationship between TaxTrans and LedgerTrans was many to one. Now, because of split distributions, the relationship between TaxTrans and LedgerEntry is many to many. To resolve this, a new table named TaxTransGeneralJournalAccountEntry was introduced.

TaxUncommitted stores several accounts. For transactions that implement the Source Document Framework, these accounts are no longer used because the accounts listed below could have multiple values for a single TaxUncommitted record. Instead, the Source Document Framework or the TaxTransGeneralJournalAccountEntry table should be used to get these values.

AccountNum

For most scenarios, this is typically the Use Tax Expense, the Sales Tax Receivable, or the Sales Tax Payable account(s) from Ledger Posting Groups. Tax loads the default account from the TaxLedgerAccountGroup table. The default dimensions from the transaction line are applied to this default account to form the ledger account.

OperationAccount

These are the revenue/expense accounts from the transaction line that is being taxed.

ChargeAccount

This account is no longer required and is being removed.

TaxOffsetAccountUseTax

These are the Use tax payable account.

Performance and Legacy sales tax considerations

The section discusses some performance tips as well as support for legacy sales tax scenarios.

Legacy Tax

Now, in Microsoft Dynamics AX 2012, Tax supports the following three scenarios:

1. Legacy sales tax usage. This is how sales tax functions in Microsoft Dynamics AX 2009. The transaction-specific tax class does not inherit from TaxCalculation and the transaction does not implement TaxUncommitted, nor does the transaction make use of the Source Document Framework.
2. The transaction-specific tax class inherits from the new TaxCalculation base class and has implemented the Taxable interfaces described above. The transaction has implemented TaxUncommitted but has not implemented the Source Document Framework.
3. The transaction-specific tax class inherits from the new TaxCalculation base class and has implemented the Taxable interfaces described above. The transaction has implemented TaxUncommitted and supports the Source Document Framework.

Legacy sales tax

For Microsoft Dynamics AX 2012, this scenario will continue to be supported because some transactions will fall into this category.

TaxUncommitted without the Source Document Framework

Before a transaction can implement the Source Document Framework, the transaction must implement TaxUncommitted. The journals are an example of this.

Source Document Framework support

Free text invoice is an example of a sales tax enable transaction that supports the Source Document Framework.

Performance considerations

This section is a list of performance tips to consider when implementing sales taxes. Some of the points in this section have been mentioned previously in this document but it is useful to have them listed in one place. This is by no means a comprehensive list.

- For transactions that have a large number of transaction lines, do not calculate sales taxes on line save. Taxes are calculated on an entire document, so the addition of a new transaction line can conceivably change the sales tax amounts for every line in the transaction. When a large number of transaction lines are involved, there can be a significant delay.
- When using TaxUncommitted, ensure that sales taxes are calculated prior to posting. The reason for this is so that posting does not have to calculate sales taxes. The best time to do this is when the user is finished with the document. Ideally, this could be done as a background process.
- Related to the previous point, during posting, if TaxUncommitted records exist, do not recalculate sales taxes. Instead, use the existing TaxUncommitted records to post to TaxTrans and to retrieve sales tax amounts for posting needs.
- Tax supports loading TaxUncommitted records into TmpTaxWorkTrans. This is done by passing in true for the parameter `_loadTaxUncommitted` of `TaxCalculation::newForSourceTypeWithTaxUncommitted()`. The use of this should be avoided if possible. Ideally, re-factor any existing code to remove the dependency on having an instance of Tax that has the tax lines loaded into TmpTaxWorkTrans. The methods on TaxUncommitted can be used instead to retrieve tax amounts or to perform any necessary logic. TaxPost will post to TaxTrans using TaxUncommitted.

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

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

Worldwide +1-701-281-6500

www.microsoft.com/dynamics

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

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

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

© 2011 Microsoft Corporation. All rights reserved.

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

All other trademarks are property of their respective owners.

Microsoft