Microsoft Dynamics® 2012

Print Management Integration Guide

White Paper

Print management gives users specialized control of print settings for selected Microsoft Dynamics AX reports.

December 2012

www.microsoft.com/dynamics/ax

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

Microsoft Dynamics

Table of Contents

Feature description

Print management gives users control of print settings for selected Microsoft Dynamics AX reports. Print settings include the number of copies, the printer destination, and the multi-language text that can be included on the report. Print management also includes the following features:

- The ability to modify the print settings of a report based on the contents of the data being printed. For example, send sales orders > $1000 to Printer A and sales orders > $10,000 to Printer B.
- The ability to specify print settings at various levels of the application. For example, module-level print settings apply to all documents in a given module, while customer-level print settings only apply to specific customers and transaction-level print settings apply only to specific transactions.
- The ability to override or add to the settings from higher levels in the application. For example, module-level settings can be overridden at the customer or transaction level if not desirable. Furthermore, additional settings can be added to the existing settings for similar reasons.

Why use print management?

You should consider using the print management framework when the following conditions are true:

- Your new or existing feature has reports that require specialized print management.
- You are adding a new document type to the application similar to the document types currently supported by print management. The document types supported by print management in the Microsoft Dynamics AX 2012 release are as follows:

  Accounts Payable

  - Purchase order
  - Purchase order invoice
  - Purchase order packing slip
  - Purchase order receipts list
  - Request for quote
  - Request for quote - Accept
  - Request for quote - Reject
  - Request for quote - Return

  Accounts Receivable

  - Confirmation
  - Free text invoice
  - Quotation
  - Sales order confirmation
  - Sales order invoice
  - Sales order packing slip

  Inventory management

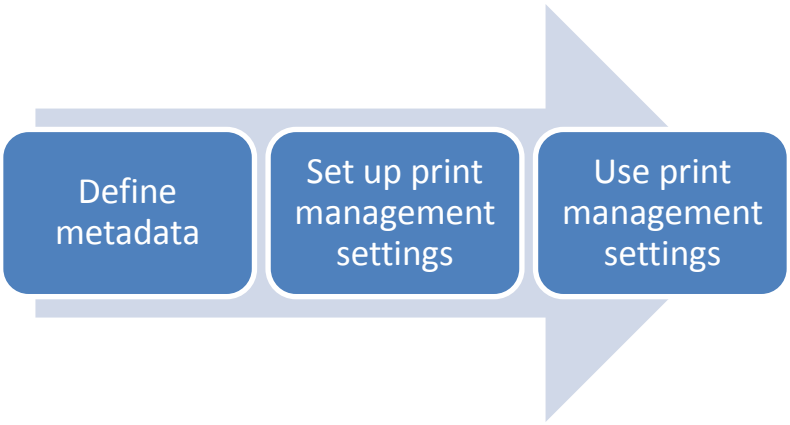
  - Picking list

Project

- Project invoice

Architectural overview

The print management subsystem can be broken into three main parts: metadata, setup, and retrieval. The metadata portion defines a series of classes that allow you to describe how print management should work. Here you can describe how many levels of defaulting are possible, what tables and columns should be involved in queries, and other pertinent information. The setup portion defines a generic user experience that can be opened throughout the application based on certain inputs. You would call this in areas where print management information has to be managed by end users. The retrieval portion defines an API which enables you to retrieve and use the print management information set up by the user.



Defining metadata

Print management metadata is defined completely in code and stems from three main classes or concepts: PrintMgmtDocType (documents), PrintMgmtNode (nodes) and PrintMgmtHierarchy (hierarchies).

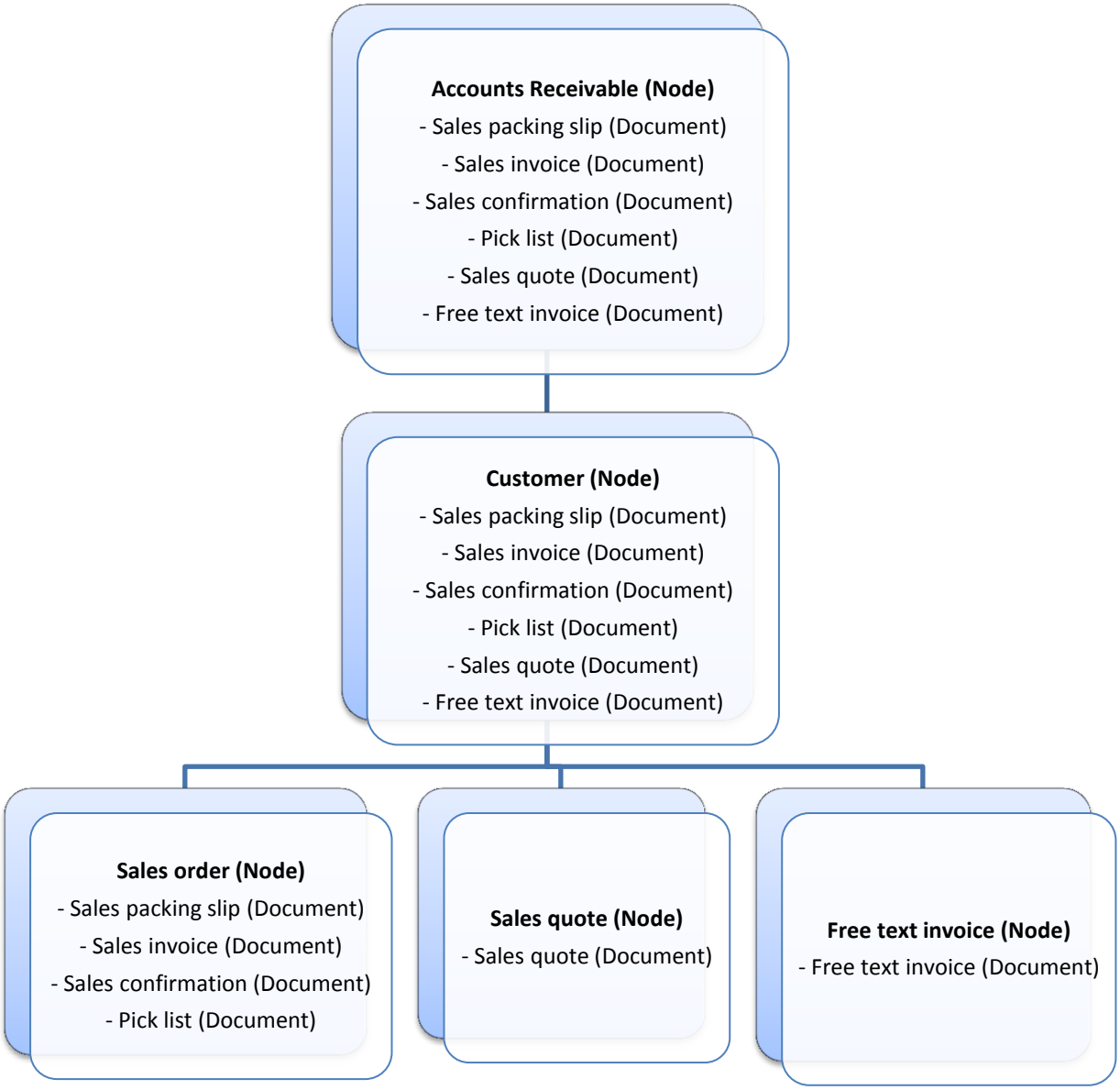
A **document** is the document or report to which specific print management settings apply. Documents are at the heart of print management because all settings are defined for documents of a particular type. Once users set up this information it is used whenever that document is printed. The documents in the print management system are defined in the PrintMgmtDocType class and are always related to specific nodes in print management.

A **node** is a particular level of defaulting in a hierarchy. For example, in the Accounts Receivable module print management settings can exist at the Sales Order Transaction level, the Customer level and the Module level. Each of these levels is considered a node and each node in print management must derive from the abstract PrintMgmtNode class. While nodes define the level at which print management settings can exist, they do not have any order by themselves which is where hierarchies help.

A **hierarchy** is a group of related nodes. Using the previous example, the nodes mentioned would all be a part of the Account Receivable Hierarchy. The ordering of nodes in a hierarchy defines the ordering of defaulting for print management settings. Each hierarchy in print management must derive from the abstract PrintMgmtHierarchy class.

To bring it all together, consider the following diagram which shows how documents, nodes and hierarchies all relate to each other to define the print management metadata for a hypothetical Accounts Receivable hierarchy.

Accounts Receivable (Hierarchy)



The Accounts Receivable node represents the module level settings and all documents present in the module apply at this level. The Customer node represents settings that apply to specific customers and once again all documents present in the module apply. At the lowest level the nodes represent transaction level settings and, as expected, only certain documents apply to each specific transaction.

To integrate with print management you may have to modify existing nodes or hierarchies to meet the needs of the documents you want to print. You may also be required to create new nodes, hierarchies or documents to accomplish the same goal. The following sections outline the steps that are required to accomplish these common integration tasks.

Adding a new document

Whether you are adding a new document to new or existing nodes the same steps are required. If you want a better understanding of the classes or methods mentioned in the steps, see the XML documentation that is included in the source code.

1. Update the PrintMgmtDocumentType enumeration
    - Add the document type to the enumeration immediately following the last element.
    - Create and assign a label for the document type that follows the pattern for existing document types.
2. Update the PrintMgmtDocType class to handle the new document type.
    - Modify the getQueryTableId method. Add the table ID of the table that will be used when enabling users to enter conditional queries for the document type. This is generally the Jour table that is used when you post, but could be any table that applies.
    - Modify the getQueryRangeFields method. Add the default fields that should be used when entering conditional queries for the new document type. The order in which they are added will also be the order in which they appear in the query window. Consider wrapping this logic with a call to isConfigurationkeyEnabled, passing the appropriate configuration key, to guarantee that the fields only appear when the functionality is enabled in the system.
    - Modify the getDefaultReportFormat method. Add a case for your PrintMgmtDocumentType enumeration. Return the name of your report, utilizing the ssrsReportStr intrinsic function.
3. Update the PrintMgmtDocTypeTest class to handle the new document type.
    - Modify the testGetDisplayName method. Add a test to guarantee that the document type appears with the correct text.
    - Modify the testGetQueryTableId method. Add a test to guarantee that the document type has the correct query table id.
    - Modify the testGetQueryRangeFields method. Add a test to guarantee that the document type has the correct default query fields.
4. Update all PrintMgmtNode derivatives to which the document type applies.
    - Modify the getDocumentTypes method. Add the document type to the list of supported document types. Wrap this logic with a call to isConfigurationkeyEnabled, passing the appropriate configuration key, to guarantee that the documents only appear when the functionality is enabled in the system.
5. Update necessary PrintMgmtNode derivatives' tests for the document type.
    - Modify the testNodeDocumentTypes method. Create or update this method to handle the document type.
6. Update necessary PrintMgmtHierarchy derivatives to which the document type applies.
    - Modify the getParentImplementation method. This method is used to define the structure of the nodes in a hierarchy. In some cases, the document type affects the logic to determine the parent and in other cases it does not. For example, Sales order invoice documents use the InvoiceAccount field of the SalesTable to find their parent node, whereas Sales order confirmation documents use the CustAccount field of the SalesTable. This is all determined based on the document type. If your new document type requires special behavior such as this, update the method.
7. Update necessary PrintMgmtHierarchy derivatives' tests for the document type.
    - This step is only necessary if you were required to update the getParentImplementation method.

- Update the testGetParent method. Add tests for the new document type following the existing pattern.
- Update the testGetParentSkippingLevels method. Add tests for the new document type following the existing pattern.

Adding a new layout to an existing document

A new layout may be added to an existing document to support another report option, such as an additional country/region configuration. If you want a better understanding of the classes or methods mentioned in the steps, see the XML documentation that is included in the source code.

1. Update the PrintMgmtReportFormat table to add records for the new document layout.
    - Modify the populate method. Add the appropriate condition for which the new layout should be available. Make use of the local methods inside the populate method.
2. Update the PrintMgmtDocType class to handle the new document layout.
    - Modify the getDefaultReportFormat method. Add the appropriate condition for the new layout. Return the name of your new report layout combination, utilizing the ssrsReportStr intrinsic function.

Creating or modifying nodes

New nodes are necessary when a new level of defaulting is necessary for the documents you want to print. If you want a better understanding of the classes or methods mentioned in the steps, see the XML documentation that is included in the source code.

1. Ensure a node does not already exist that meets your needs. Look at the classes that derive from the PrintMgmtNode class to determine this information. Even if an existing node defines document types you are not interested in using, you can still use the node. If none of the existing nodes meet your needs, continue to step 2.
2. Create a new enumeration value in the PrintMgmtNodeType enumeration
    - Add the node type to the enumeration immediately following the last element.
    - The name of the element should match the table to which the print management settings will be related. If the print management settings do not relate to a table, choose a name following the existing patterns present.
    - Create and assign a label for the node type. If the node references a table, use the table name label from the table it references. If not, follow the existing patterns present.
3. Create a new class that derives from the PrintMgmtNode class. Be sure to reference the existing PrintMgmtNode derivatives for examples.
    - The class should be prefixed with 'PrintMgmtNode_' and should follow the naming pattern of the existing nodes. If the node is tied to a particular table, the table name should follow the prefix such as PrintMgmtNode_CustTable, for example.
    - Implement the getDisplayCaptionImplementation method.
        - This method controls the caption that will appear for the instance of the node in the Print Management window.
        - Return the appropriate label for Module level settings. See existing examples in PrintMgmtNode_Purch or PrintMgmtNode_Sales.
        - Return strfmt("@SYS108943", _tableBuffer.caption()) for account level settings. See existing example in PrintMgmtNode_CustTable or PrintMgmtNode_VendTable.
        - Return strfmt("@SYS108944", _tableBuffer.caption()) for transaction level settings. See existing examples in PrintMgmtNode_SalesTable or PrintMgmtNode_PurchTable.

        - Customize as necessary for unique situations.
    - Implement the getDocumentTypes method.
        - Return a list of document types the node supports.
        - The document types should be wrapped in configuration key checks so that they are only loaded if the configuration key is enabled. For example, isConfigurationkeyEnabled(configurationkeynum(LogisticsBasic)).
    - Implement the getIconImageResNum method.
        - This method controls the icon that will appear for the instance of the node in the Print Management window.
        - Return the resource ID used to display the icon representing this node. The resAppl macro contains the resource ID definitions for icon images. This method should return one of those resource IDs. If additional resource IDs are used, they should also be added to the build method of the ImageListAppl_PrintMgmt class.
        - Module level nodes should use the same icon that is used for the module, account level nodes should use #ImagePrintManagementAccount, and transaction level nodes should use #ImagePrintManagementTrans.
    - Implement the getNodeType method.
        - Return the appropriate PrintMgmtNodeType enumeration value.
    - Implement the getReferenceTableId method.
        - Return the table to which the node will be related. For example, if a node is tied to specific customers you would return tablenum(CustTable).
        - If the node is not related to a specific table, return the value of 0. Module level settings are examples of nodes that are not tied to specific tables.
4. Modify the PrintMgmtNode::construct method. Add a case statement to construct your new node.
5. Create appropriate unit tests.
    - The name of the test class should be identical to the node followed by the postfix 'Test'.
    - Regarding the contents of the tests, follow the example of existing node tests that are similar to your node type. For example, module-level, account level, transaction level nodes, or other.
    - To add data, follow the existing examples in the PrintMgmtTestSuite class.
        - Add a macro that defines the id of the table in the classDeclaration.
        - Add a new method that retrieves the table following the existing pattern. For example, getSalesTable, getCustInvoiceTable, and so forth.
        - Create the data in the setUpReferenceData method and make to trackInsertedRows for the table you add.
    - Once done, add the new test to the PrintMgmtTestSuite by adding the appropriate entry in the new method.
6. If the node relates to a table, update the appropriate relations and cascading deletes.
    - Create a relation on the PrintMgmtDocInstance table to the table you are referencing. For example, PrintMgmtDocInstance -> SalesTable.
    - Add a cascading delete on the table you are referencing to the PrintMgmtDocInstance table. For example, SalesTable -> PrintMgmtDocInstance.
7. Add the node to the appropriate hierarchy. See the *Creating or modifying hierarchies* section for the steps that are needed to do this.

8. Add the node to the PrintMgmt shared project.

Creating or modifying hierarchies

In some cases existing hierarchies can be modified to include or rearrange existing nodes. In other cases completely new hierarchies are necessary. Both scenarios have similar steps which are outlined later in this document. If you want a better understanding of the classes or methods mentioned in the steps, see the XML documentation that is included in the source code.

1. Create a new enumeration value in the PrintMgmtHierarchyType enumeration.
    - Add the hierarchy type to the enumeration immediately following the last element.
    - Create and assign a label for the hierarchy type following the existing patterns.
2. Create a new class that derives from the PrintMgmtHierarchy class. Be sure to reference the existing PrintMgmtHierarchy derivatives for examples.
    - The class should be prefixed with 'PrintMgmtHierarchy_' and should follow the naming pattern of the existing hierarchies.
    - Implement the getNodesImplementation method.
        - Return the list of nodes the hierarchy supports.
    - Implement the getParentImplementation method.
        - This method is used to define the structure of the nodes in a hierarchy.
        - It provides an instance of a node and a document type as input parameters and will be required to return an instance of the PrintMgmtNodeInstance class. This PrintMgmtNodeInstance class brings together a PrintMgmtNode instance with the actual table buffer record it relates with.
        - If the parent node does not reference a table, only the PrintMgmtNodeType enumeration value of the parent is required to be returned through the parmNodeDefinition property of the PrintMgmtNodeInstance class.
        - If the parent node does reference a table, the specific record of the parent node should be returned also. This is returned through the parmReferencedTable property of the PrintMgmtNodeInstance class. If the record cannot be found, a null value should be returned.
        - See existing implementations for a better understanding of what is necessary for this method.
3. Modify the PrintMgmtHierarchy::construct method. Add a case statement to construct your new hierarchy.
4. Create appropriate unit tests.
    - The name of the test class should be identical to the hierarchy followed by the postfix 'Test'.
    - Regarding the contents of the tests, follow the example of existing hierarchy tests.
    - To add data, follow the existing examples in the PrintMgmtTestSuite class.
        - Add a macro that defines the id of the table in the classDeclaration.
        - Add a new method that retrieves the table following the existing pattern. For example, getSalesTable or getCustInvoiceTable.
        - Create the data in the setUpReferenceData method and make sure to trackInsertedRows for the table you add.
    - Once done, add the new test to the PrintMgmtTestSuite by adding the appropriate entry in the new method.

5. If you are modifying an existing hierarchy, make sure to update the appropriate hierarchy tests.
    - Modify the testIsValidNodeType method. Update the count because the number of nodes added will have changed.
6. Add the hierarchy to the PrintMgmt shared project.

Set up print management

Users need the ability to set up print management for each node in a hierarchy. Using the Accounts Receivable example, this means that users should be able to set up print management settings at the module level, the customer level and at each transaction level. To make this possible you must start the user experience from the appropriate forms for each node in your hierarchy. Since the user experience is based on the metadata which defines your documents, nodes and hierarchies there is not much you need to provide to get a fully functional and reusable user experience.

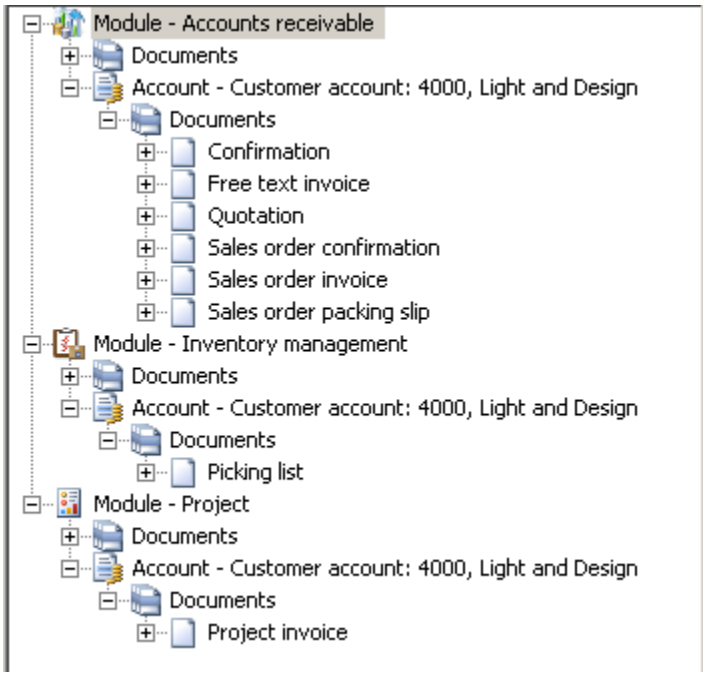
The API to start the user experience is centered on the PrintMgmtSetupContext class. To demonstrate the usage of this class, consider the following code which starts the print management user experience from the Purchase Order Details window:

```
void clicked()
{
    PrintMgmtSetupContext setupContext = new PrintMgmtSetupContext();

     // If you reference a table, always include this check
    if (purchTable_ds.cursor() == null)
        return;

    super();

     // The first parameter specifies the hierarchy to display and
     // the second specifies which node to start with. All parent
     // nodes "above" the starting node will be displayed, along
     // with the starting node.
    setupContext.addHierarchyContext(
            PrintMgmtHierarchyType::Purch,
            PrintMgmtNodeType::PurchTable);

     // This links the caller to the instance of the print mgmt form
    setupContext.parmCaller(element);

     // This is the record the node is associated with. If a node
     // is not related to a table, pass null.
    setupContext.parmReferencedTableBuffer(purchTable_ds.cursor());

    PrintMgmt::launchSetup(setupContext);
}
```

This represents a fairly common scenario and will likely suit your needs most of the time. Note that all documents associated with the PurchTable node type will be displayed in the user experience. Sometimes there are cases where you may want to display only certain documents associated to a node or want to show multiple hierarchies. A scenario such as this is outlined in the code example that follows. It shows the code that starts the print management user experience from the Customer details window:

```
void clicked()
{
    PrintMgmtSetupContext setupContext = new PrintMgmtSetupContext();
    List salesDocs = new List(Types::Enum);
    List projectDocs = new List(Types::Enum);
    List inventDocs = new List(Types::Enum);

    if (custTable_ds.cursor() == null)
        return;

    super();

     // This example shows the ability to call with multiple hierarchies
     // as well as the ability to limit which documents appear for each
     // node in those hierarchies. If the document filter list was not
     // passed, all documents would appear in each hierarchy and it
     // would not make sense to show Project or Inventory documents in
     // the Accounts Receivable hierarchy, for example.

     // Add sales documents
    salesDocs.addEnd(PrintMgmtDocumentType::Confirmation);
    salesDocs.addEnd(PrintMgmtDocumentType::Quotation);
    salesDocs.addEnd(PrintMgmtDocumentType::SalesFreeTextInvoice);
    salesDocs.addEnd(PrintMgmtDocumentType::SalesOrderConfirmation);
    salesDocs.addEnd(PrintMgmtDocumentType::SalesOrderInvoice);
    salesDocs.addEnd(PrintMgmtDocumentType::SalesOrderPackingSlip);
    setupContext.addHierarchyContext(
            PrintMgmtHierarchyType::Sales,
            PrintMgmtNodeType::CustTable,
            salesDocs);

     // Add project documents
    projectDocs.addEnd(PrintMgmtDocumentType::ProjectInvoice);
    setupContext.addHierarchyContext(
            PrintMgmtHierarchyType::Project,
            PrintMgmtNodeType::CustTable,
            projectDocs);

     // Add invent documents
    inventDocs.addEnd(PrintMgmtDocumentType::InventPickList);
    setupContext.addHierarchyContext(
            PrintMgmtHierarchyType::Invent,
            PrintMgmtNodeType::CustTable,
            inventDocs);

    setupContext.parmCaller(element);

     // Note that even though multiple hierarchies are allowed, the
     // starting node for each must reference the same table buffer.
    setupContext.parmReferencedTableBuffer(custTable_ds.cursor());

    PrintMgmt::launchSetup(setupContext);
}
```

The following screenshot illustrates the resulting user experience:



If you want a better understanding of the PrintMgmtSetupContext class, see the XML documentation that is included in the source code.

Using print management settings

Once the metadata has been defined and users can set up print management settings it is necessary to use those settings when printing documents. There are three options available to accomplish this task based on the purpose of your document:

1. Create a derivative of the FormLetterReport class and use the FormLetterReport class to retrieve and enumerate settings. In this case, your document may be part of the FormLetter class hierarchy. These documents are generated during posting and require a unique FormLetter and FormLetterReport derivative. If your document is not part of the FormLetter class hierarchy, you may still use this option by creating a FormLetterReport derivative and bypassing the FormLetter class hierarchy.
2. Call print management directly through the PrintMgmt::getSettings method. In this case, your document is not required to be a part of the FormLetter class hierarchy. These documents are independent and are not generated during posting time.
3. Use print management via the PrintMgmtReportRun class. The class simplifies retrieval and iteration over settings and output of a SQL Server Reporting Services (SRS) report using print management settings.

Even if Option #2 or 3 matches your scenario, you may find the most benefit in using Option #1. Option #1 provides some nice shortcuts and prevents you from worrying about many of the details, even if your document is not involved in the posting process. As a result, Option #2 will not be described in detail because it is mainly available for custom needs. Refer to the XML documentation of the PrintMgmtPrintContext, PrintMgmtPrintSetting and PrintMgmtPrintSettingDetail classes for more detail about Option #2, or view the implementation of the methods on the FormLetterReport abstract class.

Option #3 will be described briefly and an example of its use is included.

Using print management with FormLetterReport

For option #1, the API to retrieve user settings is centered on the abstract FormLetterReport class and its concrete derivatives. The following steps describe how to implement the necessary classes and interact with the FormLetterReport class to retrieve the print management settings. If you want a better understanding of the classes or methods mentioned in the steps, see the XML documentation that is included in the source code.

1. Create a new class that derives from the FormLetterReport class. Be sure to reference the existing FormLetterReport derivatives for examples.
    - Use a name that follows a similar pattern as the following existing derivatives:
        - PurchFormLetterReport_Invoice
        - PurchFormLetterReport_PackingSlip
        - SalesFormLetterReport_Confirm
        - SalesFormLetterReport_Invoice
    - Implement the getDefaultPrintJobSettings method. Based upon the PrintSetupOriginalCopy enumeration value provided, return the default print job setting. This is the default that will be used if no printer destination is specified on the print management setting. Normally this value can be retrieved from the FormLetter class derivative that corresponds to the FormLetterReport class derivative because default printers are set up on the posting form.
    - Implement the getPrintMgmtDocumentType method. Return the appropriate PrintMgmtDocumentType enumeration value.
    - Implement the getPrintMgmtHierarchyType method. Return the appropriate PrintMgmtHierarchyType enumeration value.
    - Implement the getPrintMgmtNodeType method. Return the appropriate PrintMgmtNodeType enumeration value.
2. Modify the FormLetterReport::construct method. Add a case statement to construct your new form letter report.
3. Create appropriate unit tests.
    - The name of the test class should be identical to the form letter report class followed by the postfix 'Test'.
    - Regarding the contents of the tests, follow the example of existing form letter report tests that are present.
4. Create a new class that derives from the SrsPrintMgmtFormLetterController class. Be sure to reference the existing SrsPrintMgmtFormLetterController derivatives for examples.
    - Use a name that follows a similar pattern to the following existing derivatives:
        - VendInvoiceDocumentController
        - PurchPackingSlipController
        - SalesConfirmController
        - SalesInvoiceController
    - Implement the main method.
        - Initialize a new instance of your controller class.
        - Using your specific PrintMgmtDocumentType enumeration, construct an instance of the PrintMgmtDocType class. Call the PrintMgmtDocType getDefaultReportFormat method to

get the print management report format name. Pass this report name to the controller class by using the parmReportName method.

- Pass the function's Args to the controller class by using the parmArgs method.
- Normally, the report controller's dialog will need to be suppressed. This is the case when the settings for the report were available in the posting form, and the list of documents was generated during posting. Suppress the dialog by calling the controller's parmShowDialog method with a false value.
- Initiate the report process by calling the controller's startOperation method.

- Implement the initFormLetterReport method.
  - Retrieve the list of documents to print from the controller's Args. Normally, this will be the list of documents that was generated during posting. This list should be available in the controller's Args as the object parameter.
  - Initialize a new instance of your FormLetterReport class by using the FormLetterReport constructor method. Assign this instance to the formLetterReport class variable.
  - Pass the print type (copy or original) to the formLetterReport instance by using the parmPrintType method. Normally, this setting will be available in the controller's Args as the enumeration parameter.
- Implement the runPrintMgmt method.
  - This method will loop through the list of documents that was retrieved in the initFormLetterReport method and print each document.
  - For each document, set the print management settings in the formletterReport variable by using the loadPrintSettings method.
  - Print each document by using the outputReports() method.

The following code examples create a controller class that retrieves and uses the print management settings for the Sales invoice report.

In the class declaration of the controller class, declare variables of type RecordSortedList and PrintCopyOriginal, and also a table variable of the type being printed.

```
    RecordSortedList    journalList;
    PrintCopyOriginal   printCopyOriginal;
    CustInvoiceJour     custInvoiceJour;
```

In the main method of your controller class, initialize an instance of the controller, and set the initial values based on the context.

```
public static void main(Args _args)
{
    // Construct an instance of the controller class.
    SrsPrintMgmtFormLetterController formLetterController =
        new SalesInvoiceController();

    // Set the initial values on the controller.
    formLetterController.parmReportName(
        PrintMgmtDocType::construct(
            PrintMgmtDocumentType::SalesOrderInvoice).getDefaultReportFormat());
    formLetterController.parmArgs(_args);
    formLetterController.parmShowDialog(false);

    // Initiate the report process
    formLetterController.startOperation();
}
```

In the initFormLetterReport method of your controller class, initialize the formLetterReport variable and set the initial values based upon the context.

```
protected void initFormLetterReport()
{
    // Retrieve the list of documents to be printed
    if (this.parmArgs().record())
    {
        // Get journal list from the selected record/s
        journalList =
            FormLetter::createJournalListCopy(this.parmArgs().record());
    }
    else
    {
        journalList = this.parmArgs().object();
    }

    // Construct an instance of the FormLetterReport class and save it as
    // an instance variable for later use in other methods.
    formLetterReport =
        FormLetterReport::construct(PrintMgmtDocumentType::SalesOrderInvoice);

    // printCopyOriginal determines whether this is a reprint or not.
    printCopyOriginal = this.parmArgs().parmEnum();
    formLetterReport.parmPrintType(printCopyOriginal);

    super();
}
```

In the runPrintMgmt method, process each record corresponding to a document, retrieve the settings, and print reports for each document. Note that this method is called on the client tier, so any additional business logic that accesses the database or otherwise executes on the server tier should be done carefully to avoid unnecessary remote procedure calls.

```
protected void runPrintMgmt()
{
    if (!journalList)
    {
        throw error("@SYS26348");
    }
    journalList.first(custInvoiceJour);

    do
    {
        if (!custInvoiceJour)
        {
            throw error("@SYS26348");
        }

        // Load all print settings associated with the current record.
        // The custInvoiceJour table is the record being printed and the
        // user-defined queries will be run against this table to see which
        // print settings apply.
        // The custInvoiceJour.salesTable() is the transaction that the
        // print management settings are related with. This is the record
        // that starts the defaulting process for the settings that should
        // be considered.
        formLetterReport.loadPrintSettings(
            custInvoiceJour,
            custInvoiceJour.salesTable(),
            custInvoiceJour.LanguageId);

        this.outputReports();
    }
    while (journalList.next(custInvoiceJour));
}
```

Your document may have a requirement to show the footer text that is available from print management setup. This is one of the print settings loaded in the previous example. When editing the report in Visual Studio, create a hidden string parameter in the report called “IdentificationText,” with properties “Allow Blank” and “Allow null” set to true. To use the parameter on the report, create a SSRS precision design, add a text box and set the “Value” property of the text box expression to “=Parameters!IdentificationText.Value”.

Your document may also have a requirement to show whether it is an original or a copy. If so, this must be determined as each report is printed for the document. Add a variable called “PrintType” to your data contract to store this value. Then override the outputReport() method, and add the value to the data contract before the call to super().

Using print management with PrintMgmtReportRun

Create a new class that derives from the SrsPrintMgmtController class. The base class already contains a class variable of type PrintMgmtReportRun.

```
    PrintMgmtReportRun printMgmtReportRun
```

In the class declaration of your controller class, declare a table variable of the type being printed.

```
    CustInvoiceJour     custInvoiceJour;
```

In this scenario, you may choose to display the controller's dialog by not making a call to the parmShowDialog method from your main method. Also, because the print settings will be supplied by the print management framework, you may want to exclude them from the controller's dialog by overriding the showPrintSettings method.

```
public boolean showPrintSettings()
{
    // When using print management settings, do not show printer settings
    // to the user on the report dialog
    return false;
}
```

In the initPrintMgmtReportRun method of your controller class, initialize the printMgmtReportRun variable.

```
protected void initPrintMgmtReportRun()
{
    printMgmtReportRun = PrintMgmtReportRun::construct(
        PrintMgmtHierarchyType::Sales,
        PrintMgmtNodeType::CustInvoiceTable,
        PrintMgmtDocumentType::SalesFreeTextInvoice);

    printMgmtReportRun.parmReportRunController(this);
}
```

In the runPrintMgmt method, process each record corresponding to a document, retrieve the settings, and print reports for each document. In the following example, the controller's query variable is used, assuming that the controller's dialog was displayed. Note that this method is called on the client tier, so any additional business logic that accesses the database or otherwise executes on the server tier should be done carefully to avoid unnecessary remote procedure calls.

```
protected void runPrintMgmt()
{
    QueryRun queryRun;

    printCopyOriginal = this.parmArgs().parmEnum();

    if (query)
    {
        queryRun = new QueryRun(query);

        if (!queryRun.next())
        {
            throw error("@SYS26348");
        }

        do
        {
            if (queryRun.changed(custInvoiceJour.TableId))
            {
                custInvoiceJour = queryRun.get(custInvoiceJour.TableId);

                this.initPrintMgmtReportRun();

                printMgmtReportRun.load(
                    custInvoiceJour,
                    custInvoiceJour.salesTable(),
                    custInvoiceJour.LanguageId);

                this.outputReports();
            }
         } while (queryRun.next());
    }
}
```

If you want a better understanding of this approach, see the XML documentation that is included in the SrsPrintMgmtController class. Also, be sure to reference the existing SrsPrintMgmtController derivatives for examples.

Conclusion

Now your document can take advantage of the rich features of print management. If you have additional questions about the details behind the print management subsystem, refer to the XML documentation included in the source code or the online help.

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