**PureConnect®**

**2018 R5**

Generated:

12-November-2018

Content last updated:

08-February-2018

See Change Log for summary of changes.

**GENESYS™**

# Using Active Reports to Create Reports for Interaction Reporter

## Developer's Guide

### Abstract

This document is a reference to author and integrate reports for Interaction Reporter using ActiveReports 6.1.2814.0.

For the latest version of this document, see the PureConnect Documentation Library at: http://help.genesys.com/cic.

# Table of Contents

# Audience

This guide is a reference to create and integrate reports for Interaction Reporter using ComponentOne ActiveReports 6.1.2814.0.

This document assumes that you:
- Have installed ActiveReports on your development system.
- Are familiar with the basic concepts of report creation as illustrated on the ActiveReports help document installed with the product.

To use this guide, you should be familiar with the following .NET objects:
- `System.Data.DataSet`
- `System.Data.DataTable`

You should also be familiar with these programming concepts in C# .NET objects:
- Inheritance
- Implementing interfaces

# Defining the Data

The first step in report creation is defining the raw data. The reports in Interaction Reporter retrieve data from the PureConnect database by using stored procedures. The reports also retrieve some data using IceLib calls combined with database data for readability and display purposes.

For a report you create, you can use any method to retrieve data as long as the final object can bind to an `ActiveReports.DataSource` object. You can combine data from multiple sources in any way that you require and display the data in a report. For example, you can use:

- A `DataTable` within a `DataSet`.
- A `DataTable` by itself.
- An array of `DataRows` returned from a `DataTable.Select()` method.
- Any collection of objects with the object properties bound to the text boxes on the report. This includes a collection that results from a call to a `Linq` expression.

You can refer to ComponentOne documentation for additional object types.

.NET DataSet objects can have multiple `DataTable` objects. For example, a stored procedure could execute multiple `SELECT`s, and the report could utilize all of them. A report could present summarized and detailed data on the same report or create a main report that calls a subreport.

## Create the Stored Procedures for a Report

You can choose whether to use a stored procedure to source the data for a report. A report typically has the following stored procedures:

- Main procedure to contain all data for the report
- Count procedure to return the number of rows that meet the criteria from the `@WhereClause`. This is the procedure executed when the **Show Count** button is pressed in Interaction Reporter.
- Sample procedure to allow exploring a small set of data from the parameter screen.

When you create stored procedures, consider the following:

- In MS SQL Server, wrap the `SELECT` statement in an `EXEC` command. As a best practice, use `sp_executesql` for better performance and improved security.
- Most shipping reports take `@WhereClause` and `@OrderClause` input parameters that the engine passes to the procedure when the report executes.
- You can configure whatever parameters you require for a custom data as long as you pass the parameters when the report executes.

The following example shows SQL Server stored procedures for the Fax Detail report:

```
-------------------------------------------------------------------
--sprpt_FaxDetail
-------------------------------------------------------------------
IF EXISTS
    (   SELECT * FROM sysobjects
        WHERE id = object_id(N'dbo.[sprpt_FaxDetail]')
        and OBJECTPROPERTY(id, N'IsProcedure')



 = 1)
    DROP PROCEDURE dbo.[sprpt_FaxDetail]
GO
CREATE PROCEDURE [dbo].[sprpt_FaxDetail]( @WhereClause nvarchar(4000),
@OrderClause nvarchar(2048))
AS
SET NOCOUNT ON;
exec(
/* Fax detail activity: */
'SELECT
feh.Siteid,
Sendername,
Direction,
successflag,
```

```
RemoteCSId,
RemoteNumber,
EnvelopeId,
FaxId,
Direction,
Speed,
CallIdKey,
ProcessingDatetime,
ProcessingDatetimeGmt,
FaxTimeStamp,
EnvelopeTimeStamp,
PortNumber,
Duration,
Speed,
PageCount,
FailureType,
LastName,
FirstName
FROM FaxEnvelopeHist feh
left outer join Individual i on i.icuserid = feh.SenderName '
+ @WhereClause + ' ' + @OrderClause)
go
-----------------------------------------------------------------------
--sprpt_FaxDetail_samp
-----------------------------------------------------------------------
IF EXISTS
( SELECT * FROM sysobjects
WHERE id = object_id(N'dbo.[sprpt_FaxDetail_samp]')
and OBJECTPROPERTY(id, N'IsProcedure') = 1)
DROP PROCEDURE dbo.[sprpt_FaxDetail_samp]
GO
CREATE PROCEDURE [dbo].[sprpt_FaxDetail_samp](@ColName varchar(50),
@RecCountSamp integer, @Distinct integer)
AS
SET NOCOUNT ON;
DECLARE @DistinctStr varchar(8)
IF @Distinct = 1
    SET @DistinctStr = 'Distinct'
ELSE
    SET @DistinctStr = ''
exec(
/* Fax detail activity: */
'SELECT ' + @DistinctStr + ' top ' + @RecCountSamp + ' ' + @ColName +
' FROM FaxEnvelopeHist')
go
-----------------------------------------------------------------------
--sprpt_FaxDetail_count
-----------------------------------------------------------------------
IF EXISTS
( SELECT * FROM sysobjects
WHERE id = object_id(N'dbo.[sprpt_FaxDetail_count]')
and OBJECTPROPERTY(id, N'IsProcedure') = 1)
DROP PROCEDURE dbo.sprpt_FaxDetail_count
GO
CREATE PROCEDURE [dbo].sprpt_FaxDetail_count( @WhereClause
nvarchar(4000))
AS
SET NOCOUNT ON;
exec('SELECT count(*) from FaxEnvelopeHist ' + @WhereClause)
GO
```

The following example shows Oracle stored procedures for the Fax Detail report:

```
-----------------------------------------------------------------------
--sprpt_FaxDetail
-----------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE sprpt_FaxDetail
(i_WhereClause IN varchar2
,i_OrderClause IN varchar2
,o_prc        OUT sys_refcursor)
```

```
AS
v_str varchar2(4000) := 'SELECT
feh.Siteid,
Sendername,
Direction,
successflag,
RemoteCSId,
RemoteNumber,
EnvelopeId,
FaxId,
Direction,
Speed,
CallIdKey,
ProcessingDatetime,
ProcessingDatetimeGmt,
FaxTimeStamp,
EnvelopeTimeStamp,
PortNumber,
Duration,
Speed,
PageCount,
FailureType,
LastName,
FirstName
FROM FaxEnvelopeHist feh
left outer join Individual i on i.icuserid = feh.SenderName '
|| i_WhereClause || ' ' || i_OrderClause
;
BEGIN
    dbms_output.put_line(v_str);    open


 o_prc for v_str;
    commit;EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error in sprpt_FaxDetail




 procedure.');
        DBMS_OUTPUT.PUT_LINE(sqlerrm);
        RAISE;
END;
/
-----------------------------------------------------------------------
--sprpt_FaxDetail_samp
-----------------------------------------------------------------------
create or replace PROCEDURE sprpt_FaxDetail_samp
(i_ColName  IN varchar2
,i_RecCount IN integer
,i_Distinct IN integer
,o_prc      OUT sys_refcursor
)
AS
v_DistinctStr varchar2(8);
v_str         varchar2(4000);BEGIN
    IF i_Distinct = 1 THEN
        v_DistinctStr := 'Distinct';
    ELSE
        v_DistinctStr := '';
    END IF;
/* Fax detail activity: */
    v_str := 'SELECT * FROM (SELECT ' || v_DistinctStr ||




 ' ' ||
```

```
i_ColName ||
  ' FROM FaxEnvelopeHist) WHERE ROWNUM <= ' || i_RecCount
  ;    dbms_output.put_line(v_str);    open



 o_prc for v_str;
   commit;EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in sprpt_FaxDetail_samp
procedure.');
      DBMS_OUTPUT.PUT_LINE(sqlerrm);
      RAISE;
END;
/
----------------------------------------------------------------------
--sprpt_FaxDetail_count
----------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE sprpt_FaxDetail_count
(i_WhereClause IN varchar2
,O_Count        OUT NUMBER
)
AS
v_str varchar(4000) := 'SELECT count(*) from FaxEnvelopeHist ' ||
i_WhereClause;
BEGIN
   dbms_output.put_line(v_str);
   execute immediate(v_str) into o_count;
EXCEPTION
   WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE('Error in sprpt_FaxDetail_count
procedure.');
   DBMS_OUTPUT.PUT_LINE(sqlerrm);
   RAISE;
END;
/
```
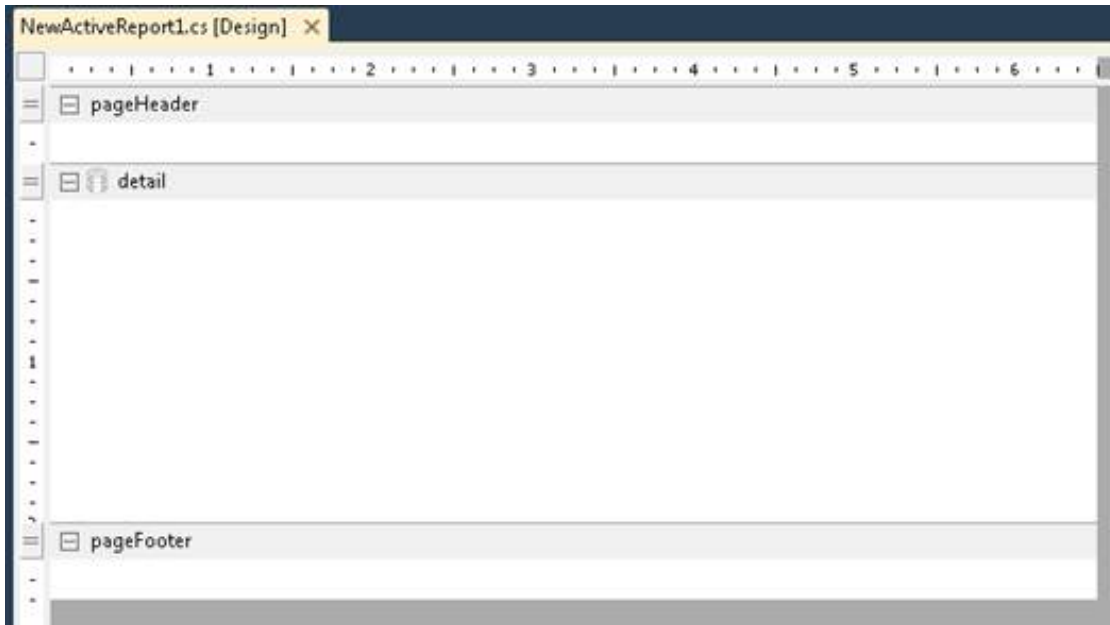
# Creating a report

## Create a C# Class in Visual Studio

1. Open a new instance of Visual Studio.
2. Add a new item.
3. Select ActiveReports 6 (code-based) and name the report.
4. Click **Add** to add the report to your project and open it in design view. By default, your report contains a page header section, a detail section, and a page footer section.

# Conform a Report to Interaction Reporter

Add the following references from the location of your installation of IC Business Manager:

- i3trace_dotnet_reader_interop-w32r-1-2.dll
- i3trace_dotnet_reader_interop-w32r-1-4.dll (If you have it.)
- ININ.Common.dll
- ININ.Reporting.Common.dll
- ININ.Reporting.Historical.Engine.Module.dll
- ININ.Reporting.Historical.Engine.Objects.dll
- ININ.Reporting.Historical.Engine.Reports.dll
- ININ.Reporting.MetaDataAPI.dll

You must also complete the following to conform your report to Interaction Reporter:

- In the report's code, inherit from ReportBase to access standard report page header, footer, styles, and several useful utilities (Refer to "Appendix C: Properties and Methods").
- Add the IReport interface to the inheritance chain. Implement at least IReport's GetData to retrieve your data. This method is executed when the report is executed from the parameters page.
- If you would like to override the GetCount and GetSample methods, you can. These methods are implemented in ReportBase using information from the report metadata you enter in Interaction Administrator.

Because the ReportBase is a master report, you cannot change the report header, page header, report footer, or page footer sections. By definition of a master report in ActiveReports, a custom report is the detail and additional groups you add between these sections. The following properties on ReportBase have controls that you can change:

- PHeader is the DataDynamics.ActiveReports.PageHeader with get and set capability.
- GetReportHeader1 returns the DataDynamics.ActiveReports.ReportHeader section. You can override the title by using this property:

```
GetReportHeader1.Controls["txtReportTitle"].Text

 = "My Custom title";
GetReportHeader1.Controls["txtSubtitle"].Text

 = "My Custom title";
```

The following example is code from the Fax report. In this example, the:

- GetSQLFragment() is the method to return the WHERE clause as a string based on the ParameterValue data stored in the GetReportData instance of ReportData.
- ReportTransaction is a class located in ININ.Reporting.Historical.Engine.Objects that wraps the methods to retrieve data from the database.
- StartQueryTimer() method and StopQueryTimer() method start and stop the timer and log this metric in the ICBusinessManager log file.

# Design the Layout

To design the layout of a report, you can:

- Place the query results in the Detail section.
- Add a group break.
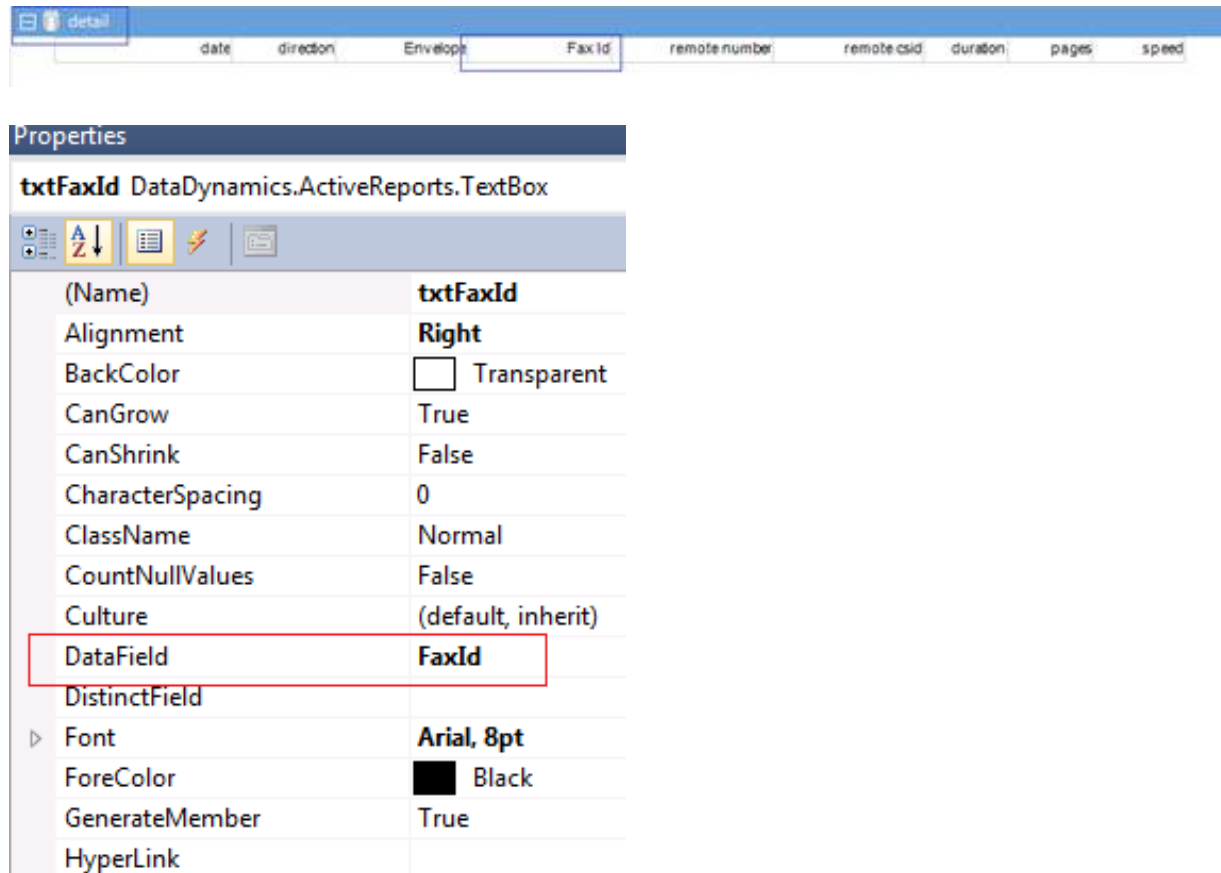- Add a group summary.
- Add a subreport.

## Place the Query Results in the Detail Section

Every column returned by your query is added to the ActiveReport.Fields collection. You can add custom data to this field collection that could serve you later to create complex calculations or just to handle a better layout.

Select the data from your query or from your custom data that you would like to display in the body of the report. The data generates one instance of the Detail section for each record from your query.

Place textboxes in the detail section to map to each Field. Select the **Datafield** property of the textbox. Enter the exact name or alias of the column from your query result that you want to display.

The following illustration shows adding the **Faxid** column as a **Datafield** property.
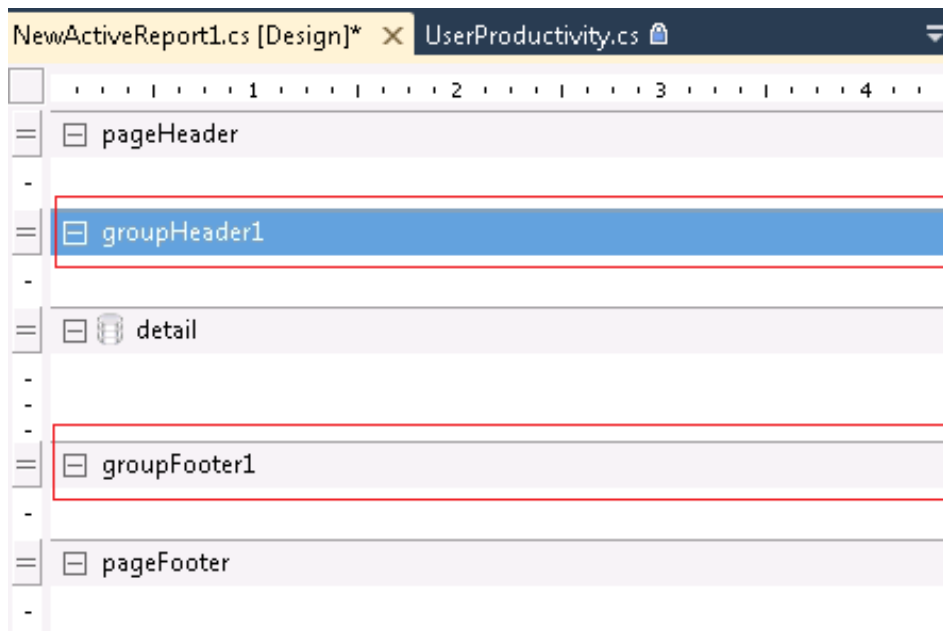


## Add a Group Break to a Report

Add a group break to a report as needed. A group break uses a field from the query.

1. Select any section on the report. Right-click and select **Insert > Group Header/Footer**.

Two new sections appear on the layout that correspond to the header and footer of the new group.



2.  Select the group header, right-click and select **Properties**. Name the group and set the **DataField** property to map the column from your data that drives the group breaks in the report. Set the **RepeatStyle** property to **OnPage** to repeat all elements in this section on each page.
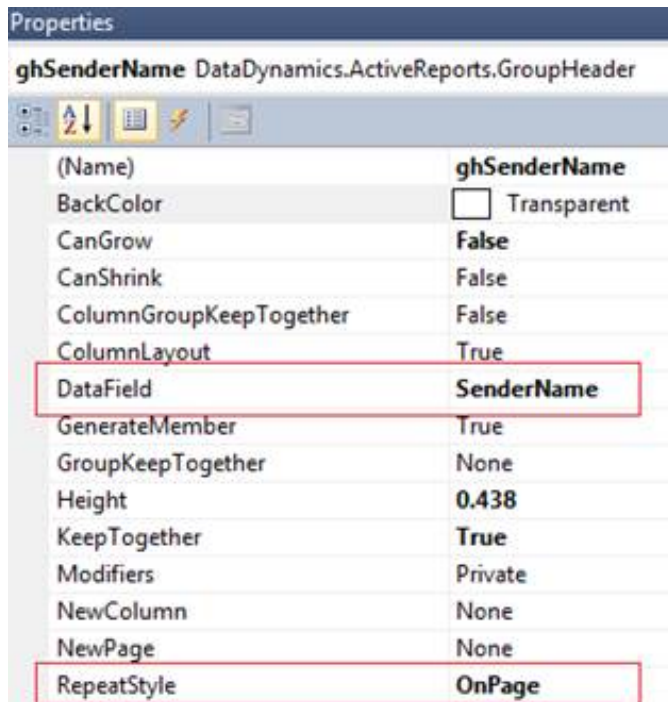


3.  Add elements (for example, labels or textboxes) to the group header. In the FaxDetail report example, the **SenderName** group contains labels that serve as titles for each column in the detail section.

FaxDetail.cs [Design]

reportHeader1

**Title**
*SubTitle*

debug info

pageHeader

gHSiteId

ghSenderName

| Sender: sender | | | | | | | | Site ID: |

| Processing Date | Direction | Envelope | Fax ID | Remote Number | Remote CSId | Duration | Pages | Speed |

ghSenderName2

detail

| date | direction | Envelope | Fax ID | remote number | remote csid | duration | pages | speed |

gfSenderName2

gfSenderName

| | Succeeded | Failed | Total | Duration | Avg Duration |
|---|---|---|---|---|---|
| Total Inbound | is | if | countIn | durationIn | durationIn |
| Total Outbound | os | is | countOut | durationOut | durationIn |
| Total | ts | tf | tc | td | ta |

gfSiteId

pageFooter

| Date | Category | Title footer | reportInfoPages |

reportFooter1

debug info

4. Continue to add as many group breaks as the report requires. For example, create a group break for **SiteId**, **UserId**, and **Date**.

## Add a Group Summary

You can place summarized data in the group header or footer by using the data from the detail section.

1. Set the **DataField** property to the field name on which you base the summary.
2. Set the **SummaryFunc** property to the type of arithmetic aggregation function that applies to the summary.
3. Set the **SummaryGroup** property to the group header name to which this summary relates.
4. Set the **SummaryRunning** property to **Group** if the group footer contains the summary field. Set the property to **None** if the group header contains the summary.
5. Set the **SummaryType** property to **SubTotal** for summaries placed in groups.



## Additional Properties for Report Groups

Use the **KeepTogether** property to indicate whether a section prints in its entirety on the same page. Set the property to:

- **True** to print the section on the same page without any page breaks. If the section is too large for the current page or can fit fully on the next page, the **KeepTogether** property is ignored.
- **False** to print the section across two or more pages.

Use the **GroupKeepTogether** property to indicate whether the group header and footer sections print as a single block on the same page. Set the property to:

- **None** to split the block across pages. The property defaults to **None**.
- **All** to print the block on the same page without any page breaks. If the block does not fit on one page, ActiveReports prints the block across two or more pages. Use the **FirstDetail** property to prevent any widowed group header sections. The group header always prints with at least one detail record.

## Add a Subreport

You can add one or more subreports to the main report. Limit the use of subreports in a repeating section because this consumes memory and can result in an Out of Memory error. To use a subreport in a repeating section, instantiate the subreport in the ReportStart event.

1. From the toolbox, drag a subreport object into the main report and use the name subRptSummary.

| Properties | ▾ 뮤 ✕ |
| --- | --- |
| **subRptSummary** DataDynamics.ActiveReports.SubReport | ▾ |
| (Name) | **subRptSummary** |
| CanGrow | True |
| CanShrink | True |
| GenerateMember | True |
| ▷ Location | **0, 0.375 in** |
| Modifiers | Private |
| ReportName | **subReport1** |
| ▷ Size | **10, 0.438 in** |
| Tag | |
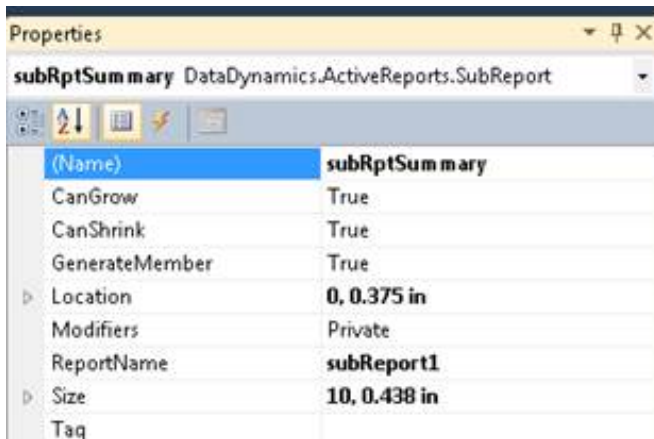
2. Add an ActiveReports class to the main report folder and use the name SubReportSummary.
3. In the main report, declare the constructor.

```
SubReportSummary _SumSubRpt = null;
```

4. In the ReportStart event of the Main report, initialize the subreport.

```
//Initialize subreport
    if ( _SumSubRpt == null)
    {
        _SumSubRpt = new SubReportSummary();
        this.subRptSummary.Report = _SumSubRpt;
    }
```

5. In the Format event of the section where you place the subreport, set the data source for the subreport. Use a table from the GetData method of the main report that corresponds to the data to display on the subreport. You can use a complete table or part of a table based on the current value currently being iterated in the main report.

   The following example uses a complete table for the subreport:

```
string filter = String.Format("InteractionID = '{0}'", ReportDataSet.Tables[ReportCommonConstants.FIRST_TABLE_INDEX].Rows[0]["InteractionID"].ToString());
subDetail.Report.DataSource = ReportDataSet.Tables["Events"].Select(filter, "serverutc");
```

# Configuring a Report to Appear and Run in IC Business Manager

You configure a report in Interaction Administrator so that you can run the custom report from Interaction Reporter in IC Business Manager. You can add the report to an existing category or create a category in Interaction Administrator. After you add the report to a category, you can:

1. Enter general report information.
2. Add data source information.
3. Add stored procedure information, if your report requires a stored procedure.
4. Add and configure parameters.
5. Set access control.

## General and Data Source Information

The following illustration shows the general report information and the data source information for the Fax Summary report.
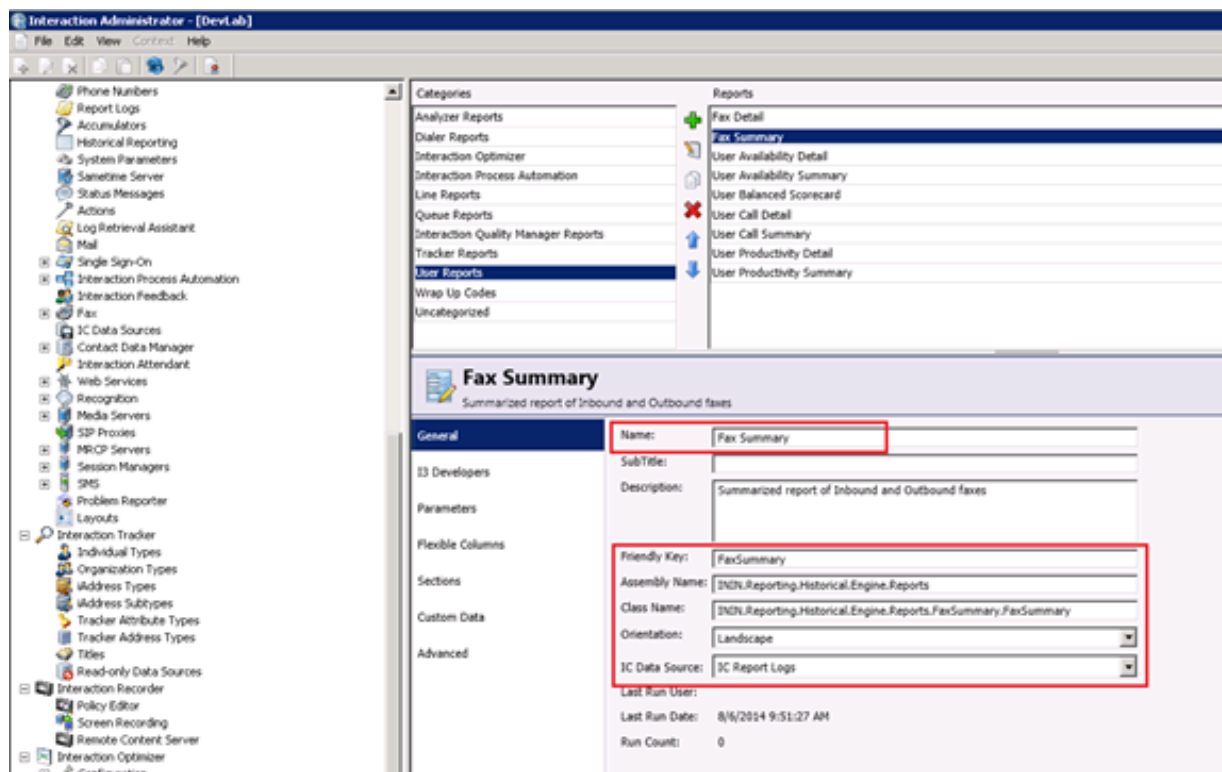
# Stored Procedure Information

The following illustration shows the stored procedure information for the Fax Summary report.



# Parameters

A report can filter data by using parameters (for example, a date range). Interaction Reporter provides parameters that you can add to a report.

You can create a parameter based on classes. For example, to create a Date or a Time parameter, use the following class: `ININ.Reporting.HistoricalEngine.Module.Parameters.DateTimeRange.`

The following illustration shows the Assembly Name and the Class Name for the Date parameter in the Fax Summary report.

The following illustration shows the User Control Assembly Name and the User Control Class Name for the Date parameter in the Fax Summary report.



The following table lists the assembly names and class names for parameters available in Interaction Reporter.

| Name | class_name | user_control_class_name |
|------|-----------|------------------------|
| Group Order | ...ViewModels.AnalyzerKeywordSections | ...Views.DualListSelector |
| Interval | ...ViewModels.AnalyzerScoringInterval | ...Views.ComboBoxParameter |
| Group Order | ...ViewModels.AnalyzerScoringSections | ...Views.DualListSelector |
| ACD Logged In | ...ViewModels.Boolean | ...Views.ComboBoxParameter |

| Group Report by | ...ViewModels.CalibrationGroupBy | ...Views.ComboBoxParameter |
|---|---|---|
| Calibration Only | ...ViewModels.CalibrationOnlyYesNo | ...Views.ComboBoxParameter |
| Call Type | ...ViewModels.CallType | ...Views.ComboBoxParameter |
| Date Time Range | ...ViewModels.DateTimeRange | ...Views.DateTimeRange |
| Period Type | ...ViewModels.DialerInterval | ...Views.ComboBoxParameter |
| Workgroup Queue | ...ViewModels.DistributionQueue | ...Views.TextBoxParameter |
| Call Duration | ...ViewModels.Duration | ...Views.Duration |
| Display Legends | ...ViewModels.FormattingYesNo | ...Views.ComboBoxParameter |
| Interaction Direction | ...ViewModels.InteractionDirection | ...Views.ComboBoxParameter |
| Media Type | ...ViewModels.InteractionType | ...Views.MediaType |
| Process | ...ViewModels.IpaProcess | ...Views.DualListSelector |
| Process Status | ...ViewModels.IpaProcessStatus | ...Views.ComboBoxParameter |
| Line | ...ViewModels.LineComboBox | ...Views.ComboBoxParameter |
| Subtotal By | ...ViewModels.LineGroupBy | ...Views.MultiCheckbox |
| Line Group | ...ViewModels.LineGroupComboBox | ...Views.ComboBoxParameter |
| Media Type | ...ViewModels.MediaType | ...Views.MediaType |
| Top N Results | ...ViewModels.NumericUpDownBase | ...Views.NumericUpDown |
| Date Range | ...ViewModels.OptimizerDateRange | ...Views.OptimizerDateRangeYear |
| Date Time | ...ViewModels.OptimizerDateRangeMonth | ...Views.OptimizerDateRangeMonth |
| exception | ...ViewModels.OptimizerFilterViewModel | ...Views.OptimizerFilterView |
| GroupBy | ...ViewModels.OptimizerGroupByComboBox | ...Views.ComboBoxParameter |
| Interval | ...ViewModels.OptimizerIntervalComboBox | ...Views.ComboBoxParameter |
| Order By | ...ViewModels.OptimizerOrderByViewModel | ...Views.OptimizerOrderByView |
| Target | ...ViewModels.OptimizerTargetViewModel | ...Views.NumericUpDown |
| Time zone | ...ViewModels.OptimizerTimeZoneComboBox | ...Views.ComboBoxParameter |
| Target Answered Service Level Percentage | ...ViewModels.Percentage | ...Views.NumericUpDown |
| Questionnaire Name | ...ViewModels.QuestionnaireName | ...Views.DataDrivenSelection |
| Interval Configuration | ...ViewModels.QueueInterval | ...Views.ComboBoxParameter |
| Group Order | ...ViewModels.QueueSections | ...Views.DualListSelector |
| Media Type | ...ViewModels.RecorderMediaType | ...Views.MediaType |
| Scheduling Unit | ...ViewModels.SchedulingUnitViewModel | ...Views.SchedulingUnitView |
| Service Level Format | ...ViewModels.ServiceLevelFormat | ...Views.ComboBoxParameter |
| Status | ...ViewModels.Status | ...Views.ComboBoxParameter |
| Report Flag | ...ViewModels.StringParameter | ...Views.TextBoxParameter |
| Target Service Level Calculation | ...ViewModels.TargetAnsweredCalc | ...Views.ComboBoxParameter |
| User | ...ViewModels.User | ...Views.AutoCompleteComboBox |

| Last Name | ...ViewModels.UserLastName | ...Views.AutoCompleteComboBox |
|---|---|---|
| Recorded IC User | ...ViewModels.UserList | ...Views.UserList |
| Group Order | ...ViewModels.WrapUpSections | ...Views.DualListSelector |
| Contacted | ...ViewModels.YesNo | ...Views.ComboBoxParameter |

For the parameters listed in the above table:

- Use `ININ.Reporting.Historical.Engine.Module` as the assembly name.
- Replace … in the class_name and user_control_class_name with `ININ.Reporting.Historical.Engine.Module.Parameters`.

# Access Control

You can set the report to use access control. The access control for a report follows the same rules that Interaction Administrator uses for access control in other products.

The following illustration shows that the Fax Summary report requires access control.



The following illustration shows access control for Interaction Reporter reports in Interaction Administrator.

# Running a Report

You can visualize, save filters, and export a report to different formations. Once you configure a report in Interaction Administrator, you can launch the report for Interaction Reporter in IC Business Manager.

The following illustration shows the Fax Summary report running in IC Business Manager.
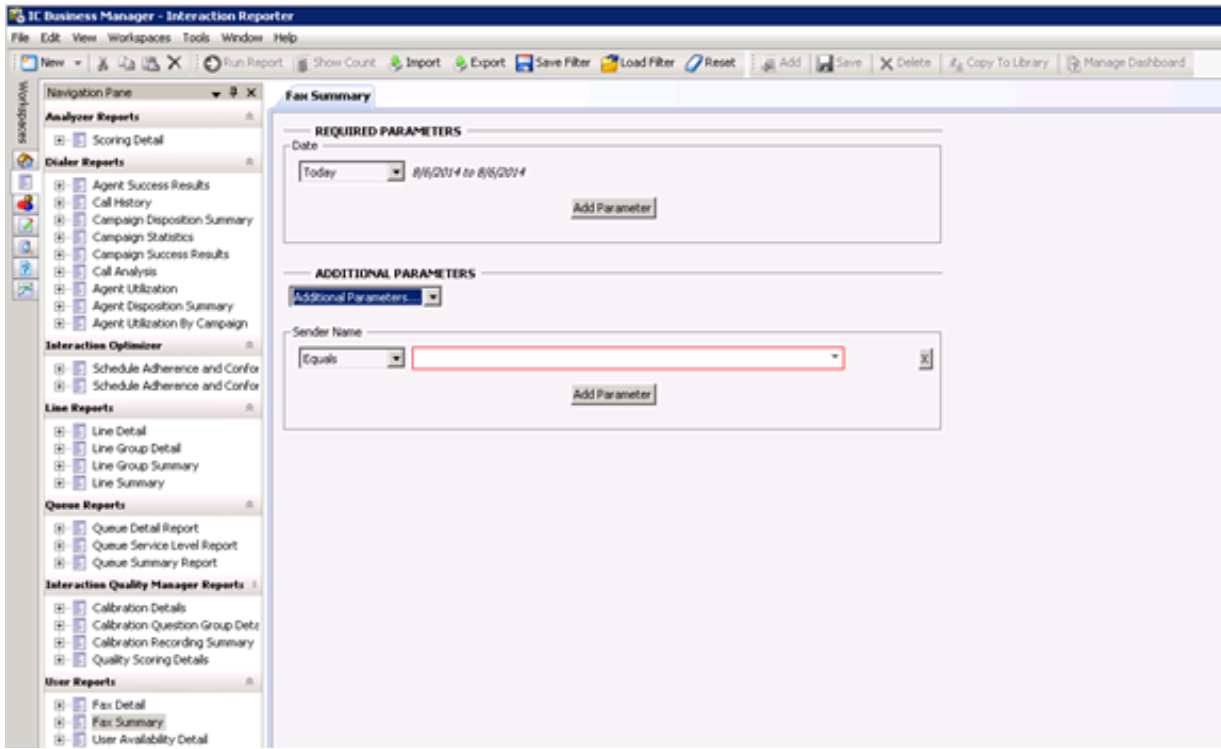
# Appendix A: Order of ActiveReport Events

Events typically happen in the following order:
- ReportStart event
- DataInitialize Event
- FetachData Event
- Section Events

## ReportStart event

Execution begins and validates any change made to the report structure.

## DataInitialize Event

Any access to data source properties could raise this event. Data source is opened.

## FetchData Event

This event occurs just after each row is read from the Data Source but before it goes to the report. If no data is returned from your procedure, the NoData event is raised.

> **Important!**
> The DataInitialize and FetchData events are the only events in which the Fields collection should ever be referenced. On the DataInitialize event, you can add new fields to the collection. On the FetchData event, you can add the custom data to your field.

Depending on the type of fields or properties set on the report, certain pages might not render completely until all data is available. For example, a summary field located on a group header. ActiveReports delays rendering this particular group header until all data is available for the calculation.

## Section Events

- Format Event
- BeforePrint Event

### Format Event

This event occurs after the data is loaded and bound to the controls within the section. Use this event to set or change the properties of a control or load subreport controls with subreports.

When CanGrow or CanShrink properties are set to true, the growing and shrinking of controls contained in this section, and the section itself, takes place in this event. Thus, the height of a control or section cannot be obtained in this event.

### BeforePrint Event

This event occurs just before ActiveReports renders the section to the canvas. Use this event to resize a control or get an accurate height of the section or control. You may resize a control in this event but you cannot resize the section itself.

# Appendix B: Creating Group Breaks Based on Unbound Data

In Interaction Reporter, the page header is inherited from the ReportBase and is not accessible while designing your report. You can add an additional page header by creating a group break based on data that is common to the entire report. You can create this common data by adding a custom field to the Fields collection.

On the DataInitialize event, add a new field called customPageHeader to the Fields collection as follows:

```
private void UserProductivity_DataInitialize(object sender, EventArgs e)
   {
      this.Fields.Add("customPageHeader");
   }
```

On the FetchData event, add the data to this field. The idea is to have the same data for every row so that we can create a break group on this common data and create a level that serves as another page header.

```
private void UserProductivity_FetchData(object sender, FetchEventArgs eArgs)


   {
   //For each row, add data to the CustomPageHeader field
   this.Fields["customPageHeader"].Value = "pageheadergrouping";
   }
```

In this example, in addition to the fields that map each column returned by your query, the Fields collection now includes the CustomPageHeader field. The data for each row stored in the CustomPageHeader field is the pageheadergrouping string. You can now create a group break based on this unbound data.

---

# Appendix C: Properties and Methods

The following table lists some of the properties of ReportBase that you will find usefull. The ReportBase inherits from DataDynamics.ActiveReports.ActiveReport.

| Property Name | Scope | Returns | Description |
|---|---|---|---|
| SessionService | public, get | ININ.Cafe.Interface.Connection. IICSessionService | Provides access to CIC session variable, which is necessary to make other IceLib calls |
| GetReportData | public, get, set | ININ.Reporting.Historical.Engine.Objects.ReportData | The ReportData object with parameters completed from the Parameters page |
| WhereClause | public, get, set | String | String that can be used to hold the result of ReportData.GetSQLFragment() |
| ReportDataSet | public, get, set | System.Data.DataSet | Used to house the results of whatever method is used to get data for the report. |
| GetReportRecordCount | public, get | int | Returns the count of the records in the primary table in the ActiveReport's DataSource property |
| HasRecords | public, get | bool | Returns true or false depending if the results from GetReportRecordCount > 0 |
| LinkedReport | protected | ININ.Reporting.Historical.Engine.Objects.ReportData | The ReportData object for a report linked to the main report via a hypertext link. |
| GetPageFooter | public, get | DataDynamics.ActiveReports.Section | Access to the PageFooter, pageFooter, and the controls within it |
| GetPageHeader | public, get | DataDynamics.ActiveReports.Section | Access to the PageHeader, pageHeader, and the controls within it |
| GetReportHeader1 | public, get | DataDynamics.ActiveReports.Section | Access to the ReportHeader, reportHeader1 and the controls within it |
| ElapsedQueryTime | public, get | System.TimeSpan | Holds the time the query took to execute |

The following table lists some of the methods of ReportBase that you will find usefull. You can override almost all of these methods.

| Name | Scope | Returns | Parameters | Description |
|---|---|---|---|---|
| UpdateProgress | protected | void | None or increment | Updates the progress bar as the report runs. Should be called from the Detail section, typically |
| StartQueryTimer | protected | void | None | Starts the query timer |
| StopQueryTimer | protected | void | None | Stops the query timer |
| FixHeights | protected | void | Array of DataTynamics.ActiveReports.ARControl | Iterates through the controls passed and adjusts each one's height to the tallest control in the collection. |

The following table contains some of the utilities that you will find useful in ININ.Reporting.Common.ReportCommonUtils. These utilities are all `public static` so they can be called without an instantiated object reference.

| Name | Parameters | Returns | Description |
|---|---|---|---|
| ToString | object | String | A save conversion of any object to a string; accounts for null values |
| FixSQLString | String | String | Corrects illegal strings found in SQL statements before processing them |
| IsNumeric | Object | bool | Determines if a value can be safely converted to a number; accounts for null values |
| SafeConvertToInt32 | Object | Int32 | Safely converts any object to a Int32 value; accounts for null values |
| SafeConvertToInt64 | Object | Int32 | Safely converts any object to a Int64 value; accounts for null values |
| SafeConvertToDouble | Object | Double | Safely converts any object to a Double value; accounts for null values |
| SafeConvertToDateTime | Object | DateTime | Safely converts any object to a DateTime value; accounts for null values |
| ConvertFromGMT | Object | DateTime | Checks for a valid date then returns the date converted from GMT to the local client machine's time |

The following table contains some of the utilities that you will find useful in ININ.Reporting.Historical.Engine.Objects.ReportEngineUtils. These utilities are all `public static` so they can be called without an instantiated object reference.

| Name | Parameters | Returns | Description |
| --- | --- | --- | --- |
| GetParameterValuesByFriendlyKey | ReportData, string | IEnumerable<ParameterValueDataBase> | Gets the parameter entered by the user on the Parameters screen based on the FriendlyKey of the parameter |
| MillisecondsToHHMMSS | Object, bool (include milliseconds or not) | String | Takes the total millisecond and returns a string in the format: HH:MM:SS |
| GetSubtraction | Object (first addend), object (second addend) | Double | Safely subtracts values; null values accounted for |
| GetDivision | Object (numerator), object (denominator) | Double | Safely performs a division; null values, invalid strings and division-by-zero accounted for |
| NegativeDisplay | Object | String | Safely formats a number to display as a negative number; null values and invalid strings accounted for |

# Change Log

| Change Log Date | Changed... |
|---|---|
| 01-August-2014 | Initial version. |
| 01-January-2015 | Corrected title from Interactive Reporter to Interaction Reporter. Updated copyright and trademark information. |
| 14-July-2015 | Updated cover page and screen shots for rebranding. |
| 23-January-2018 | Conversion to HTML. |
| 08-February-2018 | Rebranding terminology. |