Microsoft Dynamics® AX 2012

Report programming model for Microsoft Dynamics AX 2012

White Paper

Microsoft SQL Server Reporting Services is the primary reporting platform for Microsoft Dynamics AX. This document describes the programming model to use with the Microsoft Dynamics AX reporting framework.

Date: March 2012

Ashok Srinivasan, Senior Software Development Engineer

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

## Overview .......................................................................................................................... 4

## Overview of concepts ......................................................................................................... 4

## Use cases .......................................................................................................................... 6

## Patterns for converting reports ......................................................................................... 7
   Using a temp table marshaller ............................................................................................ 8
   Use case 1: The report caller has a temporary table that is needed as part of business logic processing to render the report ................................................................. 8
   Use case 2: The report caller context is a class that contains business logic that needs to be used to process the report .................................................................

## Pseudo-temp table ............................................................................................................ 10

## Best practices for reports ................................................................................................ 10

## Practices to avoid for report development ...................................................................... 10

## Print management ............................................................................................................ 10

## Pre-processing RDP ......................................................................................................... 12

## Modifying Query ............................................................................................................... 14

## Reporting Services Time-out Concepts .......................................................................... 14
   Long running reports ........................................................................................................ 14
   Time-out concepts when connecting to Reporting Services ............................................. 14
   Time-out concepts when connecting from Reporting Services to the AOS .................. 15

## Code examples ................................................................................................................ 15
   RDP class ......................................................................................................................... 15

## Executing reports ........................................................................................................... 19

## Sample RDP-based reports ............................................................................................ 20
   RDP example 1: Validation is performed on the contract, default ordering, and grouping .......... 20
   RDP example 2: The UI requires grouping and ordering ......................................................... 23
   RDP example 3: The UI requires many groups and a three-column format .......................... 27
   RDP example 4: The UI requires a group check box .............................................................. 29
   RDP example 5: Condition logic is used to enable UI fields .................................................. 31
   RDP example 6: The report uses a temp table in the caller args .......................................... 33
   RDP example 7: The report uses print management ............................................................... 34
   RDP example 8: The report uses a custom dialog box and UI event overrides ...................... 34
   RDP example 9: The report uses a pre-processed RDP class .............................................. 34

## Sample query-based reports ........................................................................................... 34
Query example 1: Modify the run-time query based on parameter values, without a custom dialog box ........................................................................................................................................34
Query example 2: Override the query init method ........................................................................................................................................36
Overview

Microsoft® SQL Server® Reporting Services (SSRS) is the primary reporting platform for Microsoft Dynamics® AX. The new report programming model is built on the SysOperation framework. The SysOperation framework replaces the RunBase model.

Overview of concepts

The following table describes the concepts for the report programming model.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Class</th>
<th>Description</th>
<th>When to override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report data provider (RDP)</td>
<td><strong>Object</strong></td>
<td>A contract used to specify parameters in an RDP class.</td>
<td>Every RDP class that requires input parameters is associated with a contract. Nested contracts are supported by RDP contracts.</td>
</tr>
<tr>
<td>Report definition language (RDL) contract</td>
<td><strong>SrsReportRdlDataContract</strong></td>
<td>A framework contract used for query-based, data method-based, and OLAP-based reports, and for reports that contain static parameters created in a report. This contract provides a weakly typed representation of parameters. It contains methods that can be used to get or set values. It also contains a map of parameter names and the <strong>SrsReportParameter</strong> class.</td>
<td>Override when you must do the following: • Add custom validation to parameters. Call super() before you validate. • Add custom initialization to your parameters. • Add your own UI builder for the parameters. Use the <strong>SrsReportNameAttribute</strong> attribute on the overridden class to specify which report uses this contract, and to bind the contract to the report. Use this attribute for reports that bind to a query or data method to tie the contract to the report.</td>
</tr>
<tr>
<td>Print contract</td>
<td><strong>SrsPrintDestinationSettings</strong></td>
<td>A contract that contains all the relevant contracts after a report RDL is parsed. This contract contains an instance of the RDP contract, RDL contract, print contract, and query contract.</td>
<td>Do not override.</td>
</tr>
<tr>
<td>Query contract</td>
<td>No direct class. A map that contains the parameter name and query object.</td>
<td>A contract that provides the query contracts that are used in the report.</td>
<td>Do not override.</td>
</tr>
<tr>
<td>Report contract</td>
<td><strong>SrsReportDataContract</strong></td>
<td>A contract that contains all the relevant contracts after a report RDL is parsed. This contract contains an instance of the RDP contract, RDL contract, print contract, and query contract.</td>
<td>Do not override.</td>
</tr>
</tbody>
</table>
### Concept | Class | Description | When to override
---|---|---|---
Controller | **SrsReportRunController** | The controller of the Model View Controller (MVC) pattern. Given the report name, the controller does the following:  
- Call **SrsReportRunInterface** to parse RDL.  
- Get the report contracts.  
- Create the necessary UI builders and invoke them.  
- Call **validate** on contracts. Validation occurs on the server through a service call.  
- Save to the SysLastValue table.  
- After the form is displayed, and the user clicks **OK**, run the report. | Override when you must do the following:  
- Change the contract before you run it. For example, to change the query based on parameters in the form, override the **modifyReportContract** method.  
- React to form control events. In this case, override the method, and provide the override events.  
- Add basic validation that is not part of the contract, or not at the table level. In this case, override the **validate** method, and call the **super** method.  
- Change the name of the report that is being run based on a parameter. In this case, override the **modifyReportContract** method, and set **SrsReportDataContract.parmReportName**.  
- Modify the company or culture. In this case, override the **modifyReportContract** method, and set the company and culture on **SrsReportRdlDataContract**.  

| Group and order | **SrsReportDataContractUIBuilder** | You specify the group and order as follows:  
- If you are using an RDP class, specify them on the contract or in Visual Studio.  
- If you are using a query or data method, specify them by using report designer.  
- If you are using a mix of an RDP class, query, and data method, specify them by using Visual Studio. | In Visual Studio, you can create groups and orders, and preview them in Visual Studio and the Microsoft Dynamics AX client.  

| UI builders | **SrsReportDataContractUIBuilder** | UI builders do the following:  
- Lay out the fields on the contract.  
- Transfer data from a field to the contract, and bind fields to contract members.  
The group and order are specified in the contract in the Visual Studio designer.  
The **SrsReportDataContractUIBuilder** class provides the capability to show the date effective tab and valid values that are specified in the report.  
The **SysOperationAutomaticUIBuilder** class provided by the SysOperation framework renders the UI for a given data contract. | Override when you must do the following:  
- Provide additional grouping or ordering. For example, you can provide radio buttons for groups.  
- Change the layout from one column to multiple columns. For example, you can display three columns of parameters on the form.  
Note: To react to control events, you do not need to override the UI builder, because the control override methods need to be specified in the controller.  
Evaluate how much you need to override. For example, you can change the layout from one to three columns without an override. In the UI builder, override the build method, update the current form group property columns to 3, and then call **super()**. The base class lays out the fields.  


### Validation

<table>
<thead>
<tr>
<th>Concept</th>
<th>Class</th>
<th>Description</th>
<th>When to override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>RDP contract</td>
<td><strong>SrsReportRdIDataContract</strong> or its override</td>
<td>Contracts must implement the SysOperationValidatable interface that provides a <code>validate</code> method. For RDP contracts, implement the <code>validate</code> method, and return <code>True</code> or <code>False</code>. Use the <code>error</code> method, and write to the Infolog. The error messages in the Infolog are marshaled from the service to the client, and then rendered. Do not throw an error. For RDL contracts, the <code>SrsReportRdlDataContract</code> class implements default validation that performs basic parameter validation. It also validates the company and culture. When you override this class, first call <code>super()</code> to allow the base class to validate, and then implement your validation logic.</td>
</tr>
</tbody>
</table>

### Initialization

<table>
<thead>
<tr>
<th>Concept</th>
<th>Class</th>
<th>Description</th>
<th>When to override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>RDP contract</td>
<td>Initialization provides support for overriding the contract and implementing your contract initialization.</td>
<td></td>
</tr>
</tbody>
</table>

### Use cases

The following table describes use cases when you develop reports.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Developer action</th>
<th>Unit testing artifact</th>
</tr>
</thead>
</table>
| A report is bound to a query, data method, or OLAP, and validation logic needs to be added on the parameters. | 1. Extend the `SrsReportRdIDataContract` class, and override the `validate` method.  
2. On the contract, add the `SrsReportNameAttribute('MyReportName')` attribute.  
Do not override the controller. | Test the validation logic on the data contract. |
| A report is bound to an RDP class, and validation logic needs to be added on the parameters. | The RDP data contract implements the SysOperationValidatable interface.  
Do not override the controller. | Test the RDP contract. |
| A report is bound to an RDP class and needs to change the UI dialog box for the report. The framework’s support for grouping and ordering is not sufficient for the report’s needs. | 1. Extend the `SrsReportDataContractUIBuilder` class, and implement the UI customizations.  
2. On the RDP contract, add a `SysOperationContractProcessingAttribute` attribute that specifies the name of the UI builder to use.  
Do not override the controller. | Test the UI builder by using the dialog API. |
| A report is bound to a query, data method, or OLAP, and needs to have grouping or ordering. | Use Visual Studio to group and order report parameters.  
Do not override the controller. A contract override is required. |                                |
| A report uses RDL and needs to change the UI dialog box for the report. The framework’s support for grouping and ordering is not sufficient for the report’s needs. | 1. Extend the `SrsReportDataContractUIBuilder` class, and implement UI customizations.  
2. Use the `SysOperationContractProcessingAttribute` attribute on the custom RDL contract.  
Do not override the controller. | Test the UI builder by using the dialog API. |
## Use case

<table>
<thead>
<tr>
<th>Use case</th>
<th>Developer action</th>
<th>Unit testing artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>A report needs to change the name of the report design used at run time.</td>
<td>Extend the SrsReportRunController class, override the preRunModifyContract method, and then provide a different name for the report in the SrsReportDataContract class.</td>
<td>Test the controller.</td>
</tr>
</tbody>
</table>

## Patterns for converting reports

Report patterns emerged when Microsoft Dynamics AX reports were converted for the current release. The following table describes the report task, the programming model pattern, and a sample report that uses the pattern in Microsoft Dynamics AX.

<table>
<thead>
<tr>
<th>Report task</th>
<th>Report programming model pattern</th>
<th>Sample report</th>
</tr>
</thead>
</table>
| An RDP report needs simple grouping and validation.                         | 1. Create an RDP contract.  
2. Implement the SysOperationValidatable class and validation.  
3. Do parameter grouping on the contract.                                 | AssetDepreciationLedger_IT InventABC                                                       |
| An RDP report needs complex grouping, such as horizontal, vertical, nested, or multiple groups. | Do parameter grouping by using Visual Studio tools for Microsoft Dynamics AX.                  |                                                |
| A query-based, data method-based, or OLAP-based report needs grouping.     | Do parameter grouping by using Visual Studio tools for Microsoft Dynamics AX.                  |                                                |
| A custom field needs to be created in the dialog box, because, for example, the field is not a report parameter. The field is added based on certain conditions. The custom dialog box fields are added, and the report parameters are populated based on them. | Create a UI builder:  
1. Move the code in the addCustomCtrl method to the UILBuilder.build method.  
2. Move the getFromDialog method to the UILBuilder.getFromDialog method, and populate the contract.  
3. Associate the UI builder class with the contract by using the attribute SysOperationContractProcessingAttribute(classtr(UIBuilderName), SysOperationDataContractProcessingMode::CreateUIBuilderForRootContractOnly).  
4. Optional: Create a controller, and if further modification of the contract is required before the report is run, add the logic to the preRunModifyContract method. | AgreementFollowup BOMPartOf |
| Code needs to react to dialog box field events, such as when a field is modified. | Create a UI builder:  
- In postRun(), register override methods for field events, and associate the fields events with methods. | BudgetFundsAvailableUIBuilder |
| A dialog box field needs to be changed to have a different button type than the default, or a field needs to turn on a flag on a grouping. | Create a UI builder:  
- In the postBuild method, get the dialog box field from the binding information, and change the field. | BudgetDetailsUIBuilder |

---

*Report Programming Model for Microsoft Dynamics AX 2012*
<table>
<thead>
<tr>
<th>Report task</th>
<th>Report programming model pattern</th>
<th>Sample report</th>
</tr>
</thead>
<tbody>
<tr>
<td>A helper class needs to turn visibility of a report parameter on or off, so that new custom parameters can be added.</td>
<td>• Change the report parameter in the report design so that it is hidden. The framework takes care of not displaying the parameter in the Microsoft Dynamics AX form.</td>
<td>AgreementFollowUpController</td>
</tr>
<tr>
<td>A helper class needs to add a caption for the report.</td>
<td>This task should be avoided. The framework uses the label on the menu item that launches the report as a caption. For cases where the same report is used in multiple reports, override the <code>controller.prePromptModifyContract</code> method, and set <code>parmDialogCaption</code>.</td>
<td>CustAccountStatementInt</td>
</tr>
<tr>
<td>A dialog box field needs to be enabled or disabled based on certain conditions.</td>
<td>Create the UI builder: 1. Override the <code>getFromDialog</code> method if you want to get the custom field value from the dialog box. 2. Override the <code>postBuild</code> method to enable or disable fields.</td>
<td>AgreementFollowUpController</td>
</tr>
<tr>
<td>A query is modified based on the caller args before the UI is rendered.</td>
<td>Create a controller: • Override the <code>prePromptModifyContract</code> method.</td>
<td>CustAccountStatementInt</td>
</tr>
<tr>
<td>A query is modified before the report is run.</td>
<td>Create a controller: • Override the <code>preRunModifyContract</code> method.</td>
<td>AgreementFollowUpController</td>
</tr>
<tr>
<td>The report name needs to be changed based on the caller args or a UI parameter.</td>
<td>Create a controller: • Override the <code>preRunModifyContract</code> method.</td>
<td>CustInterestNoteHelper</td>
</tr>
<tr>
<td>The helper class needs to use print management.</td>
<td>Create a controller: 1. Override the <code>run</code> method. 2. Override the <code>preRunModifyContract</code> method, and also override the <code>prePromptModifyContract</code> method as needed.</td>
<td>Cheque_US</td>
</tr>
<tr>
<td>The report parameter UI should not be displayed.</td>
<td>Create a controller: • In the main method, before the <code>controller.startOperation</code> method, call <code>controller.parmShowDialog(false)</code>.</td>
<td>SalesConfirm</td>
</tr>
<tr>
<td>The report needs to use a temp table from a caller.</td>
<td>Create a controller: 1. Get the temp table from the caller, and use the <code>SRSTmpTblMarshaller</code> class to save the data. 2. Develop an RDP class to extract this data and perform business logic processing.</td>
<td>PurchPackingSlip ProjectInvoice</td>
</tr>
<tr>
<td>The report needs to use print management.</td>
<td>See the &quot;Print management&quot; section.</td>
<td></td>
</tr>
</tbody>
</table>

### Using a temp table marshaller

Use of a temp table marshaller needs to be very limited. The following are the use cases for a temp table marshaller.

**Use case 1: The report caller has a temporary table that is needed as part of business logic processing to render the report**

- Use an RDP class to write the business logic.
• Use a controller class.
• Use a marshaller inside the controller class to pass the temp table that is used in the RDP class.
• Use an RDP contract to pass any other necessary details.
• Use the SrsTmpTblMarshallerContract class as a nested contract in the RDP contract.
• Do not process business logic in the controller class.

**Example: The Cheque_US report**
The Cheque_US report is an example of a report that uses a temp table marshaller.

**Best practices:**
• In the controller class, use a marshaller to send a temp table that contains the data that must be processed.
• In an RDP class, get the temp table, perform business logic processing on it, and return the data.
• The advantage is that the RDP class is always called as a service on the Application Object Server (AOS) instance, so the report runs more efficiently.

**Practices to avoid:**
• Do not use a controller class to process all the business logic, and then use a marshaller to store the final data that needs to be shown.
  
  In RDP, data is read from the marshaller, and the dataset is returned to SSRS.
• The problem is that the business logic processing in the controller is run on the client and creates many database calls.

**Use case 2: The report caller context is a class that contains business logic that needs to be used to process the report**
• If the caller class can be packed or unpacked, use the SRSTmpDataStore table to store the packed value.
• If the caller class cannot be packed or unpacked, check whether relevant parameters from the caller class can be captured to create the logic in an RDP class.
• If neither of the previous actions is possible, process the data in a controller class.
• Insert the processed data in a marshaller, and then use an RDP class to get the data out.
**Pseudo-temp table**

Many reports from the previous release use the following pseudo-temp pattern:

- The table contains a sessionId column.
- Business logic processing occurs on the client, and data is then written to the table together with the sessionId.
- The report uses a query that has the sessionId as a predicate.

**Note:** Do not use this approach for reports from the current release. This approach is not framework approved. Because the data is not cleaned up immediately, it is vulnerable to security attacks and information disclosure.

**Best practices for reports**

- Use an RDP class to write the business logic.
- Use a controller class.
- Use a marshaller inside the controller to pass the temp table that is used in the RDP class.
- Use an RDP contract to pass any other necessary details.
- Use the `SrsTmpTblMarshallerContract` class as a nested contract in your RDP contract.
- Do not process business logic in a controller class.
- When a Surrogate Foreign Key (SFK) control is in a Microsoft Dynamics AX client reporting dialog, the following are required:
  - A SFK field in the table must have an EDT.
  - The EDT must have the Reference table property set to the natural key table.

**Practices to avoid for report development**

The following list describes practices that you should avoid when you define reports:

- Do not specify a parameter if parameters are not used anywhere in the code or report.
- Do not use an RDP class if a query can provide the same result.
- Do not use a helper class that provides no value.
- Do not define a label or caption for parameters if the same value can come from an EDT.
- Do not use a UI builder to hide report parameter visibility.
- Do not use UI builder to specify a report caption, except when a menu item launches multiple reports.

**Print management**

The following list describes print management–based reports:

1. Data needs to be processed based on user selection in a form. For example, a user selects three sales orders that must be invoiced.
2. For each sales order, the relevant print management settings are loaded into code by using the print management framework.
3. One or more print management settings are produced.
4. For each print management setting for the same report, a report should be printed.
If a report requires that business logic be run to render the data, the business logic should run once, for all print management settings. Running the business logic each time that the same report is rendered for different print management settings has the following effects:

- Data can become inconsistent. For example, the sales order copy might have an extra line that the original did not have.
- Performance is reduced, because the same business logic is run repeatedly.

By using an RDP class to have business logic, you can materialize the data. Make sure that the RDP class is run once for each sales order that is selected in the form. To ensure that the RDP class is only run once, the framework provides the concept of a pre-processing RDP. As a best practice, use a pre-processing RDP to perform business logic.

**Pre-processing RDP**

The following steps describe how to develop a print management report that has a pre-processing RDP class.

1. Implement a pre-processing RDP:
   1. Extend the RDP class from the `SrsReportDataProviderPreProcess` class.
   2. The tables that store data that is used by the report should have the following properties set:
      - `Table Type` = Regular
      - `CreatedBy` = Yes
      - `CreatedTransactionId` = Yes

2. Explicitly set the User Connection object on the temp table(s) that are used in the RDP class to return data. You must set the User Connection before the tables can be used. It is a best practice to set it in the beginning of `processReport()` method. For example, the `SalesInvoiceDP` class uses the `salesInvoiceTmp` class variable of the `SalesInvoiceTmp` table. To set the User Connection, provide the `parmUserConnection` to the Data Provider class like the following:

   ```
salesInvoiceTmp.setConnection(this.parmUserConnection());
   ```

3. Open the report in Visual Studio and refresh the data source to include the new `CreatedTransactionId` field. To do this, in Model Editor, right-click the dataset and then click **Refresh**.

4. In Solution Explorer, right-click the solution and click **Deploy Solution** to deploy the new report.

5. In Microsoft Dynamics AX development workspace, generate the Incremental CIL. To do this, right-click the **AOT**, point to **Add-ins**, and then click **Incremental CIL generation from X++**.

6. Implement the controller class:
   - When print management uses, for example, the **Form Letter** report:
     1. Extend the controller class from the `SrsPrintMgmtFormLetterController` class.
     2. Override the `initFormLetterReport` method to initialize the relevant form letter class based on input arguments. Use the `formLetterReport` member variable to store the input arguments. Call the `super` method before you exit the method.
     3. Override the `runPrintMgmt` method:
        1. Write a loop to go over data that was selected in the form, and that needs to be printed.
        2. Load the relevant print settings for the given business record.
        3. Call the `outputReports` base class.
4. Override the `preRunModifyContract` method to modify the RDP contract parameters before the report is run. This method is called as part of the `outputReports` method.

5. Override the `outputReport` method to set the footer text or any other special parameters before each copy of report is printed.

6. Create a static main method. In most of these reports, the report parameter dialog box is not displayed. You can suppress the dialog box by using `parmShowDialog(false)`.

- When print management uses direct print management classes:
  1. Extend the controller class from the `SrsPrintMgmtController` class.
  2. Override the `initPrintMgmt` method.
  3. Override the `runPrintMgmt` method:
     1. Write a loop to go over data that was selected in the form, and that needs to be printed.
     2. Load the relevant print settings for the given business record.
     3. Call the `outputReports` base class.
  4. Override the `preRunModifyContract` method to modify the RDP contract parameters before the report is run. This method is called as part of the `outputReports` method.
  5. Override the `outputReport` method to set the footer text or any other special parameters before each copy of report is printed.
  6. Create a static main method. In most of these reports, the report parameter dialog box is not displayed. You can suppress the dialog box by using `parmShowDialog(false)`.
  7. Set the UserConnection for each table that is returned from the RDP:

In the RDP class for the tables being returned, you must set the UserConnection before it can be used. The following code example illustrates returning the `SalesInvoiceTmp` table:

```java
[SRSReportDataSetAttribute(tableStr(SalesInvoiceTmp))]
public SalesInvoiceTmp getSalesInvoiceTmp()
{
    select salesInvoiceTmp;
    return salesInvoiceTmp;
}
```

The following code illustrates setting the user connection to use on the `SalesInvoiceTmp` table. This code is added to the `processReport` method in the RDP class. You must set the UserConnection before code is executed for each of the tables that are returned from the RDP:

```java
// Set the userconnection to use on table.
// This is required to ensure that createdTransactionId of inserted record is different than default transaction.
salesInvoiceTmp.setConnection(this.parmUserConnection());
```

For a complete example, see the `SalesInvoiceDP`, and `ProjInvoiceDP` class.
Modifying Query

The reporting framework does not invoke init on query in the SysIOperationFramework class. Invoking init is specific to forms. You can override the controller and call the init in the PrePrompt or PreRunModifyContract class.

For the error you are getting: “Pre-Processed RecId parameter not found. Cannot process report. Indicates a development error”

After you have changed the RDP class, compile it and then go to Visual Studio, then open the report and do a refresh on the dataset (which refers to the RDP), the new parameter will get added. Now re-deploy the report and you should be ok.

Reporting Services Time-out Concepts

A report time-out can occur in various places when a report is rendered. It is recommended that you move long running reports to the pre-processed model. Consider the following to improve reports with large datasets:

1. Is the report a query based report or a RDP based report?
2. Investigate whether there is inefficiency in the way the data is accessed.
3. Can you add an index to table?
4. If RDP is being used, is the code working correctly and does not have unnecessary loops?
5. Can you improve the user performance by only showing the first 100K records?
6. Can you provide a parameter that the end user can filter the data on?

Long running reports

Long running reports are classified as reports that take more than 10 minutes to process the business data. If the Reporting Services to AOS communication takes longer than 10 minutes, a time-out will occur and the report will not render. Examples of long running reports can occur in the processReport method of RDP based reports. You must change the RDP based report to use the pre-processed format. The RDP business logic processing occurs before Reporting Services is called. RDP Example 9 illustrates an example of a pre-processed RDP class.

The following sections describe time-out concepts when connecting to or from Reporting Services.

Time-out concepts when connecting to Reporting Services

This section describes how time-outs can impact report viewing like viewing the report using print to screen, print to file, or print to email.

The following list describes scenarios when connecting to Reporting Services causes a time-out:

- **Execution session time-out**: Time-out occurs when using a URL in the report viewer. The time can be increased using a script.
- **Asp.NET time-out**: The Reporting Services built-in web server times-out. The web.config contains the execution time-out value. The default time-out value is 90 seconds as shown in the following web.config:
  
  ```xml
  <httpRuntime maxRequestLength="100000" executionTimeout="90000" /> 
  ```
- **Web services time-out**: Use web services to connect to Reporting Services and the web service proxy has a time-out property that can be set. Setting to -1 provides infinite time. In reporting, you can set the time-out value to -1 in code.

For more information, see [Time-out Values for Report Processing](#).

### Time-out concepts when connecting from Reporting Services to the AOS

This section describes how time-outs can impact reports when connecting from Reporting Services to the AOS.

- **Data execution time-out**: Specifies the time-out for data processing to occur. This needs to be set in RDL for data extensions. If omitted, it defaults to infinite as set by the reporting framework.
- **Report processing time-out**: Specifies time taken to process the report, including data execution. This can be set per report or site level using Report Manager. The default is **Do not timeout report execution**.

The following screenshot illustrates setting the site level settings:

![Time-out settings](image)

The following screenshot illustrates setting the report level settings:

![Time-out settings](image)

- **WCF time-out**: The data extension in Reporting Services connects to the AOS using Windows Client Foundation (WCF). You can configure the binding to allow a large time-out.
  - On the client side (report extension configuration): Binding.SendTimeout
  - On server side (AOS configuration): Binding.ReceiveTimeout

### Code examples

#### RDP class

The following code example illustrates how to define an RDP class, including the attributes and extension of the `SrsReportDataProviderPreProcess` class.

```csharp
[ SRSReportQueryAttribute(queryStr 'ProjPrintInvoice'), SRSReportParameterAttribute(classStr 'ProjInvoiceContract_New') ]
```
class ProjInvoiceDP_New extends SrsReportDataProviderPreProcess

The following screenshots illustrate the properties that are set on the temp table that is created for an RDP class.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>10424</td>
</tr>
<tr>
<td>Name</td>
<td>ProjInvoiceTmp</td>
</tr>
<tr>
<td>Label</td>
<td>Project invoice</td>
</tr>
<tr>
<td>FormRef</td>
<td></td>
</tr>
<tr>
<td>ListPageRef</td>
<td></td>
</tr>
<tr>
<td>ReportRef</td>
<td></td>
</tr>
<tr>
<td>PreviewPatRef</td>
<td></td>
</tr>
<tr>
<td>SearchLinkRefId1</td>
<td></td>
</tr>
<tr>
<td>SearchLinkRefId2</td>
<td></td>
</tr>
<tr>
<td>TitleField1</td>
<td></td>
</tr>
<tr>
<td>TitleField2</td>
<td></td>
</tr>
<tr>
<td>TableType</td>
<td>Regular</td>
</tr>
<tr>
<td>ModifiedInstanceId</td>
<td>No</td>
</tr>
<tr>
<td>CreatedDateTime</td>
<td>No</td>
</tr>
<tr>
<td>CreatedBy</td>
<td>Yes</td>
</tr>
<tr>
<td>CreatedInstanceId</td>
<td>Yes</td>
</tr>
<tr>
<td>OccEnabled</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The following code examples illustrate how to define a controller class. You can see the class and all the method code by clicking AOT > Classes > ProjInvoiceController. The following examples illustrate how to extend the SrsPrintMgmtFormLetterController class, main method, initFormLetterReport method, runPrintManagement method, preRunModifyContract method, and the outputReport method.

class ProjInvoiceController_New extends SrsPrintMgmtFormLetterController

public client static void main(Args _args)
{
    ProjInvoiceController_New controller = new ProjInvoiceController_New();
    controller.parmReportName(#ReportName);
    controller.parmArgs(_args);
    controller.parmShowDialog(false);
    controller.startOperation();
}

protected void initFormLetterReport()
{
    ProjPrintInvoice projPrintInvoice;
    printCopyOriginal = this.parmArgs().parmEnum();
    if (args && args.caller())
    {
        if (classIdGet(args.caller()) == classNum(ProjFormLetter_Invoice))
        {
            // Code for initFormLetterReport method
        }
    }
}
projFormLetter = args.caller();

if (classIdGet(args.caller()) == classNum(ProjPrintInvoice))
{
    projPrintInvoice = args.caller();
}

if (args.record())
{
    journalList = FormLetter::createJournalListCopy(args.record());
}
else
{
    journalList = args.object();
}

this.filterReprintedOriginals_IS();

formLetterReport = FormLetterReport::construct(PrintMgmtDocumentType::ProjectInvoice);
formLetterReport.parmPrintType(printCopyOriginal);

if (projFormLetter)
{
    formLetterReport.parmDefaultCopyPrintJobSettings(new SRSPrintDestinationSettings(projFormLetter.printerSettingsFormletter(PrintSetupOriginalCopy::Copy)));
    formLetterReport.parmDefaultOriginalPrintJobSettings(new SRSPrintDestinationSettings(projFormLetter.printerSettingsFormletter(PrintSetupOriginalCopy::Original)));
    formLetterReport.parmUsePrintMgmtDestinations(projFormLetter.usePrintManagement());
}
else if (printCopyOriginal == PrintCopyOriginal::OriginalPrint)
{
    // Always use the print mgmt destinations when reprinting for the OriginalPrint case.
    formLetterReport.parmUsePrintMgmtDestinations(true);
}
else if (projPrintInvoice)
{
    formLetterReport.parmDefaultCopyPrintJobSettings(new SRSPrintDestinationSettings(projPrintInvoice.printerSettingsPrintInvoice()));
    formLetterReport.parmDefaultOriginalPrintJobSettings(new SRSPrintDestinationSettings(projPrintInvoice.printerSettingsPrintInvoice()));
    formLetterReport.parmUsePrintMgmtDestinations(false);
    formLetterReport.parmUseUserDefinedDestinations(true);
}

// Init projcredit invoice flag. Used to get invoicetext footer.
projCreditInvoicing = CustParameters::find().CreditInvoicingReport;

projInvoiceContract = this.parmReportContract().parmRdpContract() as ProjInvoiceContract;
super();

protected void runPrintMgmt()
{
    if (this.parmArgs().menuItemName() == menuitemOutputStr(ProjPrintInvoice))
    {
        this.createJournalListFromQuery();
    }

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

    journalList.first(projInvoiceJour);

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

    do
    {
        formLetterReport.loadPrintSettings(projInvoiceJour, ProjInvoiceTable::find(projInvoiceJour.ProjInvoiceProjId), projInvoiceJour.LanguageId);
        this.outputReports();
    } while (journalList.next(projInvoiceJour));
}

public void preRunModifyContract()
{
    SRSTmpDataStore srsTmpDataStore;

    // set the current projinvoice jour recid
    projInvoiceContract.parmProjJourRecId(projInvoiceJour.RecId);
    projInvoiceContract.parmCountryRegionISOCode(SysCountryRegionCode::countryInfo());

    // set proforma and packed form letter. We dont need to do this in preRunModifyContract(),
    // since this needs to be set only once.
    if (projFormLetter && projFormLetter.proforma())
    {
        projInvoiceContract.parmProforma(true);

        // pack container and put into srstmptable.
        // Pack the class and insert into the temporary store.
        srsTmpDataStore.Value = projFormLetter.parmFormletterProformaPrintPacked();
        srsTmpDataStore.insert();
}
// Set the rec id to contract parameter
projInvoiceContract.parmFormLetterRecordId(srsTmpDataStore.RecId);
}

super();
}

protected void outputReport()
{
    SRSPrintDestinationSettings srsPrintDestinationSettings;
    SRSPrintMediumType srsPrintMediumType;

    // For EVERY print mgmt setting
    // Set the invoice text.
    projInvoiceContract.parmInvoiceTxt(this.invoiceTxt());

    if (SysCountryRegionCode::isLegalEntityInCountryRegion(#isoIS))
    {
        srsPrintDestinationSettings = formLetterReport.getCurrentPrintSetting().parmPrintJobSettings();
        srsPrintMediumType = srsPrintDestinationSettings.printMediumType();
        if ((printCopyOriginal == PrintCopyOriginal::Original || printCopyOriginal == PrintCopyOriginal::OriginalPrint)
            && srsPrintMediumType == SRSPrintMediumType::Printer)
        {
            ProjInvoiceJour::updatePrinted(projInvoiceJour, srsPrintDestinationSettings.numberOfCopies());
        }
    }

    super();
}

Executing reports
1. The RDL is stored in the Application Object Tree (AOT).
2. To deploy a Microsoft Dynamics AX reporting project in Microsoft Visual Studio®, you must start Visual Studio with administrative privileges, or you must start Microsoft Dynamics AX and the Developer Workspace with administrative privileges. Right-click the icon for Visual Studio or Microsoft Dynamics AX, and then click Run as administrator. The following table describes the options for deploying reports.

<table>
<thead>
<tr>
<th>Deployment option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Dynamics AX</td>
<td>Reports can be deployed individually from the development workspace in Microsoft Dynamics AX. In the AOT, expand the SSRS Reports node, expand the Reports node, right-click the report, and then click Deploy Element. The reports are deployed for all the translated languages.</td>
</tr>
</tbody>
</table>
### Deployment option

<table>
<thead>
<tr>
<th>Deployment option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visual Studio</td>
<td>Reports can be deployed individually from Visual Studio. In Solution Explorer, right-click the reporting project that contains the reports that you want to deploy, and then click <strong>Deploy</strong>. The reports are deployed only for the neutral, or invariant, language.</td>
</tr>
<tr>
<td>Windows PowerShell</td>
<td>The default reports that are provided with Microsoft Dynamics AX can be deployed by using Windows PowerShell. For more information, see <strong>Deploy the Default Reports</strong>.</td>
</tr>
</tbody>
</table>

3. Reports follow the naming convention of `ReportName.ReportDesign`. The Visual Studio project name is not included in the name.

4. When you add a report to a menu item, select the `SSRSReport` object type. You can then select a report and report design in the AOT by using a drop-down list of deployed reports. The following screenshot illustrates the menu item properties that you set when you create a menu item for a report.

<table>
<thead>
<tr>
<th>HelpExt</th>
<th>SSRSReport</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>AgreementFollowUp</td>
</tr>
<tr>
<td>Object</td>
<td>Report</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
</tbody>
</table>

### Sample RDP-based reports

The following sections provide projects, code, and information about reports that had common scenarios when they were converted to use RDP and the current report programming model.

**RDP example 1: Validation is performed on the contract, default ordering, and grouping**

The following example provides information about how the `InventABC` report was converted to use the current report programming model. This report includes validation on the contract, default ordering, and grouping.

- The `InventABC` report was migrated to the current report programming model.
- Validation on the `InventABCHelper` class was moved to the contract.
- The helper class was removed.

The following screenshot illustrates the parameter form for the `InventABC` report that ships with Microsoft Dynamics AX.
The following code illustrates the InventABC contract.

```
[DataContractAttribute]
public class InventABCContract implements SysOperationValidatable
{
    TransDate       fromDate;
    TransDate       toDate;
    InterestPct     interest;
    Percent         categoryA;
    Percent         categoryB;
    Percent         categoryC;
    ABCModel        abcModel;
}
```
The following code illustrates the **validate** method.

```java
public boolean validate()
{
    boolean isValid = true;
    if (this.parmABCModel() == ABCModel::Link)
    {
        if (this.parmInterest() <= 0)
        {
            error("@SYS8377");
            isValid = false;
        }
        if (! this.parmToDate())
        {
            error("@SYS24515");
            isValid = false;
        }
    }
    if (this.parmInterest() < 0)
    {
        error("@SYS118311");
        isValid = false;
    }
    if (this.parmCategoryA() < 0)
    {
        error("@SYS1724");
        isValid = false;
    }
    if (this.parmCategoryB() < 0)
    {
        error("@SYS10103");
        isValid = false;
    }
    if (this.parmCategoryC() < 0)
    {
        error("@SYS1723");
        isValid = false;
    }
    if ((this.parmCategoryA() + this.parmCategoryB() + this.parmCategoryC()) != 100)
    {
        error("@SYS17104");
        isValid = false;
    }
    if (this.parmFromDate() && this.parmToDate() && this.parmFromDate() > this.parmToDate())
    {
        error("@SYS91020");
        isValid = false;
    }
    return isValid;
}
```
The following screenshot illustrates how the **InventABC** report appears in the report viewer.

![InventABC report screenshot]

RDP example 2: The UI requires grouping and ordering

The following example provides information about how the **AssetDepreciationLedger_IT** report was converted to use the current report programming model. This report provides an example of a report that requires grouping and ordering in the UI.

- The **AssetDepreciationLedger_IT** report was migrated to new model.
- Grouping and ordering were added to the contract to provide groups and the order of parameters.

The following steps explain how to do add groups and set the order of parameters:

1. Two groups are required for this contract, so define two group attributes on the class declaration. You can specify the order in which the groups are laid out.

2. Specify which group each data member is part of. Specify the order in which the data members appear in the group. For example, in the **Period** group, the **FromDate** data member should be first, followed by the **ToDate** data member.

If you do not specify a group for a data member, the dialog box includes the data member as part of the **Parameters** root group.

The following screenshot illustrates the parameter form for the **AssetDepreciationLedger_IT** report that ships with Microsoft Dynamics AX.
The following code illustrates the AssetDepreciationLedger contract that specifies the grouping and ordering.

```csharp
[DataContractAttribute,
 SysOperationGroupAttribute('Period',"@SYS40","1"),
 SysOperationGroupAttribute('Detailed',"@SYS95926","2")
]

public class AssetDepreciationLedgerContract_IT
{
    FromDate    fromDate;
    ...
}
```

The following blocks of code illustrate how the parameters on the report are grouped and ordered by using the attributes SysOperationGroupMemberAttribute and SysOperationDisplayOrderAttribute:

```csharp
[DataMemberAttribute('FromDate'),
 SysOperationLabelAttribute(literalstr("@SYS71135")),
 SysOperationHelpTextAttribute(literalstr("@SYS71135")),
 SysOperationGroupMemberAttribute('Period'),
 SysOperationDisplayOrderAttribute('1')
]

public FromDate parmFromDate(FromDate _fromDate = fromDate)
{
    fromDate = _fromDate;
    return fromDate;
}
```

```csharp
[DataMemberAttribute('ToDate'),
 SysOperationLabelAttribute(literalstr("@SYS71136")),
 SysOperationHelpTextAttribute(literalstr("@SYS71136")),
 SysOperationGroupMemberAttribute('Period'),
 SysOperationDisplayOrderAttribute('2')
]

public ToDate parmToDate(ToDate _toDate = toDate)
{
}
```
toDate = _toDate;
return toDate;
}

[DataMemberAttribute('DetailedPrint'),
SysOperationLabelAttribute(literalstr("@SYS95927")),
SysOperationGroupMemberAttribute('Detailed'),
SysOperationDisplayOrderAttribute('1')
]

public boolean parmDetailedPrint(boolean _detailedPrint = detailedPrint)
{

detailedPrint = _detailedPrint;
return detailedPrint;
}

The following screenshot illustrates how the **AssetDepreciationLedger_IT** report appeared in the report viewer before it was converted. Note that the parameters are empty, because a helper class was used.
The following screenshot illustrates the AssetDepreciationLedger_IT report when it has two groups, Period and Detailed. The Period group reflects the specified order of the parameters.

The following code illustrates how you can change the DetailedPrint data member so that it is not part of a group.

```csharp
[DataMemberAttribute("DetailedPrint"),
 SysOperationLabelAttribute(literalstr("@SYS95927")),
 // SysOperationGroupMemberAttribute("Detailed"),
 SysOperationDisplayOrderAttribute("1")
]

public bool parmDetailedPrint(bool _detailedPrint = detailedPrint)
{
    detailedPrint = _detailedPrint;
    return detailedPrint;
}
```
The following screenshot illustrates the **Detailed print** parameter when no group is specified for it. Notice that it is part of root **Parameters** group.

![Screenshot of report parameters]

**RDP example 3: The UI requires many groups and a three-column format**

The following example provides information about how the **CustInvoiceSettled_TransDate** report was converted to use the current report programming model. This report provides an example of a report that requires three columns in the UI.

- The **CustInvoiceSettled_TransDate** report was migrated to the current report programming model.
- The grouping was moved from the helper class to the contract.
- The UI builder was used to override the group control so that three columns are shown in the UI.
The following screenshot illustrates the parameter form for the **CustInvoiceSettled_TransDate** report that ships with Microsoft Dynamics AX.

![Parameter Form for CustInvoiceSettled_TransDate Report](image)

The following code illustrates the **CustInvoiceSettled_TransDate** contract when the **SysOperationContractProcessingAttribute** attribute is attached to specify the UI builder class.

```csharp
[DataContractAttribute,
 SysOperationContractProcessingAttribute(
 classstr (CustInvoiceSettled_TransDateUI_ES),
 SysOperationDataContractProcessingMode::CreateUIBuilderForRootContractOnly),
 SysOperationGroupAttribute(
 'CustAccount', "@SYS7149", '1'),
 SysOperationGroupAttribute('PostingDate', "@SYS14475", '2'),
 SysOperationGroupAttribute('DueDate', "@SYS14588", '3'),
 SysOperationGroupAttribute('Status', "@SYS25587", '4'),
 SysOperationGroupAttribute('MethodPaym', "@SYS21698", '5'),
 SysOperationGroupAttribute('BillId', "@SYS71453", '6'),
 SysOperationGroupAttribute('Invoice', "@SYS4726", '7'),
 SysOperationGroupAttribute('Amount', "@SYS53072", '8')
]

public class CustInvoiceSettled_TransDateCtct_ES
{
    ...  

    The following code illustrates how to set the **Amount** parameter group for the fromAmount and toAmount parameters.

```csharp
[DataMemberAttribute('FromAmount'),
 SysOperationLabelAttribute(literalstr("@SYS53072")),
 SysOperationGroupMemberAttribute('Amount'),
 SysOperationDisplayOrderAttribute('1')
]

public AmountMST parmFromAmount(AmountMST _fromAmount = fromAmount)
{
    fromAmount = _fromAmount;
    return fromAmount;
}

```
DataMemberAttribute('ToAmount'),
SysOperationLabelAttribute(literalstr('@SYS53072')),
SysOperationGroupMemberAttribute('Amount'),
SysOperationDisplayOrderAttribute('1')
}

public AmountMST parmToAmount(AmountMST _toAmount = toAmount)
{
    toAmount = _toAmount;
    return toAmount;
}

The following code illustrates how to override the UI builder to include three columns in the UI.

public class CustInvoiceSettled_TransDateUI_ES extends SrspReportDataContractUIBuilder
{
}

public void build()
{

    FormBuildGroupControl grp;
    // Get the form group used in template form and set it to use 3 columns.
    grp = this.dialog().curFormGroup();
    grp.frameType();
    grp.columns(3);
    // now let the framework draw the controls and take care of everything.
    super();
}

RDP example 4: The UI requires a group check box

The following example provides information about how the BOMPartOf report was converted to use the current report programming model. This report provides an example of a report that requires a group check box in the UI.

- The BOMPartOf report was migrated to the current report programming model.
- An RDP contract was created.
- The UI builder was used together with overrides so that the UI includes a group check box.
- Before the conversion, the helper class did the following:
  - Add the fields to the dialog box.
  - Get the fields from the dialog box.
  - Set the report parameters.
  - Provide the grouping.
The following screenshot illustrates a parameter form that includes the group check box.

To see the full implementation, in the development workspace, click AOT > Classes > BOMPartOfUIBuilder. The following code illustrates how to create a form group control, and how to use the control in the build method, the getFromDialog method, and the getSearchIntervalFromDialog method.

```java
public void build()
{
    FormBuildGroupControl formBuildGroupControl;
    super();
    formBuildGroupControl = this.dialog().curFormGroup();
    formBuildGroupControl.columns(2);
}

public void getFromDialog()
{
    super();
    this.getSearchIntervalFromDialog(this.dialog());
}

protected void getSearchIntervalFromDialog(Dialog _dialog)
{
    BOMPartOfContract contract = this.dataContractObject();
    FormGroupControl dialogSearchIntervalGroup = _dialog.formRun().control(_dialog.formRun().controlId('Date'));
    contract.parmSearchInterval(dialogSearchIntervalGroup.optionValue());
}
```
The following code illustrates the contract for the **BOMPartOf** report.

```csharp
[DataContractAttribute,
    SysOperationContractProcessingAttribute(classstr(BOMPartOfUIBuilder),
    SysOperationDataContractProcessingMode: CreateUIBuilderForRootContractOnly),
    SysOperationGroupAttribute('Date', "@SYS25853", '4'),
    SysOperationGroupAttribute('ViewGroup', "@SYS5252", '5')
]
public class BomPartOfContract
{
...
}
```

**RDP example 5: Condition logic is used to enable UI fields**

The following example provides information about how the **CustAccountStatementInt** report was converted to use the current report programming model. This report provides an example of a report that requires condition logic to determine whether a UI field should be enabled.

- The **CustAccountStatementInt** report was migrated to the current report programming model.
- A UI builder was created to do the following:
  - Allow the report framework to create fields based on the contract.
  - Override the **postBuild** method to disable the field based on a condition.
- The helper class was removed. Before the conversion, the helper class created the control and read from the dialog box.

The following screenshot illustrates the parameter form when the **BankDocumentType** field is disabled and is not displayed.
The following screenshot illustrates the parameter form when the `BankDocumentType` field is enabled and is displayed.
The following code for the UI builder illustrates how to override the `postBuild` method to provide the condition logic to set the visibility of the field.

```java
public void postBuild()
{
    DialogField dialogField;
    super();
    contract = this.dataContractObject();
    contract.parmBankLCExportEnable(BankLCExportFeatureChecker::checkBankLCExportEnabled());

    // after framework has built all fields, enable/disable the dialogfield for
    BankLCBankDocumentType
    dialogField = this.bindInfo().getDialogField(contract,
                                            methodstr(CustAccountStatementIntContract, parmBankLCBankDocumentType));

    if (dialogField && !contract.parmBankLCExportEnable())
    {
        dialogField.visible(false);
    }
}
```

RDP example 6: The report uses a temp table in the caller args

For an example of a report that uses a temp table in the caller args, see the classes that are defined for the `Cheque` report.
RDP example 7: The report uses print management

The following reports provide examples of reports that use print management:

- PurchPackingSlip
- SalesConfirm
- ProjInvoice

RDP example 8: The report uses a custom dialog box and UI event overrides

For an example of a report that uses a custom dialog box and UI event overrides, see the classes that are defined for the BudgetDetails report.

RDP example 9: The report uses a pre-processed RDP class

You must set the user connection on the tables that are used in the report. For examples of reports that set parmUserConnection, see the report data provider classes that are defined in the SalesInvoiceDP and SalesConfirmDP classes.

Sample query-based reports

The following sections provide projects, code, and information about reports that had common scenarios when they were converted to use a query that is defined in the AOT and the current report programming model.

Query example 1: Modify the run-time query based on parameter values, without a custom dialog box

The following example provides information about how the AssetBasis report was converted to use the current report programming model. This report provides an example of a report that modifies the run-time query based on the values of the fromDate and toDate parameters, without using a custom dialog box.

- The AssetBasis report was migrated to the current report programming model.
- Code was moved from the helper class to the controller.
- Simple validation logic on the helper class was also moved to the controller. Most validation was performed automatically.

The following screenshot illustrates the parameter form that includes the from date and to date parameters for the AssetBasis report that ships with Microsoft Dynamics AX.
The following code illustrates how to modify the query contract based on the `fromDate` and `toDate` parameters in the `preRunModifyContract` method of the `AssetBasisController` class.

```java
public class AssetBasisController extends SrsReportRunController {

protected void preRunModifyContract() {

#define.parameterFromDate('FromDate')
#define.parameterToDate('ToDate')
#define.parameterDepreciationBookID('DepreciationBook')

SrsReportRdlDataContract contract = this.parmReportContract().parmRdlContract();

date fromDate = contract.getParameter(#parameterFromDate).getValueTyped();
date toDate = contract.getParameter(#parameterToDate).getValueTyped();
AssetDepreciationBookId depreciationBookId = contract.getParameter(#parameterDepreciationBookID).getValueTyped();

Query query = this.getFirstQuery();

// Modify the query contract based on fromDate & toDate.
SrsReportHelper::addFromAndToDateRangeToQuery(query, fromDate, toDate, tableNum(AssetBasis), fieldNum(AssetBasis, UsedFromDate));

// Modify the query contract based on DepreciationBookId.
SrsReportHelper::addParameterValueRangeToQuery(query, tableNum(AssetBasis), fieldNum(AssetBasis, DepreciationBookId), depreciationBookId);
}
```
Query example 2: Override the query init method

The reporting framework does not invoke the `init` method on a query. You can override the controller and call the `init` method on the `PrePrompt` class or the `PreRunModifyContract` class.
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.

© 2012 Microsoft Corporation. All rights reserved.