Microsoft Dynamics® AX 2012

Implementing the Budget Control Framework for Microsoft Dynamics AX 2012 Applications

White Paper

This document highlights the new budget control framework that was added to Microsoft Dynamics AX 2012. Topics discussed include key integration patterns for new financial frameworks, how to make budget requests, and how to use the ledger budget control framework with core integration patterns.

Date: July 2011

http://microsoft.com/dynamics/ax

Author: Kim Kroetsch, Senior Development Lead

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

Table of Contents

Overview

In Microsoft Dynamics® AX 2009, there was no provision for budget control. By introducing a ledger budget control framework, Microsoft Dynamics AX 2012 now offers budget control solutions for public and private sector organizations.

To support the new functionality, the ledger budget control framework has been integrated with the ledger, the new accounting distributions framework, the chart of accounts, and the new account and financial dimensions framework. It has also been integrated with currency and exchange rate types. Basic integrations have been created for budget transactions, accounting journals, and source documents. These integrations are implemented for a subset of accounting journal types and source document types to support budget control within the expenditure cycle.

This document does not discuss all of the functionality within ledger budget control, but instead focuses on development patterns and how they are implemented. Note that ledger budget control does not affect budget control for the **Project** module in any way.

Audience

This white paper targets 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
Budget control	A practice of authorizing expenditure only when budget funds can be reserved to meet future payment commitments.
Budget control dimension	A combination of active financial dimensions values that are used to allocate budget funds to pay for planned activities.
Budget control rule	The encoding of a business decision to check committed and actual expenditure against available budget funds that are allocated for detailed or aggregate activities defined by valid budget control dimension value combinations.
Budget group	A set of financial dimension values in a budget dimension hierarchy that is used to calculate aggregate budget funds that are allocated to superordinate financial dimension values by summing budget funds that are allocated to subordinate financial dimension values.

Purpose

This white paper discusses how to implement the new budget control framework. It also discusses key integration patterns for the new financial frameworks, how to make budget requests in the ledger budget control framework, and how to use the ledger budget control framework with core integration patterns.

The information provided in this white paper is intended for developers who need to perform the following tasks for ledger budget control:

- Add calls to the ledger budget control framework for existing integrations, such as a budget transaction.
- Add a new integration to ledger budget control.

Implementing the budget control framework

In Microsoft Dynamics AX 2012, budget control is based on the ledger for a legal entity. Budget control can be used to monitor budgets and to provide feedback about the availability of funds during an entry for budget transactions, accounting journals, or source documents, for example, purchase orders and invoices.

The budget control framework allows organizations to actively manage their expenses by providing notifications whenever transaction amounts for accounts exceed the funds available in their budgets.

Users can define the calculation that determines the budget funds that are available and specify the transactions that are subject to budget control. Additionally, they can specify dimensions and dimension values separately from the main account for budget control.

Budget control configuration

Configuration plays a key part in the budget control framework. Configuration defines all the parameters and settings that control the behavior of the budget control API. The user enters parameter and setting values on the **BudgetControlConfiguration** form in the application to set up budget control for a ledger.

All tables in the budget control configuration have the **SaveDataPerCompany** table property set to **No**. The BudgetControlConfiguration table is the header table for the budget control configuration. This table contains a reference to the Ledger table (in its PrimaryLedger field) that identifies the legal entity associated with the budget control configuration data. This table also has a status field (IsActive) that tracks the active version of the configuration that is referenced by the budget control framework during document processing. Only one budget control configuration can be active for a ledger. While a configuration is being edited, a draft version is tracked. After completing the draft, the user initiates the activation process from the form. This process performs the validation required to allow the draft version to become active.

Examples of budget control configuration settings that are entered on the **BudgetControlConfiguration** form are as follows:

- The budget control configuration is created for a single account structure. Based on the dimension attributes enabled for budgeting a ledger and on the account structure specified for budget control, the user can select a set of budget control dimension attributes on the form.
- By using the dimension attributes selected, the user can set the criteria for the dimension values that budget control can be applied to. These values are specified through budget control rules and budget groups, which are shared pools of budget funds.
- The budget control configuration provides an option that specifies the user's ability to continue processing in "over budget" or "over budget threshold" conditions. This option can be specified at a user group level for users belonging to a particular group.

Budget control API

The core API for the ledger budget control framework is provided in the **BudgetControlProcessor** class. **The BudgetControlProcessor** class performs budget check processing according to budget control configuration parameters and settings. Budget check processing includes validation, calculation of budget funds available, and notification in case of budget overrun.

The **BudgetControlProcessor** class is implemented as a singleton class that is managed by the **Application** class instance located on the server tier and associated with each client session. The **BudgetControlProcessor** class is called from the **ttsNotifyPostBegin**, **ttsNotifyPreCommit**, **ttsNotifyCommit**, and **ttsNotifyAbort** methods on the **Application** class to execute the logic with the **ttsBegin**, **ttsCommit** and **ttsAbort** statements.

Core budget check logic is implemented in X++ on the **BudgetControlProcessor** class and also in Transact-SQL in stored procedures. The Transact-SQL code in the form of stored procedures is

installed when the budget control configuration is activated (by a user action on the **BudgetControlConfiguration** form) or the first time that budget control logic is invoked. The Transact-SQL implementations are on the **BudgetControlSQLObjects** class.

The sequence of logic for the budget check starts in X++ code, where data is written to the budget control tracking system tables. This prepares the data for methods in the **BudgetControlProcessor** class that call stored procedures to perform budget check processing and to maintain summary budget control tracking data. The BudgetSource, BudgetSourceTracking, BudgetSourceTrackingDetail, BudgetSourceTrackingRelievingDetail, and BudgetCheckResultErrorWarningDetail tables comprise the budget control tracking system maintained by the **BudgetControlProcessor** class and its helper classes. The structure of the BudgetSourceTrackingSummary table is defined for the purpose of performing the budget check calculation of funds that are available. This table contains a summary of the data in the budget control tracking system. Both the data in the table and performing the calculation are managed by the stored procedure.

To make a request for processing to the budget control framework, there must be an open transaction. The budget control processing request is passed to the **BudgetControlProcessor** object during the open transaction. Requests are collected and processed on the outermost **ttsCommit** statement.

Budget requests

There are three types of budget requests, as follows:

- Check requests
    - A check request inquires about transaction amounts in the budget.
    - Within check requests, there are three types of details that can be captured, as follows:
        - Details of the amounts.
        - Amounts to be returned.
        - Amounts to be relieved.
- Remove requests
    - A remove request removes a BudgetSource record.
- Return requests
    - A return request returns unrelieved funds for a record.

The following sections describe each type of budget request, its supporting class, and the information required by the request.

Check requests

The **BudgetControlCheckRequest** class supports requests to check the budget.

Any amount details provided with a check request are given in the transaction currency.

The amounts are converted to the accounting currency of the ledger by the budget control framework, using the exchange rate type specified on the ledger for budget.

The following table lists information that is collected by an instance of the **BudgetControlCheckRequest** class to include in a check request. These details are applied to any type of check request.

Data	Description	Required
parmBudgetControlCategory	BudgetControlCategory of the check request.	Yes
parmBudgetDate	Date of the check request.	Yes
parmBudgetSourceId	RecId of the BudgetSource for the check request.	Yes
parmTransactionNumber	TransactionNumber of the check request.	No
parmUser	User submitting the check request.	Yes
setBudgetModel	BudgetModelId and BudgetModel.DataAreaId.	No
parmIsSimulatedBudgetSourceId	RecId of the BudgetSource being simulated for the check request.	No
parmIsCarryforward	Boolean indicating whether it is a carry forward check request.	No

The **BudgetCheckDetail** class supports requests to check the budget for details of the amounts. The following details are specific to each amount.

Data	Description	Required
LegalEntity.RecId	The ID for the legal entity record to check budget.	Yes
LedgerDimensionBase	The ledger dimension to check budget.	No
Amount	The currency amount to check budget.	No
CurrencyCode	The currency to check budget.	No

The **BudgetReturnDetail** class supports requests to check the budget for amounts to be returned. The following details are specific to each amount.

Data	Description	Required
LegalEntity.RecId	The ID for the legal entity record to return budget.	Yes
LedgerDimensionBase	The ledger dimension to return budget.	No
Amount	The currency amount to return budget.	No
CurrencyCode	The currency to return budget.	No

The **BudgetRelievingDetail** class supports requests to check the budget for amounts to be relieved. The following details are specific to each amount.

Data	Description	Required
LegalEntity.RecId	The ID for the legal entity record to check budget.	Yes
LedgerDimensionBase	The ledger dimension to check budget.	No
Amount	The currency amount to check budget.	No
CurrencyCode	The currency to check budget.	No

LegalEntity.RecId to relieve amounts from	The ID for the legal entity record to relieve budget from.	Yes
LedgerDimensionBase to relieve amounts from	The ledger dimension to relieve budget from.	No
Amount to be relieved	The currency amount to be relieved.	No
CurrencyCode associated to the amounts to be relieved	The currency to relieve the amounts.	No

Remove requests

A request to remove a BudgetSource record and its details from budget control requires a BudgetSource reference.

Return requests

A request to return remaining unrelieved funds for a BudgetSource record requires the following information.

Data	Description	Required
BudgetSource.RecId	The recId for the BudgetSource for the return request	Yes
TransDate	Date of the return request	Yes
userId	User submitting the return request	Yes

Budget request code patterns

The following code examples demonstrate patterns for how check requests, remove requests, and return requests can be set up and how the **BudgetControlProcessor** class is accessed and invoked.

Check requests

The following code example demonstrates how to set up a budget check request.

```
static void Job1(Args _args)
{
    BudgetControlCheckRequest   budgetControlCheckRequest;
    BudgetControlProcessor      budgetControlProcessor;
    BudgetSource                savedBudgetSource;
    BudgetSource                savedBudgetSourceToRelieveBudget;
    LegalEntityRecId            primaryLedgerLegalEntityId;
    BudgetModel                 budgetModel;
    LedgerDimensionAccount      ledgerAccount;
    AmountCur                   amount;
    CurrencyCode                currency;

    ttsBegin;

    // Setup necessary variables

    primaryLedgerLegalEntityId = Ledger::find(Ledger::current()).PrimaryForLegalEntity;

    budgetControlCheckRequest = BudgetControlCheckRequest::construct();
    budgetControlCheckRequest.parmTransactionNumber('000001');
    budgetControlCheckRequest.parmBudgetControlCategory(BudgetControlCategory::Original);
```

```
        budgetControlCheckRequest.parmBudgetDate(today());
        budgetControlCheckRequest.parmBudgetSourceId(savedBudgetSource.RecId);
        budgetControlCheckRequest.parmUser(curUserId());
        budgetControlCheckRequest.setBudgetModel(budgetModel.ModelId, budgetModel.dataAreaId);
        budgetControlCheckRequest.parmIsCarryforward(false);

        budgetControlCheckRequest.addLedgerDimensionDetail(
            BudgetCheckDetail::newBudgetCheckDetail(
                primaryLedgerLegalEntityId,
                ledgerAccount,
                amount,
                currency));

        budgetControlCheckRequest.addRelievingLedgerDimensionDetail(
            BudgetRelievingDetail::newBudgetRelievingDetail(
                primaryLedgerLegalEntityId,
                ledgerAccount,
                amount,
                currency,
                savedBudgetSourceToRelieveBudget.RecId,
                primaryLedgerLegalEntityId,
                ledgerAccount,
                amount,
                currency));

        budgetControlProcessor = appl.budgetControlProcessor();
        budgetControlProcessor.addBudgetCheckRequestToProcessAtCommit(budgetControlCheckRequest);

        ttsCommit;
    }
```

Remove requests

The following code example demonstrates how to perform a remove request.

```
    static void Job2(Args _args)
    {
        BudgetSourceRecId       savedBudgetSourceId;
        BudgetControlProcessor  budgetControlProcessor;

        ttsBegin;

        budgetControlProcessor = appl.budgetControlProcessor();
        budgetControlProcessor.addBudgetSourceToRemoveAtCommit(savedBudgetSourceId);

        ttsCommit;
    }
```

Return requests

The following code example demonstrates how to set up a request to return unrelieved funds.

```
static void Job3(Args _args)
    {
        BudgetSourceRecId       savedBudgetSourceId;
```

```
    BudgetControlProcessor  budgetControlProcessor;

    ttsBegin;

    budgetControlProcessor = appl.budgetControlProcessor();

    budgetControlProcessor.addBudgetSourceToReturnRemainingAtCommit(
            savedBudgetSourceId,
            today(),
            curUserId());

    ttsCommit;
}
```

Core integration patterns

We have provided core integrations for budget transactions, accounting journals, and source documents in the ledger budget control framework.

The physical data model for the Budget Control tables can be found in the Appendix section.

The BudgetSource table holds foreign key references for each type of base integration to support referencing and querying of budget check results.

Integration pattern for budget transactions

Integrating a budget transaction with the ledger budget control framework is supported by the **BudgetControlBudgetTransactionProcessor** class and associated classes, as follows:

- **BudgetSourceBudgetTransactionLine** class: Provides methods for saving and finding BudgetSource records that contain a BudgetTransactionLine reference.
- **BudgetControlBudgetTransactionProcessor** class: Uses the **BudgetSourceBudgetTransactionLine** class to create or find the BudgetSource records for records referencing BudgetTransactionLine records. It handles the creation of budget check requests and budget removal requests for budget transactions, and calls the **BudgetControlProcessor** class.

Integration pattern for accounting journals

Integrating an accounting journal with the ledger budget control framework is supported by the **BudgetControlAccountingJournalProcessor** and **BudgetControlGeneralJournalEntry** classes and associated classes, as follows:

- **BudgetSourceLedgerJournalTrans** class: Provides methods for saving and finding BudgetSource records that contain a reference to an unposted journal voucher represented in the LedgerJournalTrans table.
- **BudgetSourceGeneralJournalEntry** class: Used for interacting with BudgetSource records that contain a reference to a posted journal voucher represented in the GeneralJournalEntry table.
- **BudgetControlAccountingJournalProcessor** class: For unposted accounting journal vouchers, this class uses the **BudgetSourceLedgerJournalTrans** class to create or find the BudgetSource records for the vouchers represented in the LedgerJournalTrans table. The **BudgetControlAccountingJournalProcessor** class handles the creation of budget check requests and budget removal requests for vouchers and interacts with the **BudgetControlProcessor** class.
- **BudgetControlGeneralJournalEntry** class: For accounting journal transaction reversals and budget control processing of accounting journals during posting, the

BudgetControlGeneralJournalEntry class uses the **BudgetSourceGeneralJournalEntry** class to create or find the BudgetSource records for the vouchers being posted or reversed. The **BudgetControlJournalEntry** class collects the posting ledger account and amount details and formulates the budget check requests that will be passed to the **BudgetControlAccountingJournalProcessor** class that will interact with the **BudgetControlProcessor** class.

Integration pattern for source documents

Integrating a source document with the ledger budget control framework is supported by the **BudgetControlSourceDocumentProcessor** class and associated classes, as follows:

- **BudgetSourceSourceDocumentLine** class: Provides methods for saving and finding BudgetSource records that contain a SourceDocumentLine reference.
- **BudgetControlSourceDocumentProcessor** class: Uses the **BudgetSourceSourceDocumentLine** class to handle the creation of budget check requests (including check, return, and relieving details), budget return requests and budget removal requests, and interacts with the **BudgetControlProcessor** class.

Adding integration patterns

This section describes the steps that you need to take to add integrations to the budget control framework (excluding budget transactions, accounting journals, or source documents, which were discussed in the Core integration patterns section).

Data update

To support a new budget control integration, you will need to make the following updates to objects defined in the Data Dictionary of the Application Object Tree (AOT).

- **BudgetControlSourceIntegrator enumeration**

  Add a value to the **BudgetControlSourceIntegrator** enumeration. Each value in this enumeration corresponds to a document that can be enabled for budget control in the budget control configuration.

  Examples:

  - **BudgetTransaction**: Budget transaction type integration
  - **PurchaseOrder** and **VendorInvoice**: Source document type integrations
  - **DailyJournal** and **FixedAssetJournal**: Accounting journal type integrations

- **BudgetSourceType enumeration**

  Add a value to the **BudgetSourceType** enumeration. This enumeration serves to discriminate the types of references on the BudgetSource table. Current enumeration values are **BudgetTransactionLine**, **SourceDocumentLine**, and **LedgerJournalEntry**.

  The enumeration value determines which foreign key field or fields will be populated for a particular BudgetSource record. When a foreign key field is added to the new integrator, a corresponding value must be added to the enumeration.

- **BudgetSource table**

  Add a new foreign key relation to the table being referenced on the BudgetSource table. The foreign key relation can be one or more fields, depending on the composition of the unique identifier for the referenced table.

  Add a new method to the BudgetSource table that checks for the existence of the record referenced by the new foreign key relation. Name the method according to the name of the

foreign key relation. For example, for the BudgetTransactionLine foreign key, there is a method named **existBudgetTransactionLine**.

Update the **existSourceIntegratorReference** method on the BudgetSource table to add a case for the new **BudgetSourceType** enumeration value. The case calls the new method that you added for checking the existence of the related record.

Code update

We recommend that you make the following code update changes:

1. Create a processor class

   Create a processor class to support a new integration. Your processor class should follow the pattern of exposing static methods where the integrating document is passed as a parameter or where details identifying the integrating document are passed as parameters.

   Use separate methods for each type of request supported by the integration: check budget, remove budget, and return budget. The core integration patterns for budget transactions, accounting journals and source documents are different and can serve as models for a new budget control integration.

   When the caller submits a new integration document to the processor, the processor should execute the following steps:

   - Prepare the appropriate budget control request.
   - Begin the transaction.
   - Call the **BudgetControlProcessor** methods to add the request.
   - Commit the transaction.

2. Extend the BudgetSourceIntegrator class

   Extend the **BudgetSourceIntegrator** class to support a new integration. The extension should be used to save and find the BudgetSource records based on the reference specific for the integration document.

   In addition to creating the extension class, you must update the **newBudgetSourceIntegrator** method of the **BudgetSourceIntegrator** class to add the new integration document. This is done to support constructing the new type.

3. Extend the BudgetSourceCollectionIntegrator class

   Extend the **BudgetSourceCollectionIntegrator** class to support a new integration. This extension class is used by the **BudgetCheckResults** form to retrieve error and warning details for a collection of BudgetSource records. The extension class should be implemented to retrieve the BudgetSource records collection for the integration document header, when the integration document has a header and lines pattern.

   In addition to writing the extension class, you must update the **newBudgetSourceCollectionIntegrator** method of the **BudgetSourceCollectionIntegrator** class to add the new integration document header. This is done to support constructing the new type.

Revising the BudgetControlConfiguration form

You must update the **BudgetControlConfiguration** form if you need to enable budget control for the new integrator type. You can pattern your code on the following examples.

1. Add the following two edit methods to the data source for **the BudgetControlConfiguration** form:

- The first method is used to edit the budget control source integrator to enable it for budget control.

  Your method must call the **setSourceIntegratorEnabled** method on the BudgetControlIntegration data source located on the **BudgetControlConfiguration** form. To refresh the form display when the value changes, call the appropriate **setEnabled[PageName]** method. These methods manage the process of enabling the control on the tab page.

  The **editDailyJournalEnabled** method in the following code provides an example:

```
public edit NoYes editDailyJournalEnabled(boolean _set, NoYes _isEnabled)
{
    BudgetControlSourceIntegrator sourceIntegrator =
BudgetControlSourceIntegrator::DailyJournal;
    boolean isSet;

    if (_set)
    {
        // Set whether daily journal budget control is enabled.
        isSet = this.setSourceIntegratorEnabled(sourceIntegrator, _isEnabled);

        if (isSet)
        {
            element.setEnabledJournals();
        }
    }

    return enabledSourceIntegrators.in(sourceIntegrator);
}
```

- The second method is used to edit the budget control source integrator to specify whether the integrator will perform a budget check at line item entry.

  This method must call the **setSourceIntegratorDoBudgetCheckOnEntry** method on the BudgetControlIntegration data source located on the **BudgetControlConfiguration** form.

  The **editDailyJournalDoCheckOnEntry** method in the following code provides an example:

```
public edit NoYes editDailyJournalDoCheckOnEntry(boolean _set, NoYes
_doBudgetCheckOnEntry)
{
    BudgetControlSourceIntegrator sourceIntegrator =
BudgetControlSourceIntegrator::DailyJournal;

    if (_set)
    {
        // Set whether daily journal do budget check on entry.
        this.setSourceIntegratorDoBudgetCheckOnEntry(sourceIntegrator,
_doBudgetCheckOnEntry);
    }

    return doBudgetCheckOnEntrySourceIntegrators.in(sourceIntegrator);
}
```

2. Add controls to the form to expose the edit methods that were added:

- If the budget control source integrates with the source document framework, the controls should be added to the source document tab page and follow the pattern that exists for the other controls on the tab page.
- If the budget control source integrator is a journal, the controls should be added to the journal tab page and follow the pattern that exists for the other controls on the tab page.
- If the budget control source integrator is not a source document or a journal, a new tab page should be added. However, it should still follow the pattern established by the source documents and journals tab pages.

3. Add methods as needed to maintain the form controls. Depending on whether the budget control source integrator is a source document, a journal, or a custom budget control integrator, the **setEnabledSourceDocuments**, **setEnabledJournals,** or the new **setEnabled[PageName]** method should be updated to manage the respective new control by following the existing patterns. Similarly the **setVisibleSourceDocuments, setVisibleJournals,** or the new **setVisible[PageName]** method should be updated to manage the visibility of the respective new control by following the existing patterns.

Budget check result indicator icon

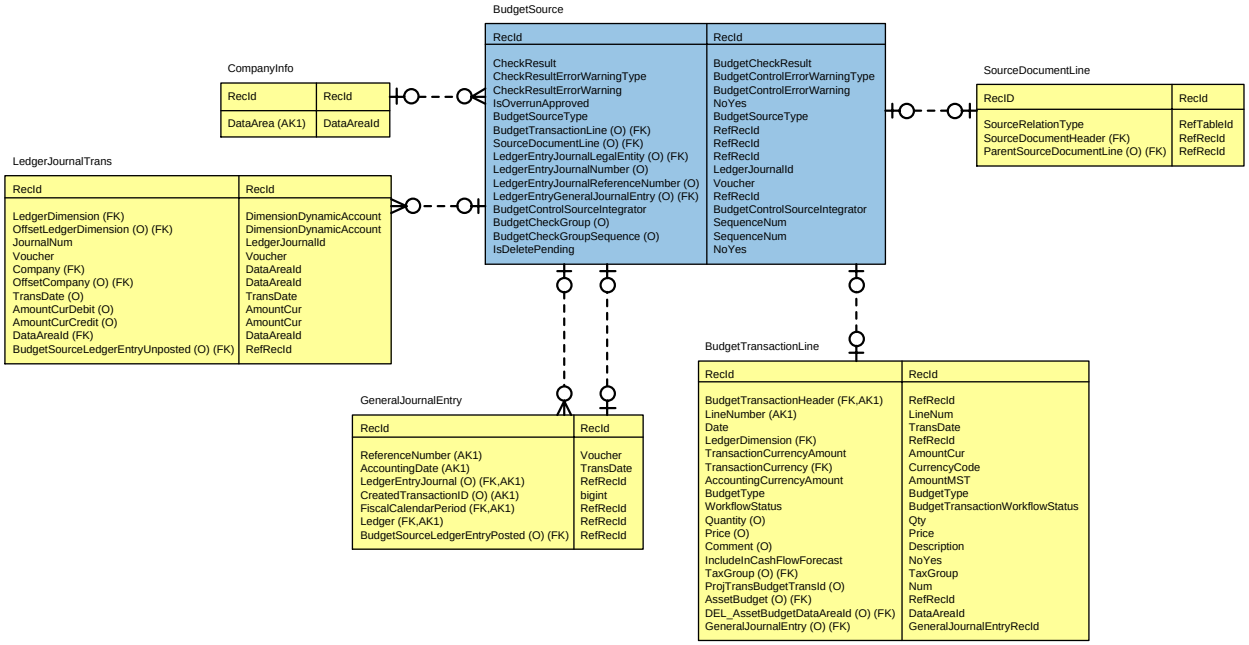
The budget control framework provides support for displaying icons that indicate the result of budget check requests on entry forms. There are four different results that can be displayed: "no check," "passed," "passed with warnings," and "failed."

The **BudgetControlResult** class provides the following helper methods:

- **createBudgetCheckResultImages**
  Creates the image list with four different result icons
- **getBudgetCheckResultImagePos**
  Retrieves the position of the icon in the image list, based on the budget check result
- **getBudgetCheckResultToolTip**
  Creates the tool tip text based on the budget check result

Appendix

Budget Control Tables



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