Blackbaud CRM Custom Reports Guide

09/19/2012 Blackbaud CRM 2.94 Blackbaud CRM Custom Reports US

BlackbaudCRMCustomReports-2012

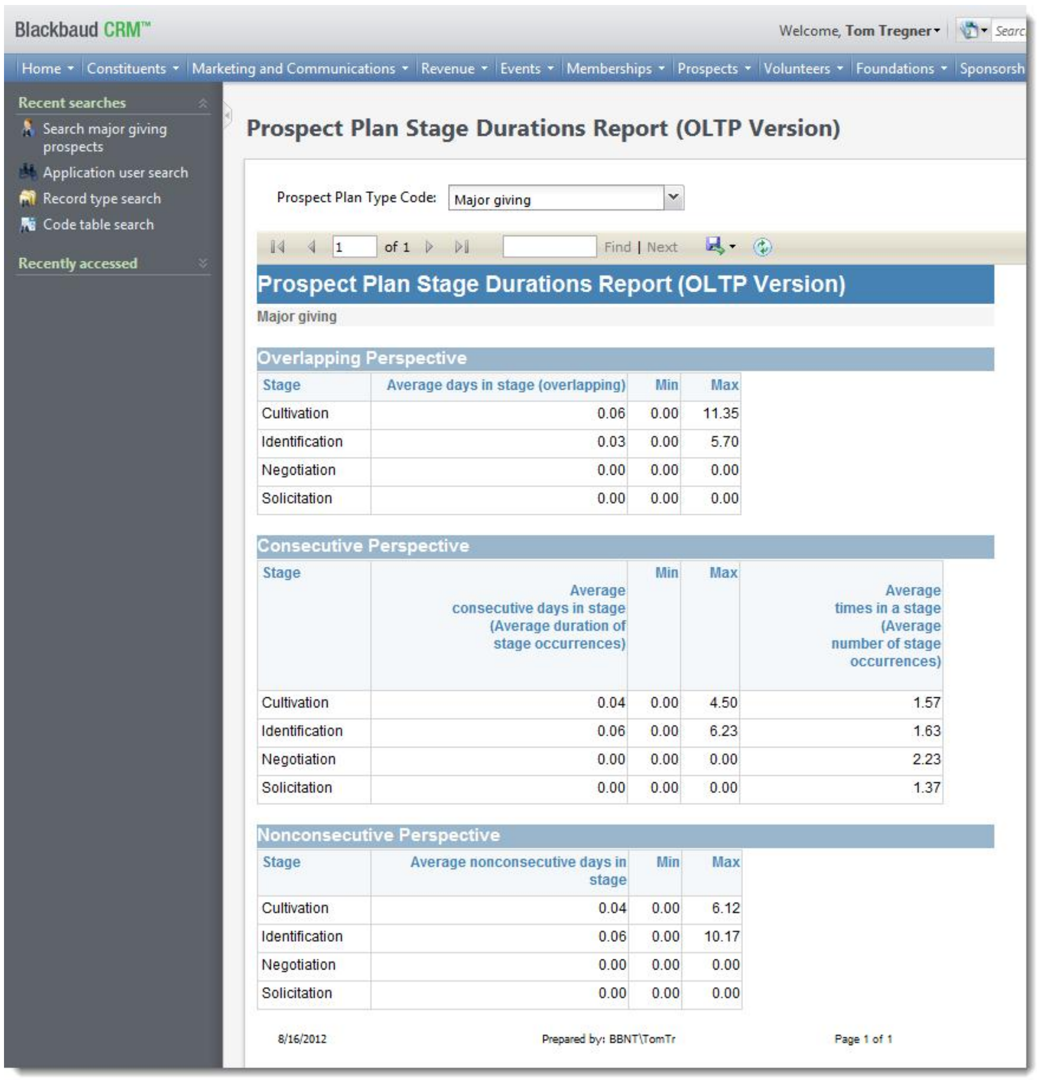
Contents

Creating Blackbaud CRM Reports

This is a document is about creating reports for ***Blackbaud CRM***. In this guide we explore the creation of a report about time periods for stages in a prospect plan. Along the way, we declare a design for the report, explore the application for features related to the report, and look at the transactional database and the data warehouse database data structures related to the report.

We will build a report based on the design which finds data in the transactional database. Then we will create reports which gather data from the data warehouse database. In the course of reporting from the data warehouse, we will explore extending the data warehouse to create a better structure for the report data.

This guide also includes an appendix of related, general topics: Appendix on page 132



Prospect Plan Stage Durations Report (OLTP Version)

Prospect Plan Type Code: Major giving

Prospect Plan Stage Durations Report (OLTP Version)

Major giving

Overlapping Perspective

Stage	Average days in stage (overlapping)	Min	Max
Cultivation	0.06	0.00	11.35
Identification	0.03	0.00	5.70
Negotiation	0.00	0.00	0.00
Solicitation	0.00	0.00	0.00

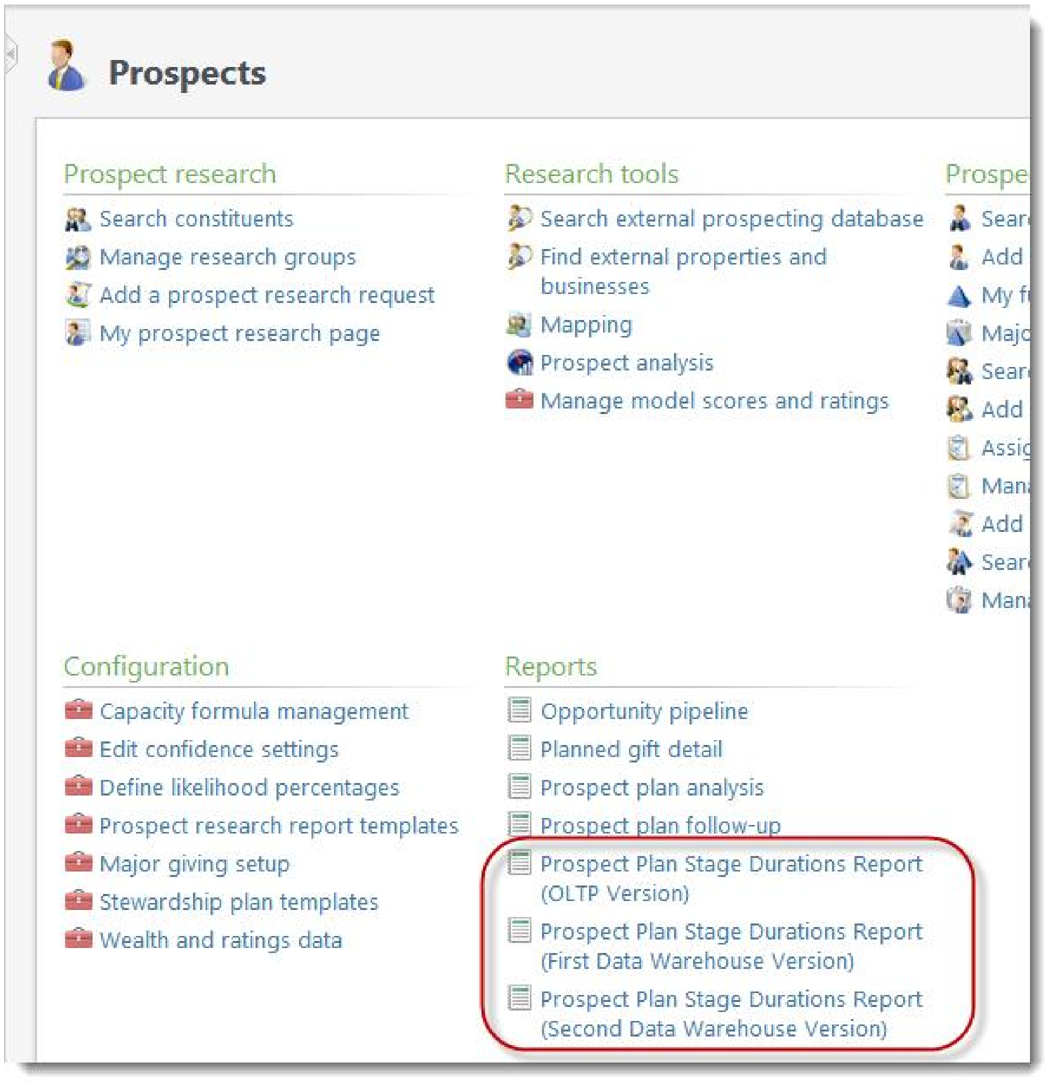
Consecutive Perspective

Stage	Average consecutive days in stage (Average duration of stage occurrences)	Min	Max	Average times in a stage (Average number of stage occurrences)
Cultivation	0.04	0.00	4.50	1.57
Identification	0.06	0.00	6.23	1.63
Negotiation	0.00	0.00	0.00	2.23
Solicitation	0.00	0.00	0.00	1.37

Nonconsecutive Perspective

Stage	Average nonconsecutive days in stage	Min	Max
Cultivation	0.04	0.00	6.12
Identification	0.06	0.00	10.17
Negotiation	0.00	0.00	0.00
Solicitation	0.00	0.00	0.00

8/16/2012 Prepared by: BBNT\TomTr Page 1 of 1



Prospects

Prospect research
- Search constituents
- Manage research groups
- Add a prospect research request
- My prospect research page

Research tools
- Search external prospecting database
- Find external properties and businesses
- Mapping
- Prospect analysis
- Manage model scores and ratings

Prospe
- Sear
- Add
- My f
- Majo
- Sear
- Add
- Assig
- Mana
- Add
- Sear
- Mana

Configuration
- Capacity formula management
- Edit confidence settings
- Define likelihood percentages
- Prospect research report templates
- Major giving setup
- Stewardship plan templates
- Wealth and ratings data

Reports
- Opportunity pipeline
- Planned gift detail
- Prospect plan analysis
- Prospect plan follow-up
- Prospect Plan Stage Durations Report (OLTP Version)
- Prospect Plan Stage Durations Report (First Data Warehouse Version)
- Prospect Plan Stage Durations Report (Second Data Warehouse Version)

Code Samples for the Report

The code samples for this document are located at:

https://www.blackbaud.com/files/support/infinitydevcasestudies/ProspectPlanStageDurationsReports.zip

Projects

The code samples include a Blackbaud Infinity catalog project. The project includes the specs necessary to load the report into the application. Reports are primarily defined by RDL files and these are included in the project as resources. But there are also specs which define the relationship of the Blackbaud Infinity feature to the RDL (Report Specs). There are also specs to expose the report in the Blackbaud Infinity application UI (Page Specs and Task Specs with the Report Spec). Finally there are specs to assist with loading the features into the Blackbaud Infinity application (Package Specs).

Because ***Visual Studio 2010*** does not provide much functionality for editing RDL files, the RDL files for this sample were edited in another flavor of ***Visual Studio***, Business Intelligence Development Studio. Report Builder 2 could also have been used. Those files were copied to the catalog project.

The UI for the report's parameter is implemented with a Blackbaud Infinity UI Model. This requires a Blackbaud Infinity UI model project. That project is contained in the same solution as the catalog project.

The solution also contains a project for Blackbaud Data Warehouse revisions extensions. Because ***Visual Studio 2010*** does not provide much functionality for editing SSIS packages, the files for the ETL extensions to the data warehouse were edited in Business Intelligence Development Studio. Rather than copy those to the ***Visual Studio 2010*** solution, those files were maintained in a separate project.

Files

There are some environment-specific post-build commands in the projects. To use the samples, you will have to modify those for your environment. Once built, these files must be copied to application folders:

Custom.AppFx.PlanStageDurations.Catalog.dll

from the catalog project build

builds to `Custom.AppFx.PlanStageDurations.Catalog\Custom.AppFx.PlanStageDurations.Catalog\bin\Debug` copy to Blackbaud\bbappfx\vroot\bin

Custom.AppFx.PlanStageDurations.UIModel.dll

from the UI model project build

builds to Cus-
tom.AppFx.Pl-
anStageDurations.Catalog\Custom.AppFx.PlanStageDurations.UIModel\obj\Debug

copy to Blackbaud\bbappfx\vroot\bin

ProspectPlanStageDurationsReportOLTPVersion.html

ProspectPlanStageDurationsReportDataWarehouseVersion1.html

ProspectPlanStageDurationsReportDataWarehouseVersion2.html

from the UI model project

located in

Cus-
tom.AppFx.Pl-
anStageDurations.Catalog\Custom.AppFx.PlanStageDurations.UIModel\htmlforms

copy to Blackbaud\bbappfx\vroot\browser\htmlforms

Revisions.dll

from the data warehouse revisions project build

builds to

Cus-
tom.AppFx.Pl-
anStageDurations.Catalog\Custom.AppFx.PlanStageDurations.Revisions\Revisions\obj\De

copy to Blackbaud\bbappfx\MSBuild\Datamarts\BBDW\Extend\Revisions

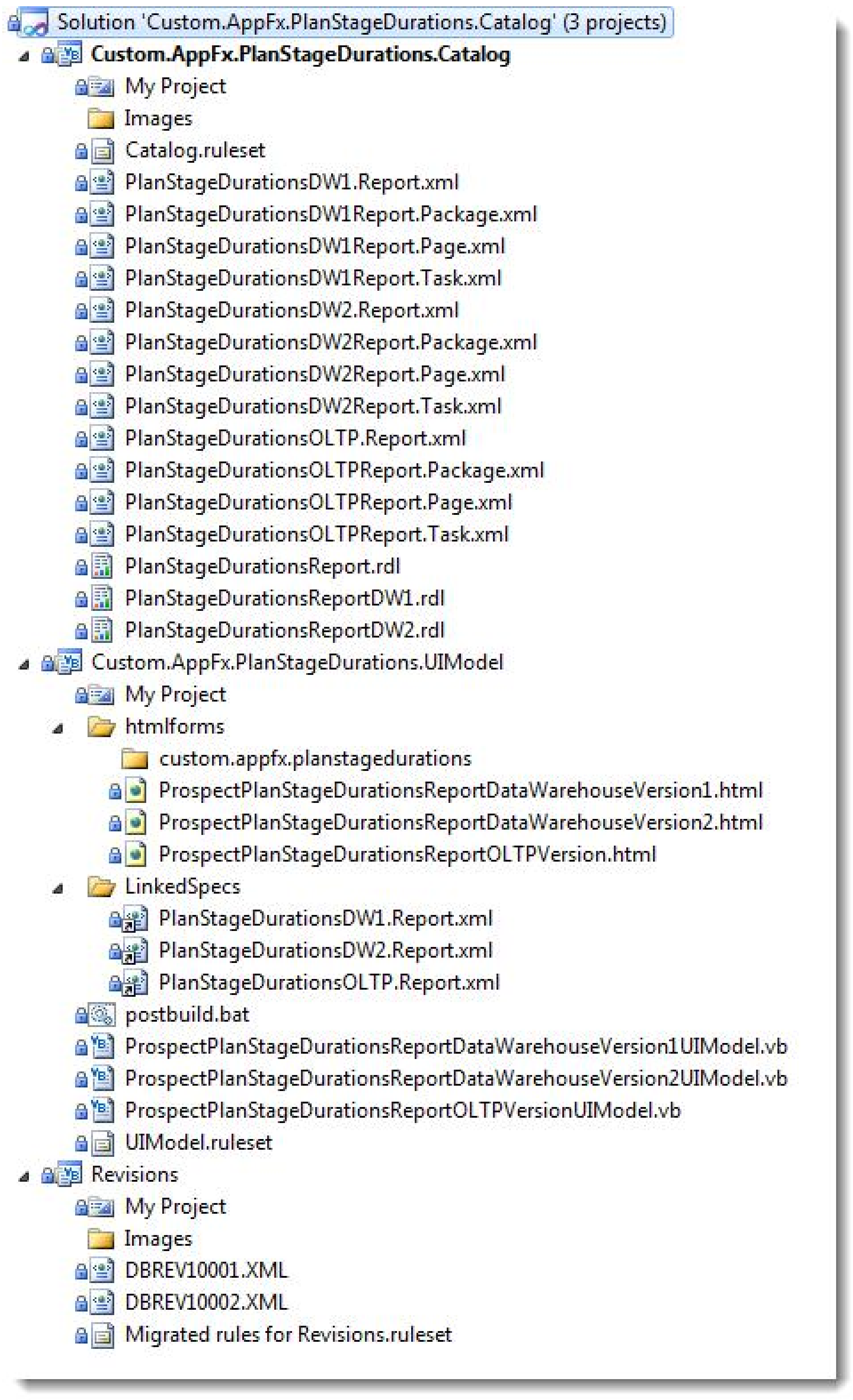
BBDW_FACT_INTERACTIONACTUALTIMES_EXT.dtsx

BBDW_FACT_PROSPECTPLANSTAGE_EXT.dtsx

BBDW_FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT.dtsx

from the Analysis Services project

copy to Blackbaud\bbappfx\MSBuild\Datamarts\BBDW\Extend\SSIS



```
Solution 'Custom.AppFx.PlanStageDurations.Catalog' (3 projects)
  Custom.AppFx.PlanStageDurations.Catalog
    My Project
    Images
    Catalog.ruleset
    PlanStageDurationsDW1.Report.xml
    PlanStageDurationsDW1Report.Package.xml
    PlanStageDurationsDW1Report.Page.xml
    PlanStageDurationsDW1Report.Task.xml
    PlanStageDurationsDW2.Report.xml
    PlanStageDurationsDW2Report.Package.xml
    PlanStageDurationsDW2Report.Page.xml
    PlanStageDurationsDW2Report.Task.xml
    PlanStageDurationsOLTP.Report.xml
    PlanStageDurationsOLTPReport.Package.xml
    PlanStageDurationsOLTPReport.Page.xml
    PlanStageDurationsOLTPReport.Task.xml
    PlanStageDurationsReport.rdl
    PlanStageDurationsReportDW1.rdl
    PlanStageDurationsReportDW2.rdl
  Custom.AppFx.PlanStageDurations.UIModel
    My Project
    htmlforms
      custom.appfx.planstagedurations
      ProspectPlanStageDurationsReportDataWarehouseVersion1.html
      ProspectPlanStageDurationsReportDataWarehouseVersion2.html
      ProspectPlanStageDurationsReportOLTPVersion.html
    LinkedSpecs
      PlanStageDurationsDW1.Report.xml
      PlanStageDurationsDW2.Report.xml
      PlanStageDurationsOLTP.Report.xml
    postbuild.bat
    ProspectPlanStageDurationsReportDataWarehouseVersion1UIModel.vb
    ProspectPlanStageDurationsReportDataWarehouseVersion2UIModel.vb
    ProspectPlanStageDurationsReportOLTPVersionUIModel.vb
    UIModel.ruleset
  Revisions
    My Project
    Images
    DBREV10001.XML
    DBREV10002.XML
    Migrated rules for Revisions.ruleset
```

Deployment

Two versions of the report use the data warehouse and extensions to the warehouse. After the `Revisions.dll` and SSIS packages are copied to the application folders, the data warehouse must be redeployed and the ETL must be reset and refreshed.

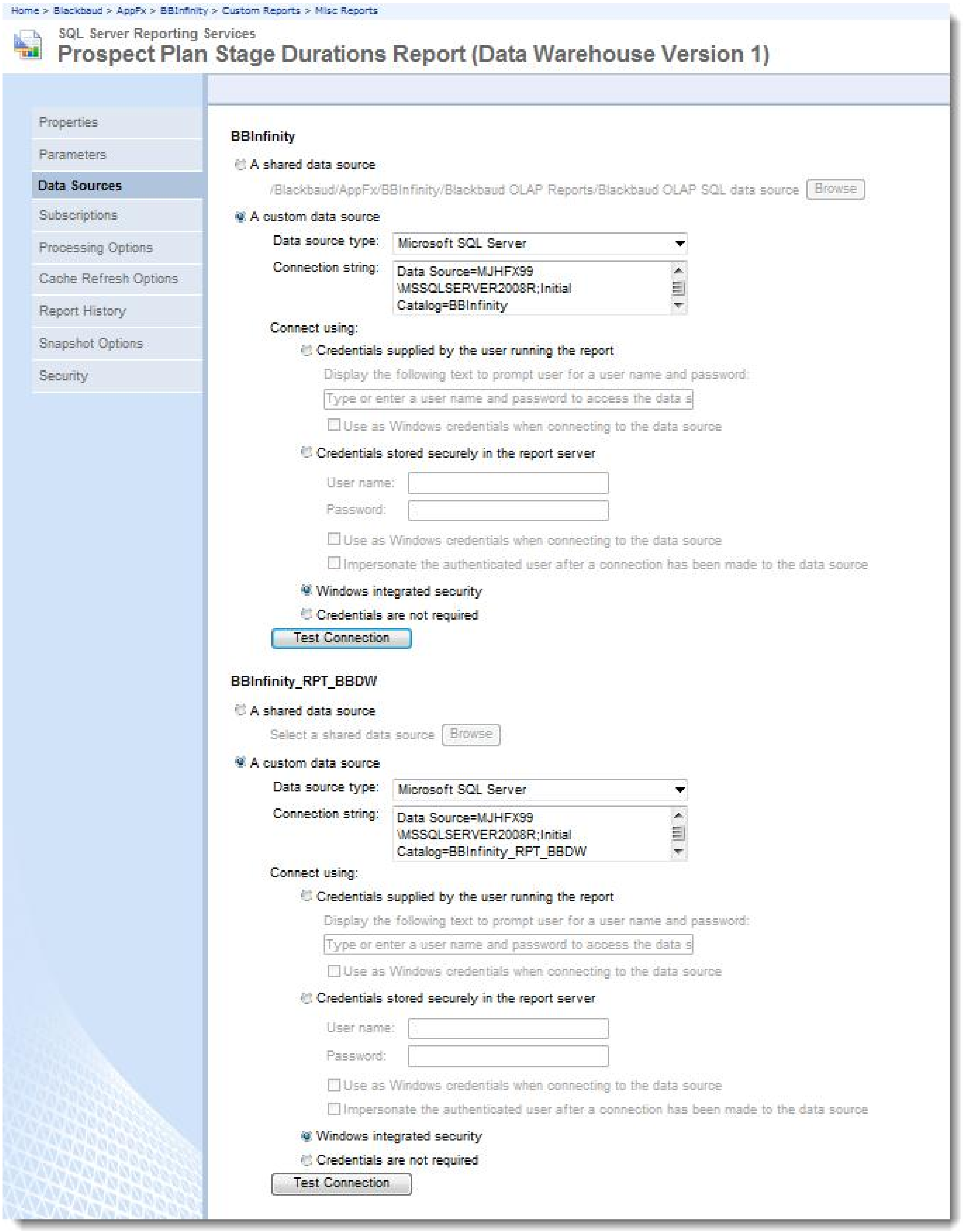
Deploy Blackbaud Data Warehouse

Note: There are no revisions in the sample to support custom security. So you will have to ensure the report user has permissions in the data warehouse database. This includes permissions to execute the stored procedures used by the reports.

All three versions include specs to expose the reports in a Blackbaud Infinity application. There are three packages in the catalog project DLL, one for each version. Each package must be loaded.

Package Specs

Warning: The loading mechanism for Report Specs overrides the data sources in the RDL when it loads the RDL. The data warehouse versions use two data sets. The extra data set uses the OLTP database as a data source. The reason for this is to avoid an extra extension to the warehouse just to support code table names used by the report parameter. So once you load the report, you will have to reconfigure that data source for the those reports in Reporting Services. You could instead remove the data set and populate the acceptable values for the parameter through the UI model only. But then there would be no drop-down for those values when the report is accessed through Reporting Services.



Home > Blackbaud > AppFx > BBInfinity > Custom Reports > Misc Reports

SQL Server Reporting Services

Prospect Plan Stage Durations Report (Data Warehouse Version 1)

- Properties
- Parameters
- **Data Sources**
- Subscriptions
- Processing Options
- Cache Refresh Options
- Report History
- Snapshot Options
- Security

BBInfinity

A shared data source

/Blackbaud/AppFx/BBInfinity/Blackbaud OLAP Reports/Blackbaud OLAP SQL data source Browse

● A custom data source

Data source type: Microsoft SQL Server

Connection string: Data Source=MJHFX99 \MSSQLSERVER2008R;Initial Catalog=BBInfinity

Connect using:

Credentials supplied by the user running the report

Display the following text to prompt user for a user name and password:

Type or enter a user name and password to access the data s

☐ Use as Windows credentials when connecting to the data source

Credentials stored securely in the report server

User name:

Password:

☐ Use as Windows credentials when connecting to the data source

☐ Impersonate the authenticated user after a connection has been made to the data source

● Windows integrated security

Credentials are not required

Test Connection

BBInfinity_RPT_BBDW

A shared data source

Select a shared data source Browse

● A custom data source

Data source type: Microsoft SQL Server

Connection string: Data Source=MJHFX99 \MSSQLSERVER2008R;Initial Catalog=BBInfinity_RPT_BBDW

Connect using:

Credentials supplied by the user running the report

Display the following text to prompt user for a user name and password:

Type or enter a user name and password to access the data s

☐ Use as Windows credentials when connecting to the data source

Credentials stored securely in the report server

User name:

Password:

☐ Use as Windows credentials when connecting to the data source

☐ Impersonate the authenticated user after a connection has been made to the data source

● Windows integrated security

Credentials are not required

Test Connection

Prospect Plan Status Durations Report

This section is the design which will be implemented in the rest of the document. If you want to skip the design details, you can skip this section and refer back to it as you read the remaining sections.

Goal

Create a report which shows average time in a stage of a prospect plan, the maximum time in stage, and the minimum time in stage.

Major Obstacle

The times associated with prospect plan stages are the start and end datetimes for prospect plan steps, which are interactions. There is no data structure in the transactional database which represents the duration of a stage in a prospect plan for a prospect. There are different ways to interpret these times to determine duration of a stage.

Design

There are different types of prospect plans. Major giving is one type. There should be a filter on the report for plan type. The goal of this design is fulfilled by selecting Major giving from the filter. But reports for each plan type are possible if a different plan type is selected from the filter.

During a prospect plan, prospects can move back and forth between stages. For example in the first step of a plan, a prospect may be in the Identification stage. In the second step, the prospect may be in the Cultivation stage. And in the third step, the prospect may return to the Identification stage. So the prospect may spend non-consecutive time in a stage.

A step occurs on a single day. It is a type of interaction. So the time in a stage can be determined from the time of the first step in a stage and the time of the first step in the next stage. But because the prospect plan can have nonconsecutive occurrences of a stage, there are different ways to consider the duration of a stage.

The report will show a nonconsecutive, consecutive, and overlapping perspective. These will be explained in more detail.

Note: Overlapping steps poses another question. If a next stage's first step starts with a time that overlaps with a step associated with a previous stage, should the end time of the step associated with the previous stage be considered or should the start time of the next step be considered? Furthermore, this can happen for more than two interactions (steps). But this will only occur in the course of a day since you can only enter one date and not a start and end date for a step.

The report will not include information for steps that are not completed.

Note: The report could also exclude prospect plans which are active.

The report should be filterable by constituency of prospect, by prospect plan type, or both.

Note: The only parameter implemented in this document and the sample is prospect plan type.

Issue 1 - Nonconsecutive Perspective

The nonconsecutive perspective primarily communicates the average days in a stage. But it does not communicate whether a prospect plan left a stage and returned to it. Also, the nonconsecutive perspective does not communicate how many times a prospect plan entered a stage. However, the calculation does account for leaving and returning. For a nonconsecutive perspective, the report will show average days in stage where the calculation is:

End time for a stage occurrence = IF there is a subsequent step in the prospect plan THEN Start time for first step after the last step in a stage occurrence ELSE IF the prospect plan is completed, End time for the last step in the plan

Keep in mind that a stage occurrence with no subsequent step will not be considered if the prospect plan is not complete. So the condition that the prospect plan be completed is always true for the preceding calculation.

Total days in a stage occurrence = End time for a stage occurrence - Start time for the first step in the stage occurrence

Total days in a stage in a prospect plan = Total of each (Total days in a stage occurrence)

Average days in stage (nonconsecutive) = Total days in a stage in a prospect plan / Number of prospect plans

In a prospect plan with the steps that follow, the nonconsecutive times in each stage are calculated:

1: Identification, 06/05/2012 6:30:00 PM - 06/05/2012 5:00:00 PM

2: Identification, 06/06/2012 7:29:00 PM - 06/06/2012 6:15:00 PM

3: Cultivation, 06/09/2012 4:58:00 PM - 06/09/2012 3:30:00 PM

4: Identification, 06/11/2012 9:51:00 AM - 06/11/2012 6:30:00 AM

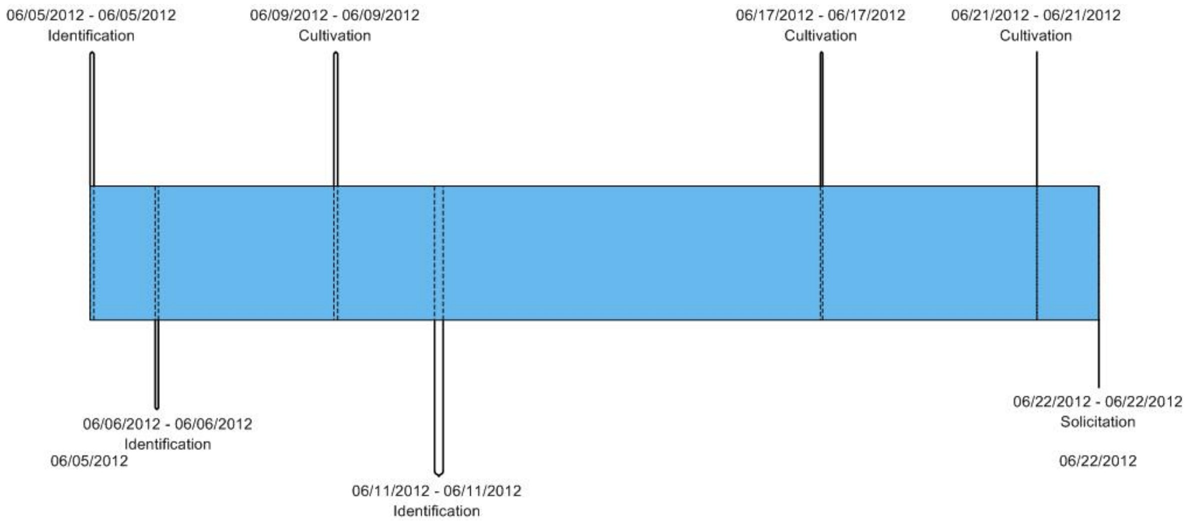
5: Cultivation, 06/17/2012 12:58:00 PM - 06/17/2012 12:05:00 PM

6: Cultivation, 06/21/2012 12:00:00 AM - 06/21/2012 12:00:00 AM

7: Solicitation, 06/22/2012 12:00:00 AM - 06/22/2012 12:00:00 AM

8: Solicitation, no actual dates yet

...



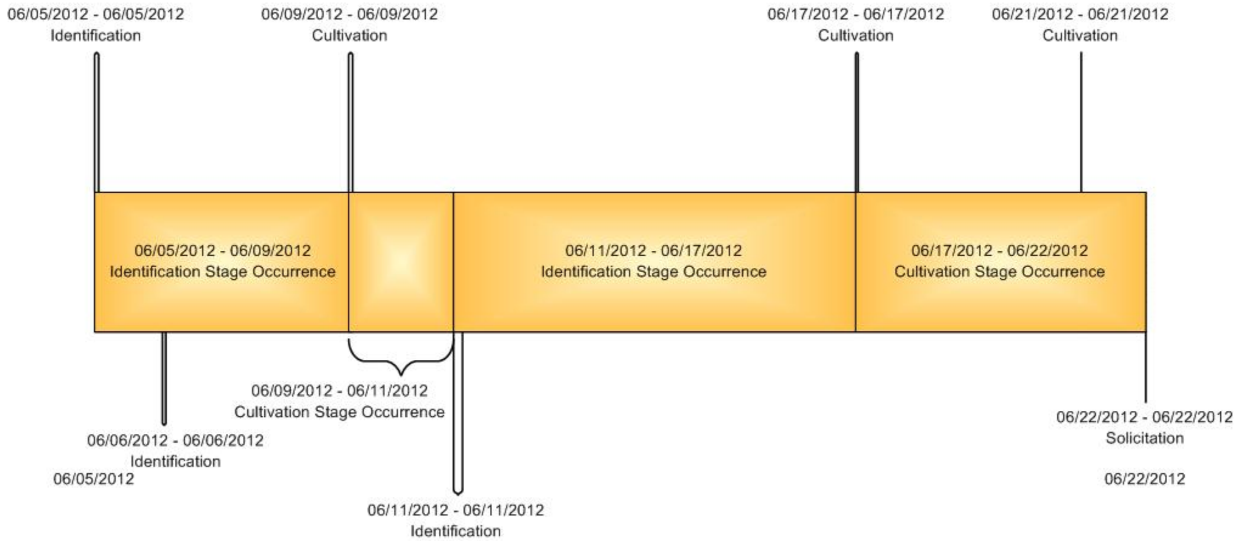
- Identification days = 3.9375 days + 6.232638889 days = 10.17013889 days

  The first Identification step started at 06/05/2012 5:000:00 PM and the next stage (Cultivation) started at 06/09/2012 3:30:00 PM. The difference is 3.9375 days. The plan returned to the Identification stage at 06/11/2012 6:30:00 AM and returned to Cultivation at 06/17/2012 12:05:00 PM. The difference is 6.232638889 days.

- Cultivation days = 1.625 + 4.496527778 = 6.121527778

- Solicitation days is considered to be indeterminate because there is an open Solicitation step

  ...



Then those times for each prospect plan are totaled and divided by the number of prospect plans. The minimum and maximum are also based on the nonconsecutive times.

Stage	Average nonconsecutive days in stage	Min	Max
Identification	xx.xx	xx.xx	xx.xx
Cultivation	xx.xx	xx.xx	xx.xx
Solicitation	xx.xx	xx.xx	xx.xx
Negotiation	xx.xx	xx.xx	xx.xx

Issue 2 - Consecutive Perspective

The consecutive perspective primarily communicates the average consecutive days in a stage. This is the average duration of stage occurrences for a stage. The perspective also communicates the average number of times prospect plans enter a stage. For a consecutive perspective, the report will show average days in a stage where the calculation is:

Total consecutive days in a stage in a prospect plan = Total days in a stage occurrence

Average consecutive days in stage = Total of (Total consecutive days in a stage in a prospect plan) / Total for all prospect plans of the total number times in a stage

The report will also show average times in a stage where the calculation is

Average times in a stage = Total number of times in a stage / Number of prospect plans

- Identification(a) = 3.9375 days
- Cultivation(a) = 1.625 days
- Identification(b) = 6.232638889 days
- Cultivation(b) = 4.496527778 days
- Total times in Identification for this plan = 2
- Total times in Cultivation for this plan = 2
- Solicitation days is considered to be indeterminate because there is an open Solicitation step

...

Stage	Average consecutive days in stage (Average duration of stage occurrences)	Min	Max	Average times in a stage (Average number of stage occurrences)
Identification	xx.xx	xx.xx	xx.xx	x.x
Cultivation	xx.xx	xx.xx	xx.xx	x.x
Solicitation	xx.xx	xx.xx	xx.xx	x.x
Negotiation	xx.xx	xx.xx	xx.xx	x.x

Issue 3 - Overlapping Perspective

The overlapping perspective includes the time spent in another stage if there are nonconsecutive occurrences of stages. It does not express the average number of times a prospect plan spends in a stage. For an overlapping perspective, the report will show average days in a stage where the calculation is:

Total days in stage = End datetime for the last step in stage in a prospect plan - Start time for first step in stage in a prospect plan

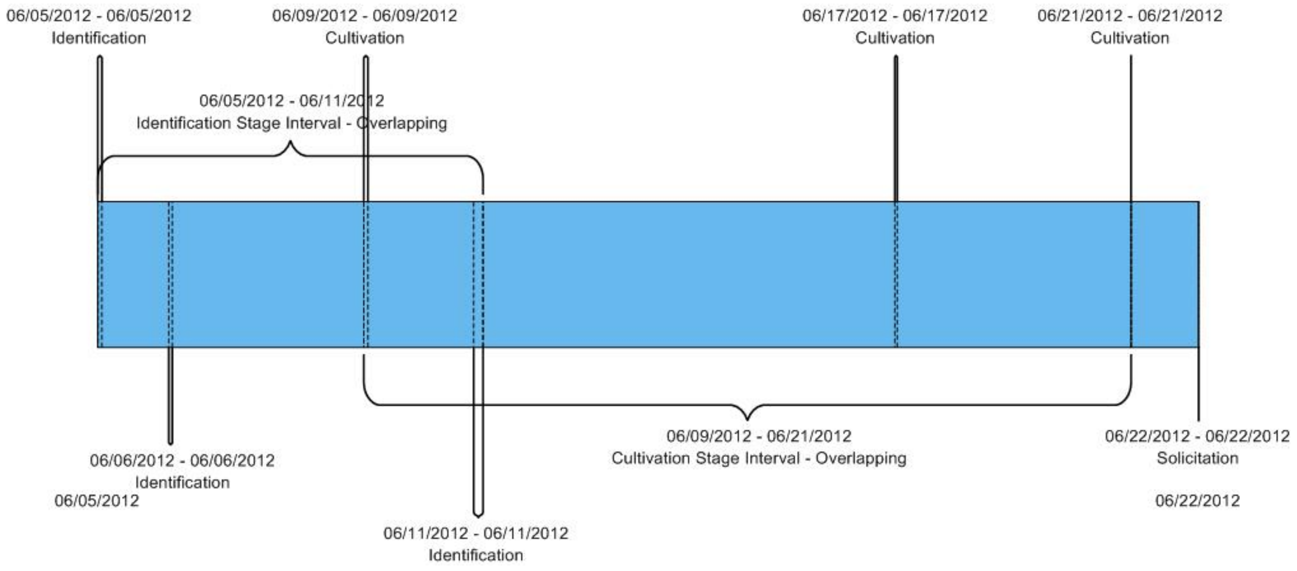
Average days in stage (overlapping) = Total days in stage / Number of prospect plans

The report will also show average times in a stage where the calculation is

Average times in a stage = Total number of times in a stage / Number of prospect plans

- Identification = 06/11/2012 9:51:00 AM - 06/05/2012 5:00:00 PM = 5.70208333333333 days
- Cultivation(a) = 11.3541666666667 days
- Solicitation days is considered to be indeterminate because there is an open Solicitation step

...



Stage	Average days in stage (overlapping)	Min	Max
Identification	xx.xx	xx.xx	xx.xx
Cultivation	xx.xx	xx.xx	xx.xx
Solicitation	xx.xx	xx.xx	xx.xx
Negotiation	xx.xx	xx.xx	xx.xx

Note: We could alternately consider the start datetime for the first step and the start datetime for the step after the last step in the stage.

Overlapping Alternative:

Total days in stage = Start time for step after last step in stage in a prospect plan - Start time for first step in stage in a prospect plan

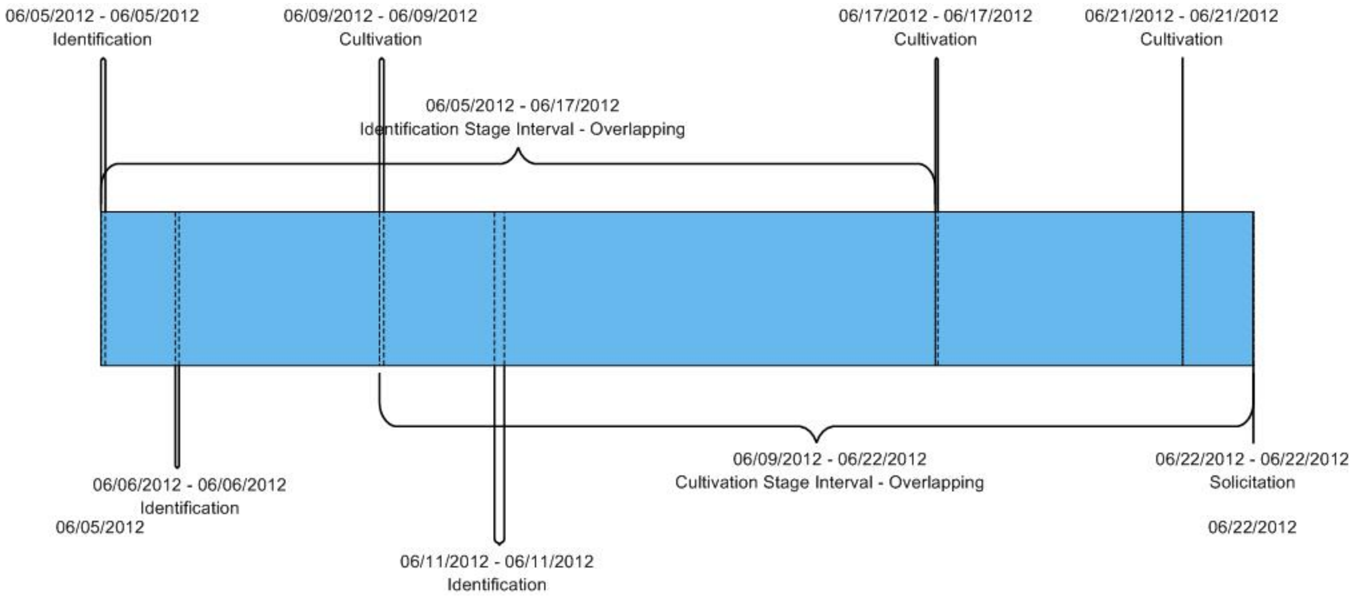
Average days in stage (overlapping) = Total days in stage / Number of prospect plans

The report will also show average times in a stage where the calculation is

Average times in a stage = Total number of times in a stage / Number of prospect plans

- Identification = 11.76944444 days
- Cultivation(a) = 12.29305556 days
- Solicitation days is considered to be indeterminate because there is an open Solicitation step

...



Finding the Data

Within these topics we will look at how to find data in the application, the transactional database, and in the data warehouse database. Then we will create Transact-SQL queries to calculate the metrics described in Prospect Plan Status Durations Report on page 11.

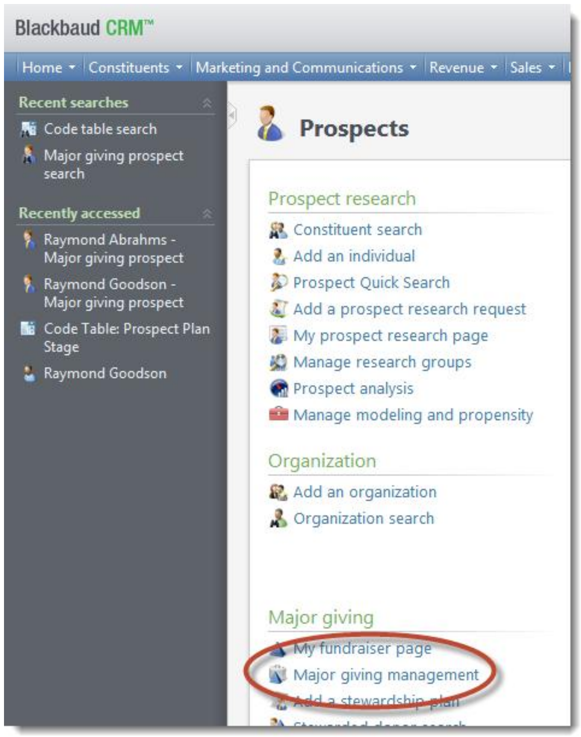
Finding the Data in the Application

For information about **Prospects** functionality in ***Blackbaud CRM***, see Prospects Guide.

This metric isn't surfaced in ***Blackbaud CRM*** as of version 2.93. For example, a KPI which shows average days in a stage could be displayed on a page in ***Blackbaud CRM***. But ***Blackbaud CRM*** does show which stage is associated with a given step and which stage is associated with a prospect plan.

To find the features that show Prospect Plan Stage information, you can browse ***Blackbaud CRM*** features through navigation, the search, or the **Administration** functional area. For more information, see Application Features on page 132.

Here is one path to information about Prospect Plan Stages: **Prospects** > **Major Giving Management** > **Major Giving Management - Prospects**.



Blackbaud CRM™

Home · Constituents · Marketing and Communications · Revenue · Sales

Recent searches
- Code table search
- Major giving prospect search

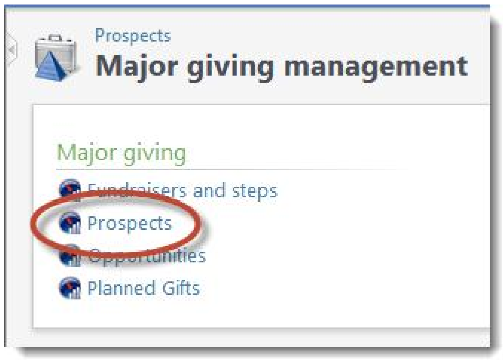
Recently accessed
- Raymond Abrahms - Major giving prospect
- Raymond Goodson - Major giving prospect
- Code Table: Prospect Plan Stage
- Raymond Goodson

Prospects

Prospect research
- Constituent search
- Add an individual
- Prospect Quick Search
- Add a prospect research request
- My prospect research page
- Manage research groups
- Prospect analysis
- Manage modeling and propensity

Organization
- Add an organization
- Organization search

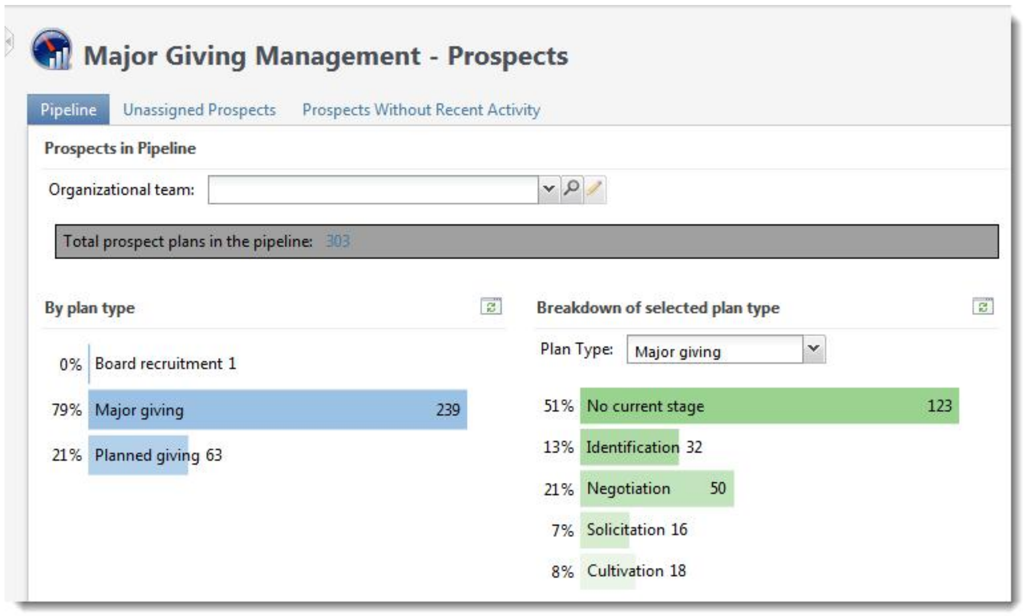
Major giving
- My fundraiser page
- Major giving management
- Add a stewardship plan



Prospects

Major giving management

Major giving
- Fundraisers and steps
- Prospects
- Opportunities
- Planned Gifts


Major Giving Management - Prospects
Pipeline
Unassigned Prospects
Prospects Without Recent Activity

Prospects in Pipeline

Organizational team:

Total prospect plans in the pipeline: 303

By plan type

0%	Board recruitment	1
79%	Major giving	239
21%	Planned giving	63

Breakdown of selected plan type

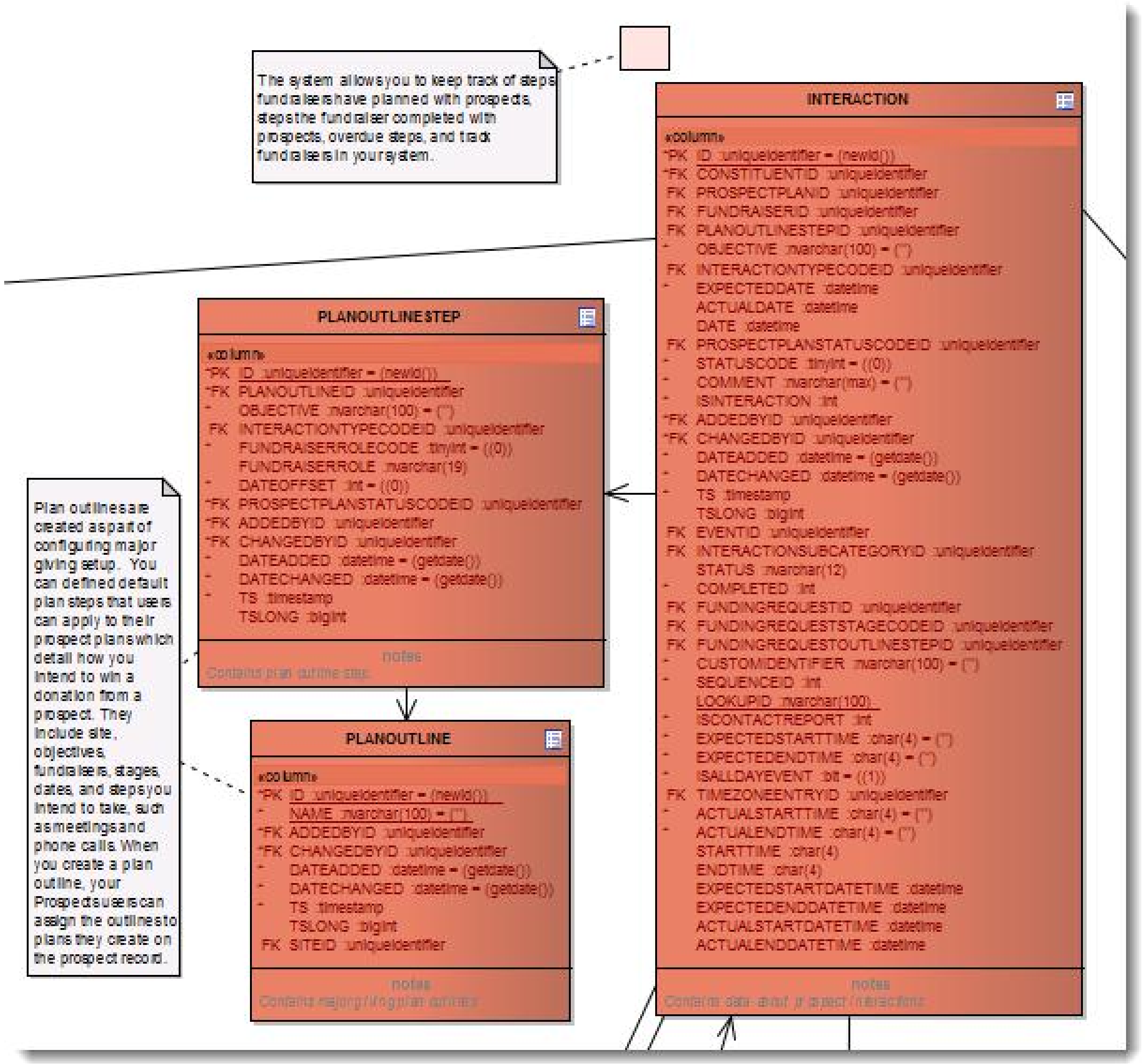
Plan Type: Major giving

51%	No current stage	123
13%	Identification	32
21%	Negotiation	50
7%	Solicitation	16
8%	Cultivation	18

Finding Data in the OLTP Database

An entity relationship diagram for **Prospects** in an Infinity database is here: Prospects ERD

One part of the diagram that relates to this report example is:



The system allows you to keep track of steps fundraisers have planned with prospects, steps the fundraiser completed with prospects, overdue steps, and track fundraisers in your system.

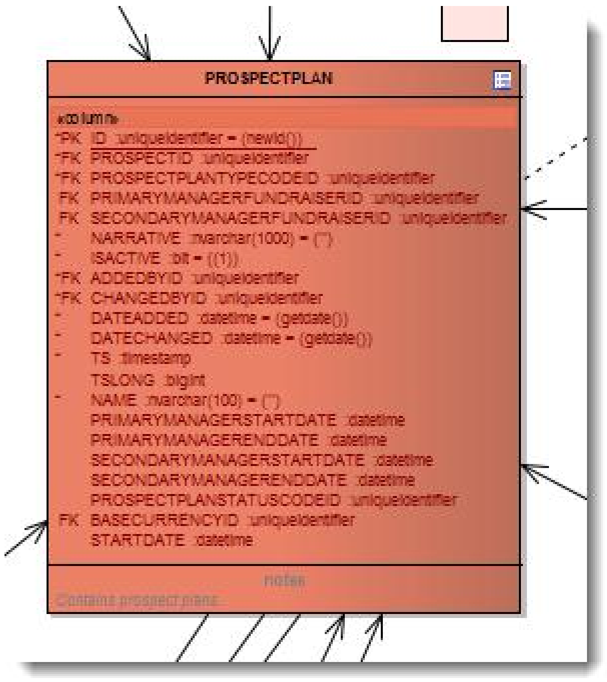
Plan outlines are created as part of configuring major giving setup. You can defined default plan steps that users can apply to their prospect plans which detail how you intend to win a donation from a prospect. They include site, objectives, fundraisers, stages, dates, and steps you intend to take, such as meetings and phone calls. When you create a plan outline, your Prospects users can assign the outlines to plans they create on the prospect record.

INTERACTION
«column»
*PK ID :uniqueidentifier = (newid())
*FK CONSTITUENTID :uniqueidentifier
FK PROSPECTPLANID :uniqueidentifier
FK FUNDRAISERID :uniqueidentifier
FK PLANOUTLINESTEPID :uniqueidentifier
* OBJECTIVE :nvarchar(100) = ('')
FK INTERACTIONTYPECODEID :uniqueidentifier
* EXPECTEDDATE :datetime
ACTUALDATE :datetime
DATE :datetime
FK PROSPECTPLANSTATUSCODEID :uniqueidentifier
* STATUSCODE :tinyint = ((0))
* COMMENT :nvarchar(max) = ('')
* ISINTERACTION :int
*FK ADDEDBYID :uniqueidentifier
*FK CHANGEDBYID :uniqueidentifier
* DATEADDED :datetime = (getdate())
* DATECHANGED :datetime = (getdate())
* TS :timestamp
TSLONG :bigint
FK EVENTID :uniqueidentifier
FK INTERACTIONSUBCATEGORYID :uniqueidentifier
STATUS :nvarchar(12)
* COMPLETED :int
FK FUNDINGREQUESTID :uniqueidentifier
FK FUNDINGREQUESTSTAGECODEID :uniqueidentifier
FK FUNDINGREQUESTOUTLINESTEPID :uniqueidentifier
* CUSTOMIDENTIFIER :nvarchar(100) = ('')
* SEQUENCEID :int
LOOKUPID :nvarchar(100)
* ISCONTACTREPORT :int
* EXPECTEDSTARTTIME :char(4) = ('')
* EXPECTEDENDTIME :char(4) = ('')
* ISALLDAYEVENT :bit = ((1))
FK TIMEZONEENTRYID :uniqueidentifier
* ACTUALSTARTTIME :char(4) = ('')
* ACTUALENDTIME :char(4) = ('')
STARTTIME :char(4)
ENDTIME :char(4)
EXPECTEDSTARTDATETIME :datetime
EXPECTEDENDDATETIME :datetime
ACTUALSTARTDATETIME :datetime
ACTUALENDDATETIME :datetime
notes
Contains data about prospect interactions
PLANOUTLINESTEP
«column»
*PK ID :uniqueidentifier = (newid())
*FK PLANOUTLINEID :uniqueidentifier
* OBJECTIVE :nvarchar(100) = ('')
FK INTERACTIONTYPECODEID :uniqueidentifier
* FUNDRAISERROLECODE :tinyint = ((0))
FUNDRAISERROLE :nvarchar(19)
* DATEOFFSET :int = ((0))
*FK PROSPECTPLANSTATUSCODEID :uniqueidentifier
*FK ADDEDBYID :uniqueidentifier
*FK CHANGEDBYID :uniqueidentifier
* DATEADDED :datetime = (getdate())
* DATECHANGED :datetime = (getdate())
* TS :timestamp
TSLONG :bigint
notes
Contains plan outline step
PLANOUTLINE
«column»
*PK ID :uniqueidentifier = (newid())
* NAME :nvarchar(100) = ('')
*FK ADDEDBYID :uniqueidentifier
*FK CHANGEDBYID :uniqueidentifier
* DATEADDED :datetime = (getdate())
* DATECHANGED :datetime = (getdate())
* TS :timestamp
TSLONG :bigint
FK SITEID :uniqueidentifier
notes
Contains majorgiving plan outlines

An interaction is associated with a plan outline step through a foreign key. The column is `PLAN-OUTLINESTEPID` of type `uniqueidentifier`. The `GUIDs` in the `INTER-ACTION.PLANOUTLINESTEPID` column correspond to `ID` in the `PLANOUTLINESTEP` table.

A plan outline step is associated with a plan outline through a foreign key. The column is `PLANOUTLINEID` of type `uniqueidentifier`. The `GUIDs` in the `PLANOUTLINESTEP.PLANOUTLINEID` column correspond to `ID` in the `PLANOUTLINE` table.

An interaction is associated with a constituent through a foreign key. The column is `CONSTITUENTID` of type `uniqueidentifier`. The `GUIDs` in the `INTERACTION.CONSTITUENTID` column correspond to `ID` in the `CONSTITUENT` table.


PROSPECTPLAN
«column»
*PK ID :uniqueidentifier = (newid())
*FK PROSPECTID :uniqueidentifier
*FK PROSPECTPLANTYPECODEID :uniqueidentifier
FK PRIMARYMANAGERFUNDRAISERID :uniqueidentifier
FK SECONDARYMANAGERFUNDRAISERID :uniqueidentifier
* NARRATIVE :nvarchar(1000) = ('')
* ISACTIVE :bit = ((1))
*FK ADDEDBYID :uniqueidentifier
*FK CHANGEDBYID :uniqueidentifier
* DATEADDED :datetime = (getdate())
* DATECHANGED :datetime = (getdate())
* TS :timestamp
TSLONG :bigint
* NAME :nvarchar(100) = ('')
PRIMARYMANAGERSTARTDATE :datetime
PRIMARYMANAGERENDDATE :datetime
SECONDARYMANAGERSTARTDATE :datetime
SECONDARYMANAGERENDDATE :datetime
PROSPECTPLANSTATUSCODEID :uniqueidentifier
FK BASECURRENCYID :uniqueidentifier
STARTDATE :datetime
notes
Contains prospect plans

An interaction is associated with a prospect plan through a foreign key. The column is `PROSPECTPLANID` and the fields are of type `uniqueidentifier`. The `GUIDs` in the `INTERACTION.PROSPECTPLANID` column correspond to `ID` in the `PROSPECTPLAN` table.

Notice `PROSPECTPLANSTATUSCODEID` appears on the `INTERACTION`, `PLANOUTLINESTEP`, and `PROSPECTPLAN` tables. `PROSPECTPLANSTATUSCODEID` indicates which stage of the plan to which a step belongs. `PROSPECTPLANSTATUSCODEID` identifies an entry on the `PROSPECTPLANSTATUSCODE` code table. The code table is called Prospect Plan *Stage*. But the table name is `PROSPECTPLANSTATUSCODE`.

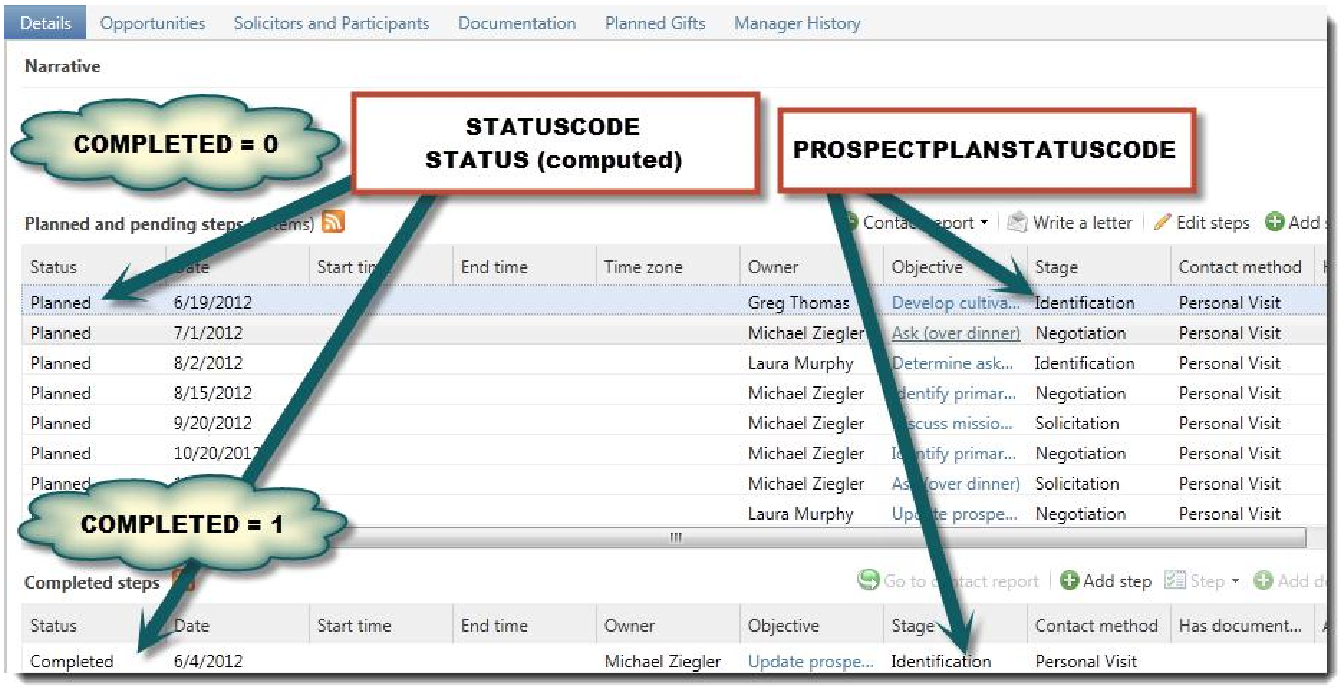
`PROSPECTPLANSTATUSCODE` (Prospect Plan Stage) can be managed from **Administration** > **Code Tables**. The category for the Prospect Plan Stage code table is Major Giving. Typical entries include **Identification**, **Cultivation**, **Solicitation**, and **Negotiation**.

The `INTERACTION` table contains other status information in the `STATUSCODE`, `STATUS`, and `COMPLETED` columns. `STATUSCODE` is a `tinyint` column that maintains these codes: `0=Planned, 1=Pending, 2=Completed, 3=Unsuccessful, 4=Cancelled, 5=Declined`. `COMPLETED` is a computed `int` column with this expression: `case when STATUSCODE in (2,3,4,5) then 1 else 0 end`. So `COMPLETED` is true (1) when `STATUSCODE` contains the `tinyint` code representation for Completed, Unsuccessful, Cancelled, or Declined. `STATUS` is a computed field which provides a translation for `STATUSCODE`:

```
CASE [STATUSCODE]
WHEN 0 THEN N'Planned'
WHEN 1 THEN N'Pending'
WHEN 2 THEN N'Completed'
WHEN 3 THEN N'Unsuccessful'
WHEN 4 THEN N'Cancelled'
WHEN 5 THEN N'Declined'
END
```

Note: There are also status-related columns for funding requests. But these columns support **Foundations** functionality.

On the PROSPECTPLAN table, PROSPECTPLANSTATUSCODEID indicates **Current plan stage**. This is the stage of the most recently completed plan step. On the INTERACTION table, PROSPECTPLANSTATUSCODEID indicates the stage associated with the interaction, which also represents a step. On the PLANOUTLINESTEP table, PROSPECTPLANSTATUSCODEID indicates the stage associated with the plan outline step. A plan outline step is not a step in a prospect plan. A plan outline step is a step in a plan outline. Plan outline steps and plan outlines are template mechanisms. Plan outlines establish the default steps created when you add a plan based on a plan outline.



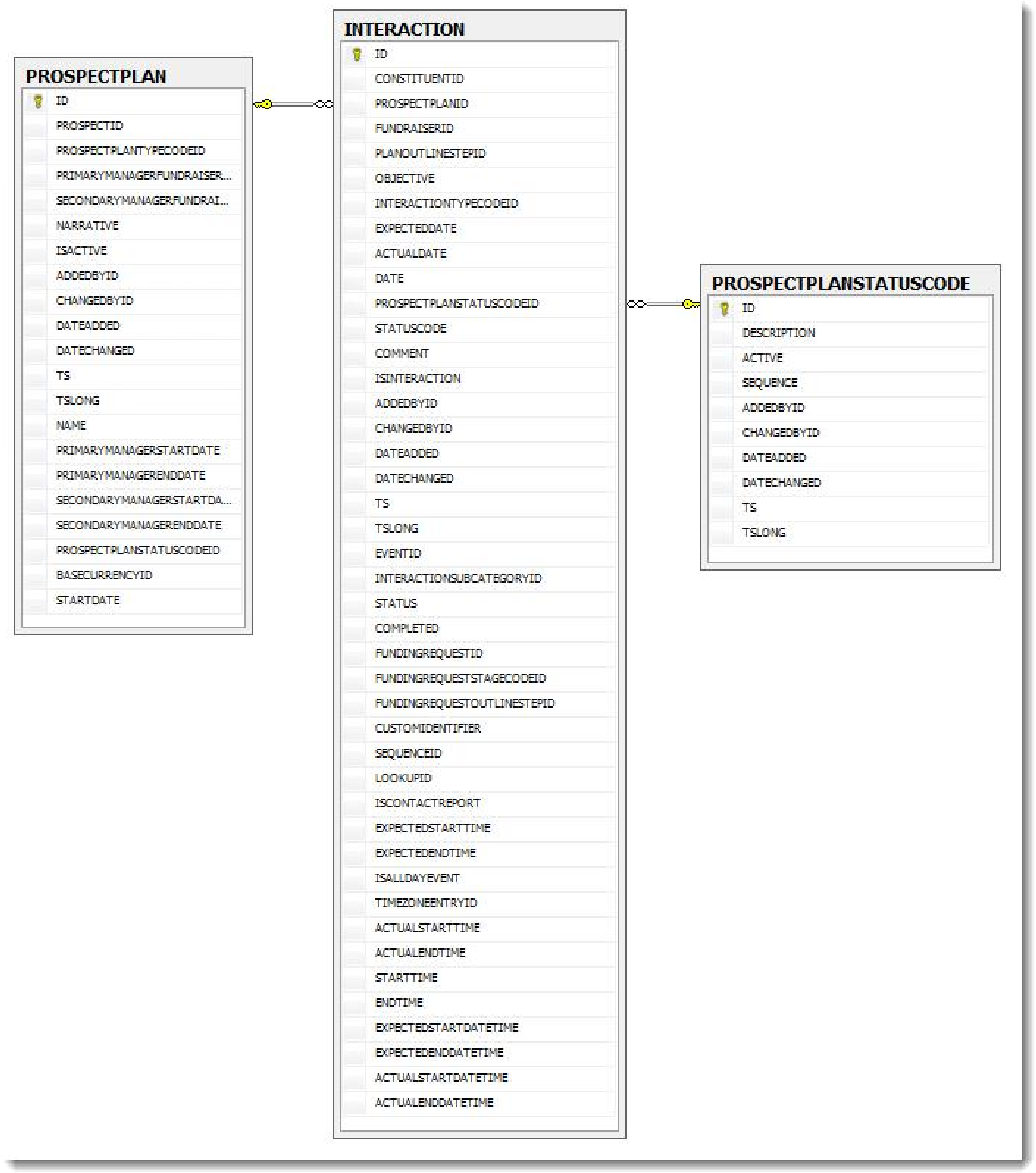
Status	Date	Start time	End time	Time zone	Owner	Objective	Stage	Contact method
Planned	6/19/2012				Greg Thomas	Develop cultiva...	Identification	Personal Visit
Planned	7/1/2012				Michael Ziegler	Ask (over dinner)	Negotiation	Personal Visit
Planned	8/2/2012				Laura Murphy	Determine ask...	Identification	Personal Visit
Planned	8/15/2012				Michael Ziegler	dentify primar...	Negotiation	Personal Visit
Planned	9/20/2012				Michael Ziegler	scuss missio...	Solicitation	Personal Visit
Planned	10/20/2012				Michael Ziegler	Identify primar...	Negotiation	Personal Visit
Planned					Michael Ziegler	Ask (over dinner)	Solicitation	Personal Visit
					Laura Murphy	Update prospe...	Negotiation	Personal Visit

Status	Date	Start time	End time	Owner	Objective	Stage	Contact method	Has document...
Completed	6/4/2012			Michael Ziegler	Update prospe...	Identification	Personal Visit	

So for the purposes of reporting, plan outlines and plan outline steps are not the a primary concern unless the goal is to audit the plan outlines and plan outline steps. But the current plan stage as maintained in PROSPECTPLANSTATUSCODEID in PROSPECTPLAN and the stages associated with steps as maintained in PROSPECTPLANSTATUSCODEID in INTERACTION are useful for reporting on the transitions between stages. INTERACTION contains all of the steps. To report on the transition between stages, INTERACTION is the table to query. But to report on the transition from the start of a plan to the current status, PROSPECTPLAN is the table to query.

A constituent can be associated with more than one plan and more than one plan of a given type. A report that breaks out information by constituent and plan type must address that complication.

Later in the document, the core of our OLTP queries will use the PROSPECTPLAN, INTERACTION, and PROSPECTPLANSTATUSCODE tables as shown in the following database diagram created in SQL Server Management Studio.


PROSPECTPLAN
ID
PROSPECTID
PROSPECTPLANTYPECODEID
PRIMARYMANAGERFUNDRAISER...
SECONDARYMANAGERFUNDRAI...
NARRATIVE
ISACTIVE
ADDEDBYID
CHANGEDBYID
DATEADDED
DATECHANGED
TS
TSLONG
NAME
PRIMARYMANAGERSTARTDATE
PRIMARYMANAGERENDDATE
SECONDARYMANAGERSTARTDA...
SECONDARYMANAGERENDDATE
PROSPECTPLANSTATUSCODEID
BASECURRENCYID
STARTDATE
INTERACTION
ID
CONSTITUENTID
PROSPECTPLANID
FUNDRAISERID
PLANOUTLINESTEPID
OBJECTIVE
INTERACTIONTYPECODEID
EXPECTEDDATE
ACTUALDATE
DATE
PROSPECTPLANSTATUSCODEID
STATUSCODE
COMMENT
ISINTERACTION
ADDEDBYID
CHANGEDBYID
DATEADDED
DATECHANGED
TS
TSLONG
EVENTID
INTERACTIONSUBCATEGORYID
STATUS
COMPLETED
FUNDINGREQUESTID
FUNDINGREQUESTSTAGECODEID
FUNDINGREQUESTOUTLINESTEPID
CUSTOMIDENTIFIER
SEQUENCEID
LOOKUPID
ISCONTACTREPORT
EXPECTEDSTARTTIME
EXPECTEDENDTIME
ISALLDAYEVENT
TIMEZONEENTRYID
ACTUALSTARTTIME
ACTUALENDTIME
STARTTIME
ENDTIME
EXPECTEDSTARTDATETIME
EXPECTEDENDDATETIME
ACTUALSTARTDATETIME
ACTUALENDDATETIME
PROSPECTPLANSTATUSCODE
ID
DESCRIPTION
ACTIVE
SEQUENCE
ADDEDBYID
CHANGEDBYID
DATEADDED
DATECHANGED
TS
TSLONG

Finding Data in the Data Warehouse Database

Blackbaud Data Warehouse Tables and Major Giving Stages

A Blackbaud Data Warehouse database has these tables:

`DIM_INTERACTION`: The Interaction dimension contains information about constituent interactions.

`FACT_INTERACTION`: The Interaction fact relates information to constituent interactions.

`DIM_PROSPECTPLAN`: Contains information about prospect plans.

`DIM_PROSPECTPLANSTATUS`: Contains information about prospect plant status codes.

`DIM_INTERACTION` includes these columns and mappings among others:

`INTERACTIONSTATUSCODE`: `dbo.[INTERACTION].[STATUSCODE]`

`INTERACTIONSTATUS`: `dbo.[INTERACTION].[STATUS]`

`ISINTERACTIONCOMPLETED`: `dbo.[INTERACTION].[COMPLETED]`

`FACT_INTERACTION` includes these columns and mappings among others:

`CONSTITUENTDIMID`: Reference key to the constituent dimension, derived from `dbo.[INTERACTION].[CONSTITUENTID]`

`CONSTITUENTSYSTEMID`: `dbo.[INTERACTION].[CONSTITUENTID]`

`INTERACTIONDIMID`: Reference key to the interaction dimension, derived from `dbo.[INTERACTION].[INTERACTIONTYPECODEID]`, `dbo.[INTERACTION].[INTERACTIONSUBCATEGORYID]`, `dbo.[INTERACTION].[STATUSCODE]`, `dbo.[INTERACTION].[ISALLDAYEVENT]`, `dbo.[INTERACTION].[ISINTERACTION]`, `dbo.[INTERACTION].[COMPLETED]`, and `dbo.[INTERACTION].[ISCONTACTREPORT]`

`PROSPECTPLANDIMID`: Reference key to the prospect plan dimension, derived from `dbo.[INTERACTION].[PROSPECTPLANID]`

`PROSPECTPLANSTATUSDIMID`: Reference key to the prospect plan status dimension, derived from `dbo.[INTERACTION].[PROSPECTPLANSTATUSCODEID]`

`DIM_PROSPECTPLAN` includes these columns and mappings among others:

`CONSTITUENTSYSTEMID`: `dbo.[PROSPECTPLAN].[PROSPECTID]`

`CONSTITUENTDIMID`: Reference key to the constituent dimension, derived from `dbo.[PROSPECTPLAN].[PROSPECTID]`

`PROSPECTSTATUS`: `dbo.[PROSPECTPLANSTATUSCODE].[DESCRIPTION]`

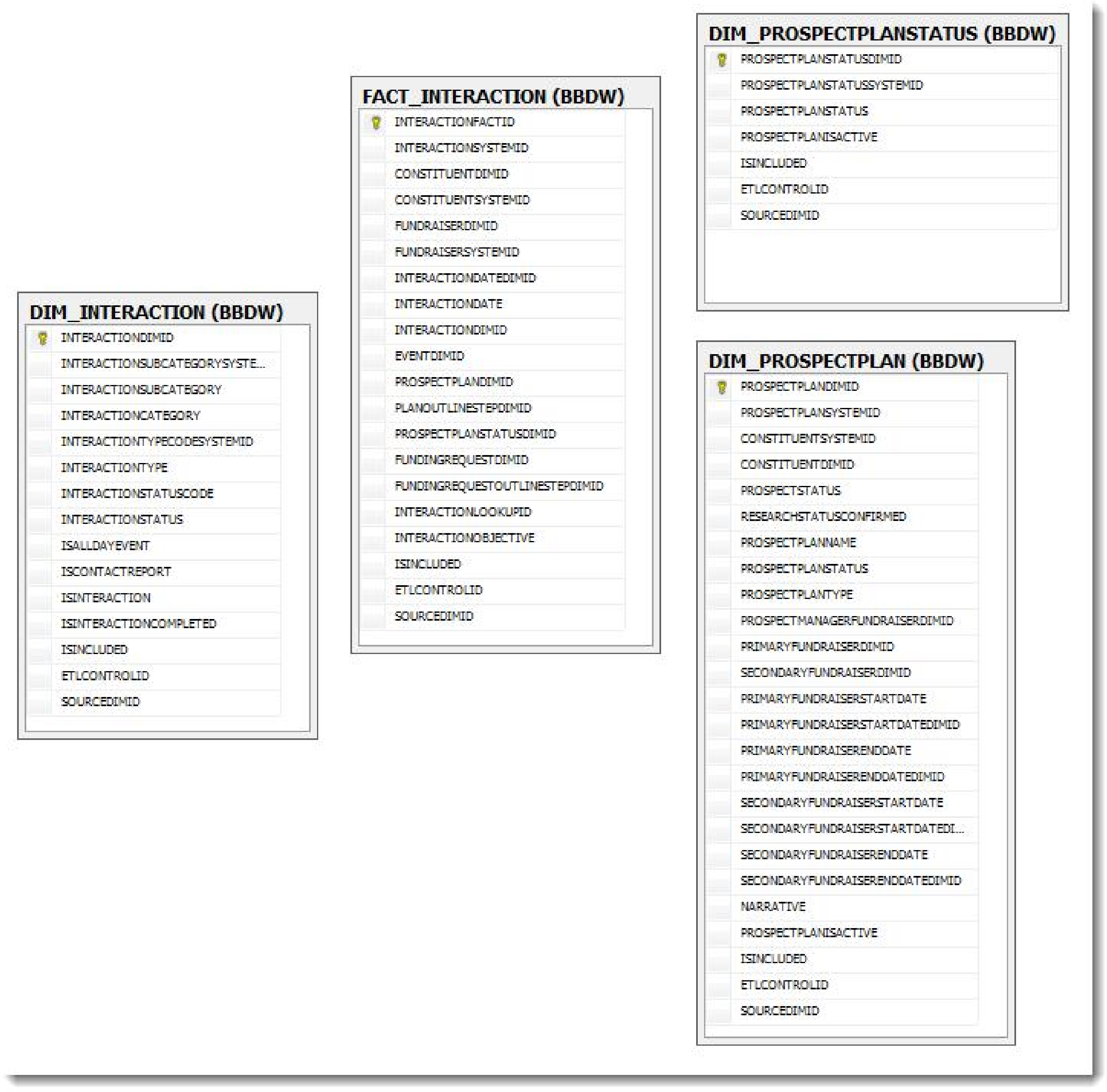
`PROSPECTPLANSTATUS`: `dbo.[PROSPECTPLANSTATUSCODE].[DESCRIPTION]`

`DIM_PROSPECTPLANSTATUS` includes these columns and mappings among others:

`PROSPECTPLANSTATUSSYSTEMID`: `dbo.[PROSPECTPLANSTATUSCODE].[ID]`

PROSPECTPLANSTATUS: dbo.[PROSPECTPLANSTATUSCODE].[DESCRIPTION]

Later in the document, the core of our first set of data warehouse queries will use the DIM_PROSPECTPLAN, DIM_INTERACTION, FACT_INTERACTION, and DIM_PROSPECTPLANSTATUS tables as shown in the following database diagram created in SQL Server Management Studio. Notice there are no foreign key relationships. This is a characteristic of the Blackbaud Data Warehouse data warehouse database. However, if you look at the primary keys for each of the dimension tables, they correspond to columns in the fact table. The relationships exist. But the database is oblivious. Removing foreign keys creates a performance gain for the warehouse.



Blackbaud Data Warehouse Views and Major Giving Stages

Warning: Prospect Plan Stage (`PROSPECTPLANSTATUSCODE`) information is not a part of these views.

A Blackbaud Data Warehouse database has these views:

`v_DIM_INTERACTION`: The interaction dimension contains information about interactions.

`v_FACT_INTERACTION`: The interaction fact table contains information about constituent interactions.

`v_DIM_INTERACTION` includes these columns and mappings among others:

`INTERACTIONSTATUSCODE`: `BBDW.[DIM_INTERACTION].[INTERACTIONSTATUSCODE]`

`INTERACTIONSTATUS`: `BBDW.[DIM_INTERACTION].[INTERACTIONSTATUS]`

`ISINTERACTIONCOMPLETED`: `BBDW.[DIM_INTERACTION].[ISINTERACTIONCOMPLETED]`

`v_FACT_INTERACTION` includes these columns and mappings among others:

`CONSTITUENTDIMID`: `BBDW.[FACT_INTERACTION].[CONSTITUENTDIMID]`

`CONSTITUENTSYSTEMID`: `BBDW.[FACT_INTERACTION].[CONSTITUENTSYSTEMID]`

`INTERACTIONDIMID`: `BBDW.[FACT_INTERACTION].[INTERACTIONDIMID]`

Code Formatting

For the Transact-SQL samples in this documentation, we will follow these guidelines.

- Enter Transact-SQL keywords in lower-case.
- Begin a custom stored procedure name with `USR_USP_`.
- When coding a `JOIN`, use a fully qualified name for fields such as `TABLENAME.FIELDNAME` instead of `FIELDNAME`.
- Indent the code.

Creating an OLTP Version

A Report Which Queries the Blackbaud CRM OLTP Database

For information about reports in ***Blackbaud CRM***, see Reports Guide. For more information about creating reports for ***Blackbaud CRM***, see Infinity Reports.

In many cases, you will get better performance from your reports if you report off of the data warehouse database. But we are going to compare a report off of the OLTP database with reports off of the data warehouse. So we will build both kinds.

Begin by creating a Report Spec. Once loaded, the Report Spec will connect the report to the application interface. For information about how to create a Report Spec, see Create a Report Spec on page 141.

Use ***SQL Server Business Intelligence Development Studio*** (***Visual Studio 2008***) to create the project because there is an editor for `RDL` files.

Logic

Here is a query which returns the stage description (Identification, Cultivation, Negotiation, Solicitation, etc.), the prospect plan name, and the actual start datetime for completed plan steps (interaction records). The query presents the results in order by prospect plan name and, within the prospect plan name, by actual start datetime.

```
select PROSPECTPLANSTATUSCODE.[DESCRIPTION],
       PROSPECTPLAN.[NAME],
       INTERACTION.[ACTUALSTARTDATETIME]
from INTERACTION
inner join PROSPECTPLANSTATUSCODE on INTERACTION.[PROSPECTPLANSTATUSCODEID] = PROSPECTPLANSTATUSCODE.[ID]
inner join PROSPECTPLAN on INTERACTION.[PROSPECTPLANID] = PROSPECTPLAN.[ID]
where (INTERACTION.[COMPLETED] = 1)
group by PROSPECTPLANSTATUSCODE.[DESCRIPTION],
       INTERACTION.[ACTUALSTARTDATETIME],
       PROSPECTPLAN.[NAME]
order by PROSPECTPLAN.[NAME],
       INTERACTION.[ACTUALSTARTDATETIME]
```

This gives us a starting place for the problem. If we create a plan in ***Blackbaud CRM*** called Test and enter the example steps from the design, the query returns these rows:

DESCRIPTION	NAME	ACTUALSTARTDATETIME
Identification	Test	06/05/2012 5:00:00 PM
Identification	Test	06/06/2012 6:15:00 PM
Cultivation	Test	06/09/2012 3:30:00 PM
Identification	Test	06/11/2012 6:30:00 AM
Cultivation	Test	06/17/2012 12:05:00 PM
Cultivation	Test	06/21/2012 12:00:00 AM
Solicitation	Test	06/22/2012 12:00:00 AM

If there were other prospect plans, there would be rows for those as well. But the results would be grouped by prospect plan name.

For each prospect plan, we want to identify the first instance of consecutive occurrences of a stage description in these results. Those rows will give us much of what we need to address the consecutive and nonconsecutive perspectives.

DESCRIPTION	NAME	ACTUALSTARTDATETIME
Identification	**Test**	**06/05/2012 5:00:00 PM**
Identification	Test	06/06/2012 6:15:00 PM
Cultivation	**Test**	**06/09/2012 3:30:00 PM**
Identification	Test	06/11/2012 6:30:00 AM
Cultivation	**Test**	**06/17/2012 12:05:00 PM**
Cultivation	Test	06/21/2012 12:00:00 AM
Solicitation	**Test**	**06/22/2012 12:00:00 AM**

For each prospect plan, we want to identify the first instance of a stage description in these results. Those rows will give us much of what we need to address the overlapping perspective. For example:

DESCRIPTION	NAME	ACTUALSTARTDATETIME
Identification	**Test**	**06/05/2012 5:00:00 PM**
Identification	Test	06/06/2012 6:15:00 PM
Cultivation	**Test**	**06/09/2012 3:30:00 PM**
Identification	Test	06/11/2012 6:30:00 AM
Cultivation	Test	06/17/2012 12:05:00 PM
Cultivation	Test	06/21/2012 12:00:00 AM
Solicitation	**Test**	**06/22/2012 12:00:00 AM**

The difference is that with the overlapping perspective, we are not concerned with when a new set of consecutive occurrences of a stage appear. In the consecutive and nonconsecutive perspectives, we need to identify occurrences of consecutive stage descriptions. In the overlapping perspective, we need to identify only the first occurrence of a stage description. But it turns out that we need information from adjacent rows to complete the picture.

For the first stage occurrence, the start time is also the start time of the first step in the plan. The end time is the end time of the last step in the occurrence. When there is a subsequent stage occurrence, the end datetime is the same as the start datetime of the first step in the subsequent occurrence. When there is no subsequent stage occurrence, the end datetime is the end datetime of the plan, which is the same as the end datetime of the last step in the stage occurrence. Since we are paring down rows to represent either a stage in the case of the overlapping perspective or a stage occurrence in the other perspectives, we need to pick which rows.

We will hold on to the last row of each occurrence or stage. For the overlapping perspective, we have what we need in those rows with the exception of the start datetime of the plan and the end datetime of the plan. We can use `MIN` and `MAX` partitioned by plan and status code to find the results we need.

```
 select i.[PROSPECTPLANSTATUSCODEID],
        i.[ACTUALENDDATETIME],
        min(i.[ACTUALSTARTDATETIME]) over (
                partition by i.[PROSPECTPLANID],
                i.[PROSPECTPLANSTATUSCODEID]
                ) as [FIRSTSTEPINSTAGEDATETIME],
        max(i.[ACTUALENDDATETIME]) over (
                partition by i.[PROSPECTPLANID],
                i.[PROSPECTPLANSTATUSCODEID]
                ) as [LASTSTEPINSTAGEDATETIME]
from [INTERACTION] as i
where i.[COMPLETED] = 1
```

For the Test plan data (`1D85EEC8-5205-4205-A5D6-9A31F4C78EA5` is the ID for Test):

PROSPECTPLANSTATUSCODEID	ACTUALENDDATETIME	FIRSTSTEPINSTAGEDATETIME	LASTSTEPINSTAGEDATETIME
1D85EEC8-5205-4205-A5D6-9A31F4C78EA5	2012-06-05 18:30:00.000	2012-06-05 17:00:00.000	2012-06-11 09:51:00.000
1D85EEC8-5205-4205-A5D6-9A31F4C78EA5	2012-06-06 19:29:00.000	2012-06-05 17:00:00.000	2012-06-11 09:51:00.000
1D85EEC8-5205-4205-A5D6-9A31F4C78EA5	2012-06-11 09:51:00.000	2012-06-05 17:00:00.000	2012-06-11 09:51:00.000
038E8841-E30B-4B32-A621-E986D75FAAF5	2012-06-22 00:00:00.000	2012-06-22 00:00:00.000	2012-06-22 00:00:00.000
3EF0AE1D-7F63-4471-BB63-EF9ACFEF168A	2012-06-17 12:58:00.000	2012-06-09 15:30:00.000	2012-06-21 00:00:00.000
3EF0AE1D-7F63-4471-BB63-EF9ACFEF168A	2012-06-21 00:00:00.000	2012-06-09 15:30:00.000	2012-06-21 00:00:00.000
3EF0AE1D-7F63-4471-BB63-EF9ACFEF168A	2012-06-09 16:58:00.000	2012-06-09 15:30:00.000	2012-06-21 00:00:00.000

We are going use aggregate calculations with these rows. We can set up a common table expression and select from that.

```
with [STEPS]
as
(
       select i.[PROSPECTPLANSTATUSCODEID],
               i.[ACTUALENDDATETIME],
               min(i.[ACTUALSTARTDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [FIRSTSTEPINSTAGEDATETIME],
               max(i.[ACTUALENDDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [LASTSTEPINSTAGEDATETIME]
       from [INTERACTION] as i
       where i.[COMPLETED] = 1
       )
select *
from [STEPS] as s
```

This returns the same results. We want the friendly name for the stage rather than the `GUID`. So we will use `INNER JOIN` to get the prospect plan status code description from the `PROSPECTPLANSTATUSCODE` code table.

```
        from [STEPS] as s
        inner join [PROSPECTPLANSTATUSCODE] as p on s.[PROSPECTPLANSTATUSCODEID] = p.[ID]
```

We also want to reduce the rows to only the last row in a stage. Also, we are going to include the stage description. So we will `GROUP BY` that.

```
        where (s.[ACTUALENDDATETIME] = s.[LASTSTEPINSTAGEDATETIME])
        group by p.[DESCRIPTION]
```

For every stage in a plan, we now have the stage description, start datetime, and end datetime. We can use the datetimes to calculate a duration and find the average, minimum, and maximum of those durations.

```
        with [STEPS]
        as (
               select i.[PROSPECTPLANSTATUSCODEID],
                       i.[ACTUALENDDATETIME],
                       min(i.[ACTUALSTARTDATETIME]) over (
                               partition by i.[PROSPECTPLANID],
                               i.[PROSPECTPLANSTATUSCODEID]
                               ) as [FIRSTSTEPINSTAGEDATETIME],
                       max(i.[ACTUALENDDATETIME]) over (
                               partition by i.[PROSPECTPLANID],
                               i.[PROSPECTPLANSTATUSCODEID]
                               ) as [LASTSTEPINSTAGEDATETIME]
               from [INTERACTION] as i
               where i.[COMPLETED] = 1
               )
        select p.[DESCRIPTION] as [STAGENAME],
               avg(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [AVGSTAGEDURATION],
               min(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MINSTAGEDURATION],
               max(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MAXSTAGEDURATION]
        from [STEPS] as s
        inner join [PROSPECTPLANSTATUSCODE] as p on s.[PROSPECTPLANSTATUSCODEID] = p.[ID]
        where (s.[ACTUALENDDATETIME] = s.[LASTSTEPINSTAGEDATETIME])
        group by p.[DESCRIPTION]
```

STAGENAME	AVGSTAGEDURATION	MINSTAGEDURATION	MAXSTAGEDURATION
Cultivation	11.3541666666667	11.3541666666667	11.3541666666667
Identification	5.70208333333333	5.70208333333333	5.70208333333333
Solicitation	0	0	0

For the full `CREATE PROCEDURE` code to be used in the Report Spec, see Overlapping Perspective Stored Procedure on page 38.

Now we have our overlapping perspective. The consecutive and nonconsecutive perspectives are more complex. The reason is those perspectives look at stage occurrences rather than just stages. A plan can go back and forth between stages. So there can be more than one stage occurrence for a given stage. For example, the plan can move from Identification to Solicitation and back to Identification. We can't partition this effectively in one query. To overcome that, we can use row comparisons. To perform row comparisons, we join a table to itself on a sequence. But we stagger the sequence.

We are still going to use a common table expression. But we maintain some additional information and create a sequence number for the steps. To create a sequence number, we use the `ROW_NUMBER` function. We also maintain the start time, prospect plan ID, and prospect plan status ID. We will discuss the parameter `@PROSPECTPLANTYPECODEID` in another section.

```
with [STEPS]
as (
       select row_number() over (
                       order by i.[PROSPECTPLANID],
                               i.[ACTUALSTARTDATETIME]
                       ) as [ALLSTEPSEQUENCENUMBER],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALENDDATETIME],
               i.[PROSPECTPLANID],
               max(i.[ACTUALENDDATETIME]) over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
               i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
               and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       ),
```

For the Test data:

ALL-STEP-SEQUENCENUMBER	ACTU-ALSTARTDATETIME	ACTU-ALENDDATETIME	PROS-PECTPLANID	FIRST-STE-PINPLANDATETIME	LAST-STE-PINPLANDATETIME	PROS-PECT-PLANSTATUSCODEID
1	2012-06-05 17:00:00.000	2012-06-05 18:30:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	1D85EEC8-5205-4205-A5D6-9A31F4C78EA5

ALL-STEP-SEQUENCENUMBER	ACTU-ALSTARTDATETIME	ACTU-ALENDDATETIME	PROS-PECTPLANID	FIRST-STE-PINPLANDATETIME	LAST-STE-PINPLANDATETIME	PROS-PECT-PLANSTATUSCODEID
			0			
2	2012-06-06 18:15:00.000	2012-06-06 19:29:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-0	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	1D85EEC8-5205-4205-A5D6-9A31F4C78EA5
3	2012-06-09 15:30:00.000	2012-06-09 16:58:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-0	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	3EF0AE1D-7F63-4471-BB63-EF9ACFEF168A
4	2012-06-11 06:30:00.000	2012-06-11 09:51:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-0	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	1D85EEC8-5205-4205-A5D6-9A31F4C78EA5
5	2012-06-17 12:05:00.000	2012-06-17 12:58:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-0	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	3EF0AE1D-7F63-4471-BB63-EF9ACFEF168A
6	2012-06-21 00:00:00.000	2012-06-21 00:00:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-0	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	3EF0AE1D-7F63-4471-BB63-EF9ACFEF168A

ALL-STEP-SEQUENCENUMBER	ACTU-ALSTARTDATETIME	ACTU-ALENDDATETIME	PROS-PECTPLANID	FIRST-STE-PINPLANDATETIME	LAST-STE-PINPLANDATETIME	PROS-PECT-PLANSTATUSCODEID
7	2012-06-22 00:00:00.000	2012-06-22 00:00:00.000	5F14DA30-E8E7-49D7-BF41-3ABA267B3F2-0	2012-06-05 17:00:00.000	2012-06-22 00:00:00.000	038E8841-E30B-4B32-A621-E986D75FAAF5

We want to grab some information from the preceding rows before we filter and create granularities for the other two perspectives. In particular, we want the prospect plan ID and prospect plan status of the preceding steps:

```
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
               s.[ACTUALENDDATETIME],
               r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
               s.[PROSPECTPLANSTATUSCODEID],
               r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
               s.[PROSPECTPLANID],
               s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.[ALLSTEPSEQUENCENUMBER]
       ),
```

For the consecutive perspective, the `[STEPS]` granularity is used to create a `[STAGEOCCURRENCES]` granularity which is used by the `[STAGE-DURATIONS]` common table expression:

```
[STAGEOCCURRENCES]
as (
       select row_number() over (
                       order by sofp.[PROSPECTPLANID],
                               sofp.[ACTUALSTARTDATETIME]
                       ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
               sofp.[ACTUALSTARTDATETIME],
               sofp.[ACTUALENDDATETIME],
               sofp.[LASTSTEPINPLANENDDATETIME],
               sofp.[PROSPECTPLANID],
               sofp.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCESFIRSTPASS] as sofp
       where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
               or (
                       (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                       and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                       )
       ),
[STAGEOCCURRENCEDURATIONS]
as (
       select so1.[ACTUALSTARTDATETIME] as [STARTDATETIME],
               so1.[ACTUALENDDATETIME] as [ENDDATETIME],
               "STAGEOCCURRENCEDURATION" = case
                       when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                               then cast(so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
                               then cast(so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       end,
               so1.[PROSPECTPLANID],
               so1.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCES] as so1
       left join [STAGEOCCURRENCES] as so2 on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.A-
LLSTAGEOCCURRENCESSEQUENCENUMBER
       )
```

The information gleaned from the row comparison is used to filter out unneeded rows in order to establish the new granularity of stage occurrence. So rather than discrete steps in a plan, the rows represent unbroken periods of time in a particular plan stage.

```
        where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
                or (
                        (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                        and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                        )
```

In `[STAGEOCCURRENCEDURATIONS]` information from the next row for a stage occurrence is used to calculate stage occurrence durations. The case statement determines qualities about the stage occurrence represented by the row and bases the duration calculation on those qualities.

```
                "STAGEOCCURRENCEDURATION" = case
                        when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                                then cast(so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                        when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
```

```
                                        then cast(so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                                end,
```

DESCRIPTION	AVGSTAGEOCCURRENCEDURATION	MINSTAGEOCCURRENCEDURATION	MAXSTAGEOCCURRENCEDURATION	AVGTIMESINSTAGE
Cultivation	3.06076388888889	1.625	4.49652777777778	2
Identification	6.23263888888889	6.23263888888889	6.23263888888889	1
Solicitation	0	0	0	1

Notice that for the original sequence number (ALLSTEPSEQUENCENUMBER), we decremented the row in the comparison and for the second one (ALL-STAGEOCCURRENCESSEQUENCENUMBER), we incremented the row. We used ALLSTAGEOCCURRENCESSEQUENCENUMBER to create a join to compare the subsequent row, we will use ALLSTEPSEQUENCENUMBER to compare the preceding row. But also notice that when we decremented, we made this provision:.

```
                where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
                       or (
                                (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                                and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                                )
```

The calculation of average, minimum, maximum, and average count of occurrences is performed in a similar way as the overlapping perspective. But the average times here are for stage occurrence durations not stage durations. Also, there is an extra calculation for average times in stage.

```
        select p.[DESCRIPTION] as [STAGENAME],
               avg(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [AVGSTAGEOCCURRENCEDURATION],
               min(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [MINSTAGEOCCURRENCEDURATION],
               max(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [MAXSTAGEOCCURRENCEDURATION],
               cast(count([STAGEOCCURRENCEDURATIONS].[PROSPECTPLANID]) as float) / cast(count(distinct ([STAGEOCCURRENCEDURATIONS].
        [PROSPECTPLANID])) as float) as [AVGTIMESINSTAGE]
        from [STAGEOCCURRENCEDURATIONS]
        inner join [PROSPECTPLANSTATUSCODE] as p on [STAGEOCCURRENCEDURATIONS].[PROSPECTPLANSTATUSCODEID] = p.[ID]
        group by p.[DESCRIPTION]
```

It is tempting to find the average number of times in a stage using the AVG function on a count created in the previous granularity. But remember there is a row for each stage occurrence. So the average would be weighted. This average calculation avoids that:

```
        cast(count([STAGEOCCURRENCEDURATIONS].[PROSPECTPLANID]) as float) /
        cast(count(distinct ([STAGEOCCURRENCEDURATIONS].[PROSPECTPLANID])) as float) as [AVGTIMESINSTAGE]
```

For the full CREATE PROCEDURE code to be used in the Report Spec, see Consecutive Perspective Stored Procedure on page 39.

The main difference between the consecutive and nonconsecutive perspective is the totaling of the stage occurrence times. We add another layer of granularity for durations and total the durations each stage occurrence.

```
[STAGEDURATIONS]
as (
       select sum(cast(sod.[STAGEOCCURRENCEDURATION] as float)) over (
                        partition by sod.[PROSPECTPLANID],
                        sod.[PROSPECTPLANSTATUSCODEID]
                        ) as [STAGEDURATION],
               RANK() over (
                        partition by sod.[PROSPECTPLANID],
                        sod.[PROSPECTPLANSTATUSCODEID] order by sod.[ACTUALSTARTDATETIME]
                        ) as [FIRSTOCCURRENCEROWINSTAGE],
               sod.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCEDURATIONS] as sod
       )
```

The average count of times in stage is not necessary for the nonconsecutive perspective:

```
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [AVGSTAGEDURATION],
       min(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [MINSTAGEDURATION],
       max(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [MAXSTAGEDURATION]
from [STAGEDURATIONS]
inner join [PROSPECTPLANSTATUSCODE] as p on [STAGEDURATIONS].[PROSPECTPLANSTATUSCODEID] = p.[ID]
where [FIRSTOCCURRENCEROWINSTAGE] = 1
group by p.[DESCRIPTION]
```

For the full `CREATE PROCEDURE` code to be used in the Report Spec, see Nonconsecutive Perspective Stored Procedure on page 41.

Overlapping Perspective Stored Procedure

```
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select i.[PROSPECTPLANSTATUSCODEID],
               i.[ACTUALENDDATETIME],
               min(i.[ACTUALSTARTDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [FIRSTSTEPINSTAGEDATETIME],
               max(i.[ACTUALENDDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [LASTSTEPINSTAGEDATETIME]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
               and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       )
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [AVGSTAGEDURATION],
       min(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MINSTAGEDURATION],
       max(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MAXSTAGEDURATION]
from [STEPS] as s
inner join [PROSPECTPLANSTATUSCODE] as p on s.[PROSPECTPLANSTATUSCODEID] = p.[ID]
where (s.[ACTUALENDDATETIME] = s.[LASTSTEPINSTAGEDATETIME])
group by p.[DESCRIPTION]
```

Consecutive Perspective Stored Procedure

```
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSCONSECUTIVE (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select row_number() over (
                      order by i.[PROSPECTPLANID],
                              i.[ACTUALSTARTDATETIME]
                      ) as [ALLSTEPSEQUENCENUMBER],
              i.[ACTUALSTARTDATETIME],
              i.[ACTUALENDDATETIME],
              i.[PROSPECTPLANID],
              max(i.[ACTUALENDDATETIME]) over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
              i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
              and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       ),
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
              s.[ACTUALENDDATETIME],
              r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
              s.[PROSPECTPLANSTATUSCODEID],
              r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
              s.[PROSPECTPLANID],
              s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.[ALLSTEPSEQUENCENUMBER]
       ),
[STAGEOCCURRENCES]
as (
       select row_number() over (
                      order by sofp.[PROSPECTPLANID],
                              sofp.[ACTUALSTARTDATETIME]
                      ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
              sofp.[ACTUALSTARTDATETIME],
                sofp.[ACTUALENDDATETIME],
                sofp.[LASTSTEPINPLANENDDATETIME],
                sofp.[PROSPECTPLANID],
                sofp.[PROSPECTPLANSTATUSCODEID]
        from [STAGEOCCURRENCESFIRSTPASS] as sofp
        where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
                or (
                        (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                        and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                        )
        ),
[STAGEOCCURRENCEDURATIONS]
as (
        select so1.[ACTUALSTARTDATETIME] as [STARTDATETIME],
                so1.[ACTUALENDDATETIME] as [ENDDATETIME],
                "STAGEOCCURRENCEDURATION" = case
                        when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                                then cast(so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                        when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
                                then cast(so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                        end,
                so1.[PROSPECTPLANID],
                so1.[PROSPECTPLANSTATUSCODEID]
        from [STAGEOCCURRENCES] as so1
        left join [STAGEOCCURRENCES] as so2 on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.A-
LLSTAGEOCCURRENCESSEQUENCENUMBER
        )
select p.[DESCRIPTION] as [STAGENAME],
        avg(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [AVGSTAGEOCCURRENCEDURATION],
        min(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [MINSTAGEOCCURRENCEDURATION],
        max(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [MAXSTAGEOCCURRENCEDURATION],
        cast(count([STAGEOCCURRENCEDURATIONS].[PROSPECTPLANID]) as float) / cast(count(distinct ([STAGEOCCURRENCEDURATIONS].
[PROSPECTPLANID])) as float) as [AVGTIMESINSTAGE]
from [STAGEOCCURRENCEDURATIONS]
inner join [PROSPECTPLANSTATUSCODE] as p on [STAGEOCCURRENCEDURATIONS].[PROSPECTPLANSTATUSCODEID] = p.[ID]
group by p.[DESCRIPTION]
```

Nonconsecutive Perspective Stored Procedure

```
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSNONCONSECUTIVE (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select row_number() over (
                       order by i.[PROSPECTPLANID],
                               i.[ACTUALSTARTDATETIME]
                       ) as [ALLSTEPSEQUENCENUMBER],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALENDDATETIME],
               i.[PROSPECTPLANID],
               max(i.[ACTUALENDDATETIME]) over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
               i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
               and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       ),
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
               s.[ACTUALENDDATETIME],
               r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
               s.[PROSPECTPLANSTATUSCODEID],
               r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
               s.[PROSPECTPLANID],
               s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.[ALLSTEPSEQUENCENUMBER]
       ),
[STAGEOCCURRENCES]
as (
       select row_number() over (
                       order by sofp.[PROSPECTPLANID],
                               sofp.[ACTUALSTARTDATETIME]
                       ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
               sofp.[ACTUALSTARTDATETIME],
               sofp.[ACTUALENDDATETIME],
               sofp.[LASTSTEPINPLANENDDATETIME],
               sofp.[PROSPECTPLANID],
               sofp.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCESFIRSTPASS] as sofp
       where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
               or (
                       (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                       and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                       )
       ),
[STAGEOCCURRENCEDURATIONS]
as (
       select so1.[ACTUALSTARTDATETIME],
               so1.[ACTUALENDDATETIME],
               "STAGEOCCURRENCEDURATION" = case
                       when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                               then cast(so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
                               then CAST(so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       end,
               so1.[PROSPECTPLANSTATUSCODEID],
               so1.[PROSPECTPLANID]
       from [STAGEOCCURRENCES] as so1
       left join [STAGEOCCURRENCES] as so2 on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.A-
LLSTAGEOCCURRENCESSEQUENCENUMBER
       ),
[STAGEDURATIONS]
as (
       select sum(cast(sod.[STAGEOCCURRENCEDURATION] as float)) over (
                       partition by sod.[PROSPECTPLANID],
                       sod.[PROSPECTPLANSTATUSCODEID]
                       ) as [STAGEDURATION],
               RANK() over (
                       partition by sod.[PROSPECTPLANID],
                       sod.[PROSPECTPLANSTATUSCODEID] order by sod.[ACTUALSTARTDATETIME]
                       ) as [FIRSTOCCURRENCEROWINSTAGE],
               sod.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCEDURATIONS] as sod
       )
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [AVGSTAGEDURATION],
       min(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [MINSTAGEDURATION],
       max(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [MAXSTAGEDURATION]
from [STAGEDURATIONS]
inner join [PROSPECTPLANSTATUSCODE] as p on [STAGEDURATIONS].[PROSPECTPLANSTATUSCODEID] = p.[ID]
where [FIRSTOCCURRENCEROWINSTAGE] = 1
group by p.[DESCRIPTION]
```

Wiring up the Report

Within these topics we will look at how to create the report (`RDL`) file to use the query we created in Creating an OLTP Version on page 27. We will create a Blackbaud Infinity Report Spec to reference the `RDL` file, a Page Spec to define a page to display the report, and a Task Spec to define a link to appear in a functional area to open the page that displays the report. We will also create a Package Spec to make it easier to add those items to the catalog. Finally, we will load everything into the application and see the report displayed.

Create an RDL File

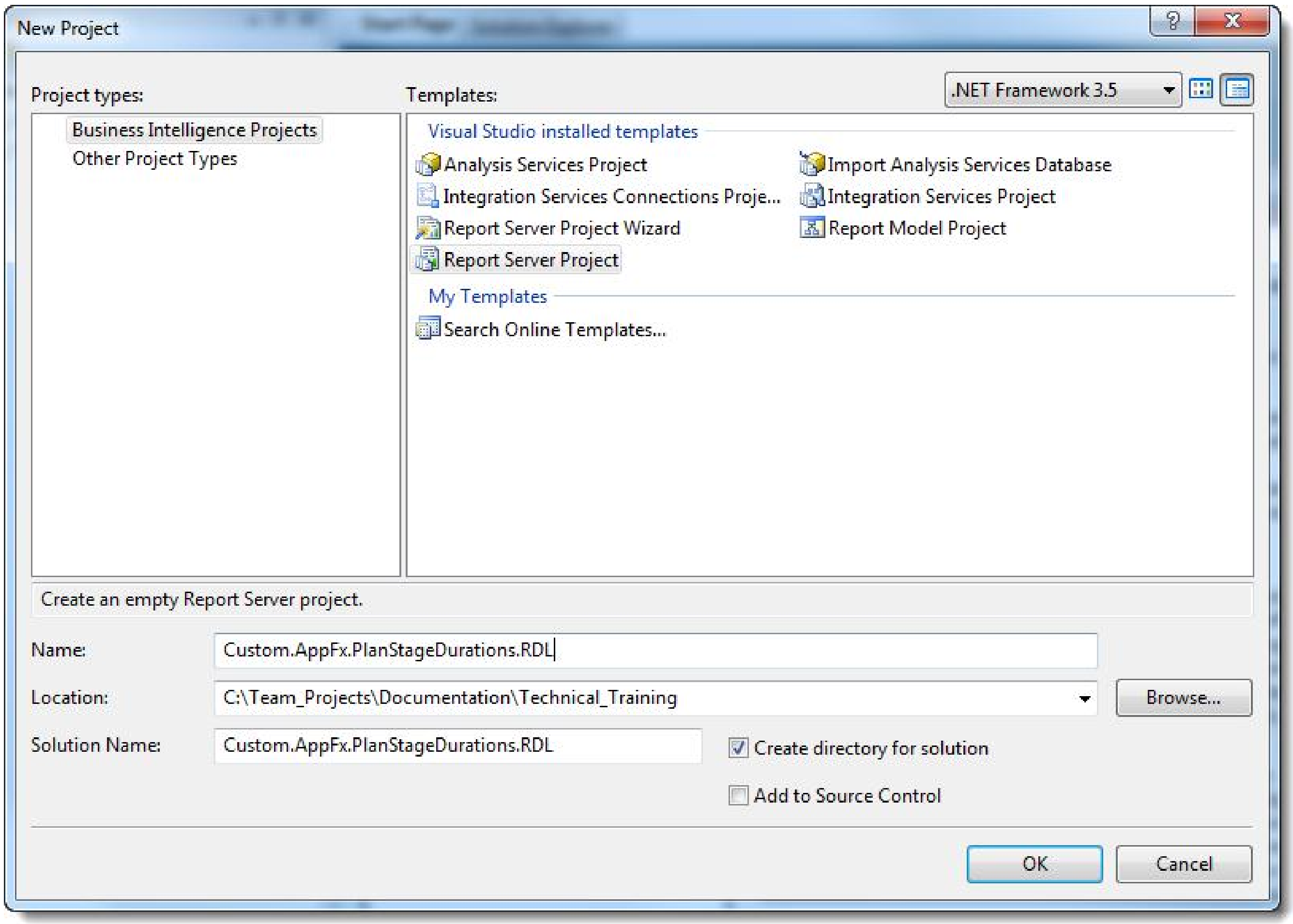
Note: These steps describe how to create the file and add the needed parameters, embedded data source, and embedded data set for the OLTP version of the report. The steps are similar for the data warehouse versions. But the stored procedures executed by the datasets in those versions are located in the data warehouse database. However, the data warehouse versions still access the OLTP database to populate the available values for the prospect plan type parameter. This could also be done from the data warehouse. But as of writing, that would require an additional extension to the data warehouse.

1. Open ***Visual Studio 2008*** with Business Intelligence Development Studio functionality installed.

   For more information, see Microsoft's MSDN article at Introducing Business Intelligence Development Studio.

2. To create a new Report Server project, click **File** > **New** > **Project**. The New Project screen appears.

   Otherwise skip this and the next step and open your existing Report Server project.


New Project
Project types:
Business Intelligence Projects
Other Project Types
Templates:
.NET Framework 3.5
Visual Studio installed templates
Analysis Services Project
Import Analysis Services Database
Integration Services Connections Proje...
Integration Services Project
Report Server Project Wizard
Report Model Project
Report Server Project
My Templates
Search Online Templates...
Browse...
OK
Cancel

Create an empty Report Server project.

Name: Custom.AppFx.PlanStageDurations.RDL

Location: C:\Team_Projects\Documentation\Technical_Training

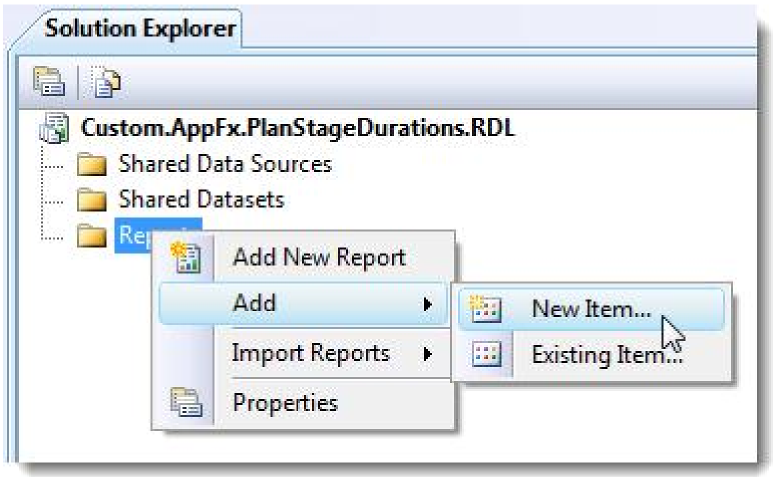
Solution Name: Custom.AppFx.PlanStageDurations.RDL

[✓] Create directory for solution

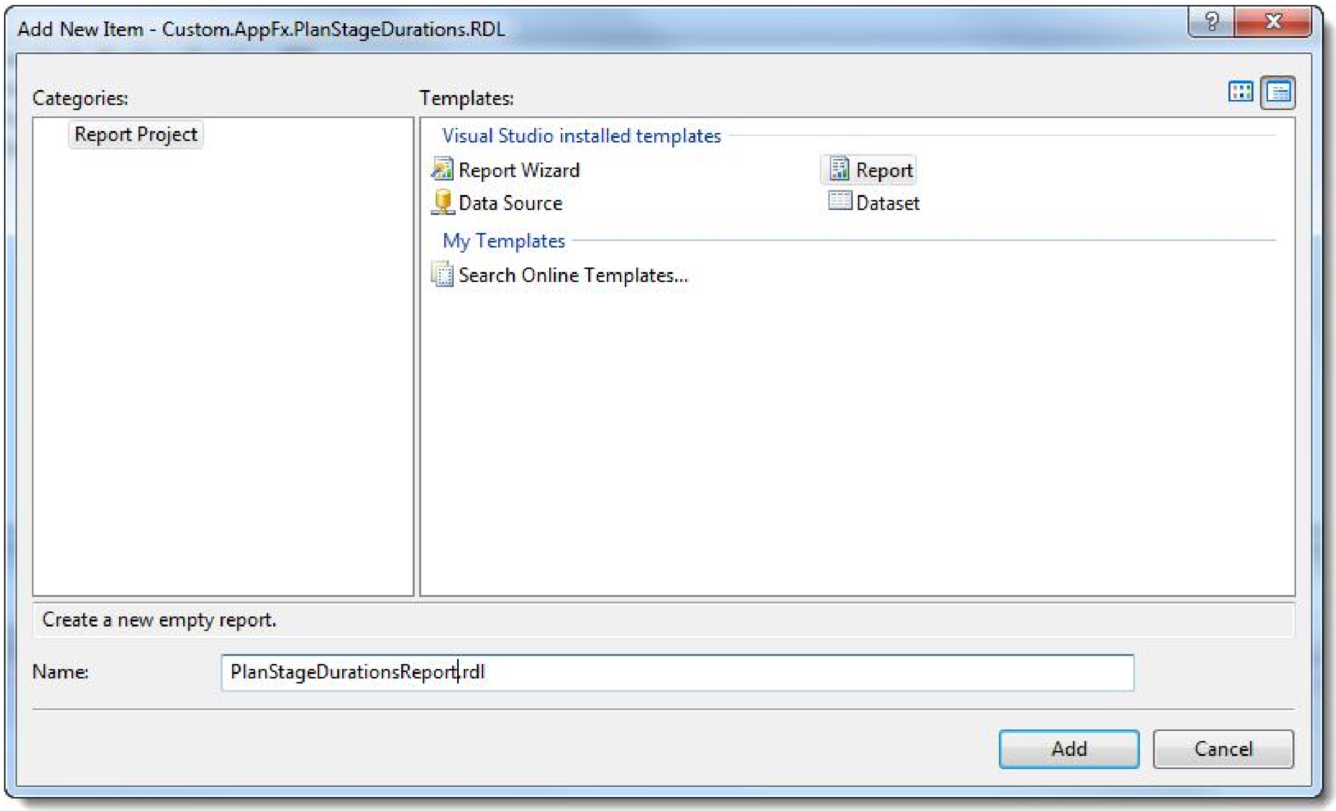
[ ] Add to Source Control

3. Select the **Report Server Project** template from the **Business Intelligence Projects** type templates, enter a name, and click **OK**. The new project appears.

4. From Solution Explorer, right-click **Reports** and select **Add** > **New Item**.



The Add New Item screen appears.

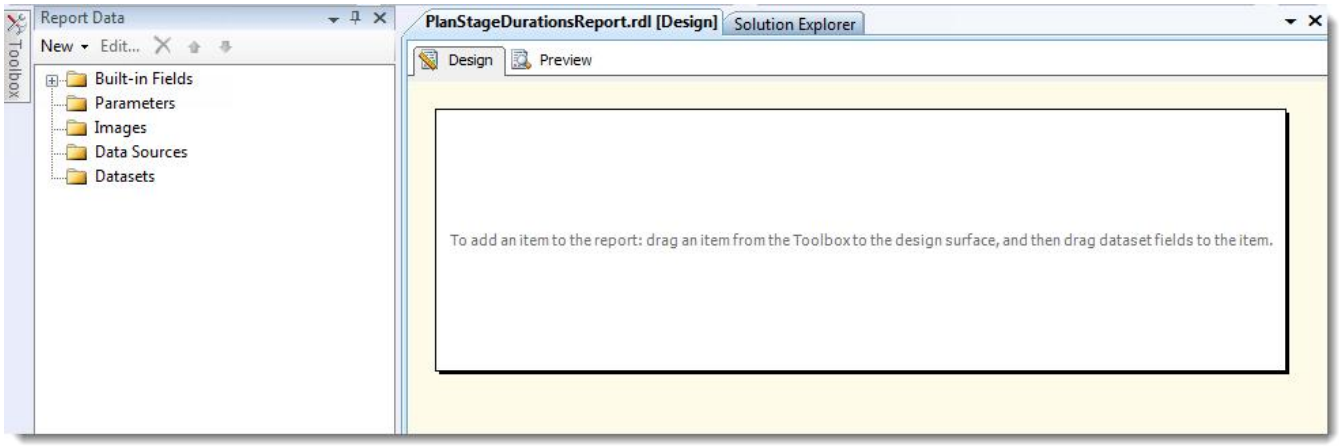

Add New Item - Custom.AppFx.PlanStageDurations.RDL
Categories:
Report Project
Templates:
Visual Studio installed templates
Report Wizard
Report
Data Source
Dataset
My Templates
Search Online Templates...
Add
Cancel

Create a new empty report.

Name: PlanStageDurationsReport.rdl

5. From **Categories**, select **Report Project** and from **Templates** select **Report**.
6. Enter a name and click **Add**.

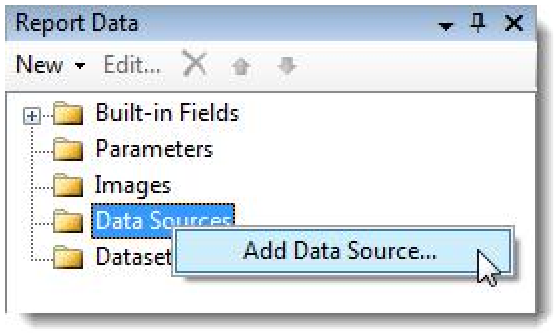
   The report appears in the designer.



To add an item to the report: drag an item from the Toolbox to the design surface, and then drag dataset fields to the item.

7. Connect the RDL to your Blackbaud Infinity OLTP database.

   From the Report Data window, right-click **Data Sources** and select **Add Data Source**.



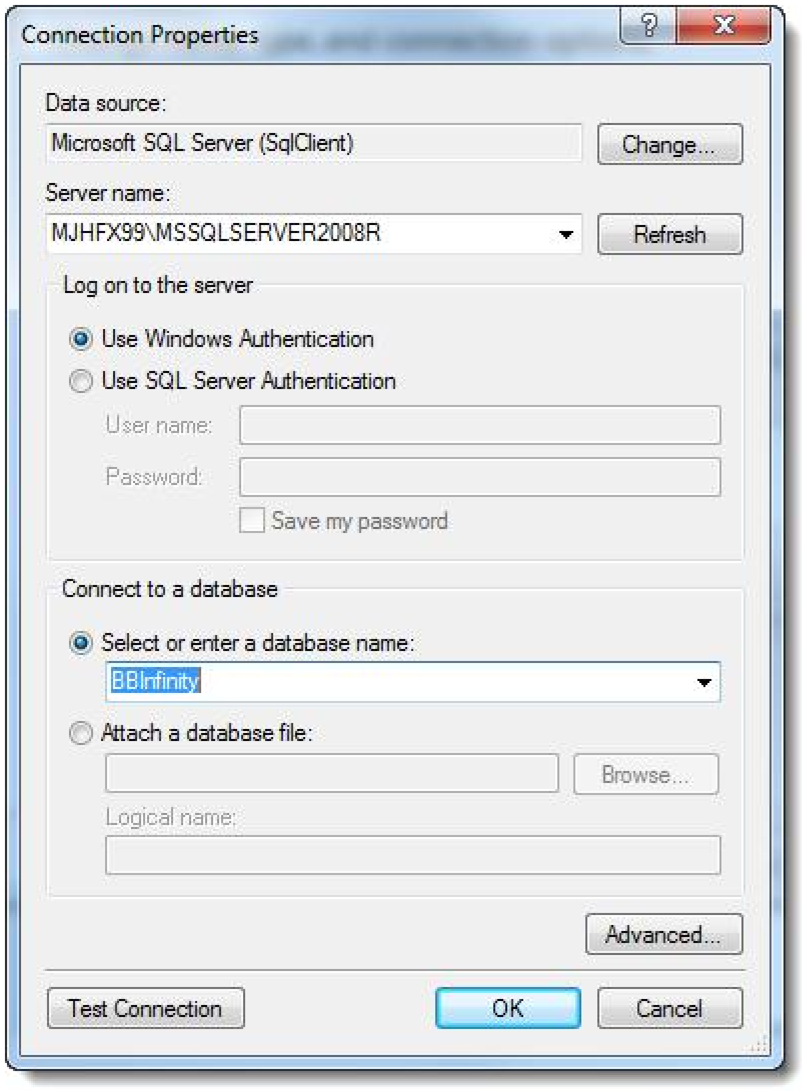
8. The Data Source Properties screen appears. Enter a name such as `BBInfinity`.

9. Select **Embedded connection** and from **Type**, select **Microsoft SQL Server**.

10. Click the **Edit** button next to **Connection string**. The Connection Properties screen appears.

11. From **Server Name**, select the name of the server which hosts your OLTP database.

12. From **Connect to a database** > **Select or enter a database name**, select the name of your OLTP database such as `BBInfinity`.



Data source: Microsoft SQL Server (SqlClient)

Server name: MJHFX99\MSSQLSERVER2008R

Log on to the server

(●) Use Windows Authentication

( ) Use SQL Server Authentication

User name:

Password:

[ ] Save my password

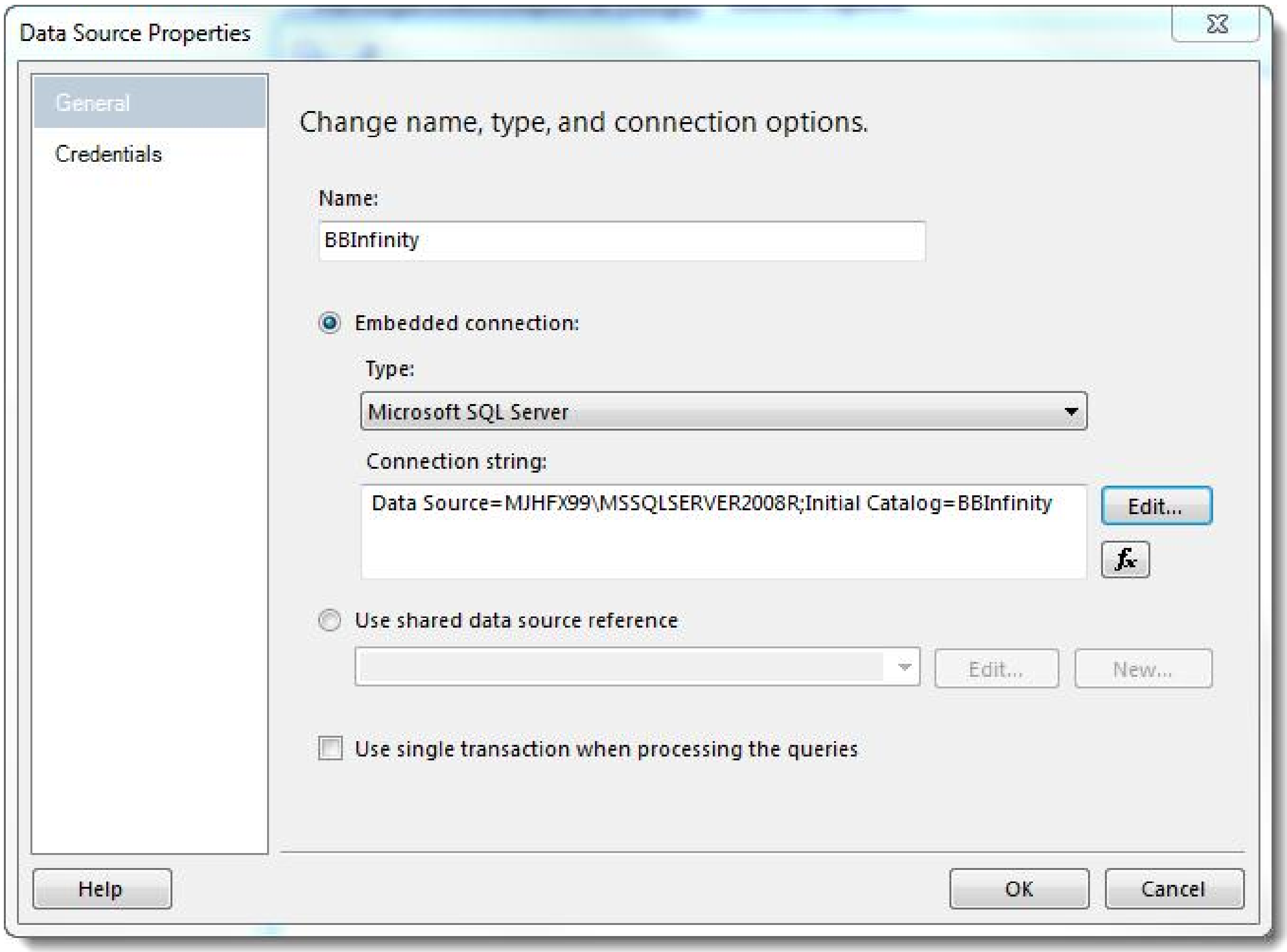
Connect to a database

(●) Select or enter a database name: BBInfinity

( ) Attach a database file:

Logical name:

13. Click **OK**. You return to the Data Source Properties screen.


Data Source Properties

Change name, type, and connection options.

Name: BBInfinity

[✓ Embedded connection:]

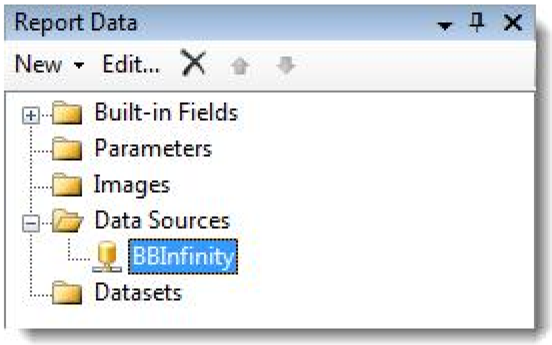
Type: Microsoft SQL Server

Connection string: Data Source=MJHFX99\MSSQLSERVER2008R;Initial Catalog=BBInfinity

Use shared data source reference

Use single transaction when processing the queries

14. Click **OK**. The Data Source is added to the RDL file.



15. From **Report Data**, create a data set from your Blackbaud Infinity OLTP database. Right-click **Datasets** and select **Add Dataset**. The Dataset Properties screen appears.

16. The goal is to create a dataset for each stored procedure in the report.

Enter a name such as `USR_USP_REPORT_PLANSTAGEDURATIONSCONSECUTIVE` and select **Use a dataset embedded in my report**.

17. From **Data source**, select the data source you created for the OLTP database.

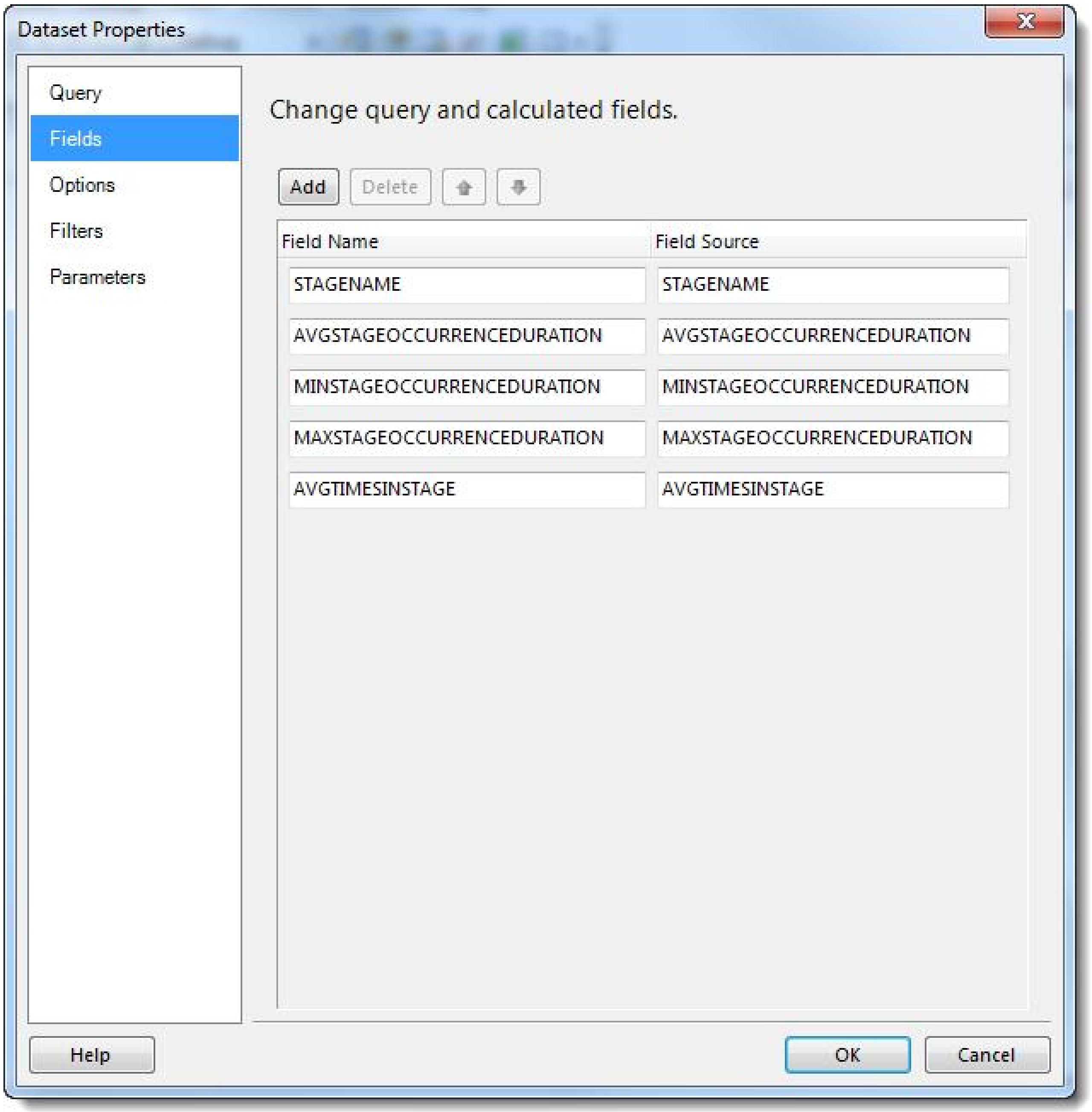
18. From **Query type**, select **Stored Procedure**.

19. From **Select or enter stored procedure name**, select the name of the stored procedure in your Report Spec.

    Note: If you create the RDL file first, you can temporarily add the stored procedure to your development database with SQL Server Management Studio. If you create the Report Spec first, you load the spec with `LoadSpec`. This way when you select the stored procedure, the fields and parameters will be rec-ognized by the dataset.

20. Click **Refresh Fields**. If the stored procedure exists in the database in your connection, the other tabs of the Dataset Properties screen will be updated. This is easier than filling those out manually.

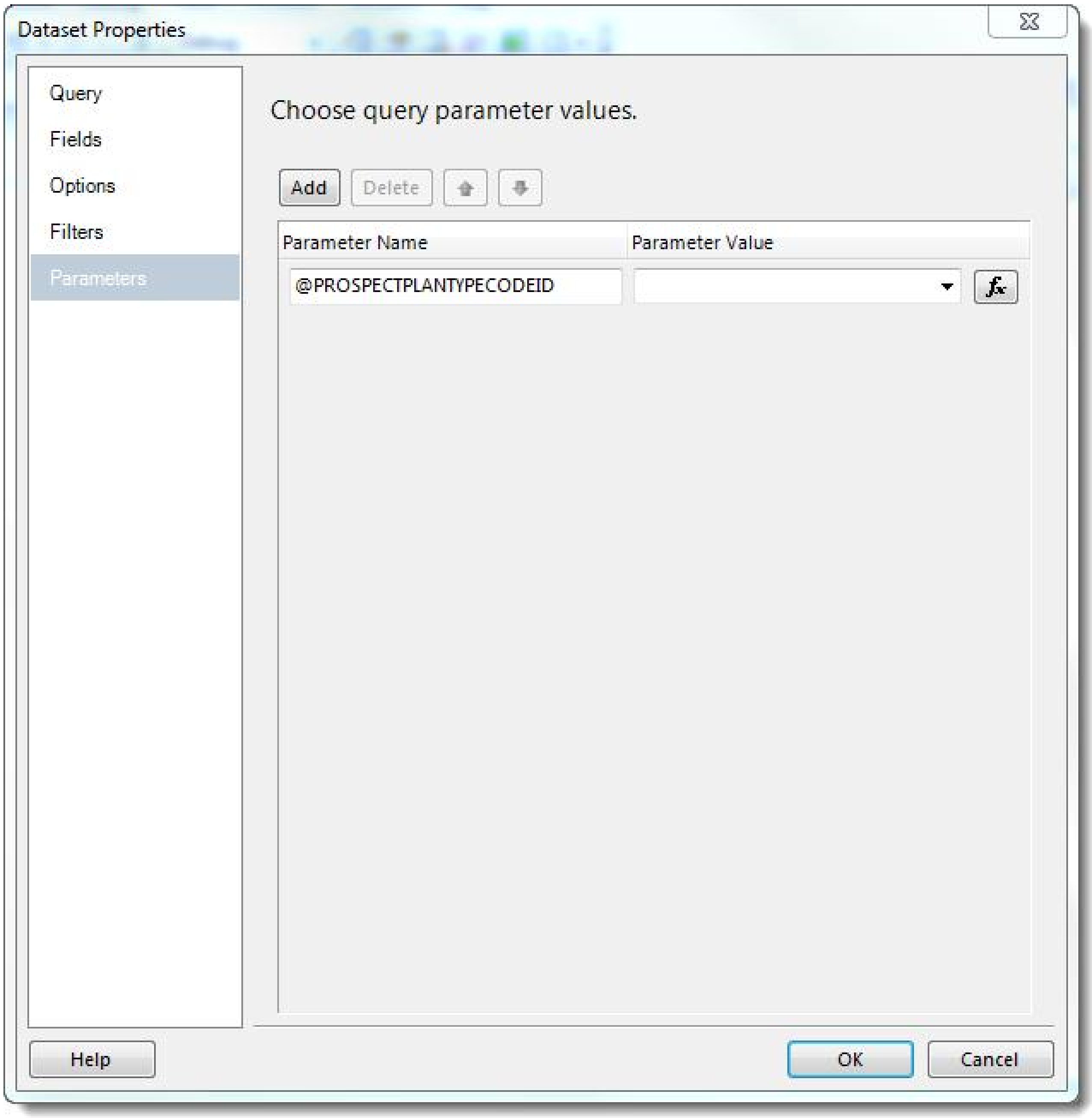
21. Open the Fields tab of the Dataset Properties screen and confirm the fields were found.



Change query and calculated fields.

Field Name	Field Source
STAGENAME	STAGENAME
AVGSTAGEOCCURRENCEDURATION	AVGSTAGEOCCURRENCEDURATION
MINSTAGEOCCURRENCEDURATION	MINSTAGEOCCURRENCEDURATION
MAXSTAGEOCCURRENCEDURATION	MAXSTAGEOCCURRENCEDURATION
AVGTIMESINSTAGE	AVGTIMESINSTAGE

22. Click the Parameters tab of the Dataset Properties screen.



Choose query parameter values.

Parameter Name	Parameter Value
@PROSPECTPLANTYPECODEID	

23. There is no report parameter to map the dataset parameter to yet. Click **OK**. ***Visual Studio*** will probably recognize the discrepancy and add the parameter. If not, right-click **Parameters** and add the `PROS-PECTPLANTYPECODEID` parameter.

24. We need to populate the available values for the `PROSPECTPLANTYPECODEID` parameter. Add a new dataset for that. Right-click **Datasets** and select **Add Dataset**.

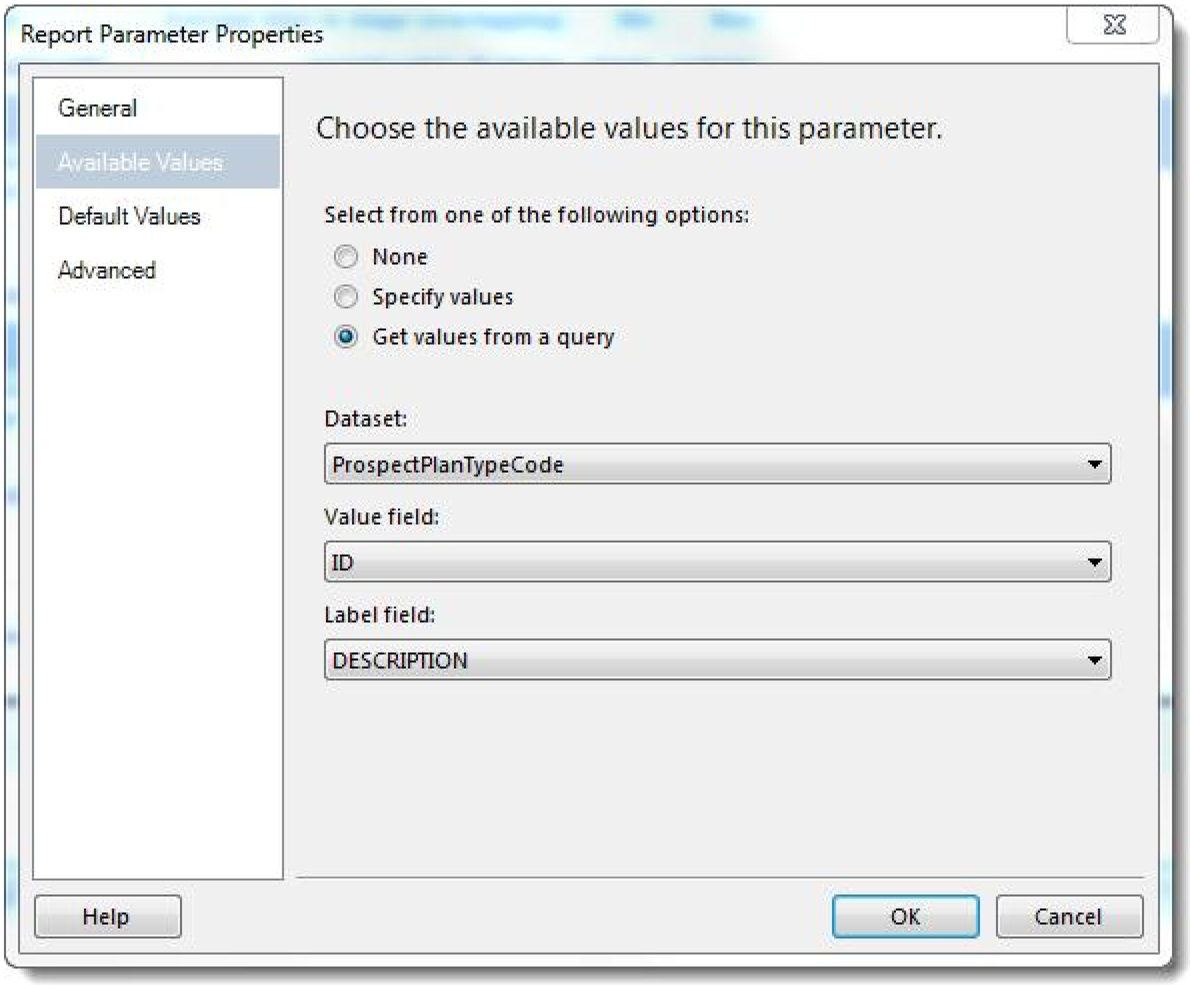
25. For **Name**, enter `ProspectPlanTypeCode`.

26. Select User a dataset embedded in my report.

27. Select the OLTP data source you created.
28. Select **Query type** > **Text**.
29. In the **Query** field, enter:

```
select [ID], [DESCRIPTION] from [PROSPECTPLANTYPECODE]
```

30. Click **OK**.
31. Return to the Parameter Properties screen for the `PROSPECTPLANTYPECODEID` parameter.
32. From Available Values, select **Get values from a query**.
33. From **Dataset**, select **ProspectPlanTypeCode**.
34. From **Value** field, select **ID**.
35. From **Label** field, select **DESCRIPTION**.



Choose the available values for this parameter.

Select from one of the following options:

None
Specify values
[✓ Get values from a query]

Dataset: ProspectPlanTypeCode

Value field: ID

Label field: DESCRIPTION

36. Click **OK**.

37. Return to the Parameters tab on the Dataset Properties screen for the stored procedure dataset.
38. You can now select a parameter value for the parameter. But click the function button next to the **Parameter Value** field. The Expression screen appears.
39. Enter this expression.

```
=Parameters!PROSPECTPLANTYPECODEID.Value
```

40. Click **OK** twice to exit those screens.
41. Right-click the `PROSPECTPLANTYPECODEID` parameter and select **Parameter Properties**.
42. Create datasets for the other two stored procedures:

```
USR_USP_REPORT_PLANSTAGEDURATIONSNONCONSECUTIVE
USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING
```

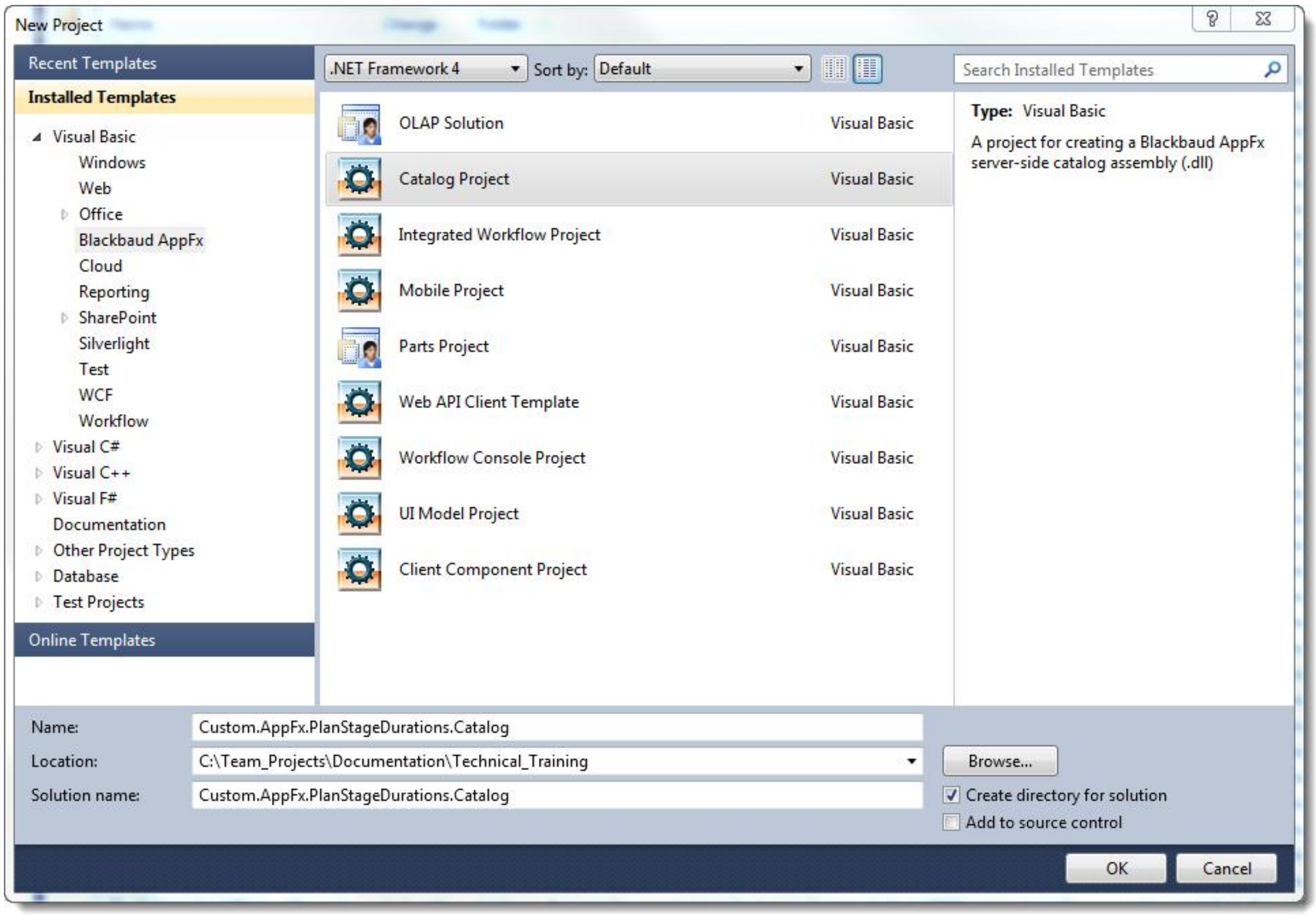
43. Create a parameter for user ID. Right-click **Parameters** and select **Add Parameter**.
44. From **Name**, enter `ALTREPORTUSERID` and from **Prompt** enter `Alt Report UserID`.
45. From the Default Values tab, select **Specify values**.
46. Click the function button next to the **Value** field. The Expression screen appears.
47. From the **Set expression for: Value** field, enter:

```
=User!UserID
```

48. Click **OK** twice.
49. Save the RDL file.

Create a Report Spec

1. To create a new Catalog Project for Blackbaud Infinity development, from ***Visual Studio*** with the Blackbaud Infinity SDK installed, click **File** > **New** > **Project**. Then from the New Project screen, click Catalog Project, enter a name and click **OK**. Otherwise, open your existing Catalog Project.


New Project

Recent Templates
Installed Templates
Visual Basic
Windows
Web
Office
Blackbaud AppFx
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Visual C#
Visual C++
Visual F#
Documentation
Other Project Types
Database
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
OLAP Solution Visual Basic
Catalog Project Visual Basic
Integrated Workflow Project Visual Basic
Mobile Project Visual Basic
Parts Project Visual Basic
Web API Client Template Visual Basic
Workflow Console Project Visual Basic
UI Model Project Visual Basic
Client Component Project Visual Basic
Browse...
OK
Cancel

Type: Visual Basic

A project for creating a Blackbaud AppFx server-side catalog assembly (.dll)

Name: Custom.AppFx.PlanStageDurations.Catalog

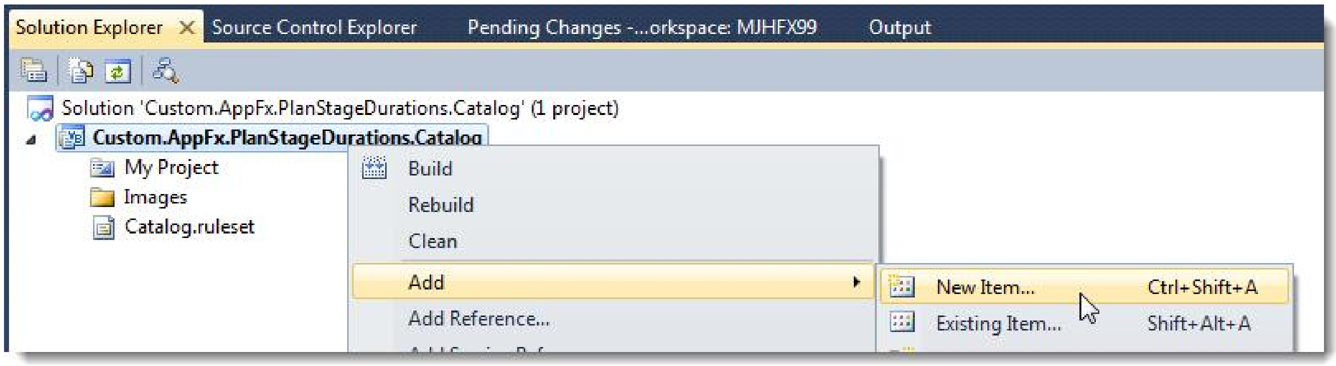
Location: C:\Team_Projects\Documentation\Technical_Training

Solution name: Custom.AppFx.PlanStageDurations.Catalog

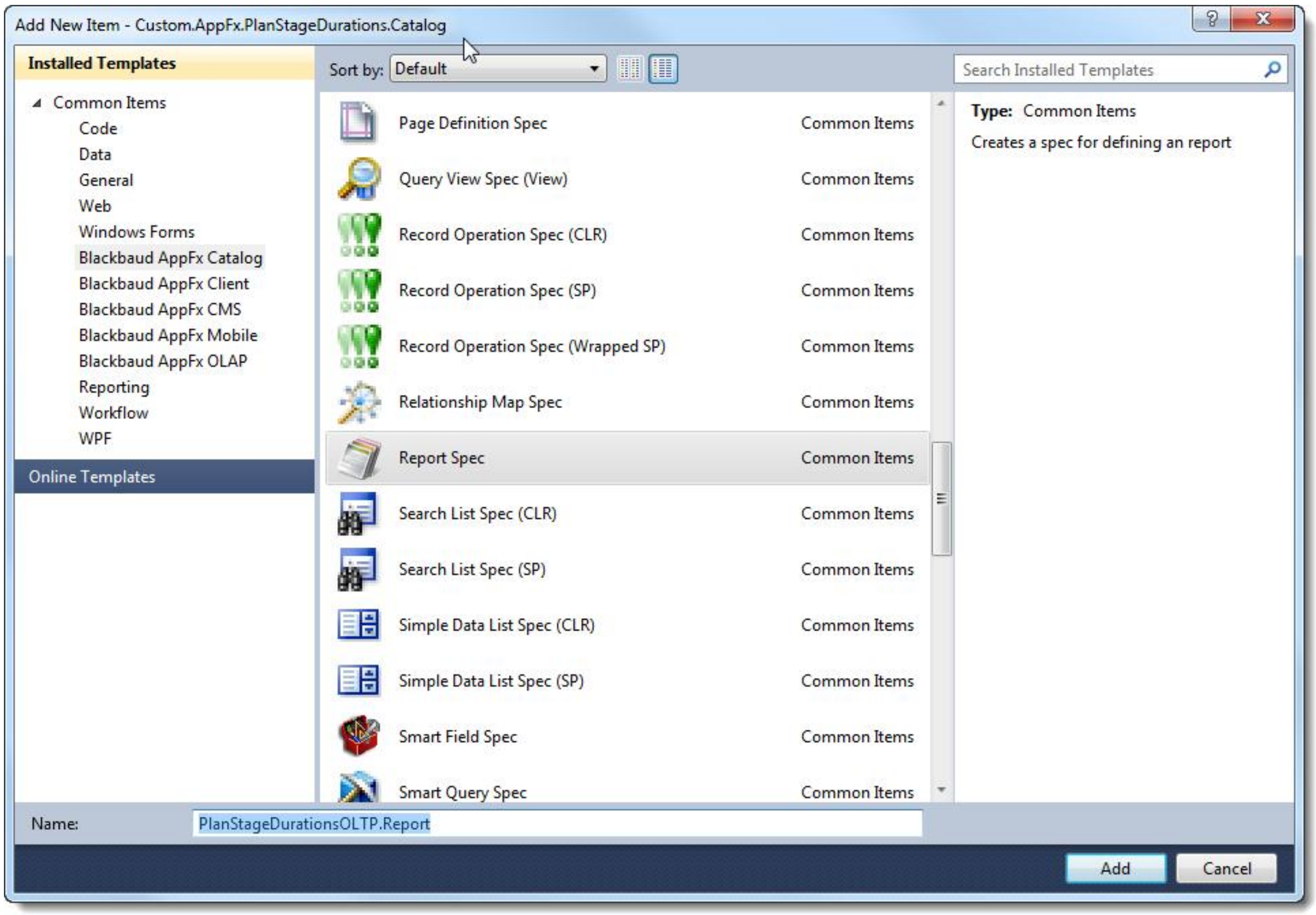
[✓] Create directory for solution

[ ] Add to source control

2. From the project node in the Solution Explorer, click **Add** > **New Item**.


Solution Explorer
Source Control Explorer
Pending Changes -...orkspace: MJHFX99
Output
Solution 'Custom.AppFx.PlanStageDurations.Catalog' (1 project)
Custom.AppFx.PlanStageDurations.Catalog
My Project
Images
Catalog.ruleset
Build
Rebuild
Clean
Add
Add Reference...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A

3. From the Add New Item screen, click **Installed Templates** > **Common Items** > **Blackbaud AppFx Catalog** > **Report Spec**.


Add New Item - Custom.AppFx.PlanStageDurations.Catalog

Installed Templates

Common Items
- Code
- Data
- General
- Web
- Windows Forms
- Blackbaud AppFx Catalog
- Blackbaud AppFx Client
- Blackbaud AppFx CMS
- Blackbaud AppFx Mobile
- Blackbaud AppFx OLAP
- Reporting
- Workflow
- WPF

Online Templates

Sort by: Default

Page Definition Spec	Common Items
Query View Spec (View)	Common Items
Record Operation Spec (CLR)	Common Items
Record Operation Spec (SP)	Common Items
Record Operation Spec (Wrapped SP)	Common Items
Relationship Map Spec	Common Items
Report Spec	Common Items
Search List Spec (CLR)	Common Items
Search List Spec (SP)	Common Items
Simple Data List Spec (CLR)	Common Items
Simple Data List Spec (SP)	Common Items
Smart Field Spec	Common Items
Smart Query Spec	Common Items

Search Installed Templates

Type: Common Items

Creates a spec for defining an report

Name: PlanStageDurationsOLTP.Report

The Report Spec appears.

```
<ReportSpec
       xmlns="bb_appfx_report"
       xmlns:common="bb_appfx_commontypes"
       ID="882fe807-2570-4900-9c89-0861070f7ea5"
       Name="PlanStageDurationsOLTP Report"
       Description="REPLACE_WITH_DESCRIPTION"
       Author="Blackbaud Product Development"
       >

       <RDLFileName>PlanStageDurationsOLTP.rdl</RDLFileName>
       <Folder>System Reports/Misc Reports</Folder>

       <DataRetrieval>
               <CreateSQL ObjectName="dbo.USP_REPORT_xxx" ObjectType="SQLStoredProc">
                       <![CDATA[
create procedure dbo.USP_REPORT_xxx
(
<list any report parameters here>
)
as
       <build the report SQL here>
                       ]]>
               </CreateSQL>
```

```
        </DataRetrieval>

    </ReportSpec>
```

4. Adjust this information:

   Name: `Prospect Plan Stage Durations Report (OLTP Version)`

   Description: `Displays the averages, minimums, and maximums of durations of plan stages and stage occurrences. Also displays an average count of stage occurrences.`

   Author: `Technical Training`

   RDLFileName: `Cus-tom.AppFx.PlanStageDurations.Catalog.PlanStageDurationsReport.rdl`

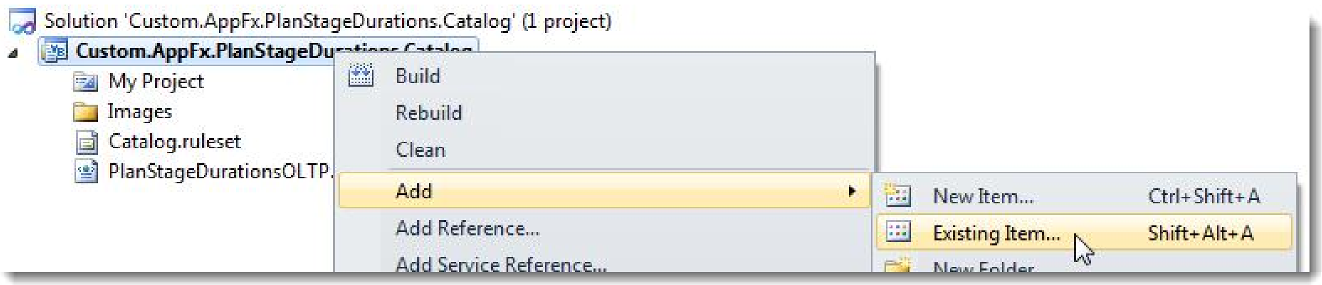
   DataRetrieval: See the code sample at Report Spec for OLTP Version on page 59.

5. Save the Report Spec.

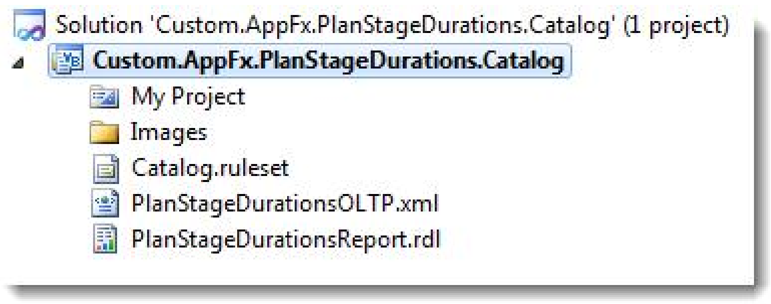
Note: At this point, if you attempt to run `LoadSpec` to test the spec, `LoadSpec` will throw an error because the `RDL` file is not available. Message shown for a different version:

```
Connecting to database 'BBInfinity' on server 'MJHFX99\MSSQLSERVER2008R'.
Loading ...C:\Users\TomTr\Documents\Visual Studio 2010\Proje-
cts\M-
ajorGivingPlanStageDurations\MajorGivingPlanStageDurations\MajorGivingPlanStageDurations.Report.xml
Uploading ReportSpec 'Plan Stage Durations - Averages Report' to
catalog...
Uploading input file C:\Users\TomTr\Documents\Visual Studio 2010\Proje-
cts\M-
ajorGivingPlanStageDurations\MajorGivingPlanStageDurations\MajorGivingPlanStageDurations.Report.xml
Error.
The specified report definition, "C:\Users\TomTr\Documents\Visual Studio
2010\Proje-
cts\M-
ajorGivingPlanStageDurations\MajorGivingPlanStageDurations\MajorGivingPlanStageDurations.rdl"
 could not be located.
Upload complete.
```

6. Add the RDL file created in Create an RDL File on page 44 to the project as an embedded resource. From Solution Explorer, right-click the project and click **Add** > **Existing Item**.

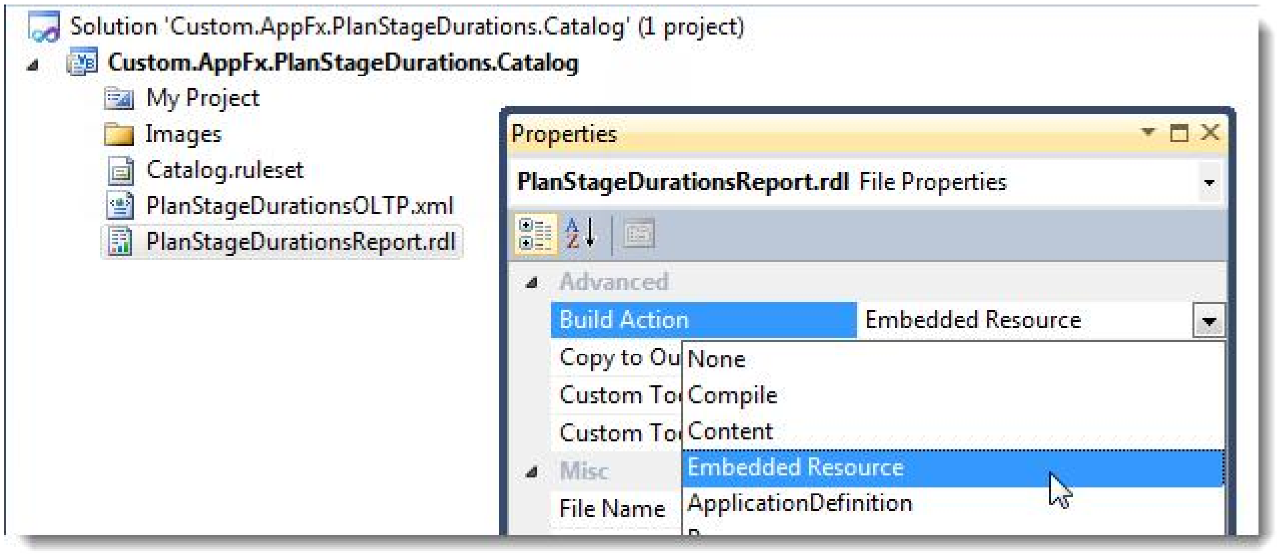


7. Browse to the `RDL` file and click **Add**.


Solution 'Custom.AppFx.PlanStageDurations.Catalog' (1 project)
Custom.AppFx.PlanStageDurations.Catalog
My Project
Images
Catalog.ruleset
PlanStageDurationsOLTP.xml
PlanStageDurationsReport.rdl

8. Right-click the `RDL` file and select **Properties**.

9. From **Build Action**, select **Embedded Resource**.


Solution 'Custom.AppFx.PlanStageDurations.Catalog' (1 project)
Custom.AppFx.PlanStageDurations.Catalog
My Project
Images
Catalog.ruleset
PlanStageDurationsOLTP.xml
PlanStageDurationsReport.rdl
Properties
PlanStageDurationsReport.rdl File Properties
Advanced
Build Action
Embedded Resource
Copy to Ou
Custom To
Custom To
Misc
File Name
None
Compile
Content
Embedded Resource
ApplicationDefinition

10. Save the project.

    As you update the RDL file in your Report Server project, you will have to update this version of the RDL. You could alternately update this file from ReportBuilder 2.0, Business Intelligence Development Studio outside of the context of the project, or with an XML editor.

Report Spec for OLTP Version

Note: This version includes stored procedure parameters and UIModel items described elsewhere in the documentation.

```
<ReportSpec
    xmlns="bb_appfx_report"
    xmlns:common="bb_appfx_commontypes"
    ID="882fe807-2570-4900-9c89-0861070f7ea5"
    Name="Prospect Plan Stage Durations Report (OLTP Version)"
    Description="Displays the averages, minimums, and maximums of durations of plan stages and stage occurrences. Also
displays an average count of stage occurrences."
    Author="Technical Training"
    >

       <RDLFileName>Custom.AppFx.PlanStageDurations.Catalog.PlanStageDurationsReport.rdl</RDLFileName>
       <Folder>Custom Reports/Misc Reports</Folder>

       <DataRetrieval>
               <CreateSQL ObjectName="dbo.USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING" ObjectType="SQLStoredProc">
                       <![CDATA[
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select i.[PROSPECTPLANSTATUSCODEID],
               i.[ACTUALENDDATETIME],
               min(i.[ACTUALSTARTDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [FIRSTSTEPINSTAGEDATETIME],
               max(i.[ACTUALENDDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [LASTSTEPINSTAGEDATETIME]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
               and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       )
```

```
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [AVGSTAGEDURATION],
       min(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MINSTAGEDURATION],
       max(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MAXSTAGEDURATION]
from [STEPS] as s
inner join [PROSPECTPLANSTATUSCODE] as p on s.[PROSPECTPLANSTATUSCODEID] = p.[ID]
where (s.[ACTUALENDDATETIME] = s.[LASTSTEPINSTAGEDATETIME])
group by p.[DESCRIPTION]
                       ]]>
               </CreateSQL>

    <CreateSQL ObjectName="dbo.USR_USP_REPORT_PLANSTAGEDURATIONSCONSECUTIVE" ObjectType="SQLStoredProc">
      <![CDATA[
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSCONSECUTIVE (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select row_number() over (
                       order by i.[PROSPECTPLANID],
                               i.[ACTUALSTARTDATETIME]
                       ) as [ALLSTEPSEQUENCENUMBER],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALENDDATETIME],
               i.[PROSPECTPLANID],
               max(i.[ACTUALENDDATETIME]) over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
               i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
               and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       ),
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
               s.[ACTUALENDDATETIME],
               r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
               s.[PROSPECTPLANSTATUSCODEID],
               r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
               s.[PROSPECTPLANID],
               s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.[ALLSTEPSEQUENCENUMBER]
       ),
[STAGEOCCURRENCES]
as (
       select row_number() over (
                       order by sofp.[PROSPECTPLANID],
                               sofp.[ACTUALSTARTDATETIME]
                       ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
               sofp.[ACTUALSTARTDATETIME],
               sofp.[ACTUALENDDATETIME],
               sofp.[LASTSTEPINPLANENDDATETIME],
               sofp.[PROSPECTPLANID],
               sofp.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCESFIRSTPASS] as sofp
       where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
               or (
                       (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                       and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                       )
       ),
[STAGEOCCURRENCEDURATIONS]
as (
       select so1.[ACTUALSTARTDATETIME] as [STARTDATETIME],
               so1.[ACTUALENDDATETIME] as [ENDDATETIME],
               "STAGEOCCURRENCEDURATION" = case
                       when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                               then cast(so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
                               then cast(so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       end,
               so1.[PROSPECTPLANID],
               so1.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCES] as so1
       left join [STAGEOCCURRENCES] as so2 on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.A-
LLSTAGEOCCURRENCESSEQUENCENUMBER
       )
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [AVGSTAGEOCCURRENCEDURATION],
       min(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [MINSTAGEOCCURRENCEDURATION],
       max(cast([STAGEOCCURRENCEDURATIONS].[STAGEOCCURRENCEDURATION] as float)) as [MAXSTAGEOCCURRENCEDURATION],
       cast(count([STAGEOCCURRENCEDURATIONS].[PROSPECTPLANID]) as float) / cast(count(distinct ([STAGEOCCURRENCEDURATIONS].
[PROSPECTPLANID])) as float) as [AVGTIMESINSTAGE]
from [STAGEOCCURRENCEDURATIONS]
inner join [PROSPECTPLANSTATUSCODE] as p on [STAGEOCCURRENCEDURATIONS].[PROSPECTPLANSTATUSCODEID] = p.[ID]
group by p.[DESCRIPTION]
      ]]>
    </CreateSQL>

    <CreateSQL ObjectName="dbo.USR_USP_REPORT_PLANSTAGEDURATIONSNONCONSECUTIVE" ObjectType="SQLStoredProc">
      <![CDATA[
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSNONCONSECUTIVE (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select row_number() over (
                       order by i.[PROSPECTPLANID],
                               i.[ACTUALSTARTDATETIME]
                       ) as [ALLSTEPSEQUENCENUMBER],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALENDDATETIME],
               i.[PROSPECTPLANID],
               max(i.[ACTUALENDDATETIME]) over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
               i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
               and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       ),
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
               s.[ACTUALENDDATETIME],
               r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
               s.[PROSPECTPLANSTATUSCODEID],
               r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
               s.[PROSPECTPLANID],
               s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.[ALLSTEPSEQUENCENUMBER]
       ),
[STAGEOCCURRENCES]
as (
       select row_number() over (
                       order by sofp.[PROSPECTPLANID],
                               sofp.[ACTUALSTARTDATETIME]
                       ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
               sofp.[ACTUALSTARTDATETIME],
               sofp.[ACTUALENDDATETIME],
               sofp.[LASTSTEPINPLANENDDATETIME],
               sofp.[PROSPECTPLANID],
               sofp.[PROSPECTPLANSTATUSCODEID]
       from [STAGEOCCURRENCESFIRSTPASS] as sofp
       where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
               or (
                       (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                       and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                       )
       ),
[STAGEOCCURRENCEDURATIONS]
as (
       select so1.[ACTUALSTARTDATETIME],
               so1.[ACTUALENDDATETIME],
               "STAGEOCCURRENCEDURATION" = case
                       when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                               then cast(so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
                               then CAST(so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME] as float)
                       end,
               so1.[PROSPECTPLANSTATUSCODEID],
               so1.[PROSPECTPLANID]
       from [STAGEOCCURRENCES] as so1
       left join [STAGEOCCURRENCES] as so2 on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.A-
LLSTAGEOCCURRENCESSEQUENCENUMBER
       ),
[STAGEDURATIONS]
as (
       select sum(cast(sod.[STAGEOCCURRENCEDURATION] as float)) over (
                       partition by sod.[PROSPECTPLANID],
                       sod.[PROSPECTPLANSTATUSCODEID]
                       ) as [STAGEDURATION],
               RANK() over (
                       partition by sod.[PROSPECTPLANID],
                       sod.[PROSPECTPLANSTATUSCODEID] order by sod.[ACTUALSTARTDATETIME]
```

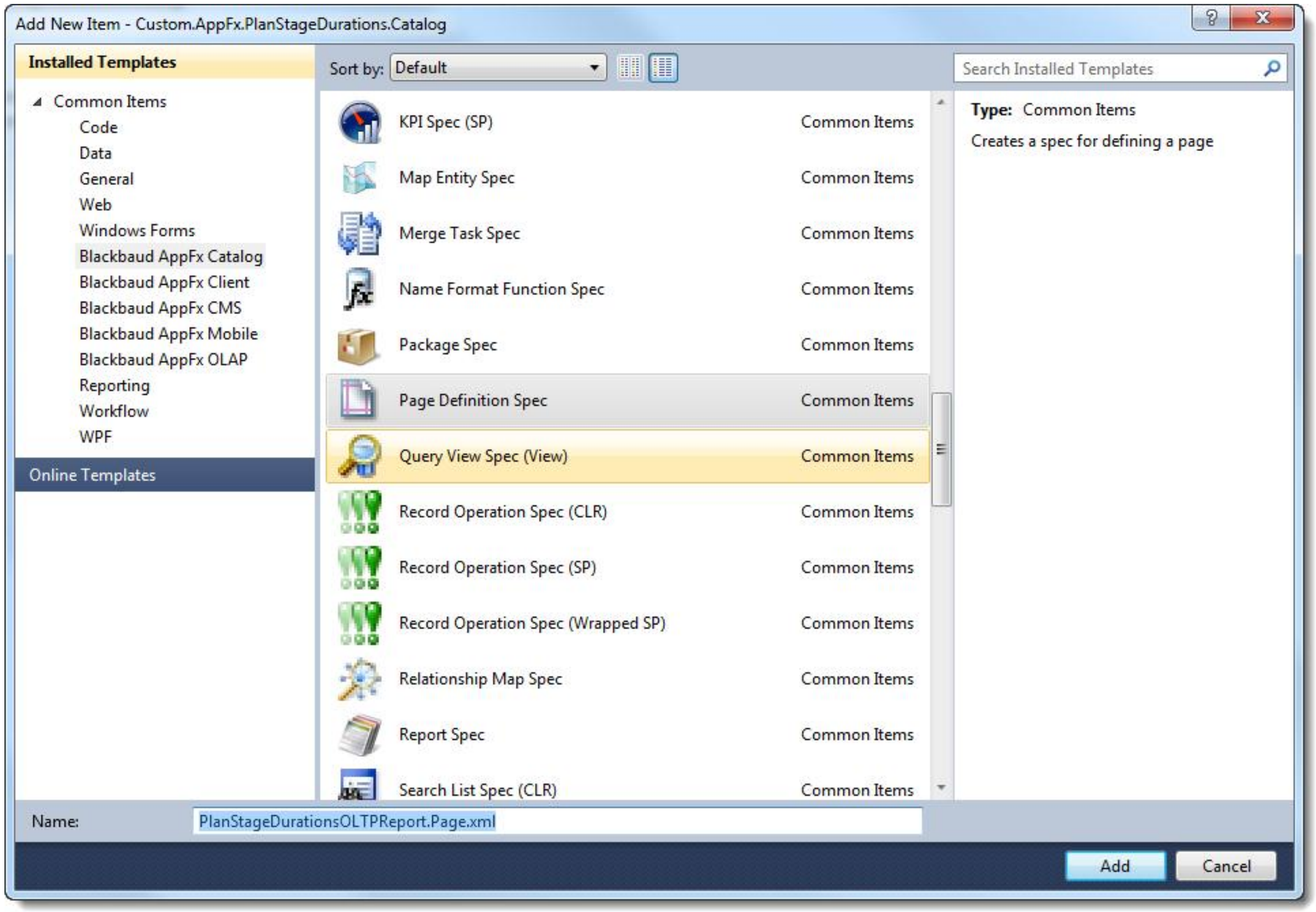
```
                        ) as [FIRSTOCCURRENCEROWINSTAGE],
                sod.[PROSPECTPLANSTATUSCODEID]
        from [STAGEOCCURRENCEDURATIONS] as sod
        )
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [AVGSTAGEDURATION],
       min(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [MINSTAGEDURATION],
       max(cast([STAGEDURATIONS].[STAGEDURATION] as float)) as [MAXSTAGEDURATION]
from [STAGEDURATIONS]
inner join [PROSPECTPLANSTATUSCODE] as p on [STAGEDURATIONS].[PROSPECTPLANSTATUSCODEID] = p.[ID]
where [FIRSTOCCURRENCEROWINSTAGE] = 1
group by p.[DESCRIPTION]
      ]]>
    </CreateSQL>
       </DataRetrieval>

  <common:FormMetaData>
    <common:FormFields>
      <common:FormField DataType="Guid" FieldID="PROSPECTPLANTYPECODEID" Caption="Prospect Plan Type Code"
Required="true">
        <common:CodeTable CodeTableName="PROSPECTPLANTYPECODE" />
      </common:FormField>
    </common:FormFields>

        <common:WebUIComponent>
            <common:UIModel AssemblyName="Custom.AppFx.PlanStageDurations.UIModel.dll" Class-
Name="C-
ustom.AppFx.PlanStageDurations.UIModel.ProspectPlanStatusDurations.ProspectPlanStageDurationsReportOLTPVersionUIModel"
/>
            <common:WebUI>
                <common:ExternalResource Url="browser/htmlforms/ProspectPlanStageDurationsReportOLTPVersion.html" />
            </common:WebUI>
        </common:WebUIComponent>




  </common:FormMetaData>
```

```
</ReportSpec>
```

Create a Page, Task, and Package

Page Spec

1. Right-click the project and select **Add** > **New Item**. The Add New Item screen appears.
2. From the Blackbaud AppFx Catalog items, select **Page Definition Spec**.
3. Enter a name such as `PlanStageDurationsOLTPReport.Page.xml`.

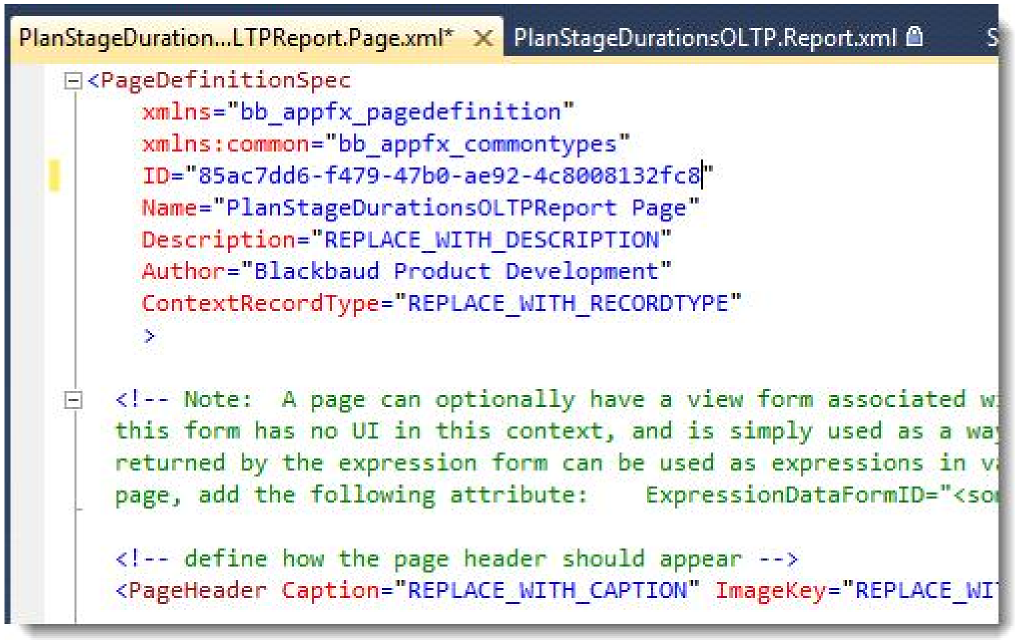


KPI Spec (SP)	Common Items
Map Entity Spec	Common Items
Merge Task Spec	Common Items
Name Format Function Spec	Common Items
Package Spec	Common Items
Page Definition Spec	Common Items
Query View Spec (View)	Common Items
Record Operation Spec (CLR)	Common Items
Record Operation Spec (SP)	Common Items
Record Operation Spec (Wrapped SP)	Common Items
Relationship Map Spec	Common Items
Report Spec	Common Items
Search List Spec (CLR)	Common Items

Type: Common Items
Creates a spec for defining a page

Name: PlanStageDurationsOLTPReport.Page.xml

4. Click **Add**. The spec appears.


PlanStageDuration...LTPReport.Page.xml*
PlanStageDurationsOLTP.Report.xml

```
<PageDefinitionSpec
    xmlns="bb_appfx_pagedefinition"
    xmlns:common="bb_appfx_commontypes"
    ID="85ac7dd6-f479-47b0-ae92-4c8008132fc8"
    Name="PlanStageDurationsOLTPReport Page"
    Description="REPLACE_WITH_DESCRIPTION"
    Author="Blackbaud Product Development"
    ContextRecordType="REPLACE_WITH_RECORDTYPE"
    >

  <!-- Note:  A page can optionally have a view form associated wi
  this form has no UI in this context, and is simply used as a way
  returned by the expression form can be used as expressions in vi
  page, add the following attribute:    ExpressionDataFormID="<sol

  <!-- define how the page header should appear -->
  <PageHeader Caption="REPLACE_WITH_CAPTION" ImageKey="REPLACE_WIT
```

5. Adjust this information:

   Name: `Prospect Plan Stage Durations Report (OLTP Version) Page`

   Description: `A page to display the Prospect Plan Stage Durations (OLTP Version) Report`

   Author: `Technical Training`

6. Remove the `ContextRecordType` attribute.
7. Change the `PageHeader Caption` attribute to `Prospect Plan Stage Durations Report (OLTP Version)`.
8. Remove the `PageHeader ImageKey` attribute.
9. Change the `Tab Caption` to `Prospect Plan Stage Durations Report (OLTP Version)`.
10. From the template `Section`, remove the `DataList` and `Actions` element.

11. Within the section, add a `Report` element.
12. To the `Report` element, add the `AutoLoad` attribute with a value of `true`.
13. To the `Report` element, add the `ID` attribute with the value of the `ID` for the report. This is the `ID` attribute in the `ReportSpec` element for the Report Spec.

```
        <Section ID="7d956395-a588-4009-8a2d-40f09a92e52b" Caption="Prospect Plan Stage Durations Report (OLTP Version)
">
          <Report AutoLoad="false" ID="882fe807-2570-4900-9c89-0861070f7ea5"></Report>
       </Section>
```

14. Remove the `PageActionGroups` element.

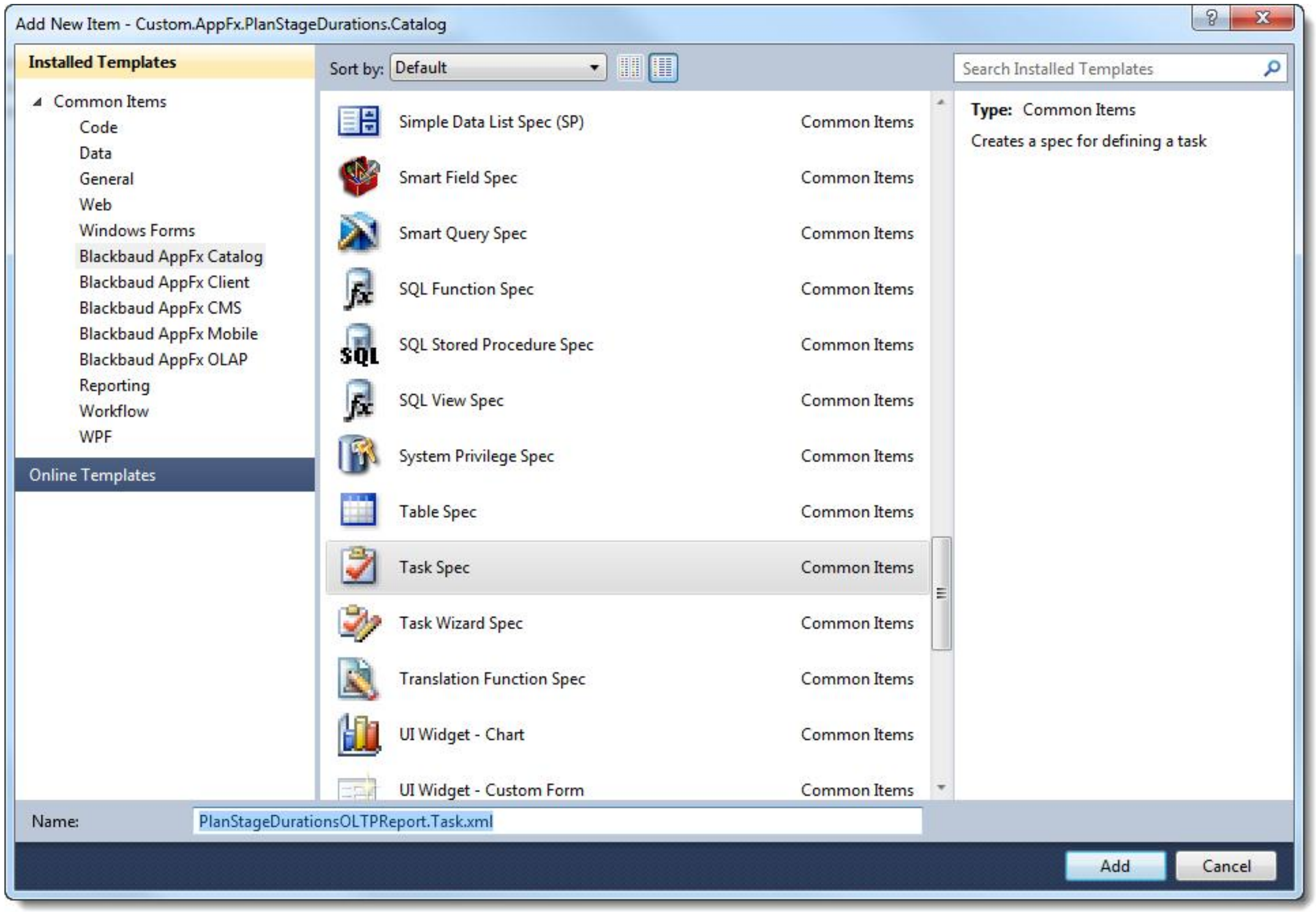
```
<PageDefinitionSpec
    xmlns="bb_appfx_pagedefinition"
    xmlns:common="bb_appfx_commontypes"
    ID="85ac7dd6-f479-47b0-ae92-4c8008132fc8"
    Name="Prospect Plan Stage Durations Report (OLTP Version) Page"
    Description="A page to display the Prospect Plan Stage Durations (OLTP Version) Report"
    Author="Technical Training"
    >

       <!-- Note:  A page can optionally have a view form associated with it as the "Expression form".  While imple-
mented as a view data form,
       this form has no UI in this context, and is simply used as a way of loading additional information associated
with the page.  The fields
       returned by the expression form can be used as expressions in various properties throughout the page.  To spec-
ify an expression form for this
       page, add the following attribute:              ExpressionDataFormID="<some guid>"-->

       <!-- define how the page header should appear -->
       <PageHeader Caption="Prospect Plan Stage Durations Report (OLTP Version)" />

       <!-- define the tabs for the page - note that if only one tab is present, then that tab's sections are promoted
to the page level (ie., the tab
       itself isn't shown -->
       <Tabs>
               <Tab ID="d850d962-3724-4616-964c-5a234ba79e61" Caption="Prospect Plan Stage Durations Report (OLTP Version)">
```

```
                        <!-- define the sections for this tab -->
                        <Sections>
                                <Section ID="7d956395-a588-4009-8a2d-40f09a92e52b" Caption="Prospect Plan Stage Durations Report (OLTP Ver-
sion)">
          <Report AutoLoad="false" ID="882fe807-2570-4900-9c89-0861070f7ea5"></Report>
                                </Section>
                        </Sections>
                </Tab>
        </Tabs>
</PageDefinitionSpec>
```

15. Save the Page Spec.

Task Spec

1. Right-click the project and select **Add** > **New Item**. The Add New Item screen appears.
2. From the Blackbaud AppFx Catalog items, select **Task Spec**.
3. Enter a name such as `PlanStageDurationsOLTPReport.Task.xml`.


Add New Item - Custom.AppFx.PlanStageDurations.Catalog

Installed Templates

- Common Items
  - Code
  - Data
  - General
  - Web
  - Windows Forms
  - Blackbaud AppFx Catalog
  - Blackbaud AppFx Client
  - Blackbaud AppFx CMS
  - Blackbaud AppFx Mobile
  - Blackbaud AppFx OLAP
  - Reporting
  - Workflow
  - WPF

Online Templates

Sort by: Default

Simple Data List Spec (SP)	Common Items
Smart Field Spec	Common Items
Smart Query Spec	Common Items
SQL Function Spec	Common Items
SQL Stored Procedure Spec	Common Items
SQL View Spec	Common Items
System Privilege Spec	Common Items
Table Spec	Common Items
Task Spec	Common Items
Task Wizard Spec	Common Items
Translation Function Spec	Common Items
UI Widget - Chart	Common Items
UI Widget - Custom Form	Common Items

Search Installed Templates

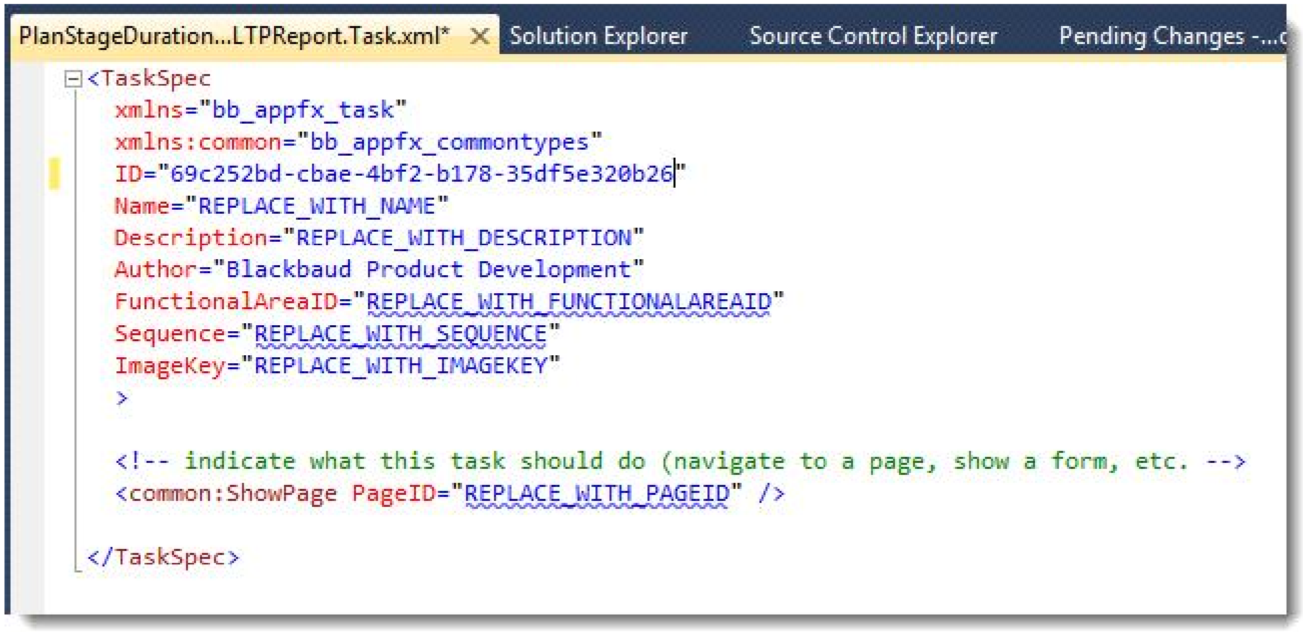
Type: Common Items

Creates a spec for defining a task

Name: PlanStageDurationsOLTPReport.Task.xml

Add Cancel

4. Click **Add**. the Task Spec appears.



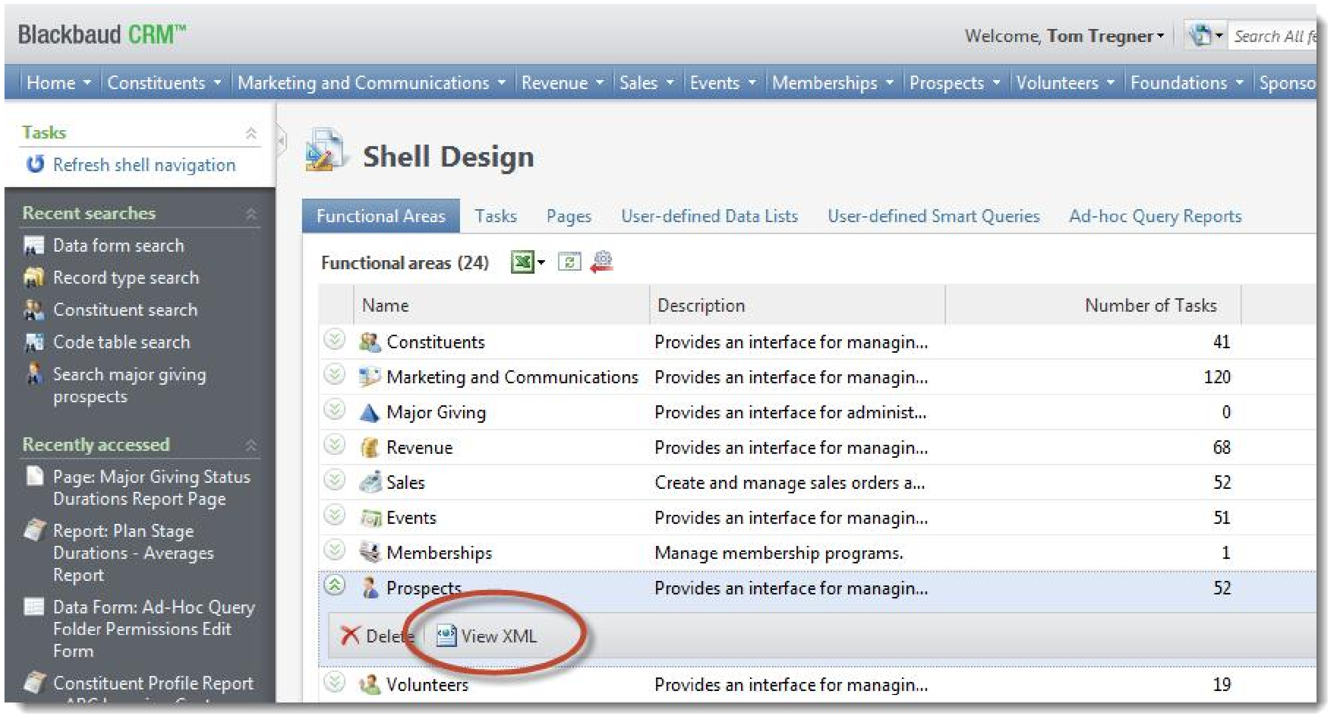
```
<TaskSpec
  xmlns="bb_appfx_task"
  xmlns:common="bb_appfx_commontypes"
  ID="69c252bd-cbae-4bf2-b178-35df5e320b26"
  Name="REPLACE_WITH_NAME"
  Description="REPLACE_WITH_DESCRIPTION"
  Author="Blackbaud Product Development"
  FunctionalAreaID="REPLACE_WITH_FUNCTIONALAREAID"
  Sequence="REPLACE_WITH_SEQUENCE"
  ImageKey="REPLACE_WITH_IMAGEKEY"
  >

  <!-- indicate what this task should do (navigate to a page, show a form, etc. -->
  <common:ShowPage PageID="REPLACE_WITH_PAGEID" />

</TaskSpec>
```

5. Adjust this information:

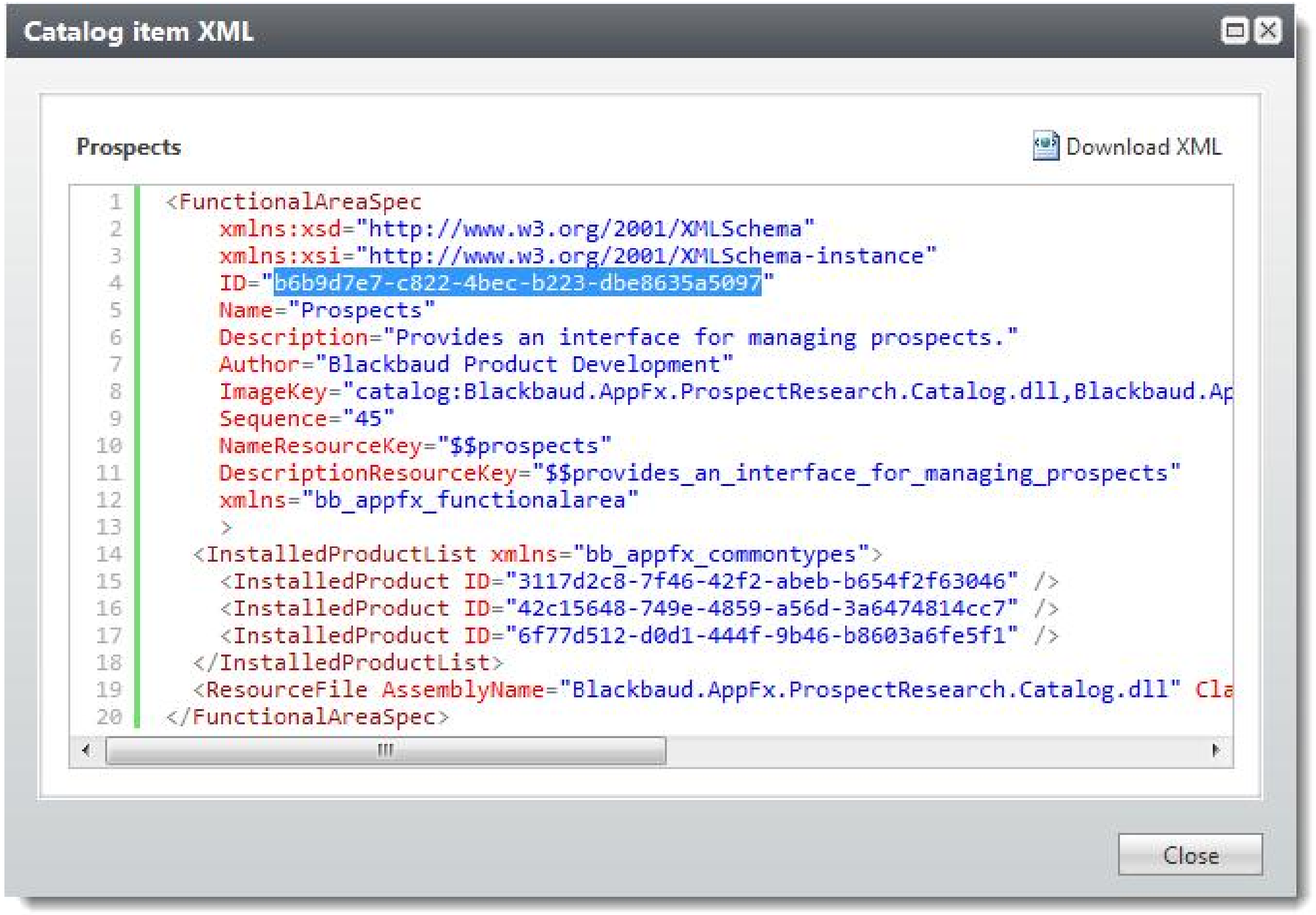
- **Name:** `Prospect Plan Stage Durations Report (OLTP Version) Page Task`
- **Description:** `A task to display the Prospect Plan Stage Durations (OLTP Version) Report Page`
- **Author:** `Technical Training`
- **FunctionalAreaID:** `b6b9d7e7-c822-4bec-b223-dbe8635a5097`

Note: This is the Prospects functional area.

To find this, look for the **Prospects** functional area in **Administration** > **Application** > **Shell Design** > Functional Areas. Then click **View XML**.



Name	Description	Number of Tasks
Constituents	Provides an interface for managin...	41
Marketing and Communications	Provides an interface for managin...	120
Major Giving	Provides an interface for administ...	0
Revenue	Provides an interface for managin...	68
Sales	Create and manage sales orders a...	52
Events	Provides an interface for managin...	51
Memberships	Manage membership programs.	1
Prospects	Provides an interface for managin...	52
Volunteers	Provides an interface for managin...	19



```
<FunctionalAreaSpec
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ID="b6b9d7e7-c822-4bec-b223-dbe8635a5097"
    Name="Prospects"
    Description="Provides an interface for managing prospects."
    Author="Blackbaud Product Development"
    ImageKey="catalog:Blackbaud.AppFx.ProspectResearch.Catalog.dll,Blackbaud.Ap
    Sequence="45"
    NameResourceKey="$$prospects"
    DescriptionResourceKey="$$provides_an_interface_for_managing_prospects"
    xmlns="bb_appfx_functionalarea"
    >
  <InstalledProductList xmlns="bb_appfx_commontypes">
    <InstalledProduct ID="3117d2c8-7f46-42f2-abeb-b654f2f63046" />
    <InstalledProduct ID="42c15648-749e-4859-a56d-3a6474814cc7" />
    <InstalledProduct ID="6f77d512-d0d1-444f-9b46-b8603a6fe5f1" />
  </InstalledProductList>
  <ResourceFile AssemblyName="Blackbaud.AppFx.ProspectResearch.Catalog.dll" Cla
</FunctionalAreaSpec>
```

- **Sequence:** 1000

  This is the `Sequence` number of the task with respect to other tasks, not the `Sequence` number for the functional area.

- **ImageKey:** Remove this attribute for now

6. Replace `<common:ShowPage PageID="REPLACE_WITH_PAGEID" />` with:

```
<common:ShowPage PageID="85ac7dd6-f479-47b0-ae92-4c8008132fc8" />
```

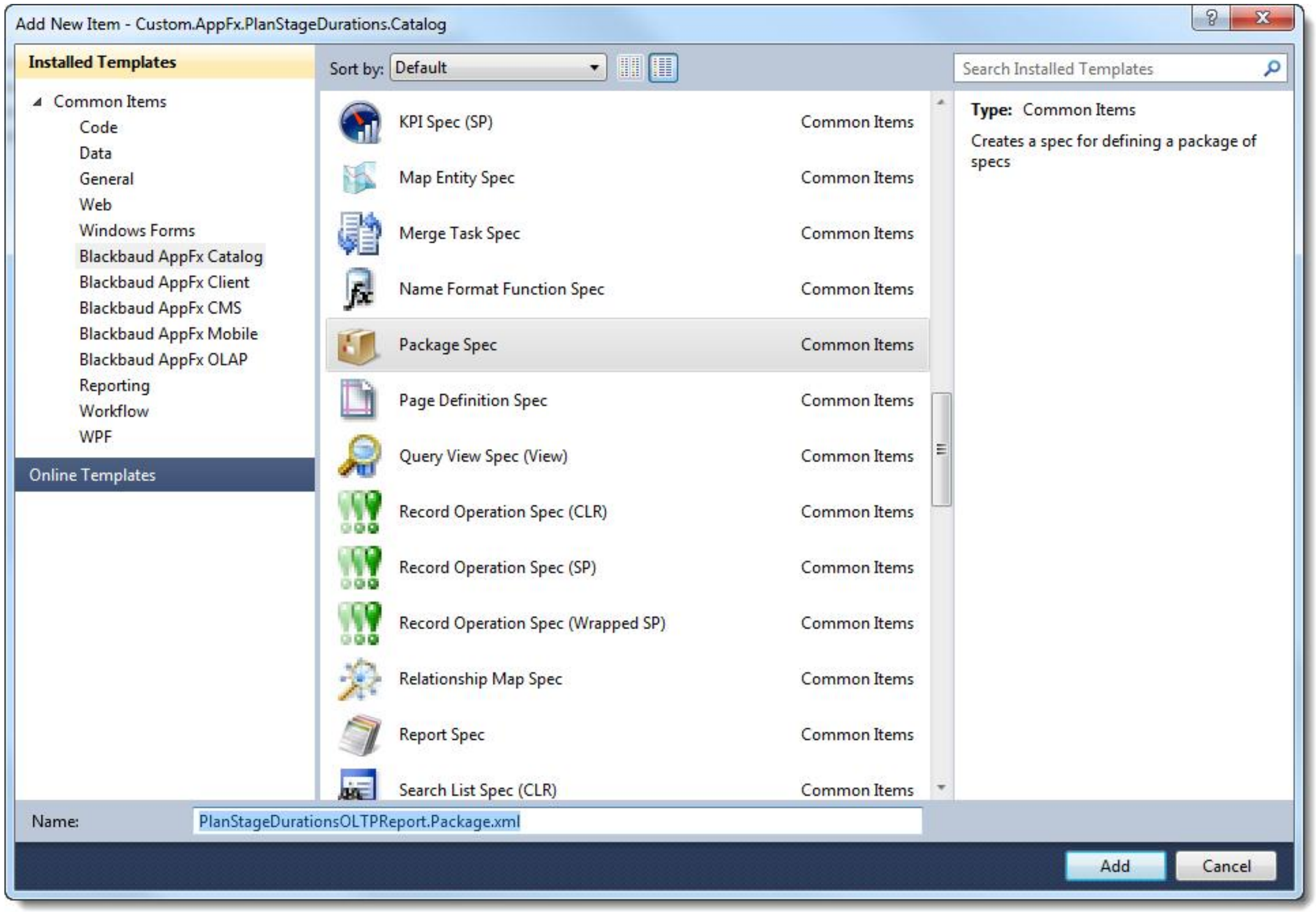
`PageID` is the `ID` for the page created in the previous section. It is the `ID` attribute of `PageDefinitionSpec` for that page.

```
<TaskSpec
       xmlns="bb_appfx_task"
       xmlns:common="bb_appfx_commontypes"
       ID="69c252bd-cbae-4bf2-b178-35df5e320b26"
       Name="Prospect Plan Stage Durations Report (OLTP Version) Page Task"
       Description="A task to display the Prospect Plan Stage Durations (OLTP Version) Report Page"
       Author="Technical Training"
       FunctionalAreaID="b6b9d7e7-c822-4bec-b223-dbe8635a5097"
       Sequence="1000"
       >

       <!-- indicate what this task should do (navigate to a page, show a form, etc. -->
       <common:ShowPage PageID="85ac7dd6-f479-47b0-ae92-4c8008132fc8" />

</TaskSpec>
```

7. Save the Task Spec.

Package Spec

1. Right-click the project and select **Add** > **New Item**. The Add New Item screen appears.
2. From the Blackbaud AppFx Catalog items, select **Task Spec**.
3. Enter a name such as `PlanStageDurationsOLTPReport.Package.xml`.



Add New Item - Custom.AppFx.PlanStageDurations.Catalog

Installed Templates

- Common Items
  - Code
  - Data
  - General
  - Web
  - Windows Forms
  - Blackbaud AppFx Catalog
  - Blackbaud AppFx Client
  - Blackbaud AppFx CMS
  - Blackbaud AppFx Mobile
  - Blackbaud AppFx OLAP
  - Reporting
  - Workflow
  - WPF

Online Templates

Sort by: Default

Search Installed Templates

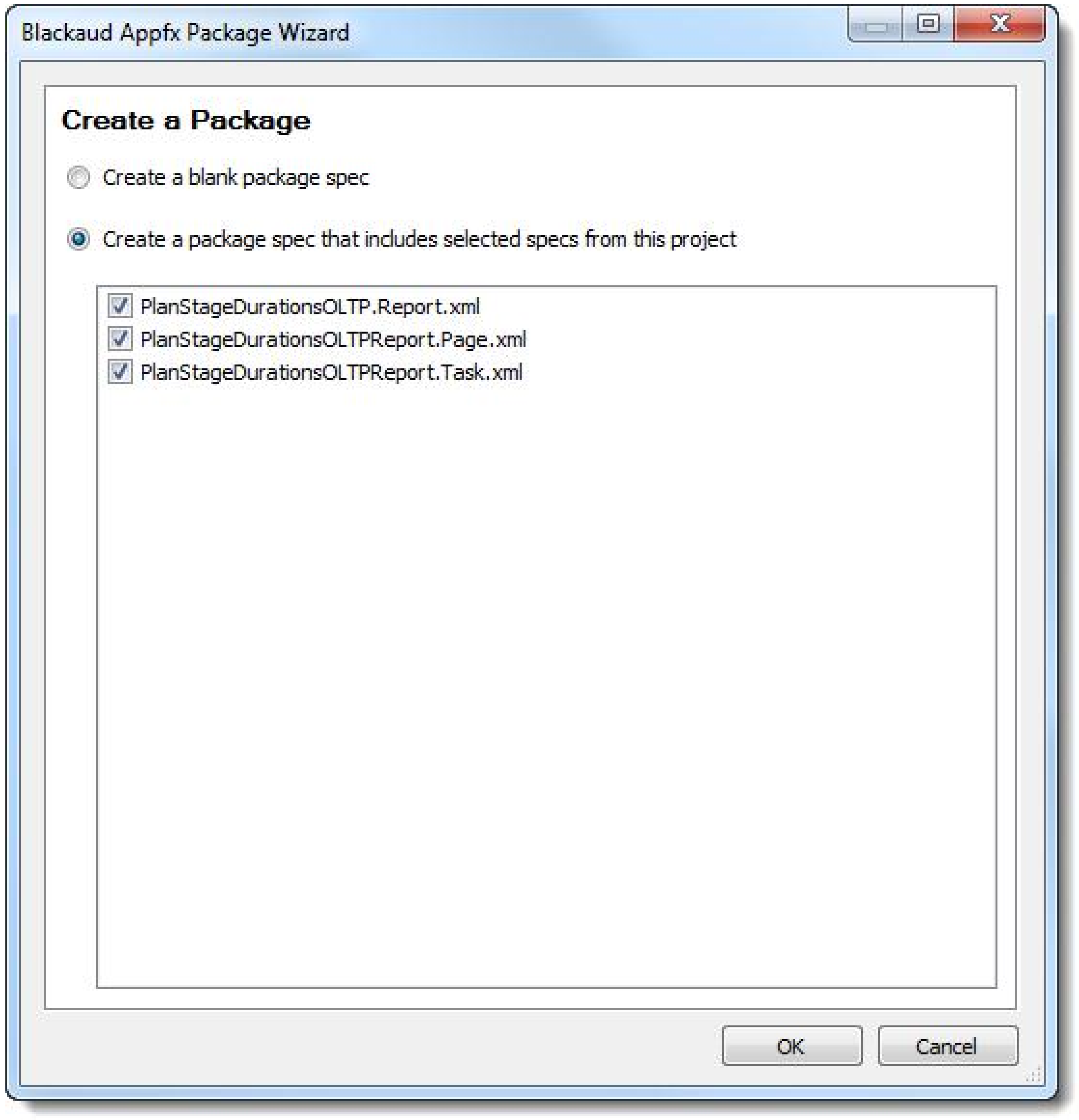
KPI Spec (SP)	Common Items
Map Entity Spec	Common Items
Merge Task Spec	Common Items
Name Format Function Spec	Common Items
Package Spec	Common Items
Page Definition Spec	Common Items
Query View Spec (View)	Common Items
Record Operation Spec (CLR)	Common Items
Record Operation Spec (SP)	Common Items
Record Operation Spec (Wrapped SP)	Common Items
Relationship Map Spec	Common Items
Report Spec	Common Items
Search List Spec (CLR)	Common Items

Type: Common Items

Creates a spec for defining a package of specs

Name: PlanStageDurationsOLTPReport.Package.xml

4. Click **Add**. The Blackbaud Appfx Package Wizard appears.
5. Click **Create a package spec that includes the selected specs from this project**.
6. Select the Report Spec created in Create a Report Spec on page 54 and the Page Spec and Task Spec created in this topic.


Blackaud Appfx Package Wizard

Create a Package

( ) Create a blank package spec

(●) Create a package spec that includes selected specs from this project

- ✓ PlanStageDurationsOLTP.Report.xml
- ✓ PlanStageDurationsOLTPReport.Page.xml
- ✓ PlanStageDurationsOLTPReport.Task.xml

7. Click **OK**. The Package Spec appears. With the comments about dependency order removed, the spec will look like this:

```
<PackageSpec
    xmlns="bb_appfx_package"
    xmlns:common="bb_appfx_commontypes"
    ID="a34984f1-f1e7-4970-b16d-ecb929e02951"
    Name="MajorGivingPlanStageDurations Package"
    Description="REPLACE_WITH_DESCRIPTION"
    Author="$author$"
    >

       <common:DependencyList>
        <common:Dependency CatalogAssembly="MajorGivingPlanStageDurations.dll" Cat-
alogItem="MajorGivingPlanStageDurations.MajorGivingPlanStageDurations.Report.xml" />
        <common:Dependency CatalogAssembly="MajorGivingPlanStageDurations.dll" Cat-
alogItem="MajorGivingPlanStageDurations.MajorGivingPlanStatusDurationsReport.Page.xml" />
        <common:Dependency CatalogAssembly="MajorGivingPlanStageDurations.dll" Cat-
alogItem="MajorGivingPlanStageDurations.MajorGivingPlanStatusDurationsReport.Task.xml" />
    </common:DependencyList>

</PackageSpec>
```

8. Adjust the `Name`, `Description`, and `Author`:

```
<PackageSpec
    xmlns="bb_appfx_package"
    xmlns:common="bb_appfx_commontypes"
    ID="903fc809-1007-490a-8f5f-c5c646b3cdb1"
    Name="Prospect Plan Status Durations (OLTP Version) Package"
    Description="A package to load the artifacts for the OLTP version of the Prospect Plan Stage Durations 
Report"
    Author="Technical Training"
    >


       <common:DependencyList>
        <common:Dependency CatalogAssembly="Custom.AppFx.PlanStageDurations.Catalog.dll" Cat-
alogItem="Custom.AppFx.PlanStageDurations.Catalog.PlanStageDurationsOLTP.Report.xml" />
        <common:Dependency CatalogAssembly="Custom.AppFx.PlanStageDurations.Catalog.dll" Cat-
alogItem="Custom.AppFx.PlanStageDurations.Catalog.PlanStageDurationsOLTPReport.Page.xml" />
```

```
        <common:Dependency CatalogAssembly="Custom.AppFx.PlanStageDurations.Catalog.dll" Cat-
alogItem="Custom.AppFx.PlanStageDurations.Catalog.PlanStageDurationsOLTPReport.Task.xml" />
    </common:DependencyList>

</PackageSpec>
```

9. Save the spec.

Load the Package

1. Build your Blackbaud AppFX Catalog project which contains the Report Spec, RDL file, Page Spec, Task Spec, and Package Spec. Right-click the project and select **Build**.
2. Open the project folder in Windows Explorer and browse to the folder which contains the application extension. Right-click the project and select, **Open folder in Windows Explorer**. Then browse to the bin folder. For example:

   ```
   C:\Team_Projects\Documentation\Technical_Train-
   ing\Cu-
   stom.AppFx.PlanStageDurations.Catalog\Custom.AppFx.PlanStageDurations.Catalog\bin\Debug
   ```

3. Copy the application extension DLL to the bin folder for the application instance. For example, copy:

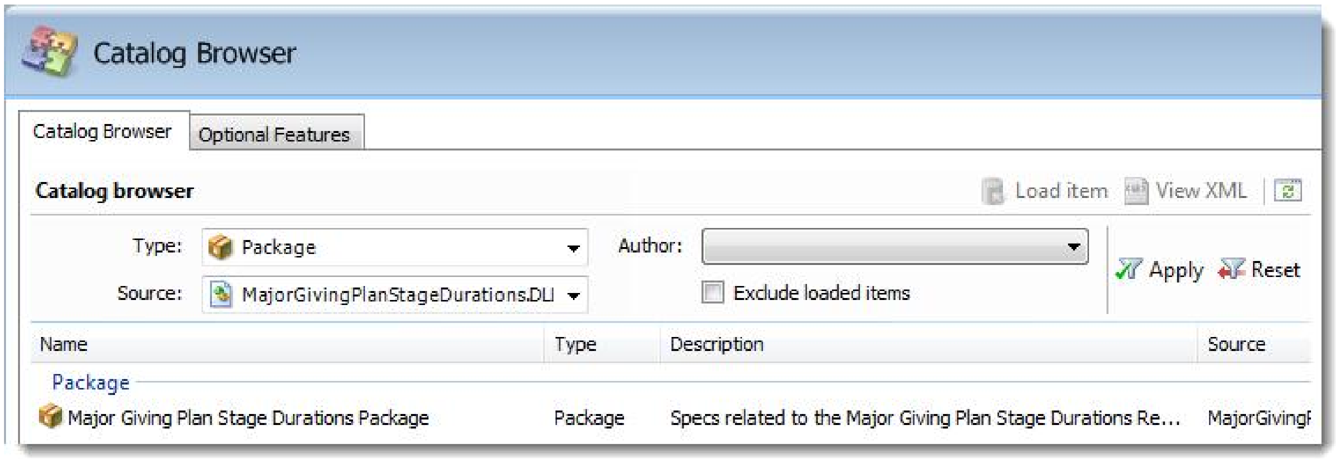
   ```
   Custom.AppFx.PlanStageDurations.Catalog.dll
   ```

   To:

   ```
   C:\Program Files\Blackbaud\bbappfx\vroot\bin
   ```

   Note: You can add a post-build command to the project to perform these steps every time you build. But you must ensure ***Visual Studio*** has rights to copy to the bin folder. For example, you may need to run ***Visual Studio*** as an Administrator for that to work.

4. From the Blackbaud Infinity based application, browse to **Administration** > **Application** > **Catalog Browser**.
5. Filter the items in the Catalog Browser for **Type**: Package and **Source**: <name of the DLL>
6. Click **Apply**.

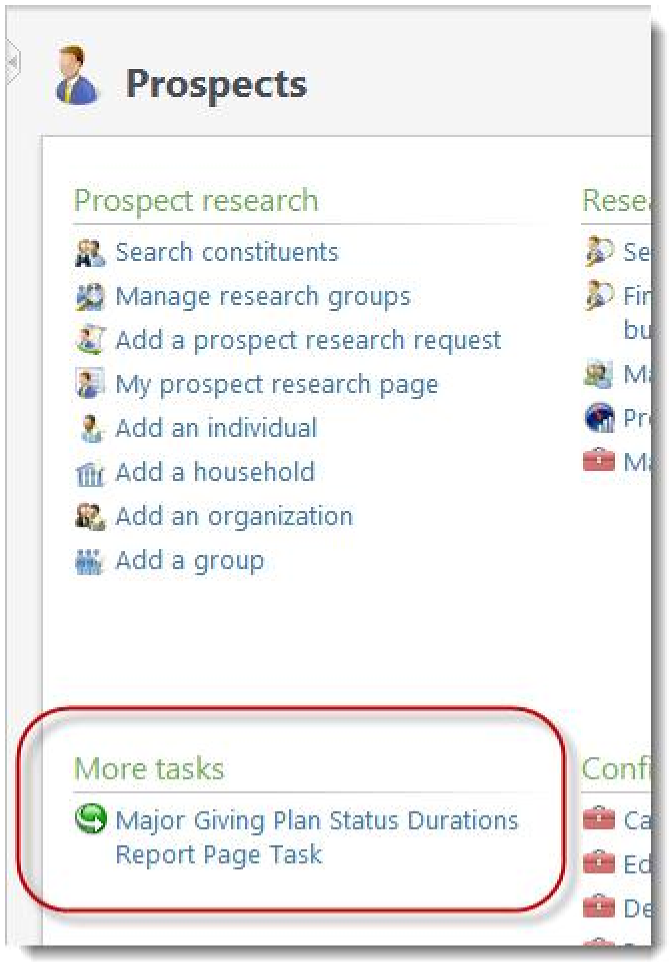


Type: Package
Source: MajorGivingPlanStageDurations.DLl
Author:
Exclude loaded items

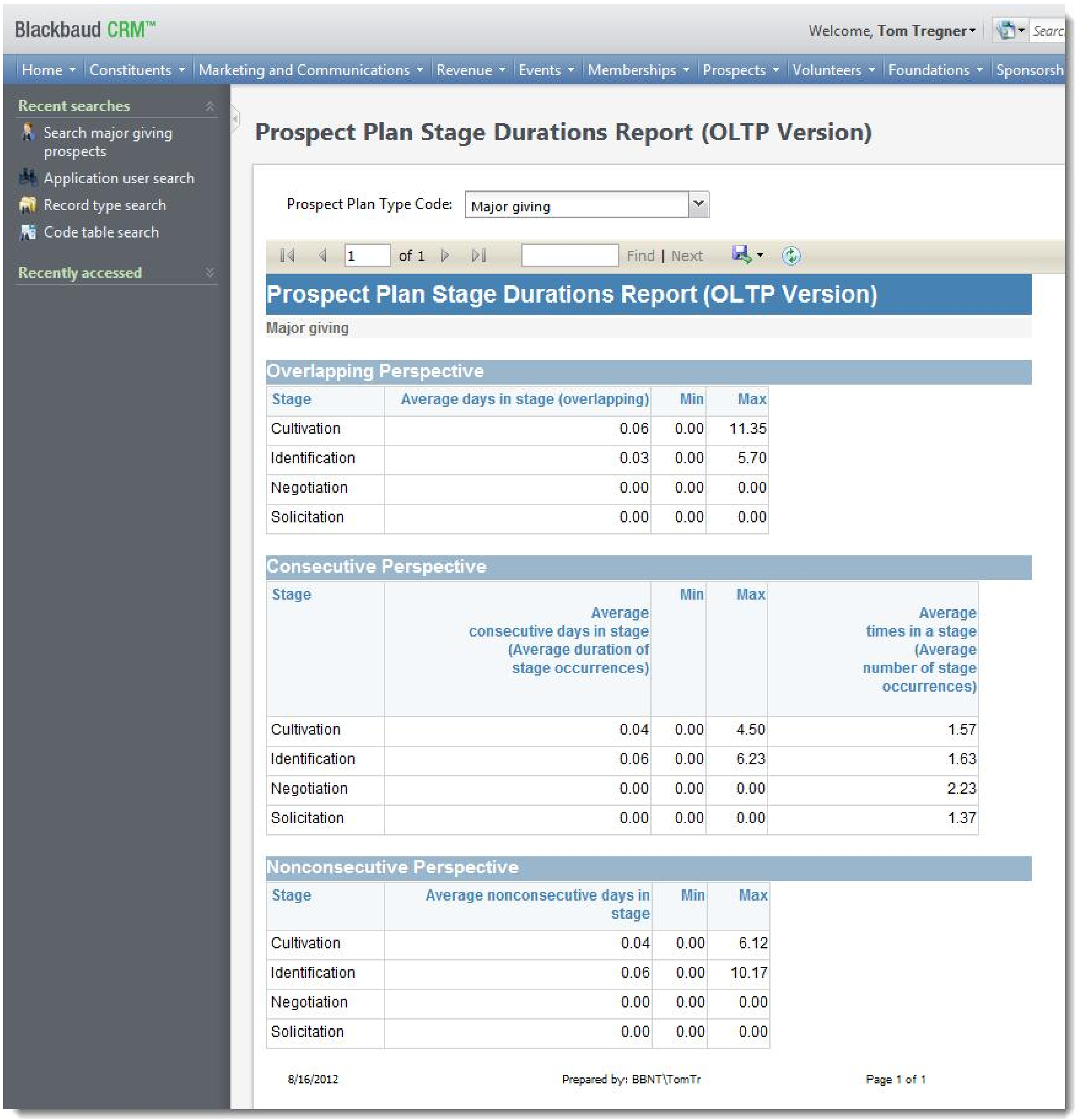
Name	Type	Description	Source
Package			
Major Giving Plan Stage Durations Package	Package	Specs related to the Major Giving Plan Stage Durations Re...	MajorGivingF

7. Highlight the package and click **Load item**.
8. Once the package loads, browse to the **Prospects** functional area. The task to open the report appears under **More tasks**.

   Note: Shown from the web shell.



9. Click the task.



Prospect Plan Stage Durations Report (OLTP Version)

Prospect Plan Type Code: Major giving

1 of 1

Prospect Plan Stage Durations Report (OLTP Version)

Major giving

Overlapping Perspective

Stage	Average days in stage (overlapping)	Min	Max
Cultivation	0.06	0.00	11.35
Identification	0.03	0.00	5.70
Negotiation	0.00	0.00	0.00
Solicitation	0.00	0.00	0.00

Consecutive Perspective

Stage	Average consecutive days in stage (Average duration of stage occurrences)	Min	Max	Average times in a stage (Average number of stage occurrences)
Cultivation	0.04	0.00	4.50	1.57
Identification	0.06	0.00	6.23	1.63
Negotiation	0.00	0.00	0.00	2.23
Solicitation	0.00	0.00	0.00	1.37

Nonconsecutive Perspective

Stage	Average nonconsecutive days in stage	Min	Max
Cultivation	0.04	0.00	6.12
Identification	0.06	0.00	10.17
Negotiation	0.00	0.00	0.00
Solicitation	0.00	0.00	0.00

8/16/2012 Prepared by: BBNT\TomTr Page 1 of 1

Polishing the Report

We have now created a report and Blackbaud Infinity specs to render the report in the application interface. But the report itself is not finished. There is a single table that displays calculations of average durations for plan stages in a table, one of the requirements in Prospect Plan Status Durations Report on page 11. No attention has been given to the layout of the report and the format of the durations. Within these topics we will format the existing report. Since we have wired our report to display in the Infinity application, most of the formatting work will be with the `RDL` file itself. But as we add parameters, we will return to the Blackbaud Infinity specs.

Formatting the Report

We won't delve into step-by-step instructions for formatting. The assumption is that the `RDL` editor comes with sufficient documentation to guide you. What follows is some information specific to this example.

Design

The report will be formatted along these guidelines:

Report Item	Guidelines
Report title	Arial 14pt Bold, White Background bar 32px high, #4682B4, flush with edges
Report param-eters	Arial, 9pt, DimGray (labels bold); background #F5F5F5
Report summary	Arial, 9pt, Black (labels bold); background #F5F5F5 Divider line between parameters and summary is 1pt #CCCCCC
Sub-report title	Arial, 11pt, White, Bold; Background #99B6CC, 24px high; 18px padding above bar Used when the summary sub-report needs to be displayed/hidden as a parameter.
Column header box	R244, G248, B251 (#F4F8FB); Border is R70 G130 B180 (#4682B4) Top is 1pt, bottom is 2pt
Column header	Arial, 9pt, Bold, R70 G130 B180 (#4682B4)

Report Item	Guidelines
text	
Report data	Arial, 9pt, Black Report rows separated by a 1pt gray line (#EEEEEE) Right-align numbers, currencies, dates, and times. Left-align text and IDs (even if numeric).
Data group parent rows	Arial, 9pt, Black; Background #F5F5F5
Sub totals	Arial, 9pt, Black, Bold
Total	Arial, 10pt, Black, Bold Top and bottom borders #4682B4
Footnotes	Arial, 8pt, black
Footer	Tahoma, 7pt, black, light; background #F5F5F5, 18px tall Footer contains date and time, prepared by and page number, as indicated below.
After first page	All pages should repeat the Title, Column headers and Footer. The summary panels and footnotes should not repeat.

Adding Parameters

The design stated: The report should be filterable by constituency of prospect, by prospect plan type, or both. So we need some way to parameterize constituency or prospect plan type. Our example will only show prospect plan type.

For the overlapping perspective as we have it so far, there is no consideration for prospect plan type. Prospect plan type is stored as PROSPECTPLANTYPECODEID on the PROSPECTPLAN table. This is a foreign key to the PROSPECTPLANTYPECODE table, a code table for prospect plan type codes.

Up until now we have avoided joins to the PROSPECTPLAN table in the stored procedures. This is made possible by using only the IDs for prospect plans. The only inner join necessary was to PROSPECTPLANSTATUSCODE to get the friendly name for the prospect plan status code. Now we not only need the prospect plan but the friendly name for the prospect plan type code. But we can avoid the join to PROSPECTPLANTYPECODE by placing that lookup in the UI for the report.

To test the join, we can modify the query inside of the stored procedure and execute it in SQL Server Management Studio:

```
declare @PROSPECTPLANTYPECODEID uniqueidentifier;
set @PROSPECTPLANTYPECODEID = '92CEC00D-F9B3-4713-A1CD-A944B1C0D58F';
with [STEPS]
as (
       select i.[PROSPECTPLANSTATUSCODEID],
               i.[ACTUALENDDATETIME],
               min(i.[ACTUALSTARTDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [FIRSTSTEPINSTAGEDATETIME],
               max(i.[ACTUALENDDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [LASTSTEPINSTAGEDATETIME]
       from [INTERACTION] as i inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1 and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       )
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [AVGSTAGEDURATION],
       min(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MINSTAGEDURATION],
       max(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MAXSTAGEDURATION]
from [STEPS] as s
```

```
inner join [PROSPECTPLANSTATUSCODE] as p on s.[PROSPECTPLANSTATUSCODEID] = p.[ID]
where (s.[ACTUALENDDATETIME] = s.[LASTSTEPINSTAGEDATETIME])
group by p.[DESCRIPTION]
```

This limits the query to prospect plans of type Major giving. The `uniqueidentifier` for the Major giving plan type code is `92CEC00D-F9B3-4713-A1CD-A944B1C0D58F`. We don't need those first two lines in our actual stored procedure. Once we are satisfied with the results, we can change the stored procedure to accept `@PROSPECTPLANTYPECODEID` as a parameter. Currently our stored procedures begin like this:

```
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING
as
```

...

We can modify the beginning of the `CREATE PROCEDURE` statement to this:

```
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
```

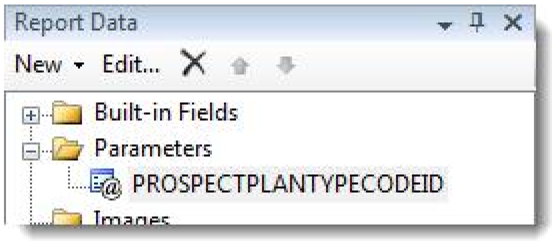
...

This example will pass one parameter, an ID for the prospect plan type code. This way the report can re-purposed for each type of prospect plan.

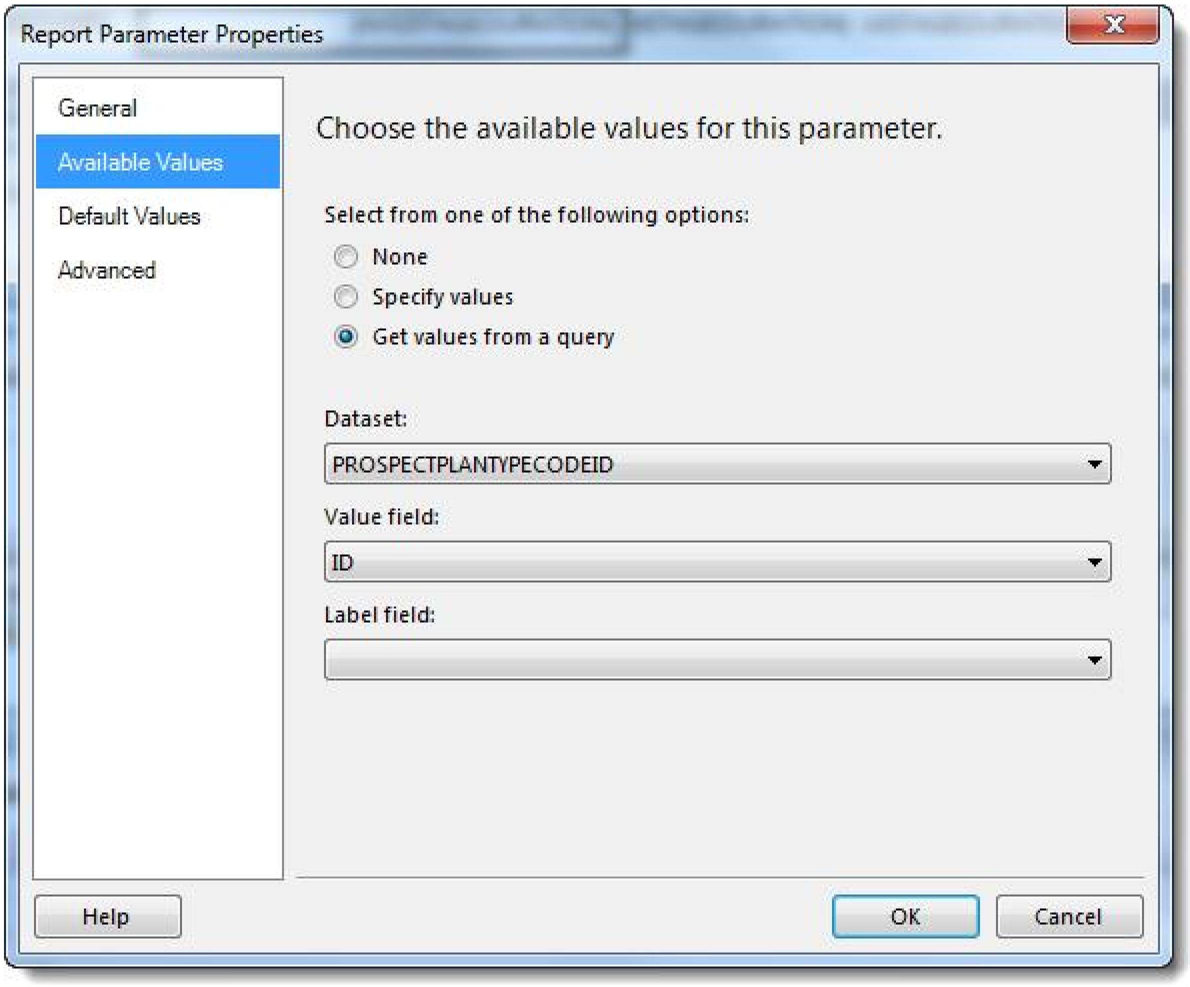
```
create procedure dbo.USR_USP_REPORT_PLANSTAGEDURATIONSOVERLAPPING (@PROSPECTPLANTYPECODEID uniqueidentifier)
as
with [STEPS]
as (
       select i.[PROSPECTPLANSTATUSCODEID],
               i.[ACTUALENDDATETIME],
               min(i.[ACTUALSTARTDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [FIRSTSTEPINSTAGEDATETIME],
               max(i.[ACTUALENDDATETIME]) over (
                       partition by i.[PROSPECTPLANID],
                       i.[PROSPECTPLANSTATUSCODEID]
                       ) as [LASTSTEPINSTAGEDATETIME]
       from [INTERACTION] as i inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1 and pl.[PROSPECTPLANTYPECODEID] = @PROSPECTPLANTYPECODEID
       )
select p.[DESCRIPTION] as [STAGENAME],
       avg(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [AVGSTAGEDURATION],
       min(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MINSTAGEDURATION],
       max(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MAXSTAGEDURATION]
from [STEPS] as s
inner join [PROSPECTPLANSTATUSCODE] as p on s.[PROSPECTPLANSTATUSCODEID] = p.[ID]
where (s.[ACTUALENDDATETIME] = s.[LASTSTEPINSTAGEDATETIME])
group by p.[DESCRIPTION]
```

Update the RDL File with the Parameter

In the RDL file, we need a to pass the parameter to the report so the report can pass the parameter to the stored procedures.



That report parameter can gather the IDs from a Dataset. For example, a dataset which holds the IDs for the `PROSPECTPLANTYPECODE` table.



Choose the available values for this parameter.

Select from one of the following options:

None

Specify values

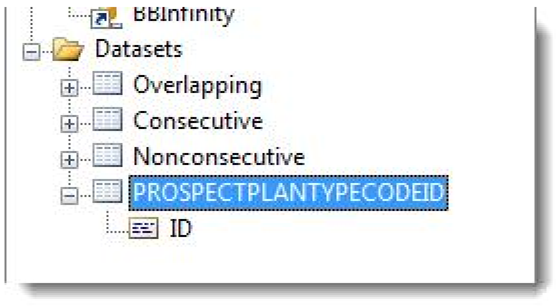
[✓ Get values from a query]

Dataset: PROSPECTPLANTYPECODEID

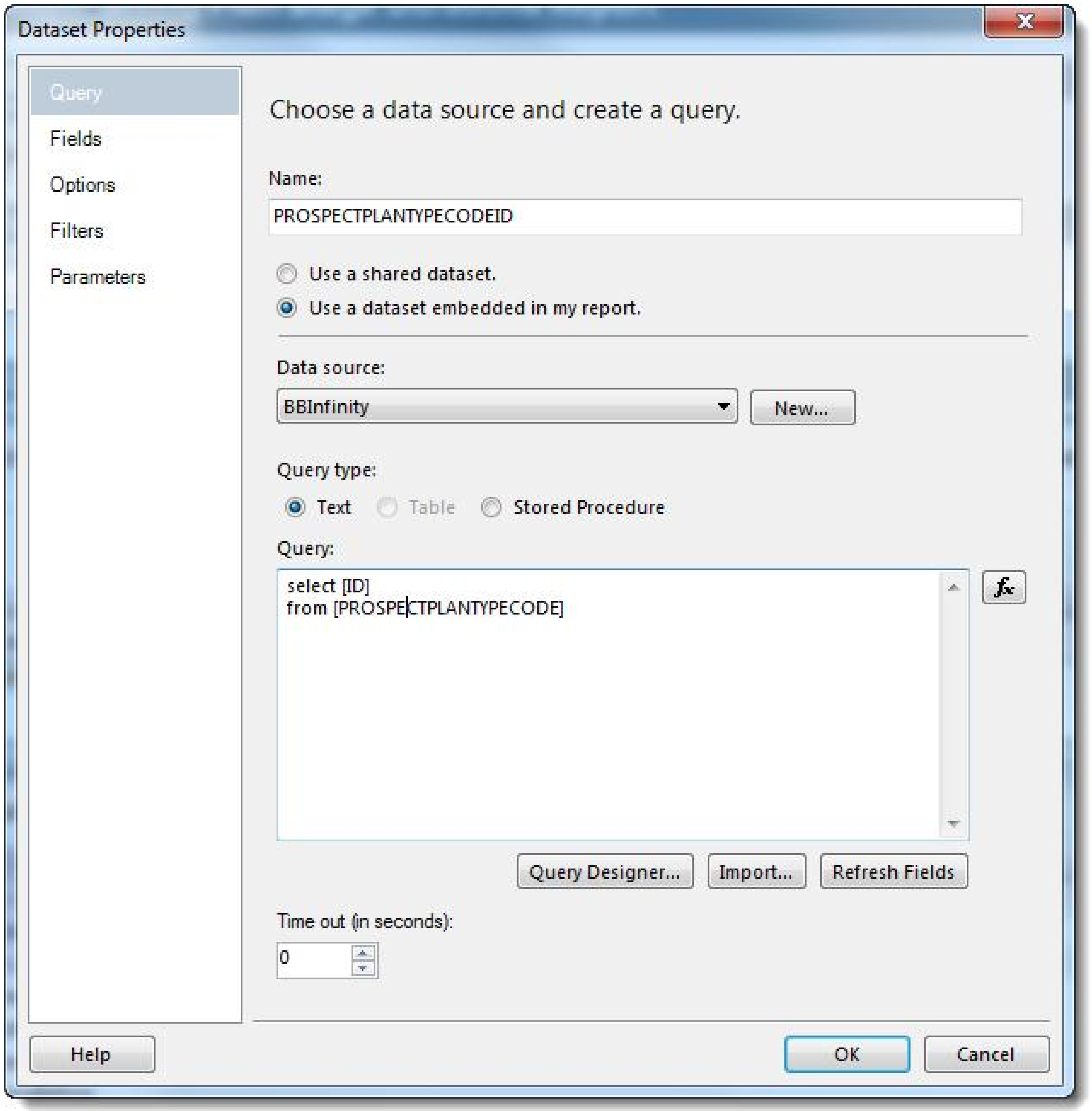
Value field: ID

Label field:

The dataset can be a simple query of the `PROSPECTPLANTYPECODE` table.



```
select [ID]
from [PROSPECTPLANTYPECODE]
```



Choose a data source and create a query.

Name: PROSPECTPLANTYPECODEID

Use a shared dataset. [✓ Use a dataset embedded in my report.]

Data source: BBInfinity

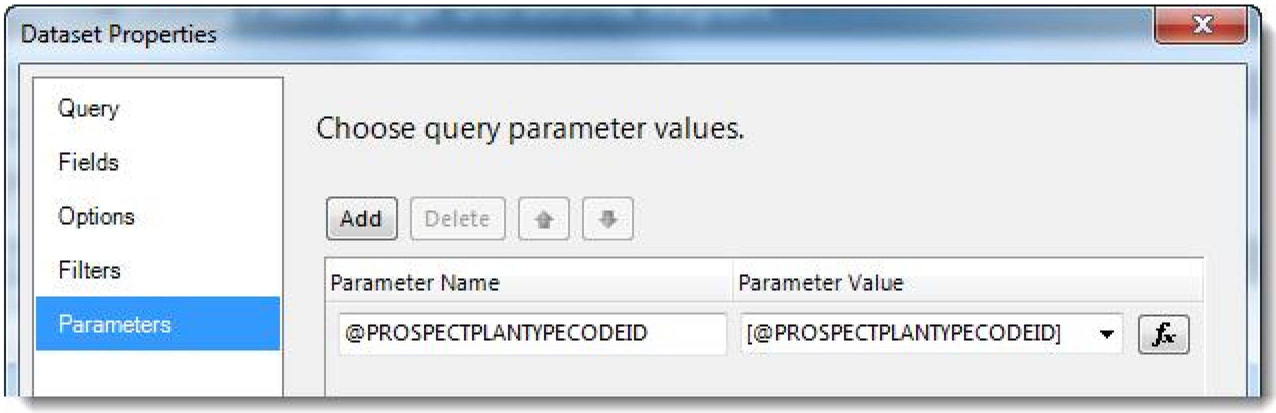
Query type: [✓ Text] Table Stored Procedure

Query:

```
select [ID]
from [PROSPECTPLANTYPECODE]
```

Time out (in seconds): 0

Each of the datasets for the stored procedures can also have a parameter.



Choose query parameter values.

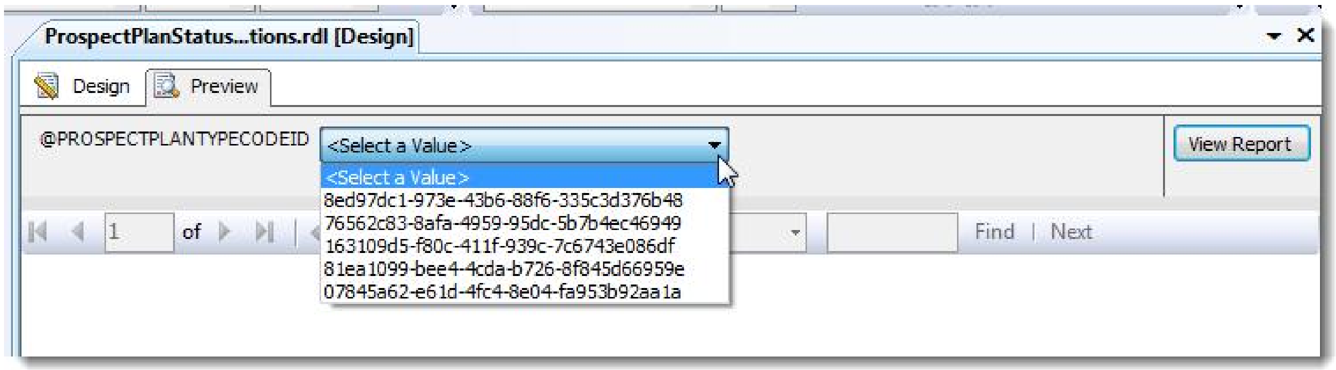
Parameter Name	Parameter Value
@PROSPECTPLANTYPECODEID	[@PROSPECTPLANTYPECODEID]

Because the parameter is a uniqueidentifier, for the Parameter Value expression:

```
=Parameters!PROSPECTPLANTYPECODEID.Value
```

This does not match what appears in the Parameter Value field. But, if you click the function button to edit the Parameter Value field with an expression editor, the above expression may appear. If that is not the expression, the report may run, but the correct parameter value may or may not be passed.

You can preview the report in Business Intelligence Development Studio at this point. The parameter appears as a drop-down of GUIDs.



@PROSPECTPLANTYPECODEID <Select a Value>

<Select a Value>
8ed97dc1-973e-43b6-88f6-335c3d376b48
76562c83-8afa-4959-95dc-5b7b4ec46949
163109d5-f80c-411f-939c-7c6743e086df
81ea1099-bee4-4cda-b726-8f845d66959e
07845a62-e61d-4fc4-8e04-fa953b92aa1a

We won't go beyond that for the RDL file. But we do have to create a friendly user interface to select those code table entries in the Blackbaud Infinity application. For that, we will add a UI Model.

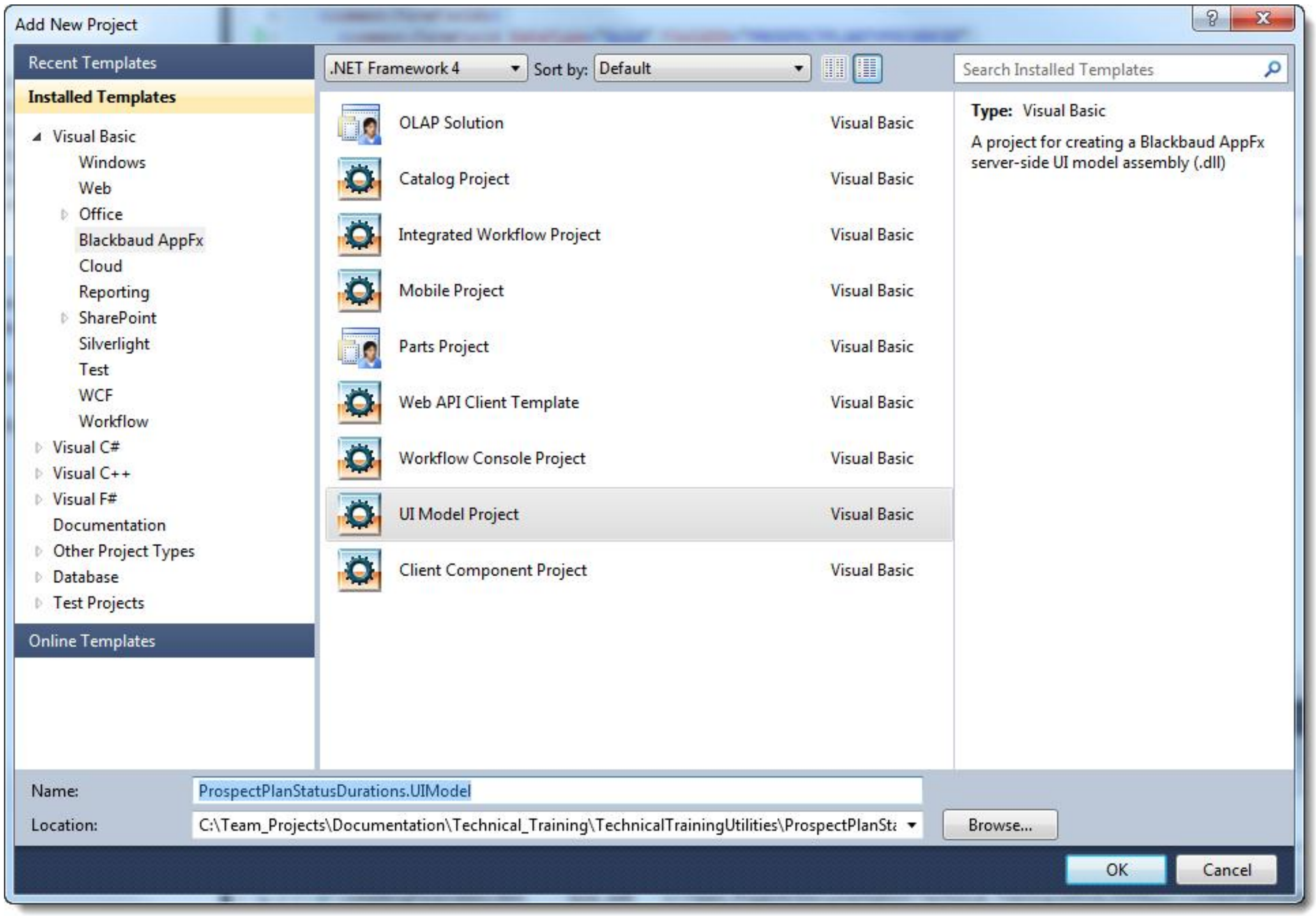
Add a UI Model for the Parameter

In the Report Spec, add form field information to map to the stored procedure parameter.

```
<common:FormMetaData>
  <common:FormFields>
    <common:FormField DataType="Guid" FieldID="PROSPECTPLANTYPECODEID"
                      Caption="Prospect Plan Type Code ID"
                      Required="true">
      <common:CodeTable CodeTableName="PROSPECTPLANTYPECODE" />
    </common:FormField>
  </common:FormFields>
```

Reload the Report Spec into the catalog.

In your solution, create a new UI Model Project.



OLAP Solution	Visual Basic
Catalog Project	Visual Basic
Integrated Workflow Project	Visual Basic
Mobile Project	Visual Basic
Parts Project	Visual Basic
Web API Client Template	Visual Basic
Workflow Console Project	Visual Basic
UI Model Project	Visual Basic
Client Component Project	Visual Basic

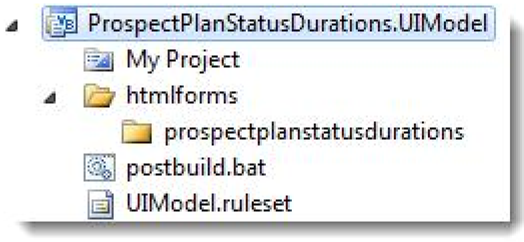
Type: Visual Basic

A project for creating a Blackbaud AppFx server-side UI model assembly (.dll)

Name: ProspectPlanStatusDurations.UIModel

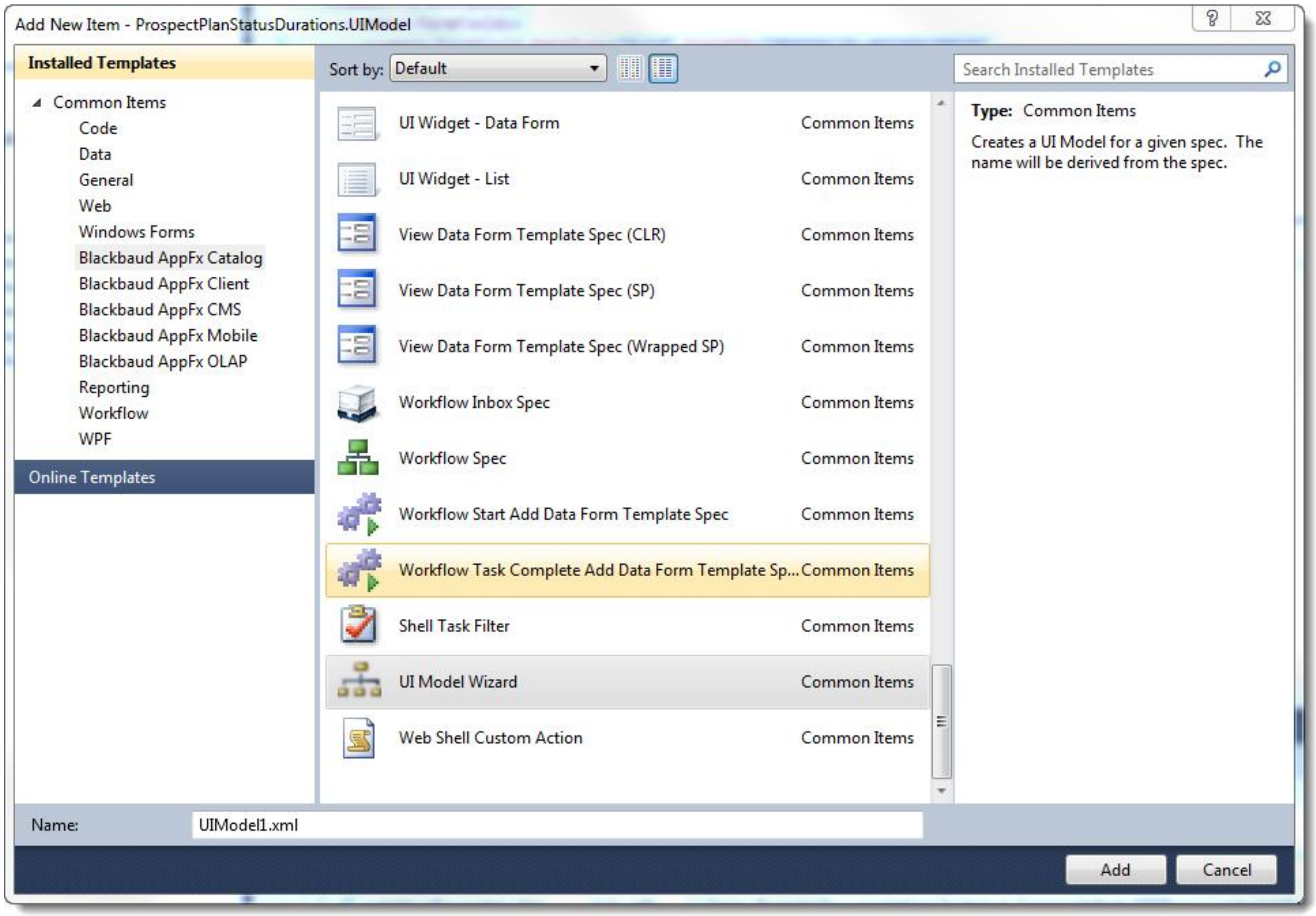
Location: C:\Team_Projects\Documentation\Technical_Training\TechnicalTrainingUtilities\ProspectPlanSt

The project appears in the solution.


ProspectPlanStatusDurations.UIModel
My Project
htmlforms
prospectplanstatusdurations
postbuild.bat
UIModel.ruleset

To the new UI Model project, add a UI Model. Right-click the project and select Add > New Item. The Add New Item screen appears.

From the Add New Item screen, select **Blackbaud AppFx Catalog** > **UI Model Wizard**. You can leave the Name field as the default.



Add New Item - ProspectPlanStatusDurations.UIModel

Installed Templates

- Common Items
  - Code
  - Data
  - General
  - Web
  - Windows Forms
  - Blackbaud AppFx Catalog
  - Blackbaud AppFx Client
  - Blackbaud AppFx CMS
  - Blackbaud AppFx Mobile
  - Blackbaud AppFx OLAP
  - Reporting
  - Workflow
  - WPF

Online Templates

Sort by: Default

UI Widget - Data Form	Common Items
UI Widget - List	Common Items
View Data Form Template Spec (CLR)	Common Items
View Data Form Template Spec (SP)	Common Items
View Data Form Template Spec (Wrapped SP)	Common Items
Workflow Inbox Spec	Common Items
Workflow Spec	Common Items
Workflow Start Add Data Form Template Spec	Common Items
Workflow Task Complete Add Data Form Template Sp...	Common Items
Shell Task Filter	Common Items
UI Model Wizard	Common Items
Web Shell Custom Action	Common Items

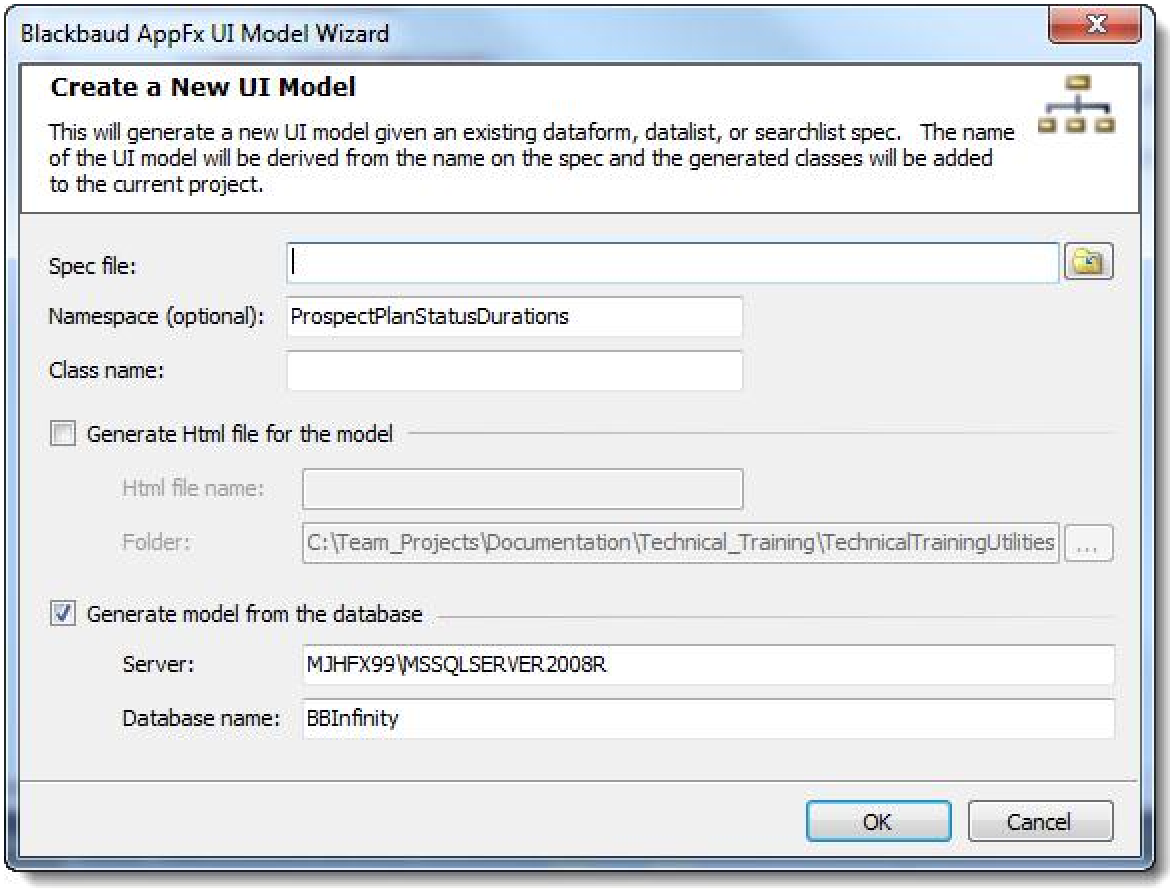
Search Installed Templates

Type: Common Items

Creates a UI Model for a given spec. The name will be derived from the spec.

Name: UIModel1.xml

Click Add. The blackbaud AppFx UI Model Wizard appears.


Blackbaud AppFx UI Model Wizard

Create a New UI Model

This will generate a new UI model given an existing dataform, datalist, or searchlist spec. The name of the UI model will be derived from the name on the spec and the generated classes will be added to the current project.

Spec file:	
Namespace (optional):	ProspectPlanStatusDurations
Class name:	

☐ Generate Html file for the model

Html file name:	
Folder:	C:\Team_Projects\Documentation\Technical_Training\TechnicalTrainingUtilities

☑ Generate model from the database

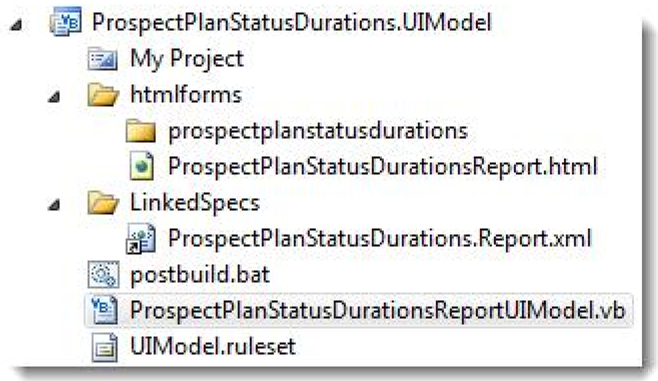
Server:	MJHFX99\MSSQLSERVER2008R
Database name:	BBInfinity

Click the folder next to the Spec file field. The Open file dialog appears.

Browse to the Report Spec in your catalog project and click Open. The Spec file, namespace, and Class name fields are populated.

Select Generate Html file for the model.

Click OK. The UI model is added.


```
ProspectPlanStatusDurations.UIModel
    My Project
    htmlforms
        prospectplanstatusdurations
        ProspectPlanStatusDurationsReport.html
    LinkedSpecs
        ProspectPlanStatusDurations.Report.xml
    postbuild.bat
    ProspectPlanStatusDurationsReportUIModel.vb
    UIModel.ruleset
```

The Report Spec FormMetaData is updated to include this:

```
        <common:WebUIComponent>
            <common:UIModel Assem-
blyName="ProspectPlanStatusDurations.UIModel.dll" Class-
Name="ProspectPlanStatusDurations.UIModel.ProspectPlanStatusDurations.ProspectPlanStatusDurationsReportUIModel"
 />
            <common:WebUI>
                <common:ExternalResource Url="browser/htmlforms/Pr-
ospectPlanStatusDurationsReport.html" />
            </common:WebUI>
        </common:WebUIComponent>
```

Create a folder called custom under prospectplanstatusdurations.

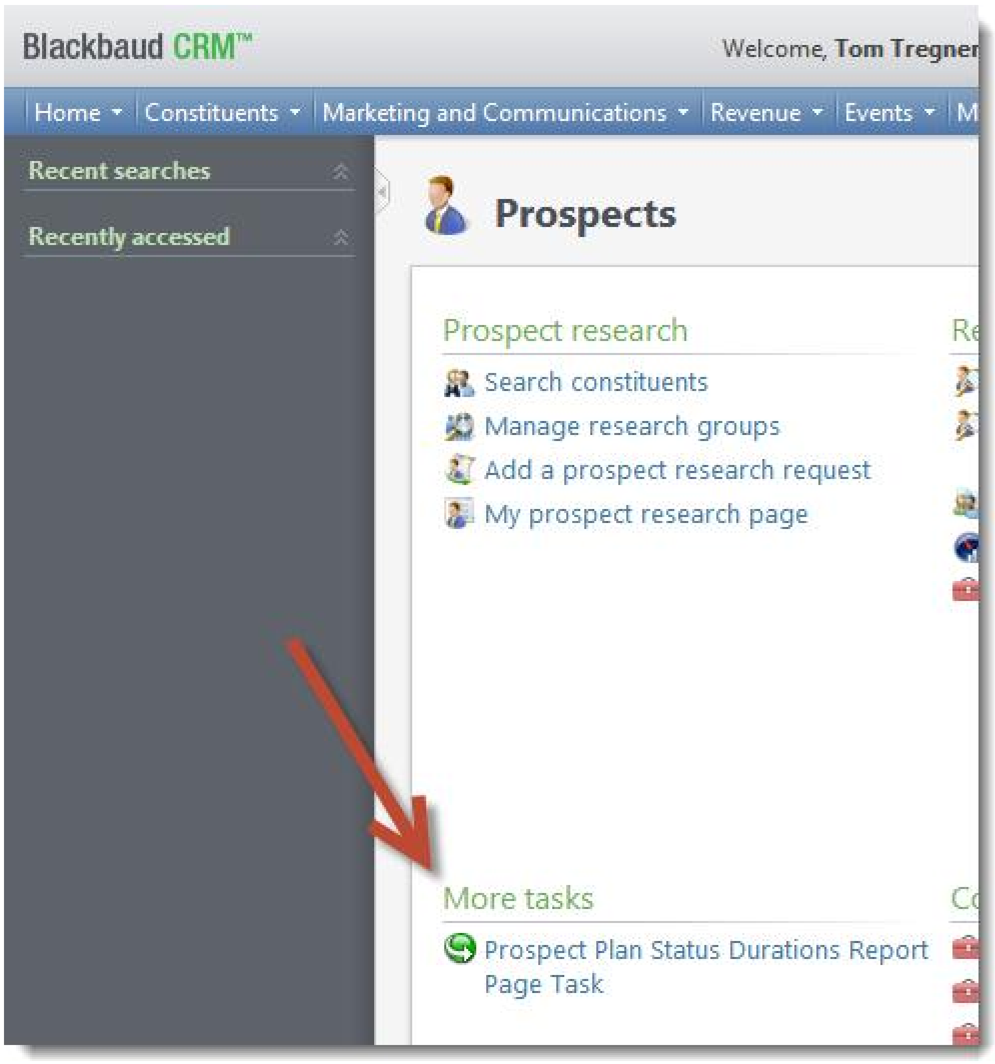
Drag ProspectPlanStatusDurationsReport.html to the custom folder.

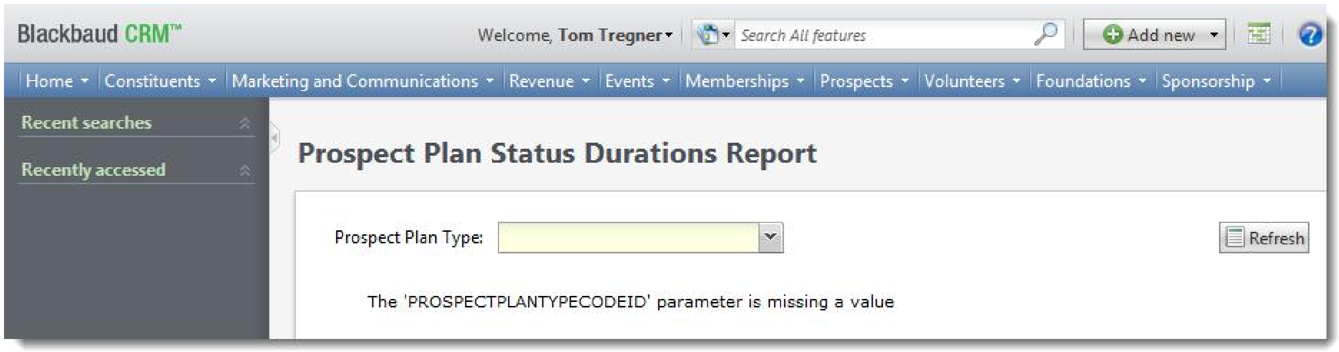
Build the project.

Copy the DLL file to vroot\bin.

Copy the HTML file to vroot\browser\htmlforms ****

You can now view the parameterized report in the Blackbaud Infinity application.

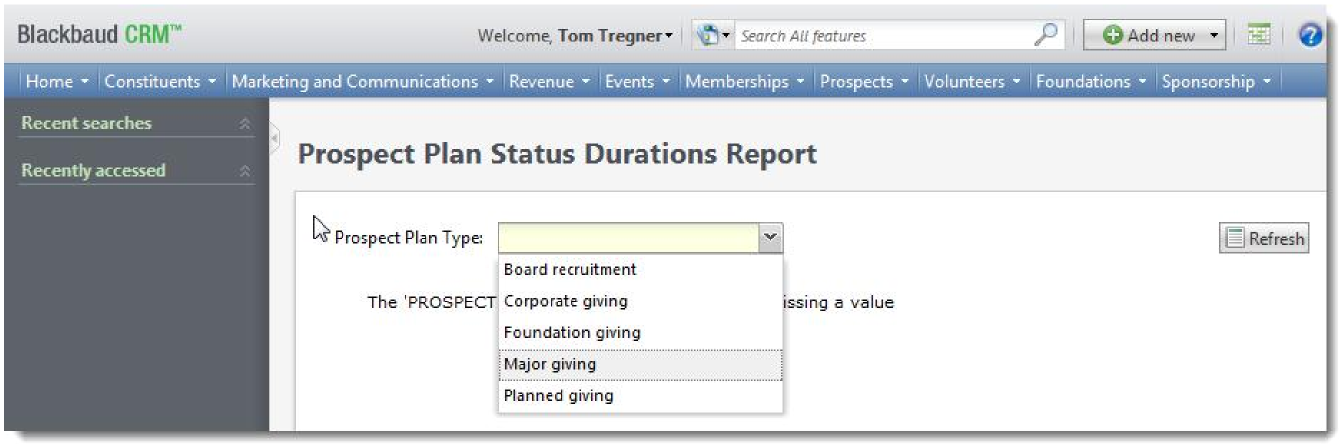




Prospect Plan Status Durations Report

Prospect Plan Type:

The 'PROSPECTPLANTYPECODEID' parameter is missing a value



Prospect Plan Status Durations Report

Prospect Plan Type:

The 'PROSPECT issing a value

To avoid the missing parameter message, in the Page Definition Spec:

```
<PageDefinitionSpec
    xmlns="bb_appfx_pagedefinition"
    xmlns:common="bb_appfx_commontypes"
    ID="681D2833-9F76-4547-81C0-8840A24ECC5E"
    Name="Prospect Plan Status Durations Report Page"
    Description="A page to display the Prospect Plan Status Durations Report"
    Author="Technical Training"
    >

  <PageHeader Caption="Prospect Plan Status Durations Report"  />

  <Tabs>
    <Tab ID="DA7C37C0-59B8-40F1-80E8-B22A3D5E02B9" Caption="Prospect Plan Status
Durations">

      <Sections>
        <Section ID="A62F6D4A-E4B9-4183-B19A-5B49DE17C741" Caption="Prospect
Plan Status Durations">
          <Report AutoLoad="false" ID="6C02DB4A-5032-4130-80BD-B99B0DCA192A"/>
        </Section>
      </Sections>
    </Tab>
  </Tabs>

</PageDefinitionSpec>
```

ProspectPlanStatusDurationsReportUIModel.vb

```
Namespace ProspectPlanStatusDurations

       Public Class ProspectPlanStatusDurationsReportUIModel

               Private Sub ProspectPlanStatusDurationsReportUIModel_Loaded(ByVal sender As
Object, ByVal e As Blackbaud.AppFx.UIModeling.Core.LoadedEventArgs) Handles
Me.Loaded

        End Sub

       End Class

End Namespace
```

ProspectPlanStatusDurationsReport.html

```
<div id="#MAP#ProspectPlanStatusDurationsReport">
  <table>
    <tr id="#MAP#PROSPECTPLANTYPECODEID_container">
      <td>
        <label id="#MAP#PROSPECTPLANTYPECODEID_caption" for="#MAP#-
PROSPECTPLANTYPECODEID_value"></label>
      </td>
      <td>
        <input id="#MAP#PROSPECTPLANTYPECODEID_value" type="text" />
      </td>
    </tr>
  </table>
  <!-- To define fields in multiple columns on the form, simply add/move the
fields to this div
        <div class="bbui-forms-fieldset-column">
            <table>
            </table>
        </div>
        -->
</div>
```

Reporting off the Warehouse

In this section, we discuss reports which query the data warehouse database. In Should the Report Query a Table or a View? on page 98 and Something is Missing from the Table or View on page 99 we cover some challenges. In Creating a Data Warehouse Version on page 100 and Extending the Warehouse with a Table to Extend the Fact on page 102, we describe the creation of a report and data warehouse extension to report using a query which is similar to our OLTP version. In Creating Another Data Warehouse Version on page 119 and Extending the Data Warehouse with New Tables and Views on page 121, we describe a more tailored extension and report.

Should the Report Query a Table or a View?

As of version 2.93 of ***Blackbaud CRM***, the data warehouse tables include prospect plan status information for interactions and prospect plans. But views in the warehouse do not. Views in Blackbaud Data Warehouse support access to warehouse data through a star schema. Stars for major giving and prospects functionality have not been created. There are three options for reporting from the warehouse for this situation.

1. Query Blackbaud Data Warehouse tables through the report
2. Extend Blackbaud Data Warehouse with views of the tables and query the views
3. Extend Blackbaud Data Warehouse with new tables, views of the tables, and query the views

The preferred method to query Blackbaud Data Warehouse is through the views which establish the star schema. If you report off of the tables directly, you run the risk of a breaking change to your report if those tables are changed. Similarly, if you extend the warehouse to create a view of the existing tables, changes to the existing tables may break the view extension and also break the report. However, in this situation, you could fix the view instead of the report.

In order to reduce the number of potential break points due to future changes in Blackbaud Data Warehouse, you could extend the warehouse with new tables and views to support prospect plan status.

To create an extension to Blackbaud Data Warehouse to support a view, you will minimally need a database revisions extension to add the view. An database revision extension and an ETL extension is necessary if you add a table. For information about how to extend Blackbaud Data Warehouse, see BBDW/OLAP Extensibility Model.

Note: Blackbaud Data Warehouse also supports OLAP extensions for the OLAP cube that is fed by the data warehouse. But OLAP reporting and extensions are not within the scope of this discussion.

Something is Missing from the Table or View

With our prospect plan stage durations example, a data warehouse query can be built which is very similar to the OLTP version. But some of the columns available in the OLTP database are not available in the data warehouse database. For example, the OLTP `INTERACTION` table has columns for actual start datetime and actual end datetime. But the fact and dimension tables only have one datetime column. That column corresponds to a column in the OLTP database which coalesces the actual start datetime and the expected start datetime. So that leaves us with some choices to make. Here are some options.

Use what is there: The only end datetime needed is the end datetime for the last step in the plan. And the interactions only have a day associated with them. So a useful metric can still be created. But it will not convey hours. In this case, we could adapt the query to use the same date for the one place where end datetime is used. But it would probably be better to rewrite the query altogether since the time parts are considered throughout. The metric would show zero days rather than one day in many situations.

Create a new table to extend the fact and a view to join the fact to the new table: This requires extending the data warehouse. But it is a fairly simple extension since we can base it on the existing table revision and SSIS package. It would allow us to model our data warehouse query on the OLTP query we already have. Unfortunately, the data is spread across a fact table and a dimension table. So we need to create an extra join to make this work. We can leave this to whoever uses the table or create a view to join the fact to the new table. We will see the benefit of placing the reporting burden on the data warehouse rather than the transactional database. But the query won't be more efficient.

Note: If we were extending the OLAP cube, we could create a fact extension to accomplish this.

Create new tables and views tailored to the reporting needs: This requires extending the data warehouse. It also requires thoughtful data modeling. And within this option, one needs to consider whether the extension will only support the specific reporting needs at hand or if there will be other reporting needs down the road. For example, with the prospect plan stage durations example, it is necessary to create sequences and many rows are eliminated in the course of the query. The extension could reflect this to make the report more efficient. But is would limit the functionality of the extension to support other reports.

Creating a Data Warehouse Version

We could reuse many of the artifacts we created for the OLTP version. But to keep both versions, it may work better to create a new Page Spec, Task Spec, and Report Spec. Since the process is the same, you can refer to Wiring up the Report on page 44. Firstly, we will show queries to the data warehouse database which are constructed in the same way as the OLTP version. There are three main issues for our prospect plan status example:

1. We must query the tables rather than the views. This is because the views do not contain enough information.
2. The tables themselves are missing a date column used in the transactional version. We can still create a useful metric without this column. But it would require revamping the approach. These sections highlight those locations in the queries: Overlapping Perspective which Mirrors the OLTP version on page 100 and Creating a Data Warehouse Version on page 100. In Extending the Warehouse with a Table to Extend the Fact on page 102, we describe the creation of an extension which allows us to mirror the OLTP version completely.
3. An extra join is required because information is spread across a fact and dimension table. In another section, Creating Another Data Warehouse Version on page 119 and Extending the Data Warehouse with New Tables and Views on page 121, we will explore a more tailored approach.

Overlapping Perspective which Mirrors the OLTP version

The warehouse has analogous tables for `INTERACTION`, `PROSPECTPLAN`, and `PROSPECTPLANSTATUS`. For the overlapping perspective, it is possible to create a very similar query. But the actual end datetime does not exist. So, the closest we can get is with `INTERACTIONDATE`, in the OLTP database and the data warehouse database, this is created through a `COALESCE` of the actual and expected datetimes. We will describe extending the data warehouse to overcome this. But for reference, this is what the query would look like `INTERACTIONDATE` was used for actual start datetime and actual end datetime. Again, if we were limited to just that datetime, it would be better to revamp the query to only consider days or to extend the warehouse to include actual start datetime and actual end datetime.

```
with [STEPS]
as (
       select i.[PROSPECTPLANSTATUSDIMID],
               i.[INTERACTIONDATE],
               min(i.[INTERACTIONDATE]) over (
                       partition by i.[PROSPECTPLANDIMID],
                       i.[PROSPECTPLANSTATUSDIMID]
                       ) as [FIRSTSTEPINSTAGEDATETIME],
               max(i.[INTERACTIONDATE]) over (
                       partition by i.[PROSPECTPLANDIMID],
                        i.[PROSPECTPLANSTATUSDIMID]
                        ) as [LASTSTEPINSTAGEDATETIME]
       from [BBDW].[FACT_INTERACTION] as i
       inner join [BBDW].[DIM_INTERACTION] as j on i.[INTERACTIONDIMID] = j.[INTERACTIONDIMID]
       where j.[ISINTERACTIONCOMPLETED] = 1
       )
select p.[PROSPECTPLANSTATUS] as [STAGENAME],
       avg(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [AVGSTAGEDURATION],
       min(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MINSTAGEDURATION],
       max(cast((s.[LASTSTEPINSTAGEDATETIME] - s.[FIRSTSTEPINSTAGEDATETIME]) as float)) as [MAXSTAGEDURATION]
from [STEPS] as s
inner join [BBDW].[DIM_PROSPECTPLANSTATUS] as p on s.[PROSPECTPLANSTATUSDIMID] = p.[PROSPECTPLANSTATUSDIMID]
where (s.[INTERACTIONDATE] = s.[LASTSTEPINSTAGEDATETIME])
group by p.[PROSPECTPLANSTATUS]
```

Extending the Warehouse with a Table to Extend the Fact

The tables used by our queries were described in Finding Data in the Data Warehouse Database on page 24. Those tables are insufficient for our design requirements because they do not include actual start datetime and actual end datetime. We don't want to alter that table because it is possible the table will be updated in a subsequent release of Blackbaud Data Warehouse. Instead, we are going to create a new table with the actual start datetime and actual end datetime which we can join to the existing fact table in queries.

Note: Another approach would be to replicate the `FACT_INTERACTION` table with the addition of those columns. This would have the advantage of avoiding an extra join in the report queries. But it would require more complex database revisions and ETL logic in the SSIS packages. Also, it increases risk because so many `INTERACTION` columns now require updates in two tables.

File for Revisions

If you don't already have a file for revisions, create an `XML` file with these contents:

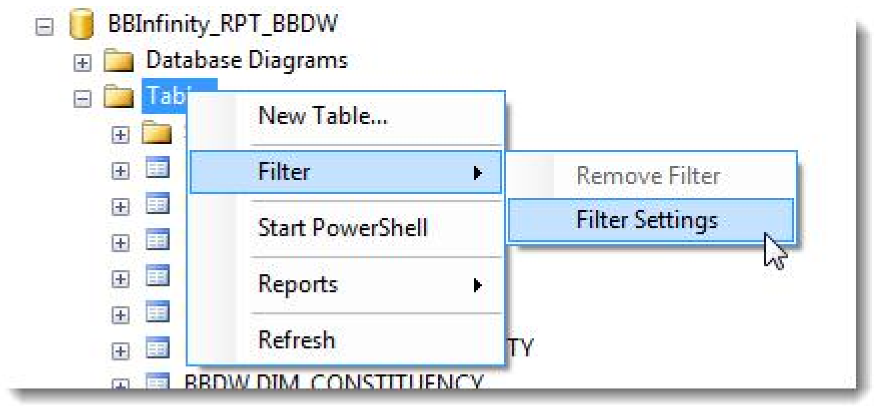
```
<?xml version="1.0" ?>
<DBRevisions xmlns="bb_appfx_dbrevisions">
       <DBRevision ID="1">
               <Comment>Extended Database Schema</Comment>
       </DBRevision>

       <DBRevision ID="5">
               <ExecuteSql>
                       <![CDATA[
if not exists (
               select *
               from sysobjects so
               where so.type = 'P'
                       and so.name = 'RESETETL_EXT'
               )
       exec sp_executesql N'create procedure BBDW.[RESETETL_EXT] as set nocount on;'
                       ]]>
               </ExecuteSql>
       </DBRevision>
```

```
    </DBRevisions>
```

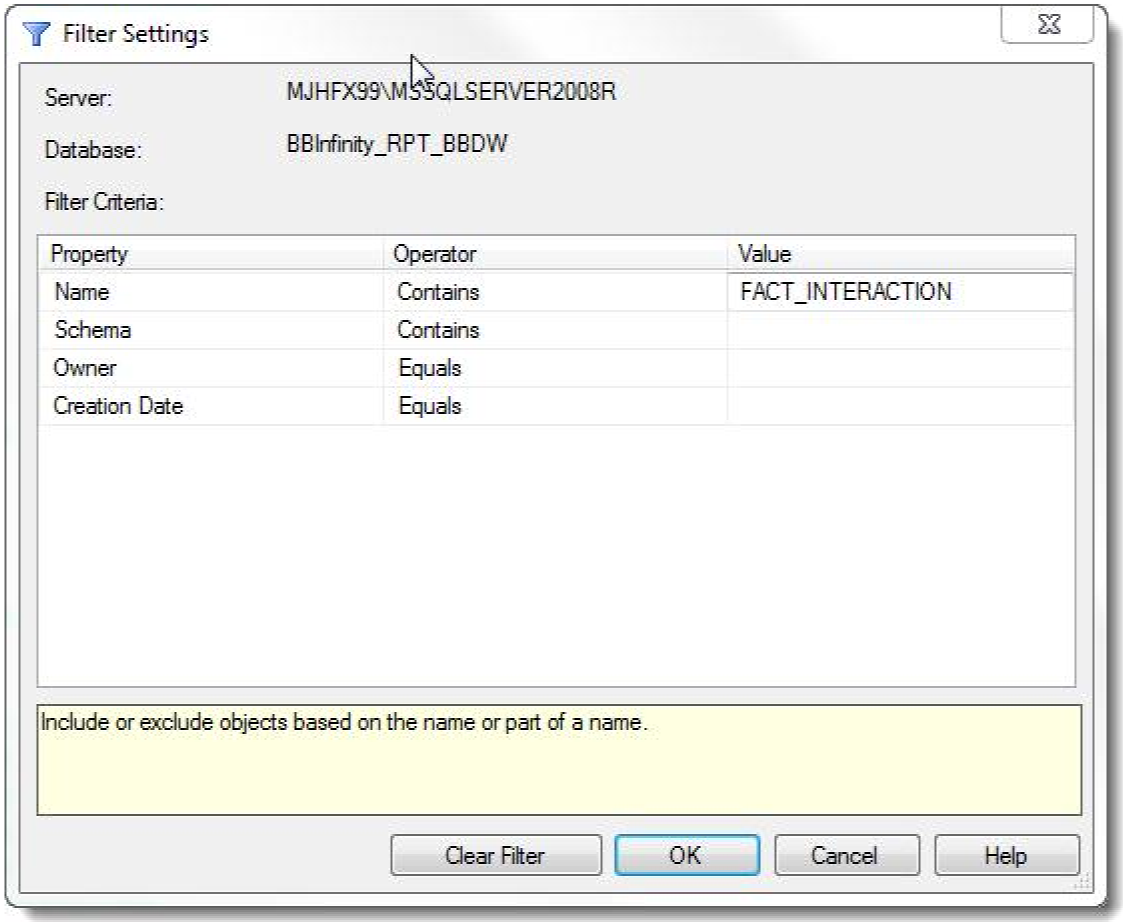
Name the file according to the naming convention for revisions extensions.

CREATE TABLE Revision

We will call the table `FACT_INTERACTIONACTUALTIMES_EXT`. One of the revisions we need is the revision which actually creates the table. Since we are replicating some of the `FACT_INTERACTION` table, we can grab some of that from ***SQL Server Management Studio***. After opening ***SSMS***, we can connect to the database and filter the tables for `FACT_INTERACTION`. To filter the tables for a database, right-click the **Tables** node for the database in Object Explorer and select **Filter** > **Filter Settings**.



In the **Name** field of the Filter Settings screen that appears, enter `FACT_INTERACTION` and click **OK**.



Server: MJHFX99\MSSQLSERVER2008R

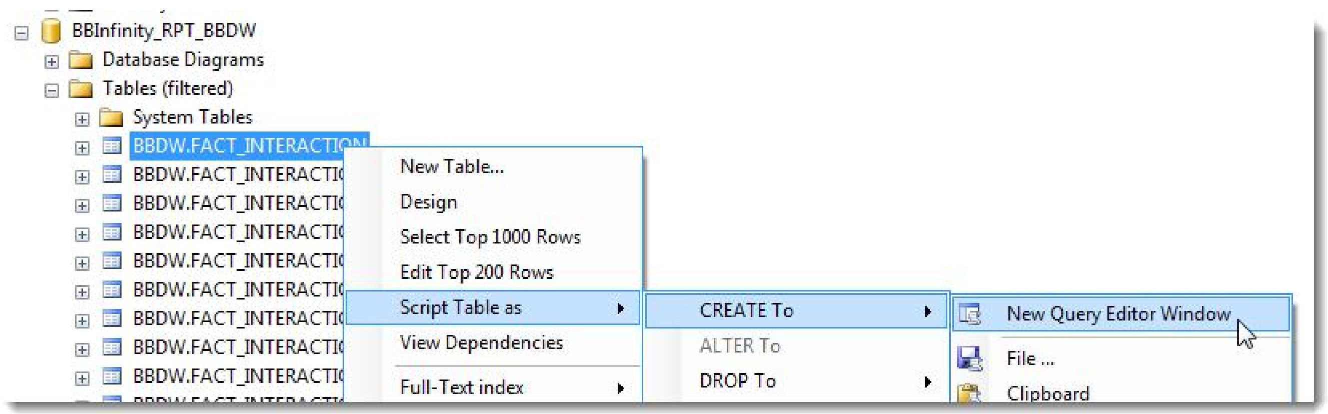
Database: BBInfinity_RPT_BBDW

Filter Criteria:

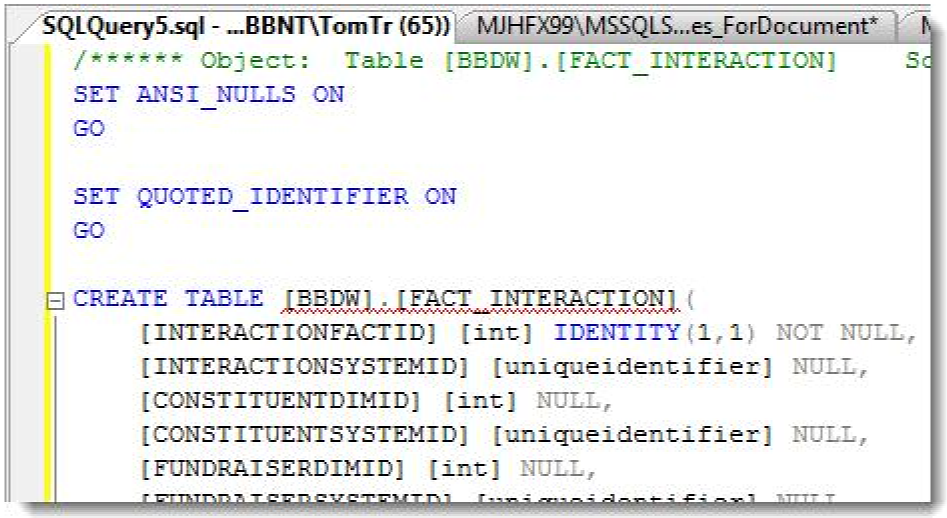
Property	Operator	Value
Name	Contains	FACT_INTERACTION
Schema	Contains	
Owner	Equals	
Creation Date	Equals	

Include or exclude objects based on the name or part of a name.

From the filtered list, right-click `FACT_INTERACTION` and select **Script Table as** > **CREATE To** > **New Query Editor Window**.



The part we need is the `CREATE TABLE` statement.


SQLQuery5.sql - ...BBNT\TomTr (65))
MJHFX99\MSSQLS...es_ForDocument*

```
/****** Object:  Table [BBDW].[FACT_INTERACTION]     Sc
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [BBDW].[FACT_INTERACTION](
    [INTERACTIONFACTID] [int] IDENTITY(1,1) NOT NULL,
    [INTERACTIONSYSTEMID] [uniqueidentifier] NULL,
    [CONSTITUENTDIMID] [int] NULL,
    [CONSTITUENTSYSTEMID] [uniqueidentifier] NULL,
    [FUNDRAISERDIMID] [int] NULL,
```

Copy this into the query editor.

```
CREATE TABLE [BBDW].[FACT_INTERACTION](
       [INTERACTIONFACTID] [int] IDENTITY(1,1) NOT NULL,
	[INTERACTIONSYSTEMID] [uniqueidentifier] NULL,
	[CONSTITUENTDIMID] [int] NULL,
	[CONSTITUENTSYSTEMID] [uniqueidentifier] NULL,
	[FUNDRAISERDIMID] [int] NULL,
	[FUNDRAISERSYSTEMID] [uniqueidentifier] NULL,
	[INTERACTIONDATEDIMID] [int] NULL,
	[INTERACTIONDATE] [datetime] NULL,
	[INTERACTIONDIMID] [int] NULL,
	[EVENTDIMID] [int] NULL,
	[PROSPECTPLANDIMID] [int] NULL,
	[PLANOUTLINESTEPDIMID] [int] NULL,
	[PROSPECTPLANSTATUSDIMID] [int] NULL,
	[FUNDINGREQUESTDIMID] [int] NULL,
	[FUNDINGREQUESTOUTLINESTEPDIMID] [int] NULL,
	[INTERACTIONLOOKUPID] [nvarchar](100) NULL,
	[INTERACTIONOBJECTIVE] [nvarchar](100) NULL,
	[ISINCLUDED] [bit] NULL,
	[ETLCONTROLID] [int] NULL,
	[SOURCEDIMID] [int] NULL,
 CONSTRAINT [PK_FACT_INTERACTION] PRIMARY KEY CLUSTERED
(
	[INTERACTIONFACTID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  =
ON) ON [BBRPT_FACTGROUP]
) ON [BBRPT_FACTGROUP]
```

Firstly, we already have a `FACT_INTERACTION table`, so we need to change that name. And we don't need the contents of the `WITH` clause. But we should leave the `CONSTRAINT` but change the name. and the `ON` for the fact group. Let's also reformat those keywords to lower-case since that is how the rest of our Transact-SQL looks. As for the columns:

- Add the columns for actual start datetime and actual end datetime.
- Change `INTERACTIONFACTID` and `INTERACTIONSYSTEMID` to `INTERACTIONACTUALTIMESFACTID` and `INTER-ACTIONACTUALTIMESSYSTEMID`.
- Retain `ISINCLUDED`, `ETLCONTROLID`, and `SOURCEDIMID`.
- Remove all other fields.

```
  <DBRevision ID="10">
    <ExecuteSql>
```

```
      <![CDATA[
create table [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT] (
       [INTERACTIONACTUALTIMESFACTID] [int] IDENTITY(1, 1) not null,
       [INTERACTIONACTUALTIMESSYSTEMID] [uniqueidentifier] null,
       [ACTUALSTARTDATETIME] [datetime] null,
       [ACTUALSTARTDATEDIMID] [int] null,
       [ACTUALENDDATETIME] [datetime] null,
       [ACTUALENDDATEDIMID] [int] null,
       [ISINCLUDED] [bit] null,
       [ETLCONTROLID] [int] null,
       [SOURCEDIMID] [int] null,
       constraint [PK_FACT_INTERACTIONACTUALTIMES_EXT] primary key clustered ([INTERACTIONACTUALTIMESFACTID] asc)
       ) on [BBRPT_FACTGROUP]
]]>
    </ExecuteSql>
  </DBRevision>
```

Drop and Create Indexes Revision for Extension Table

There should be nonclustered indexes on the date dimension columns. These need to be dropped and added as a part of the ETL process. This creates a stored procedure for that.

```
  <DBRevision ID="15">
    <ExecuteSql>
      <![CDATA[
create procedure [BBDW].[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_INDICES] @CREATE_OR_DROP bit --1 to create, 0 to
drop.
as
set nocount on;

if @CREATE_OR_DROP is null
       raiserror (
                       '@CREATE_OR_DROP must be 1 or 0 in [BBDW].[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_INDICES]',
                       16,
                       10
                       );

if @CREATE_OR_DROP = 1
begin
```

```
       --create
       if [BBDW].[UFN_INDEXEXISTS]('IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALSTARTDATETIMEDIMID') = 0
               create nonclustered index [IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALSTARTDATETIMEDIMID] on [BBDW].[FACT_INTER-
ACTIONACTUALTIMES_EXT] ([ACTUALSTARTDATEDIMID]) on [BBRPT_DIMIDXGROUP]

       if [BBDW].[UFN_INDEXEXISTS]('IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALENDDATETIMEDIMID') = 0
               create nonclustered index [IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALENDDATETIMEDIMID] on [BBDW].[FACT_INTER-
ACTIONACTUALTIMES_EXT] ([ACTUALENDDATEDIMID]) on [BBRPT_DIMIDXGROUP]
end
else
begin
       --drop
       if [BBDW].[UFN_INDEXEXISTS]('IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALSTARTDATETIMEDIMID') = 1
               drop index [IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALSTARTDATETIMEDIMID] on [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT];

       if [BBDW].[UFN_INDEXEXISTS]('IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALENDDATETIMEDIMID') = 1
               drop index [IX_FACT_INTERACTIONACTUALTIMES_EXT_ACTUALENDDATETIMEDIMID] on [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT];
end
]]>
    </ExecuteSql>
  </DBRevision>
```

CREATE TABLE Revision for Staging Table

The staging table has the same columns. Copy the revision for the main table and change the `DBRevision ID` and add `_STAGE` where the table name is used. This includes the constraint.

```
  <DBRevision ID="20">
    <ExecuteSql>
      <![CDATA[
create table [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT_STAGE] (
       [INTERACTIONACTUALTIMESSYSTEMID] [uniqueidentifier] null,
       [ACTUALSTARTDATETIME] [datetime] null,
       [ACTUALSTARTDATEDIMID] [int] null,
       [ACTUALENDDATETIME] [datetime] null,
       [ACTUALENDDATEDIMID] [int] null,
       [ISINCLUDED] [bit] null,
       [ETLCONTROLID] [int] null,
       [SOURCEDIMID] [int] null
       ) on [BBRPT_STAGEGROUP]
]]>
    </ExecuteSql>
  </DBRevision>
```

Drop and Create Indexes Revision for Staging Table

There should be an index on the `INTERACTIONACTUALTIMESSYSTEMID`. These need to be dropped and added as a part of the ETL process. This creates a stored procedure for that.

```
  <DBRevision ID="25">
    <ExecuteSql>
      <![CDATA[
create procedure [BBDW].[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INDICES] @CREATE_OR_DROP bit --1 to 
create, 0 to drop.
as
set nocount on;

if @CREATE_OR_DROP is null
       raiserror (
                        '@CREATE_OR_DROP must be 1 or 0 in [BBDW].[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INDICES]',
                        16,
                        10
                        );

if @CREATE_OR_DROP = 1
begin
       --create
       if [BBDW].[UFN_INDEXEXISTS]('IX_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INTERACTIONACTUALTIMESSYSTEMID') = 0
               create nonclustered index [IX_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INTERACTIONACTUALTIMESSYSTEMID] on [BBDW].[FACT_
INTERACTIONACTUALTIMES_EXT_STAGE] ([INTERACTIONACTUALTIMESSYSTEMID]) on [BBRPT_STAGEGROUP]
end
else
begin
       --drop
       if [BBDW].[UFN_INDEXEXISTS]('IX_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INTERACTIONACTUALTIMESSYSTEMID') = 1
               drop index [IX_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INTERACTIONACTUALTIMESSYSTEMID] on [BBDW].[FACT_INTER-
ACTIONACTUALTIMES_EXT_STAGE];
end
] ]>
    </ExecuteSql>
  </DBRevision>
```

Truncate Tables and Drop Indexes Revision

```
  <DBRevision ID="30">
    <ExecuteSql>
      <![CDATA[
alter procedure BBDW.[RESETETL_EXT]
as
set nocount on;

truncate table BBDW.[FACT_INTERACTIONACTUALTIMES_EXT];

exec BBDW.[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_INDICES] 0;

truncate table BBDW.[FACT_INTERACTIONACTUALTIMES_EXT_STAGE];

exec BBDW.[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INDICES] 0;
]]>
    </ExecuteSql>
  </DBRevision>
```

CREATE VIEW Revision

Since the data warehouse creates a star schema using views, it is helpful to add a view to support that.

```
  <DBRevision ID="35">
    <ExecuteSql>
      <![CDATA[
create view [BBDW].[v_FACT_INTERACTIONACTUALTIMES_EXT]
as
select [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT].[INTERACTIONACTUALTIMESFACTID],
       [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT].[INTERACTIONACTUALTIMESSYSTEMID],
       [BBDW].[FACT_INTERACTION].[CONSTITUENTDIMID],
       [BBDW].[FACT_INTERACTION].[CONSTITUENTSYSTEMID],
       [BBDW].[FACT_INTERACTION].[FUNDRAISERDIMID],
       [BBDW].[FACT_INTERACTION].[FUNDRAISERSYSTEMID],
       [BBDW].[FACT_INTERACTION].[INTERACTIONDATEDIMID],
       [BBDW].[FACT_INTERACTION].[INTERACTIONDATE],
       [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT].[ACTUALSTARTDATETIME],
       [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT].[ACTUALSTARTDATEDIMID],
       [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT].[ACTUALENDDATETIME],
       [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT].[ACTUALENDDATEDIMID],
       [BBDW].[FACT_INTERACTION].[INTERACTIONDIMID],
       [BBDW].[FACT_INTERACTION].[EVENTDIMID],
       [BBDW].[FACT_INTERACTION].[PROSPECTPLANDIMID],
       [BBDW].[FACT_INTERACTION].[PLANOUTLINESTEPDIMID],
       [BBDW].[FACT_INTERACTION].[PROSPECTPLANSTATUSDIMID],
       [BBDW].[FACT_INTERACTION].[FUNDINGREQUESTDIMID],
       [BBDW].[FACT_INTERACTION].[FUNDINGREQUESTOUTLINESTEPDIMID],
       [BBDW].[FACT_INTERACTION].[INTERACTIONLOOKUPID],
       [BBDW].[FACT_INTERACTION].[INTERACTIONOBJECTIVE]
from [BBDW].[FACT_INTERACTION]
left join [BBDW].[FACT_INTERACTIONACTUALTIMES_EXT] on [BBDW].[FACT_INTERACTION].[INTERACTIONSYSTEMID] = [BBDW].[FACT_
INTERACTIONACTUALTIMES_EXT].[INTERACTIONACTUALTIMESSYSTEMID]
]]>
    </ExecuteSql>
  </DBRevision>
```

Map Source to Target Revision

This revision adds `MS_Description` comments to the table.

```
  <DBRevision ID="40">
    <ExecuteSql>
      <![CDATA[
exec BBDW.USP_SCHEMA_TABLE_SETTABLECOMMENT 'FACT_INTERACTIONACTUALTIMES_EXT',
       'The Interaction Actual Times fact contains actual start and end datetimes for interactions.
        The table can be joined to the Interaction fact which relates information to constituent interactions.
         The v_FACT_INTERACTIONACTUALTIMES_EXT view does this.';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'INTERACTIONACTUALTIMESFACTID',
       'Surrogate key for Interaction fact.';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'INTERACTIONACTUALTIMESSYSTEMID',
       'dbo.[INTERACTION].[INTERACTIONID]';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'ACTUALSTARTDATETIME',
       'dbo.[INTERACTION].[ACTUALSTARTDATETIME]';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'ACTUALENDDATETIME',
       'dbo.[INTERACTION].[ACTUALENDDATETIME]';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'ISINCLUDED',
       'Flag indicating when data should be included in results.';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'ETLCONTROLID',
       'ID generated through the ETL process';

exec BBDW.USP_SCHEMA_TABLE_SETCOLUMNCOMMENT 'MS_Description',
       'FACT_INTERACTIONACTUALTIMES_EXT',
       'SOURCEDIMID',
       'Source system used';

]]>
    </ExecuteSql>
  </DBRevision>
```

Create the Project and Compile the Revisions to a DLL

If you don't already have a project for revisions extensions, create one.

Create the ETL for the New Table

1. Open Business Intelligence Development Studio.
2. Open the template `DTSX` file, `BBDW_FACT_TEMPLATE`.
   a. Click **File** > **Open** > **File**.
   b. Browse to `C:\Program Files\Blackbaud\bbappfx\MSBuild\Datamarts\BBDW\SSIS\BBDW_FACT_TEM-PLATE.dtsx`.

      Note: The location may be different for your installation.

   c. Click **Open**.
3. Save a copy in the `Extend\SSIS` folder.
   a. Click **File** > **Save Copy of BBDW_FACT_TEMPLATE.dtsx As...**
   b. The Save Copy of Package screen appears.
   c. From **Package location**, select **File System**.
   d. Click the ellipses button next to the field for Package path. The Save Package To File screen appears.
   e. Browse to `C:\Program Files\Blackbaud\bbappfx\MSBuild\Datamarts\BBDW\Extend\SSIS`.

      Note: The location may be different for your installation.

   f. Change the name to `BBDW_FACT_INTERACTIONACTUALTIMES_EXT`.
   g. Click **Save**.
4. Change the Truncate Staging task.

On the Control Flow tab for the package designer, double-click the Truncate Staging task in the Load Rows sequence. The Execute SQL Task Editor screen appears.

From **General** > **SQL Statement** > **SQLStatement**, change the query to:

```
truncate table BBDW.[FACT_INTERACTIONACTUALTIMES_EXT_STAGE];

exec BBDW.[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INDICES] 0;
```

Click **OK**.

5. Change the Load Changed Rows task.

Note: If you see a red x, it may be because the Connection Mangers are not configured for your databases. To fix, this click the `BBETL_DB_CONN_DW` and `BBETL_DB_CONN_OLTP` connection managers on the Connection Managers tab and reconfigure them.

   a. On the Control Flow tab for the package designer, double-click the Load Changed Rows task. The Data Flow appears.

      Double-click **New and changed rows from OLTP**. The OLE DB Source Editor appears.

      Change the **SQL command text** to:

```
select
  i.[ID] as [INTERACTIONACTUALTIMESSYSTEMID],
  i.[ACTUALSTARTDATETIME] as [ACTUALSTARTDATETIME],
  i.[ACTUALENDDATETIME] as [ACTUALENDDATETIME],
  1 as [ISINCLUDED]
from dbo.[INTERACTION] as i
where  (i.[DATECHANGED] > ? and i.[DATECHANGED] <= ? )
```

   b. Double-click the Date Dims Data Flow Component. The Restore Invalid Column References Editor appears.

      Remove all of the invalid references and click **OK**.

      Double-click the Date Dims Data Flow Component again. The Derived Column Transformation Editor appears.

      Copy the Expression in the grid:

```
ISNULL(INTERACTIONRESPONSEDATE) ? 0 : YEAR(INTERACTIONRESPONSEDATE) * 10000 + MONTH(INTER-
ACTIONRESPONSEDATE) * 100 + DAY(INTERACTIONRESPONSEDATE)
```

Add two new derived columns:

```
ACTUALSTARTDATEDIMID

ACTUALENDDATEDIMID
```

Adjust the expression you copied for each of these and paste the revised expressions into the Expression fields.

```
ISNULL(ACTUALSTARTDATETIME) ? 0 : YEAR(ACTUALSTARTDATETIME) * 10000 + MONTH(ACTUALSTARTDATETIME) * 100 +
DAY(ACTUALSTARTDATETIME)

ISNULL(ACTUALENDDATETIME) ? 0 : YEAR(ACTUALENDDATETIME) * 10000 + MONTH(ACTUALENDDATETIME) * 100 + DAY
(ACTUALENDDATETIME)
```

c. Remove the Check Dates Data Flow Component. Click Check Dates and press `Delete`.

d. Remove the Lookup Response Data Flow Component. Click Lookup Response and press `Delete`.

e. Remove the invalid column references in Stage Rows from the template.

   Double-click the Stage Rows Data Flow Component. The Restore Invalid Column References Editor screen appears.

   Delete the template columns. Select all of the rows and from **Column mapping** option for selected rows, select **<Delete invalid column reference>**.

   Click **OK**.

f. Double-click the Stage Rows Data Flow Component again. The OLE DB Destination Editor appears.

   Ensure the `BBETL_DB_CONN_DW` connection manager is selected under OLE DB connection manager.

   From **Name of the table or the view**, select `[BBDW].[FACT_INTERACTIONACTUALTIMES_EXT_STAGE]`.

   Click **Mappings**. The mapping should be established for every column except `INTERACTIONACTUALTIMESFACTID`.

   Click **OK**.

g. Save the package.

6. Change the Adding Staging Indices task.

a. Return to the Control Flow tab.

b. Double-click the Adding Staging Indices task.

c. From **General** > **SQL Statement** > **SQLStatement**, change the query to:

```
exec BBDW.[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_STAGE_INDICES] 1;
```

d. Click **OK**.

7. Change the Upsert task.

a. Connect Adding Staging Indices task to Upsert task

b. Double-click Upsert. The Execute SQL Task editor screen appears.

c. From **General** > **SQL Statement** > **SQLStatement**, change the query to:

```
declare @COUNTS table (
       [ACTION] varchar(28),
       [INSERTED] int,
       [UPDATED] int
       );

merge BBDW.[FACT_INTERACTIONACTUALTIMES_EXT] as t
using (
       select i.[INTERACTIONACTUALTIMESSYSTEMID],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALSTARTDATEDIMID],
               i.[ACTUALENDDATETIME],
               i.[ACTUALENDDATEDIMID],
               i.[ISINCLUDED],
               i.[ETLCONTROLID],
               i.[SOURCEDIMID]
       from BBDW.[FACT_INTERACTIONACTUALTIMES_EXT_STAGE] as i
       ) as s
       on (t.[INTERACTIONACTUALTIMESSYSTEMID] = s.[INTERACTIONACTUALTIMESSYSTEMID])
```

```
when not matched by target
       then
               insert (
                       [INTERACTIONACTUALTIMESSYSTEMID],
                       [ACTUALSTARTDATETIME],
                       [ACTUALSTARTDATEDIMID],
                       [ACTUALENDDATETIME],
                       [ACTUALENDDATEDIMID],
                       [ISINCLUDED],
                       [ETLCONTROLID],
                       [SOURCEDIMID]
                       )
               values (
                       s.[INTERACTIONACTUALTIMESSYSTEMID],
                       s.[ACTUALSTARTDATETIME],
                       s.[ACTUALSTARTDATEDIMID],
                       s.[ACTUALENDDATETIME],
                       s.[ACTUALENDDATEDIMID],
                       s.[ISINCLUDED],
                       s.[ETLCONTROLID],
                       s.[SOURCEDIMID]
                       )
when matched
       then
               update
               set t.[INTERACTIONACTUALTIMESSYSTEMID] = s.[INTERACTIONACTUALTIMESSYSTEMID],
                       t.[ACTUALSTARTDATETIME] = s.[ACTUALSTARTDATETIME],
                       t.[ACTUALSTARTDATEDIMID] = s.[ACTUALSTARTDATEDIMID],
                       t.[ACTUALENDDATEDIMID] = s.[ACTUALENDDATEDIMID],
                       t.[ISINCLUDED] = s.[ISINCLUDED],
                       t.[ETLCONTROLID] = s.[ETLCONTROLID],
                       t.[SOURCEDIMID] = s.[SOURCEDIMID]
output $action,
       case
               when deleted.[ETLCONTROLID] is null
                       then 1
               else 0
               end,
       case
               when deleted.[ETLCONTROLID] is not null
```

```
                            then 1
                    else 0
                    end
        into @COUNTS;

        select count(*) as [TOTAL],
               isnull(sum([INSERTED]), 0) as [INSERTED],
               isnull(sum([UPDATED]), 0) as [UPDATED]
        from @COUNTS
```

   d. Click **OK**.

8. Change the Adding Indices task.

   a. Double-click Adding Indices. The Execute SQL Task editor screen appears.

   b. From **General** > **SQL Statement** > **SQLStatement**, change the statement to:

```
exec BBDW.[CREATE_OR_DROP_FACT_INTERACTIONACTUALTIMES_EXT_INDICES] 1;
```

9. Adjust the package properties.

   a. Go to the Package Explorer tab.

   b. Right-click the package and select **Properties**.

   c. From **ID**, click the drop-down and select **Generate New ID**.

   d. Change the name to `BBDW_FACT_INTERACTIONACTUALTIMES_EXT`.

   e. Save the package file.

10. Copy the package to the `SSIS` folder for extensions:

```
C:\Program Files\Blackbaud\bbappfx\MSBuild\Datamarts\BBDW\Extend\SSIS
```

11. Update the package manifest (`BBDW_PackageList_EXT.txt`):

```
C:\Program Files\Blackbaud\bbappfx\MSBuild\Datamarts\BBDW\Extend\SSIS
```

```
"Enabled","Package"
"1","BBDW_FACT_INTERACTIONACTUALTIMES_EXT.dtsx"
```

Deploy and Refresh the Warehouse

For information about how to deploy and refresh the warehouse, see the online guides at Blackbaud Data Warehouse.

Change the Report

As discussed in Creating a Data Warehouse Version on page 100, the query on which the report relies can follow the same structure as the transactional database version. The difference here is the data warehouse version queries the a view created through data warehouse extensions and the stored procedures used by the RDL dataset are created through these revisions instead of a through the Report Spec.

Creating Another Data Warehouse Version

The goal of this version is to migrate the granularity logic in the stored procedures for the other versions of the report from the stored procedures to the ETL process. The load step in the SSIS packages look similar to the common table expressions in the OLTP version of the stored procedure for the report. What follows is the Transact-SQL for the load step for the stage occurrence package. The SSIS package will convert the data gathered from this query into rows for new tables defined in a set of data warehouse revisions extensions.

```
with [STEPS]
as (
       select row_number() over (
                       order by i.[PROSPECTPLANID],
                               i.[ACTUALSTARTDATETIME]
                       ) as [ALLSTEPSEQUENCENUMBER],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALENDDATETIME],
               i.[PROSPECTPLANID],
               max(i.[ACTUALENDDATETIME]) over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
               i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
       ),
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
               s.[ACTUALENDDATETIME],
               r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
               s.[PROSPECTPLANSTATUSCODEID],
               r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
               s.[PROSPECTPLANID],
               s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as t on s.[ALLSTEPSEQUENCENUMBER] + 1 = t.[ALLSTEPSEQUENCENUMBER]
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.[ALLSTEPSEQUENCENUMBER]
       ),
[STAGEOCCURRENCES]
as (
       select row_number() over (
                       order by sofp.[PROSPECTPLANID],
                               sofp.[ACTUALSTARTDATETIME]
                       ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
               sofp.[ACTUALSTARTDATETIME],
               sofp.[ACTUALENDDATETIME],
               sofp.[LASTSTEPINPLANENDDATETIME],
               sofp.[PROSPECTPLANID],
               sofp.[PROSPECTPLANSTATUSCODEID],
               pl.[PROSPECTPLANTYPECODEID],
               pl.[PROSPECTID]
       from [STAGEOCCURRENCESFIRSTPASS] as sofp
       inner join [PROSPECTPLAN] pl on sofp.[PROSPECTPLANID] = pl.[ID]
       where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
               or (
                       (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PREVIOUSSTEPPROSPECTPLANSTATUSCODEID])
                       and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                       )
       )
select so1.[ACTUALSTARTDATETIME] as [STARTDATETIME],
       so1.[ACTUALENDDATETIME] as [ENDDATETIME],
       "STAGEOCCURRENCEDURATION" = case
               when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                       then so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME]
               when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
```

```
                        then so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME]
                end,
        so1.[PROSPECTPLANID] as [PROSPECTPLANSYSTEMID],
        so1.[PROSPECTPLANSTATUSCODEID] as [PROSPECTPLANSTATUSSYSTEMID],
        so1.[PROSPECTPLANTYPECODEID] as [PROSPECTPLANTYPESYSTEMID],
        so1.[PROSPECTID] as [CONSTITUENTSYSTEMID],
        1 as ISINCLUDED
from [STAGEOCCURRENCES] as so1
left join [STAGEOCCURRENCES] as so2 on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.ALLSTAGEOCCURRENCESSEQUENCENUMBER
```

Extending the Data Warehouse with New Tables and Views

The goal of these extensions is to create structures which represent prospect plan stages and prospect plan stage occurrences. The overlapping perspective and the nonconsecutive perspective aggregates can be calculated from a table which contains rows which each represent a stage in a prospect plan. The consecutive perspective aggregates can be calculated from a table which contains rows which each represent a stage occurrence. Each row should include duration information for the perspectives in addition to the start and end times and dimension IDs for prospect plans and constituents.

Note: It is possible to implement the stored procedures for the reports through revisions also. The sample implements them from the Report Spec. The deciding factor was the ability to update the stored procedure through loading the spec as opposed to deploying the data warehouse. However, loading the stored procedures through revisions would overcome the issue caused when the spec loading mechanism overrides the data source defined in the RDL file. So if you want to use data sets with different data sources in your RDL file, loading the stored procedures accessed by those data sets through revisions is a way to avoid configuring the data sources in Reporting Services after loading the spec.

Create Table Revision for Stage Occurrence (used by Consecutive Perspective)

```
          <DBRevision ID="10">
            <ExecuteSql>
              <![CDATA[
create table [BBDW].[FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT] (
        [PROSPECTPLANSTAGEOCCURRENCEFACTID] [int] IDENTITY(1, 1) not null,
        [STARTDATETIME] [datetime] null,
        [STARTDATEDIMID] [int] null,
        [ENDDATETIME] [datetime] null,
        [ENDDATEDIMID] [int] null,
        [STAGEOCCURRENCEDURATION] [datetime] null,
        [PROSPECTPLANSYSTEMID] [uniqueidentifier] null,
        [PROSPECTPLANDIMID] [int] null,
        [CONSTITUENTSYSTEMID] [uniqueidentifier] null,
        [CONSTITUENTDIMID] [int] null,
        [PROSPECTPLANTYPESYSTEMID] [uniqueidentifier] null,
        [PROSPECTPLANSTATUSSYSTEMID] [uniqueidentifier] null,
        [PROSPECTPLANSTATUSDIMID] [int] null,
        [ISINCLUDED] [bit] null,
        [ETLCONTROLID] [int] null,
        [SOURCEDIMID] [int] null,
        constraint [PK_FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT] primary key clustered ([PROSPECTPLANSTAGEOCCURRENCEFACTID] asc)
        ) on [BBRPT_FACTGROUP]
]]>
            </ExecuteSql>
          </DBRevision>
```

Create Table Revision for Stage (used by Overlapping and Nonconsecutive Perspectives)

```
  <DBRevision ID="15">
    <ExecuteSql>
      <![CDATA[
create table [BBDW].[FACT_PROSPECTPLANSTAGE_EXT] (
       [PROSPECTPLANSTAGEFACTID] [int] IDENTITY(1, 1) not null,
       [STARTDATETIME] [datetime] null,
       [STARTDATEDIMID] [int] null,
       [ENDDATETIME] [datetime] null,
       [ENDDATEDIMID] [int] null,
       [STAGEDURATIONOVERLAPPING] [datetime] null,
       [STAGEDURATIONNONCONSECUTIVE] [datetime] null,
       [PROSPECTPLANSYSTEMID] [uniqueidentifier] null,
       [PROSPECTPLANDIMID] [int] null,
       [CONSTITUENTSYSTEMID] [uniqueidentifier] null,
       [CONSTITUENTDIMID] [int] null,
       [PROSPECTPLANTYPESYSTEMID] [uniqueidentifier] null,
       [PROSPECTPLANSTATUSSYSTEMID] [uniqueidentifier] null,
       [PROSPECTPLANSTATUSDIMID] [int] null,
       [ISINCLUDED] [bit] null,
       [ETLCONTROLID] [int] null,
       [SOURCEDIMID] [int] null,
       constraint [PK_FACT_PROSPECTPLANSTAGE_EXT] primary key clustered ([PROSPECTPLANSTAGEFACTID] asc)
       ) on [BBRPT_FACTGROUP]
]]>
```

Other Items to Support the Stage and Stage Occurrence Tables

As with other data warehouse revisions which create tables, there should be a staging table, stored procedures for indexes, table truncation, and MS_ Description comments for mapping. The revisions file should include each of these. Also, a view of each table is desirable.

But the bulk of the work comes with the SSIS packages to perform the ETL. The ETL process for these tables will be more complex than with the previous example. This is because the stage and stage occurrence tables have rows which are based on multiple rows in the Interaction table in the OLTP database. For the previous versions of the report in this document, these transformations were performed by the stored procedures used by the report. The idea behind these extensions is to transfer that workload for the report's stored procedures to the ETL process. The report stored procedures will then be simplified to calculating the aggregates and not performing the prerequisite transformations of the Interaction table.

Fortunately, we have already worked through this logic in the course of building the previous report version's stored procedures. So to create the SSIS packages for the Stage and Stage Occurrence tables, we can transfer that logic to the SSIS packages.

BBDW_FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT.dtsx

Here are is selected information from the ETL for the stage occurrence table:

Truncate Staging SQLStatement

```
truncate table BBDW.[FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT_STAGE];

exec BBDW.[CREATE_OR_DROP_FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT_STAGE_INDICES] 0;
```

New and Changed Rows from OLTP SQL command text

```
with [STEPS]
as (
       select row_number() over (
                       order by i.[PROSPECTPLANID],
                               i.[ACTUALSTARTDATETIME]
                       ) as [ALLSTEPSEQUENCENUMBER],
               i.[ACTUALSTARTDATETIME],
               i.[ACTUALENDDATETIME],
               i.[PROSPECTPLANID],
               max(i.[ACTUALENDDATETIME])
                       over (partition by i.[PROSPECTPLANID]) as [LASTSTEPINPLANENDDATETIME],
               i.[PROSPECTPLANSTATUSCODEID]
       from [INTERACTION] as i
       inner join [PROSPECTPLAN] pl on i.[PROSPECTPLANID] = pl.[ID]
       where i.[COMPLETED] = 1
       ),
[STAGEOCCURRENCESFIRSTPASS]
as (
       select s.[ACTUALSTARTDATETIME],
               s.[ACTUALENDDATETIME],
               r.[PROSPECTPLANSTATUSCODEID] as [PREVIOUSSTEPPROSPECTPLANSTATUSCODEID],
               s.[PROSPECTPLANSTATUSCODEID],
               r.[PROSPECTPLANID] as [PREVIOUSSTEPPROSPECTPLANID],
               s.[PROSPECTPLANID],
               s.[LASTSTEPINPLANENDDATETIME]
       from [STEPS] as s
       left join [STEPS] as t on s.[ALLSTEPSEQUENCENUMBER] + 1 = t.[ALL-
STEPSEQUENCENUMBER]
       left join [STEPS] as r on s.[ALLSTEPSEQUENCENUMBER] - 1 = r.
[ALLSTEPSEQUENCENUMBER]
       ),
[STAGEOCCURRENCES]
as (
       select row_number() over (
                       order by sofp.[PROSPECTPLANID],
                               sofp.[ACTUALSTARTDATETIME]
                       ) as [ALLSTAGEOCCURRENCESSEQUENCENUMBER],
               sofp.[ACTUALSTARTDATETIME],
               sofp.[ACTUALENDDATETIME],
               sofp.[LASTSTEPINPLANENDDATETIME],
               sofp.[PROSPECTPLANID],
               sofp.[PROSPECTPLANSTATUSCODEID],
               pl.[PROSPECTPLANTYPECODEID],
               pl.[PROSPECTID]
       from [STAGEOCCURRENCESFIRSTPASS] as sofp
       inner join [PROSPECTPLAN] pl on sofp.[PROSPECTPLANID] = pl.[ID]
       where sofp.[PROSPECTPLANID] <> sofp.[PREVIOUSSTEPPROSPECTPLANID]
               or (
                       (sofp.[PROSPECTPLANSTATUSCODEID] <> sofp.[PRE-
VIOUSSTEPPROSPECTPLANSTATUSCODEID])
                       and (sofp.[PROSPECTPLANID] = sofp.[PREVIOUSSTEPPROSPECTPLANID])
                       )
       )
select so1.[ACTUALSTARTDATETIME] as [STARTDATETIME],
       so1.[ACTUALENDDATETIME] as [ENDDATETIME],
       "STAGEOCCURRENCEDURATION" = case
               when so1.[ACTUALENDDATETIME] <> so1.[LASTSTEPINPLANENDDATETIME]
                       then so2.[ACTUALSTARTDATETIME] - so1.[ACTUALSTARTDATETIME]
               when so1.[ACTUALENDDATETIME] = so1.[LASTSTEPINPLANENDDATETIME]
                       then so1.[ACTUALENDDATETIME] - so1.[ACTUALSTARTDATETIME]
               end,
       so1.[PROSPECTPLANID] as [PROSPECTPLANSYSTEMID],
       so1.[PROSPECTPLANSTATUSCODEID] as [PROSPECTPLANSTATUSSYSTEMID],
       so1.[PROSPECTPLANTYPECODEID] as [PROSPECTPLANTYPESYSTEMID],
       so1.[PROSPECTID] as [CONSTITUENTSYSTEMID],
       1 as ISINCLUDED
from [STAGEOCCURRENCES] as so1
left join [STAGEOCCURRENCES] as so2
       on so1.ALLSTAGEOCCURRENCESSEQUENCENUMBER + 1 = so2.A-
LLSTAGEOCCURRENCESSEQUENCENUMBER
```

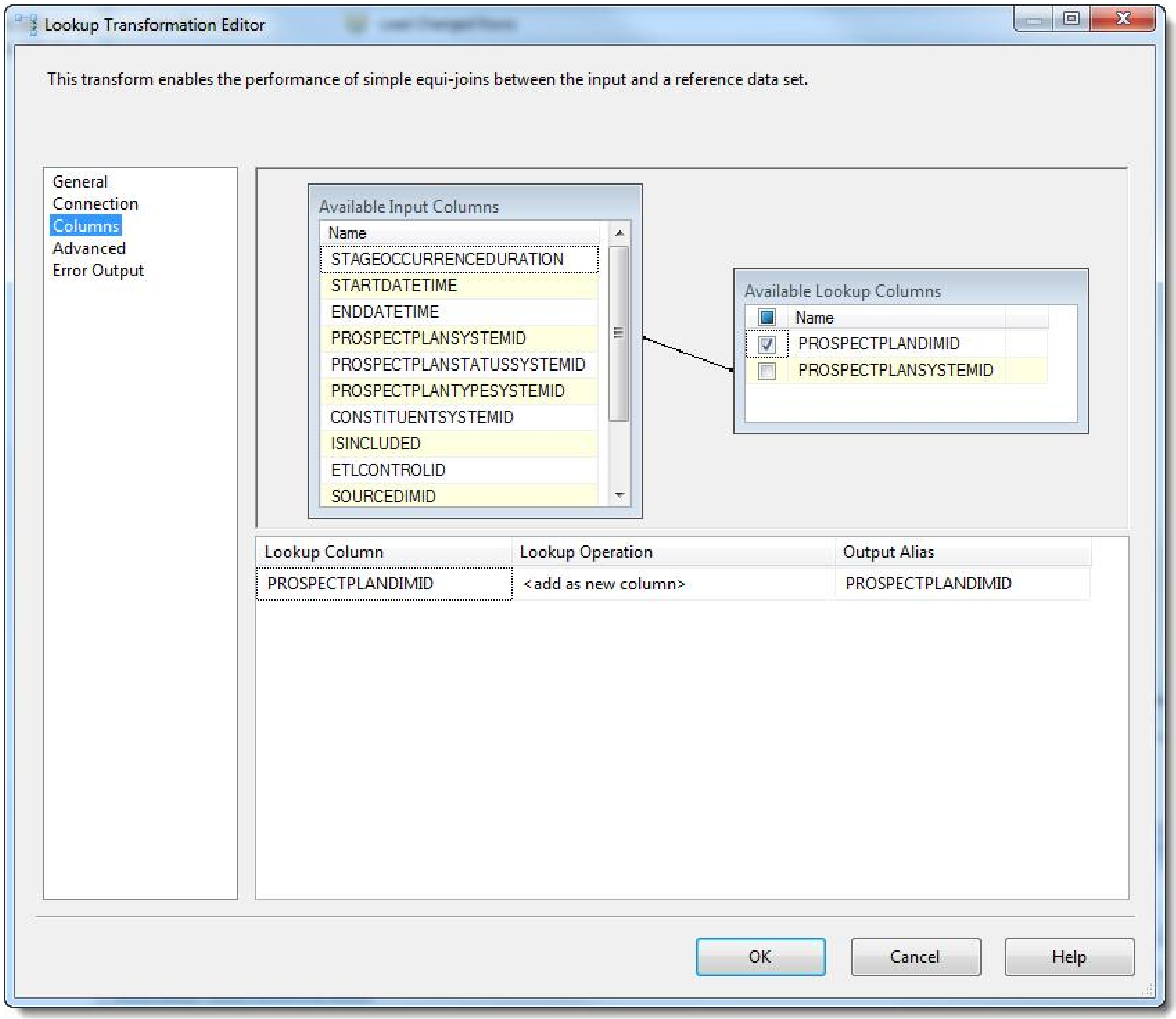
Date Dims

```
ISNULL(STARTDATETIME) ? 0 : YEAR(STARTDATETIME) * 10000 + MONTH(STARTDATETIME) *
100 + DAY(STARTDATETIME)
```

```
ISNULL(ENDDATETIME) ? 0 : YEAR(ENDDATETIME) * 10000 + MONTH(ENDDATETIME) * 100 +
DAY(ENDDATETIME)
```

Lookup Prospect Plan

```
select [PROSPECTPLANDIMID], [PROSPECTPLANSYSTEMID] from [BBDW].[DIM_PROS-
PECTPLAN]
```



This transform enables the performance of simple equi-joins between the input and a reference data set.

Available Input Columns

Name
STAGEOCCURRENCEDURATION
STARTDATETIME
ENDDATETIME
PROSPECTPLANSYSTEMID
PROSPECTPLANSTATUSSYSTEMID
PROSPECTPLANTYPESYSTEMID
CONSTITUENTSYSTEMID
ISINCLUDED
ETLCONTROLID
SOURCEDIMID

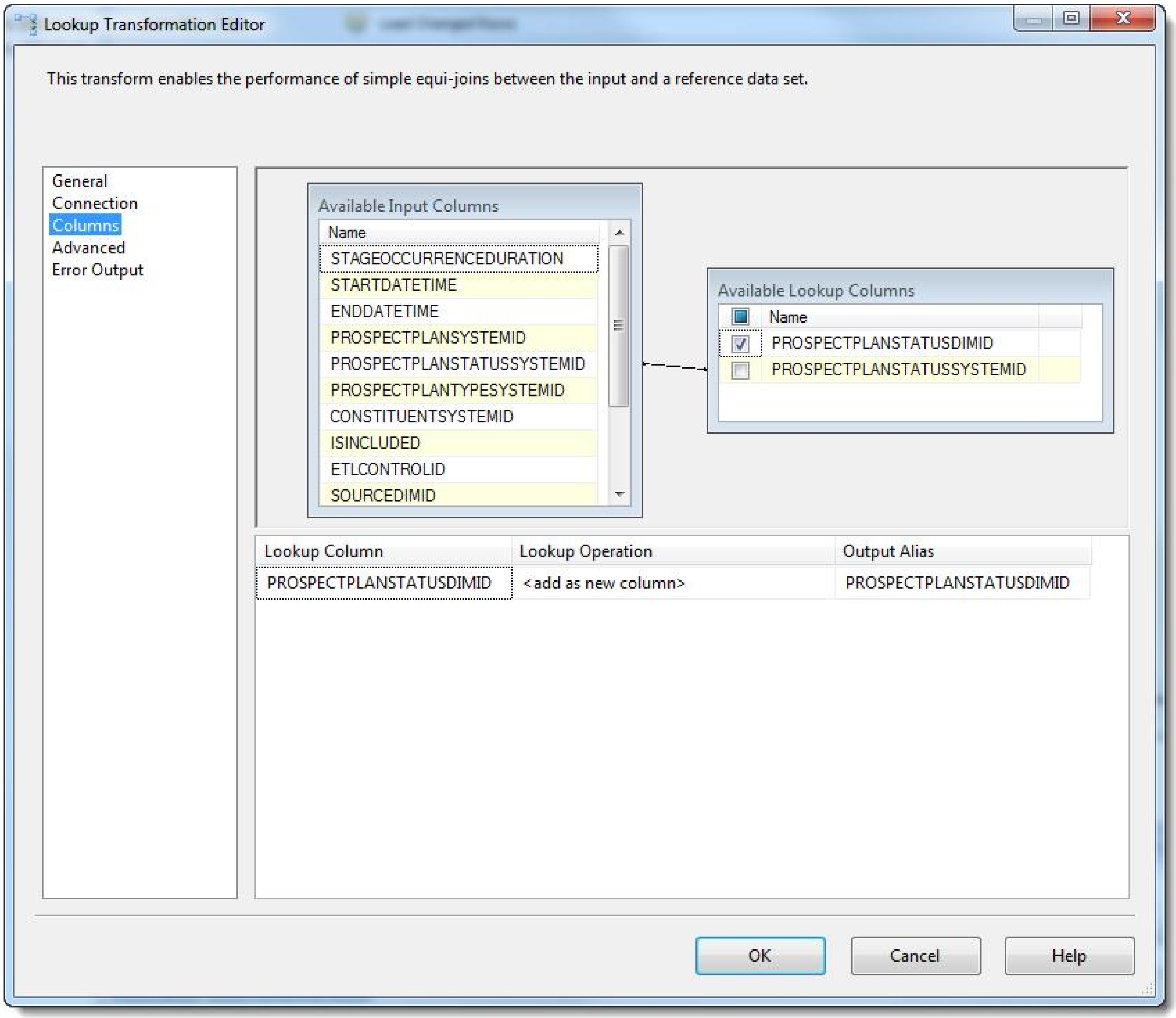
Available Lookup Columns

	Name
✓	PROSPECTPLANDIMID
	PROSPECTPLANSYSTEMID

Lookup Column	Lookup Operation	Output Alias
PROSPECTPLANDIMID	<add as new column>	PROSPECTPLANDIMID

Lookup Prospect Plan Status

```
select [PROSPECTPLANSTATUSDIMID], [PROSPECTPLANSTATUSSYSTEMID] from [BBDW].[DIM_
PROSPECTPLANSTATUS]
```



This transform enables the performance of simple equi-joins between the input and a reference data set.

Available Input Columns:

- STAGEOCCURRENCEDURATION
- STARTDATETIME
- ENDDATETIME
- PROSPECTPLANSYSTEMID
- PROSPECTPLANSTATUSSYSTEMID
- PROSPECTPLANTYPESYSTEMID
- CONSTITUENTSYSTEMID
- ISINCLUDED
- ETLCONTROLID
- SOURCEDIMID

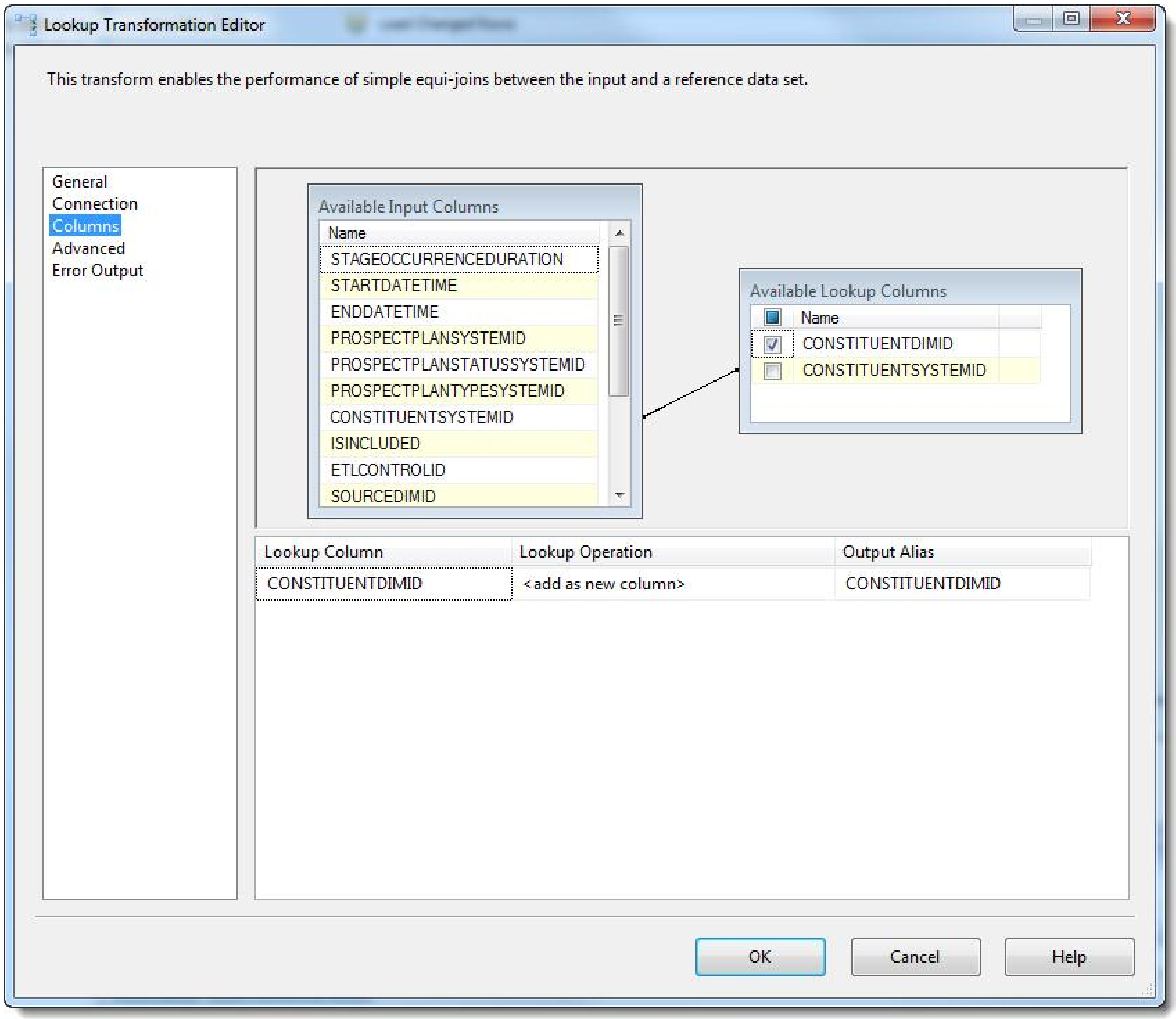
Available Lookup Columns:

- [✓] PROSPECTPLANSTATUSDIMID
- [ ] PROSPECTPLANSTATUSSYSTEMID

Lookup Column	Lookup Operation	Output Alias
PROSPECTPLANSTATUSDIMID	<add as new column>	PROSPECTPLANSTATUSDIMID

Lookup Prospect

```
select [CONSTITUENTDIMID], [CONSTITUENTSYSTEMID] from [BBDW].[DIM_PROSPECT]
```



This transform enables the performance of simple equi-joins between the input and a reference data set.

Available Input Columns

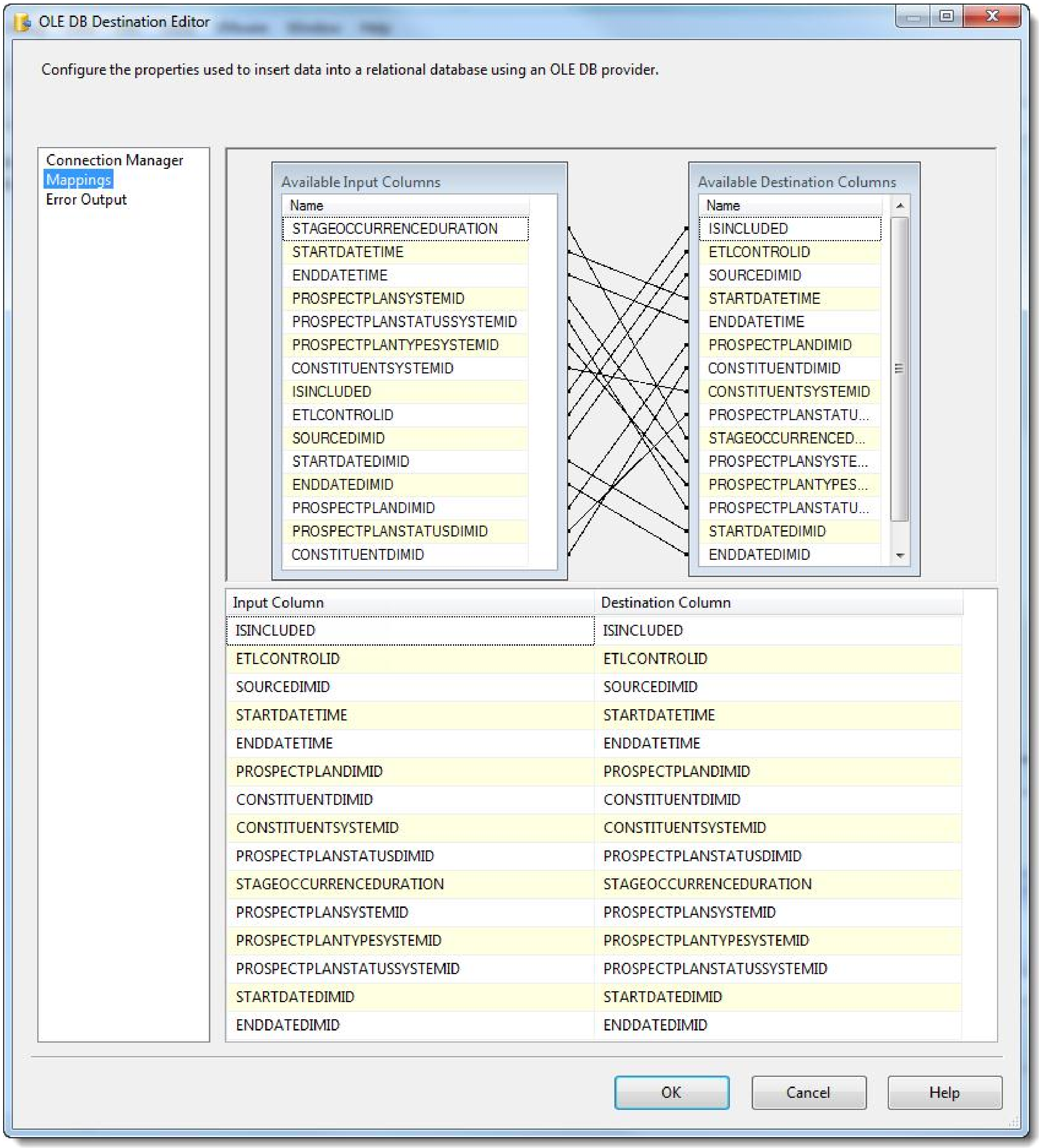
Name
STAGEOCCURRENCEDURATION
STARTDATETIME
ENDDATETIME
PROSPECTPLANSYSTEMID
PROSPECTPLANSTATUSSYSTEMID
PROSPECTPLANTYPESYSTEMID
CONSTITUENTSYSTEMID
ISINCLUDED
ETLCONTROLID
SOURCEDIMID

Available Lookup Columns

	Name
✓	CONSTITUENTDIMID
	CONSTITUENTSYSTEMID

Lookup Column	Lookup Operation	Output Alias
CONSTITUENTDIMID	<add as new column>	CONSTITUENTDIMID

Stage Rows



Configure the properties used to insert data into a relational database using an OLE DB provider.

Available Input Columns

Name
STAGEOCCURRENCEDURATION
STARTDATETIME
ENDDATETIME
PROSPECTPLANSYSTEMID
PROSPECTPLANSTATUSSYSTEMID
PROSPECTPLANTYPESYSTEMID
CONSTITUENTSYSTEMID
ISINCLUDED
ETLCONTROLID
SOURCEDIMID
STARTDATEDIMID
ENDDATEDIMID
PROSPECTPLANDIMID
PROSPECTPLANSTATUSDIMID
CONSTITUENTDIMID

Available Destination Columns

Name
ISINCLUDED
ETLCONTROLID
SOURCEDIMID
STARTDATETIME
ENDDATETIME
PROSPECTPLANDIMID
CONSTITUENTDIMID
CONSTITUENTSYSTEMID
PROSPECTPLANSTATU...
STAGEOCCURRENCED...
PROSPECTPLANSYSTE...
PROSPECTPLANTYPES...
PROSPECTPLANSTATU...
STARTDATEDIMID
ENDDATEDIMID

Input Column	Destination Column
ISINCLUDED	ISINCLUDED
ETLCONTROLID	ETLCONTROLID
SOURCEDIMID	SOURCEDIMID
STARTDATETIME	STARTDATETIME
ENDDATETIME	ENDDATETIME
PROSPECTPLANDIMID	PROSPECTPLANDIMID
CONSTITUENTDIMID	CONSTITUENTDIMID
CONSTITUENTSYSTEMID	CONSTITUENTSYSTEMID
PROSPECTPLANSTATUSDIMID	PROSPECTPLANSTATUSDIMID
STAGEOCCURRENCEDURATION	STAGEOCCURRENCEDURATION
PROSPECTPLANSYSTEMID	PROSPECTPLANSYSTEMID
PROSPECTPLANTYPESYSTEMID	PROSPECTPLANTYPESYSTEMID
PROSPECTPLANSTATUSSYSTEMID	PROSPECTPLANSTATUSSYSTEMID
STARTDATEDIMID	STARTDATEDIMID
ENDDATEDIMID	ENDDATEDIMID

Add Staging Indices

```
exec BBDW.[CREATE_OR_DROP_FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT_STAGE_INDICES] 1;
```

Upsert

```
declare @COUNTS table (
       [ACTION] varchar(28),
       [INSERTED] int,
       [UPDATED] int
       );

merge BBDW.[FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT] as t
using (
       select p.[STARTDATETIME],
               p.[STARTDATEDIMID],
               p.[ENDDATETIME],
               p.[ENDDATEDIMID],
               p.[STAGEOCCURRENCEDURATION],
               p.[PROSPECTPLANSYSTEMID],
               p.[PROSPECTPLANDIMID],
               p.[CONSTITUENTSYSTEMID],
               p.[CONSTITUENTDIMID],
               p.[PROSPECTPLANTYPESYSTEMID],
               p.[PROSPECTPLANSTATUSSYSTEMID],
               p.[PROSPECTPLANSTATUSDIMID],
               p.[ISINCLUDED],
               p.[ETLCONTROLID],
               p.[SOURCEDIMID]
       from BBDW.[FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT_STAGE] p
       ) as s
       on (t.[STARTDATETIME] = s.[STARTDATETIME])
               and (t.[ENDDATETIME] = s.[ENDDATETIME])
               and (t.[PROSPECTPLANSTATUSSYSTEMID] = s.[PROSPECTPLANSTATUSSYSTEMID])
               and (t.[PROSPECTPLANTYPESYSTEMID] = s.[PROSPECTPLANTYPESYSTEMID])
               and (t.[PROSPECTPLANSYSTEMID] = s.[PROSPECTPLANSYSTEMID])
when not matched by target
       then
               insert (
                       [STARTDATETIME],
                       [STARTDATEDIMID],
                       [ENDDATETIME],
                       [ENDDATEDIMID],
                       [STAGEOCCURRENCEDURATION],
                       [PROSPECTPLANSYSTEMID],
                       [PROSPECTPLANDIMID],
                       [CONSTITUENTSYSTEMID],
                       [CONSTITUENTDIMID],
                       [PROSPECTPLANTYPESYSTEMID],
                       [PROSPECTPLANSTATUSSYSTEMID],
                       [PROSPECTPLANSTATUSDIMID],
                       [ISINCLUDED],
                       [ETLCONTROLID],
                       [SOURCEDIMID]
                       )
               values (
                       s.[STARTDATETIME],
                       s.[STARTDATEDIMID],
                       s.[ENDDATETIME],
                       s.[ENDDATEDIMID],
                       s.[STAGEOCCURRENCEDURATION],
                       s.[PROSPECTPLANSYSTEMID],
                       s.[PROSPECTPLANDIMID],
                       s.[CONSTITUENTSYSTEMID],
                       s.[CONSTITUENTDIMID],
                       s.[PROSPECTPLANTYPESYSTEMID],
                       s.[PROSPECTPLANSTATUSSYSTEMID],
                       s.[PROSPECTPLANSTATUSDIMID],
                       s.[ISINCLUDED],
                       s.[ETLCONTROLID],
                       s.[SOURCEDIMID]
                       )
when matched
       then
               update
               set t.[STARTDATETIME] = s.[STARTDATETIME],
                       t.[STARTDATEDIMID] = s.[STARTDATEDIMID],
                       t.[ENDDATETIME] = s.[ENDDATETIME],
                       t.[ENDDATEDIMID] = s.[ENDDATEDIMID],
                       t.[STAGEOCCURRENCEDURATION] = s.[STAGEOCCURRENCEDURATION],
                       t.[PROSPECTPLANSYSTEMID] = s.[PROSPECTPLANSYSTEMID],
                       t.[PROSPECTPLANDIMID] = s.[PROSPECTPLANDIMID],
                       t.[CONSTITUENTSYSTEMID] = s.[CONSTITUENTSYSTEMID],
                       t.[CONSTITUENTDIMID] = s.[CONSTITUENTDIMID],
                       t.[PROSPECTPLANTYPESYSTEMID] = s.[PROSPECTPLANTYPESYSTEMID],
                       t.[PROSPECTPLANSTATUSSYSTEMID] = s.[PROSPECTPLANSTATUSSYSTEMID],
                       t.[PROSPECTPLANSTATUSDIMID] = s.[PROSPECTPLANSTATUSDIMID],
                       t.[ISINCLUDED] = s.[ISINCLUDED],
                       t.[ETLCONTROLID] = s.[ETLCONTROLID],
                       t.[SOURCEDIMID] = s.[SOURCEDIMID]
output $action,
       case
               when deleted.[ETLCONTROLID] is null
                       then 1
               else 0
               end,
       case
               when deleted.[ETLCONTROLID] is not null
                       then 1
               else 0
               end
into @COUNTS;

select count(*) as [TOTAL],
       isnull(sum([INSERTED]), 0) as [INSERTED],
       isnull(sum([UPDATED]), 0) as [UPDATED]
from @COUNTS
```

Adding Indices

```
exec BBDW.[CREATE_OR_DROP_FACT_PROSPECTPLANSTAGEOCCURRENCE_EXT_INDICES] 1;
```

Appendix

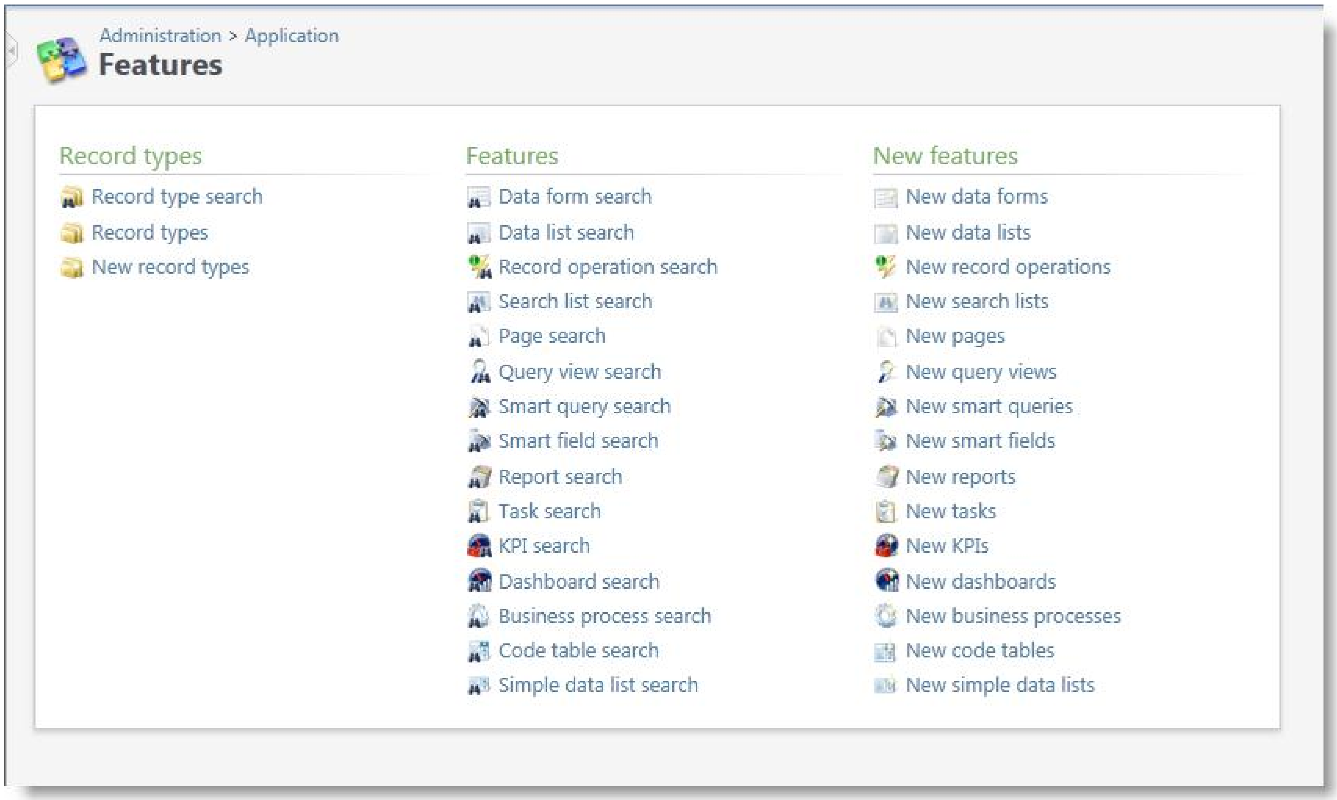
The following topics are included for your reference:

Application Features

To help surf the sea of features in the system (~20k specs at last count), Blackbaud as created a series of tasks located in a new `Application\Features` folder in **Administration**. These tasks treat our platform idioms (data forms, data lists, pages, record operations, etc) just like any other 1st-class entity in the system (constituents, interactions, revenue, etc) and allow you to search for an item and go to a page to see details about that item. We even shine a spotlight on newly added features via the New Features tasks.

This is super-useful if you're a developer wanting to get more information on a feature or see where a given feature is used. It's also nice for IT staff to be able to see the features from a low-level perspective and access the security permissions for the feature.

So when you navigate to **Administration**>**Application**> **Features**, you'll see the following series of tasks:


Administration > Application

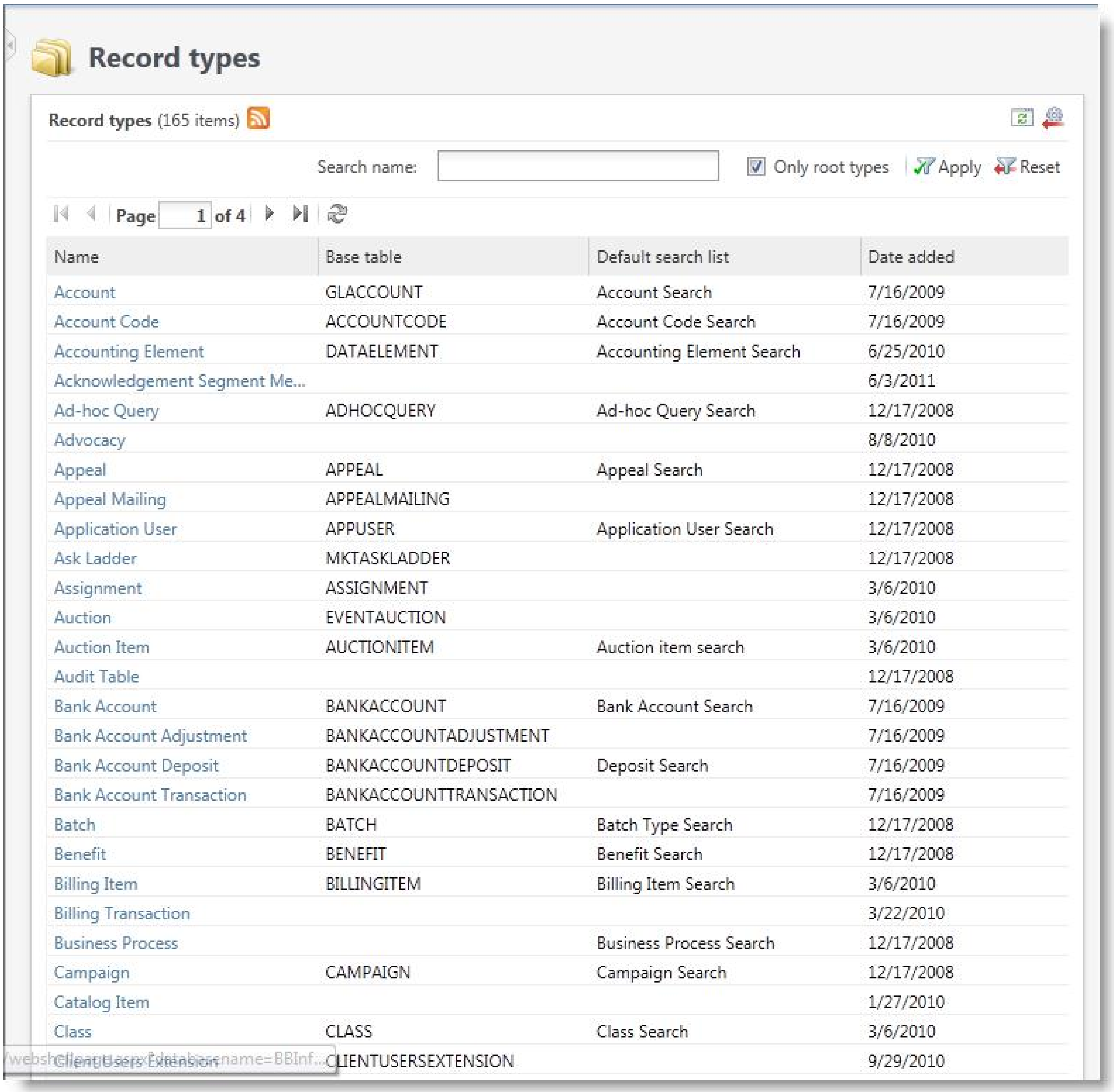
Features

Record types	Features	New features
Record type search	Data form search	New data forms
Record types	Data list search	New data lists
New record types	Record operation search	New record operations
	Search list search	New search lists
	Page search	New pages
	Query view search	New query views
	Smart query search	New smart queries
	Smart field search	New smart fields
	Report search	New reports
	Task search	New tasks
	KPI search	New KPIs
	Dashboard search	New dashboards
	Business process search	New business processes
	Code table search	New code tables
	Simple data list search	New simple data lists

Let’s dive into a few of these tasks!

Record Types

The first three tasks in the list allow you to see groups of features from a macro perspective. By using the tasks in the Record Types group, you can really get an idea of the scale and scope of a particular record type. You can search for a record type, view the list of record types (usually the “root” record types), or view a list of newly added record types.



Record types

Record types (165 items)

Search name: [ ] [✓] Only root types Apply Reset

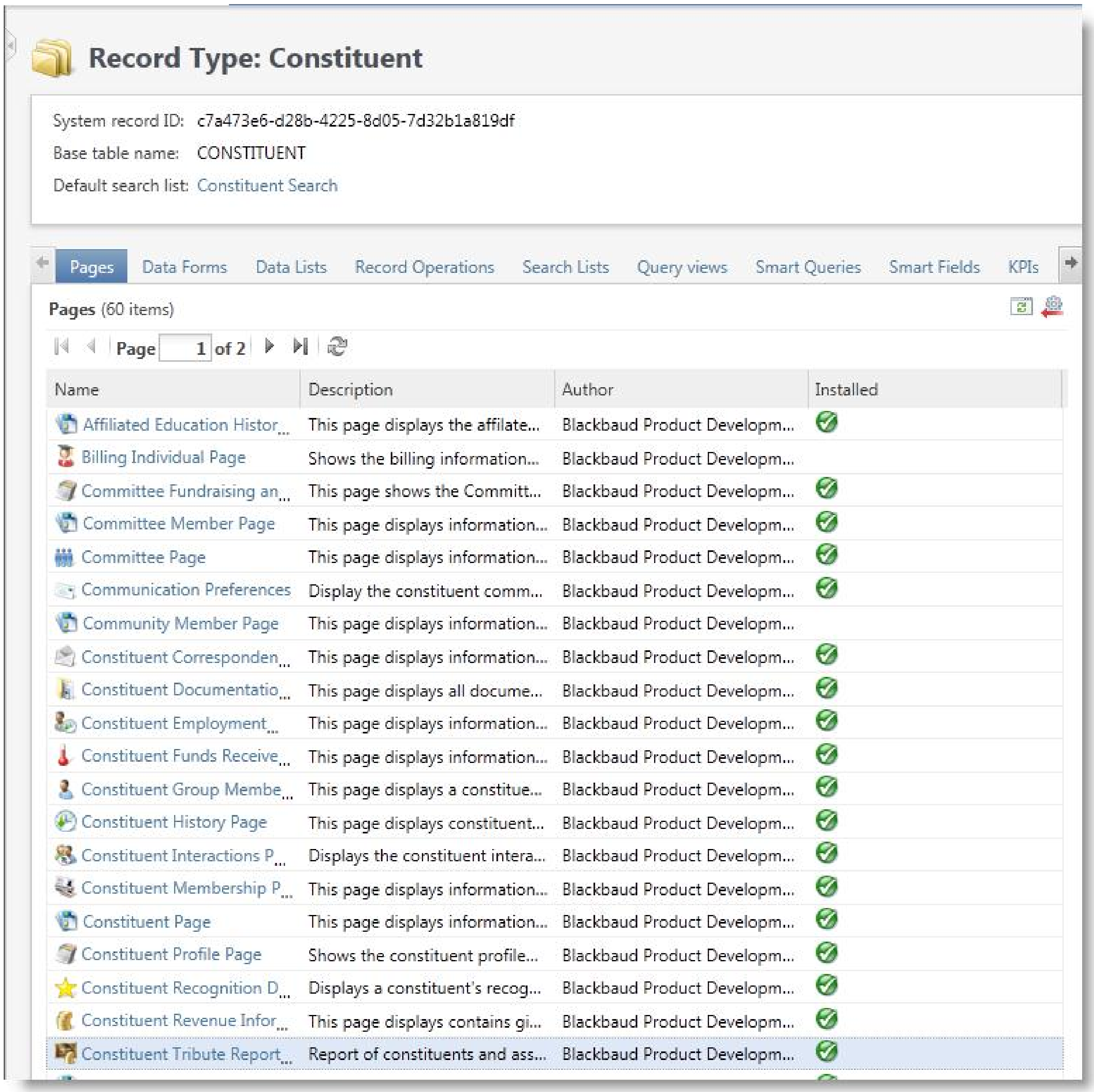
Page 1 of 4

Name	Base table	Default search list	Date added
Account	GLACCOUNT	Account Search	7/16/2009
Account Code	ACCOUNTCODE	Account Code Search	7/16/2009
Accounting Element	DATAELEMENT	Accounting Element Search	6/25/2010
Acknowledgement Segment Me...			6/3/2011
Ad-hoc Query	ADHOCQUERY	Ad-hoc Query Search	12/17/2008
Advocacy			8/8/2010
Appeal	APPEAL	Appeal Search	12/17/2008
Appeal Mailing	APPEALMAILING		12/17/2008
Application User	APPUSER	Application User Search	12/17/2008
Ask Ladder	MKTASKLADDER		12/17/2008
Assignment	ASSIGNMENT		3/6/2010
Auction	EVENTAUCTION		3/6/2010
Auction Item	AUCTIONITEM	Auction item search	3/6/2010
Audit Table			12/17/2008
Bank Account	BANKACCOUNT	Bank Account Search	7/16/2009
Bank Account Adjustment	BANKACCOUNTADJUSTMENT		7/16/2009
Bank Account Deposit	BANKACCOUNTDEPOSIT	Deposit Search	7/16/2009
Bank Account Transaction	BANKACCOUNTTRANSACTION		7/16/2009
Batch	BATCH	Batch Type Search	12/17/2008
Benefit	BENEFIT	Benefit Search	12/17/2008
Billing Item	BILLINGITEM	Billing Item Search	3/6/2010
Billing Transaction			3/22/2010
Business Process		Business Process Search	12/17/2008
Campaign	CAMPAIGN	Campaign Search	12/17/2008
Catalog Item			1/27/2010
Class	CLASS	Class Search	3/6/2010
Client Users Extension	CLIENTUSERSEXTENSION		9/29/2010

/webshell.aspx?databasename=BBInf...

You can see the features that support the record type and from here drill into the detail page for a given type of feature.

For example, if you select the Constituent record type, you’ll see just how many features we have that either require or return records of type Constituent (a lot!):



Record Type: Constituent

System record ID: c7a473e6-d28b-4225-8d05-7d32b1a819df
Base table name: CONSTITUENT
Default search list: Constituent Search

Pages (60 items)

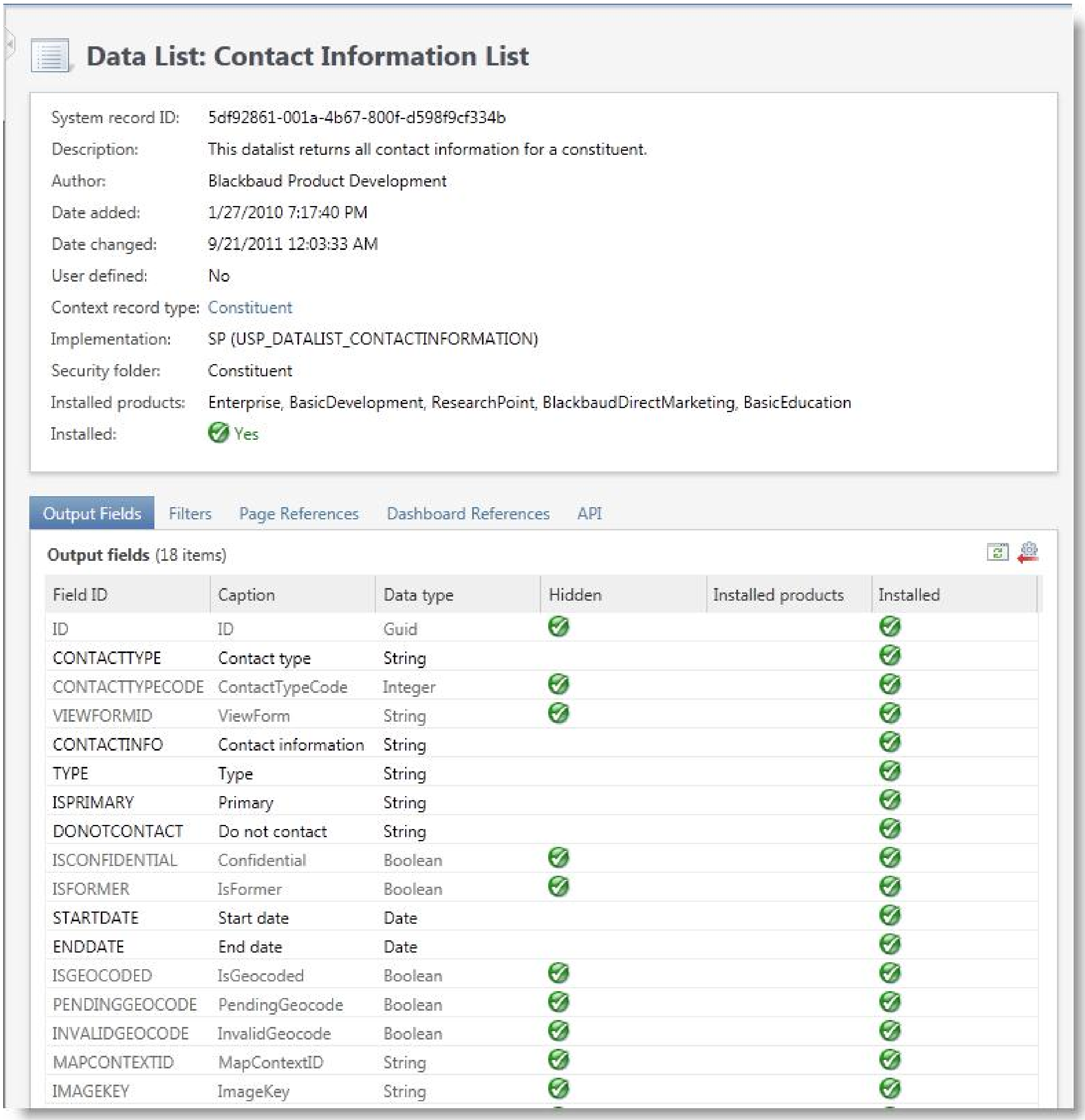
Page 1 of 2

Name	Description	Author	Installed
Affiliated Education Histor...	This page displays the affilate...	Blackbaud Product Developm...	✓
Billing Individual Page	Shows the billing information...	Blackbaud Product Developm...	
Committee Fundraising an...	This page shows the Committ...	Blackbaud Product Developm...	✓
Committee Member Page	This page displays information...	Blackbaud Product Developm...	✓
Committee Page	This page displays information...	Blackbaud Product Developm...	✓
Communication Preferences	Display the constituent comm...	Blackbaud Product Developm...	✓
Community Member Page	This page displays information...	Blackbaud Product Developm...	
Constituent Corresponden...	This page displays information...	Blackbaud Product Developm...	✓
Constituent Documentatio...	This page displays all docume...	Blackbaud Product Developm...	✓
Constituent Employment...	This page displays information...	Blackbaud Product Developm...	✓
Constituent Funds Receive...	This page displays information...	Blackbaud Product Developm...	✓
Constituent Group Membe...	This page displays a constitue...	Blackbaud Product Developm...	✓
Constituent History Page	This page displays constituent...	Blackbaud Product Developm...	✓
Constituent Interactions P...	Displays the constituent intera...	Blackbaud Product Developm...	✓
Constituent Membership P...	This page displays information...	Blackbaud Product Developm...	✓
Constituent Page	This page displays information...	Blackbaud Product Developm...	✓
Constituent Profile Page	Shows the constituent profile...	Blackbaud Product Developm...	✓
Constituent Recognition D...	Displays a constituent's recog...	Blackbaud Product Developm...	✓
Constituent Revenue Infor...	This page displays contains gi...	Blackbaud Product Developm...	✓
Constituent Tribute Report...	Report of constituents and ass...	Blackbaud Product Developm...	✓

Features

The tasks in the Features group let you search for the most popular types of features in the system (not all spec types are represented currently). When you search for and select a feature, you'll see a page with detailed information about the feature itself.

As an example, if you use the **Data List Search** task and search for the "Contact Information List", you'll be taken to the **Data List** page. Here you'll see the most relevant metadata about the list, include the ID, record type, implementation details, as well as whether or not the feature is installed. You can also see the output fields and filters, and view which pages and dashboards make use of this list (VERY handy!).



Data List: Contact Information List

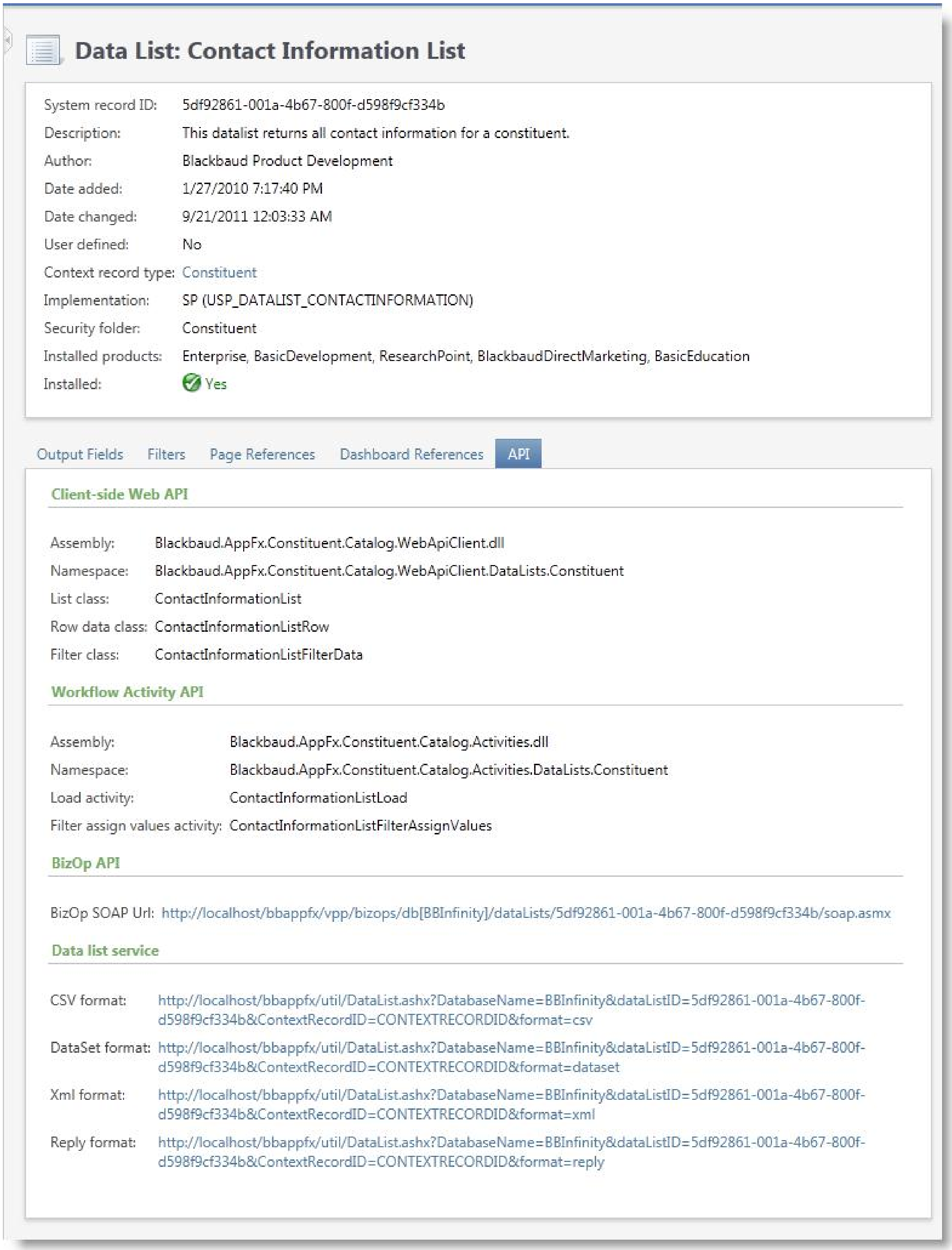
System record ID:	5df92861-001a-4b67-800f-d598f9cf334b
Description:	This datalist returns all contact information for a constituent.
Author:	Blackbaud Product Development
Date added:	1/27/2010 7:17:40 PM
Date changed:	9/21/2011 12:03:33 AM
User defined:	No
Context record type:	Constituent
Implementation:	SP (USP_DATALIST_CONTACTINFORMATION)
Security folder:	Constituent
Installed products:	Enterprise, BasicDevelopment, ResearchPoint, BlackbaudDirectMarketing, BasicEducation
Installed:	✓ Yes

Output Fields | Filters | Page References | Dashboard References | API

Output fields (18 items)

Field ID	Caption	Data type	Hidden	Installed products	Installed
ID	ID	Guid	✓		✓
CONTACTTYPE	Contact type	String			✓
CONTACTTYPECODE	ContactTypeCode	Integer	✓		✓
VIEWFORMID	ViewForm	String	✓		✓
CONTACTINFO	Contact information	String			✓
TYPE	Type	String			✓
ISPRIMARY	Primary	String			✓
DONOTCONTACT	Do not contact	String			✓
ISCONFIDENTIAL	Confidential	Boolean	✓		✓
ISFORMER	IsFormer	Boolean	✓		✓
STARTDATE	Start date	Date			✓
ENDDATE	End date	Date			✓
ISGEOCODED	IsGeocoded	Boolean	✓		✓
PENDINGGEOCODE	PendingGeocode	Boolean	✓		✓
INVALIDGEOCODE	InvalidGeocode	Boolean	✓		✓
MAPCONTEXTID	MapContextID	String	✓		✓
IMAGEKEY	ImageKey	String	✓		✓

The API tab even provides a reference for developers on how to use this data list from a variety of APIs. If you're writing .NET client-side code and want to set a reference to one of our `Black-baud.AppFx.*.WebAPIClient` assemblies (which are now being created as part of the build!!) this tab shows you which assembly contains the wrapper for this feature. You can also see the SOAP and BizOp URLs to use to get the data returned by the data list. For more information on choosing the best Infinity Web Service to suit your needs, see Introduction to the Infinity Web Service APIs.



Data List: Contact Information List

System record ID:	5df92861-001a-4b67-800f-d598f9cf334b
Description:	This datalist returns all contact information for a constituent.
Author:	Blackbaud Product Development
Date added:	1/27/2010 7:17:40 PM
Date changed:	9/21/2011 12:03:33 AM
User defined:	No
Context record type:	Constituent
Implementation:	SP (USP_DATALIST_CONTACTINFORMATION)
Security folder:	Constituent
Installed products:	Enterprise, BasicDevelopment, ResearchPoint, BlackbaudDirectMarketing, BasicEducation
Installed:	✓ Yes

Client-side Web API

Assembly:	Blackbaud.AppFx.Constituent.Catalog.WebApiClient.dll
Namespace:	Blackbaud.AppFx.Constituent.Catalog.WebApiClient.DataLists.Constituent
List class:	ContactInformationList
Row data class:	ContactInformationListRow
Filter class:	ContactInformationListFilterData

Workflow Activity API

Assembly:	Blackbaud.AppFx.Constituent.Catalog.Activities.dll
Namespace:	Blackbaud.AppFx.Constituent.Catalog.Activities.DataLists.Constituent
Load activity:	ContactInformationListLoad
Filter assign values activity:	ContactInformationListFilterAssignValues

BizOp API

BizOp SOAP Url: http://localhost/bbappfx/vpp/bizops/db[BBInfinity]/dataLists/5df92861-001a-4b67-800f-d598f9cf334b/soap.asmx

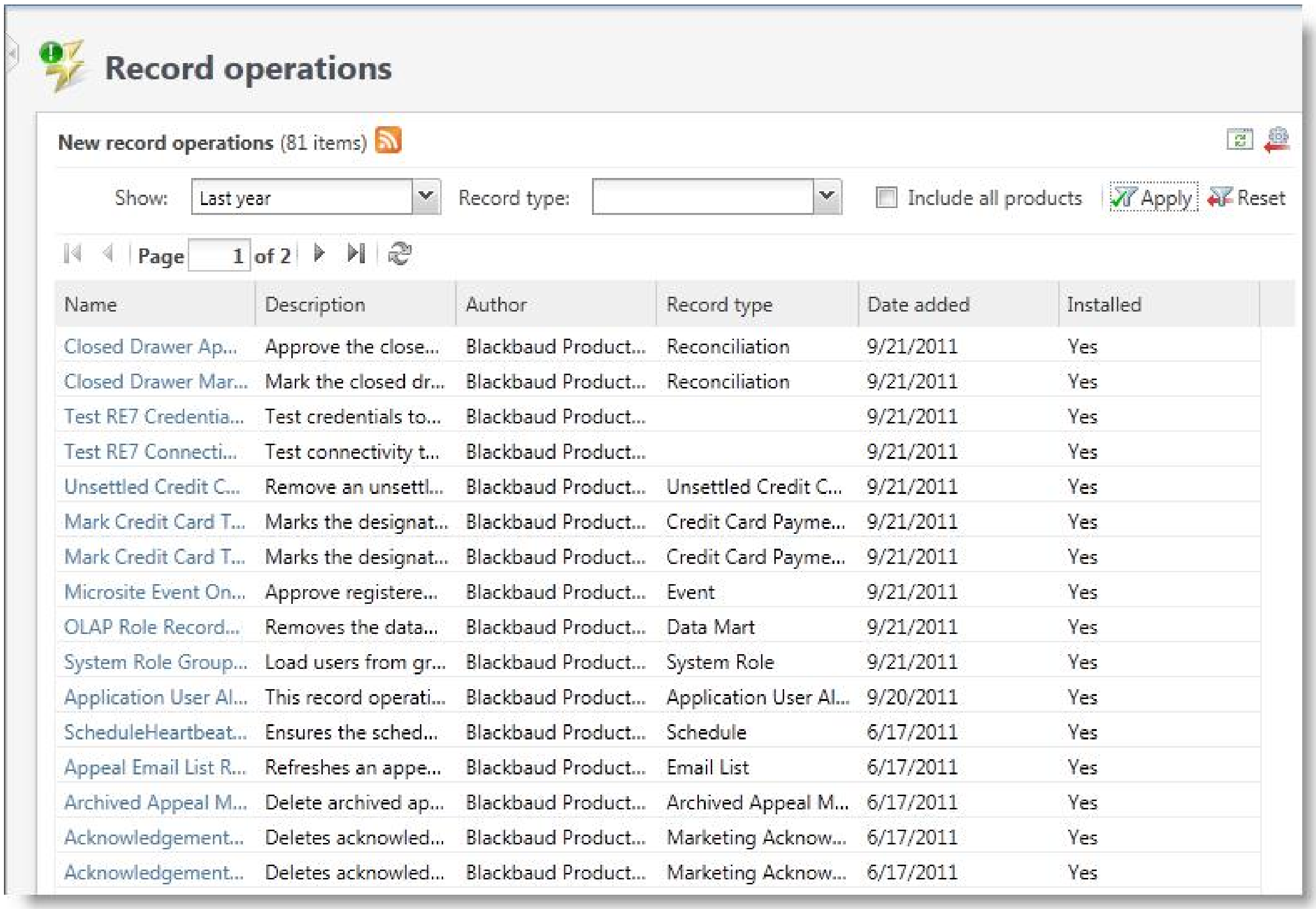
Data list service

CSV format:	http://localhost/bbappfx/util/DataList.ashx?DatabaseName=BBInfinity&dataListID=5df92861-001a-4b67-800f-d598f9cf334b&ContextRecordID=CONTEXTRECORDID&format=csv
DataSet format:	http://localhost/bbappfx/util/DataList.ashx?DatabaseName=BBInfinity&dataListID=5df92861-001a-4b67-800f-d598f9cf334b&ContextRecordID=CONTEXTRECORDID&format=dataset
Xml format:	http://localhost/bbappfx/util/DataList.ashx?DatabaseName=BBInfinity&dataListID=5df92861-001a-4b67-800f-d598f9cf334b&ContextRecordID=CONTEXTRECORDID&format=xml
Reply format:	http://localhost/bbappfx/util/DataList.ashx?DatabaseName=BBInfinity&dataListID=5df92861-001a-4b67-800f-d598f9cf334b&ContextRecordID=CONTEXTRECORDID&format=reply

New Features

The tasks in the New Features group spotlight newly added features in the past 30/60/90/etc days. This is very handy to know if you're trying to keep track of what's being added to the system. Note that these lists all support RSS feeds and notifications, so you can even get toast/email when someone adds a new spec to the system!

For example, here are the record operations that have been added in the past 30 days:


Record operations

New record operations (81 items)

Show: Last year Record type: Include all products Apply Reset

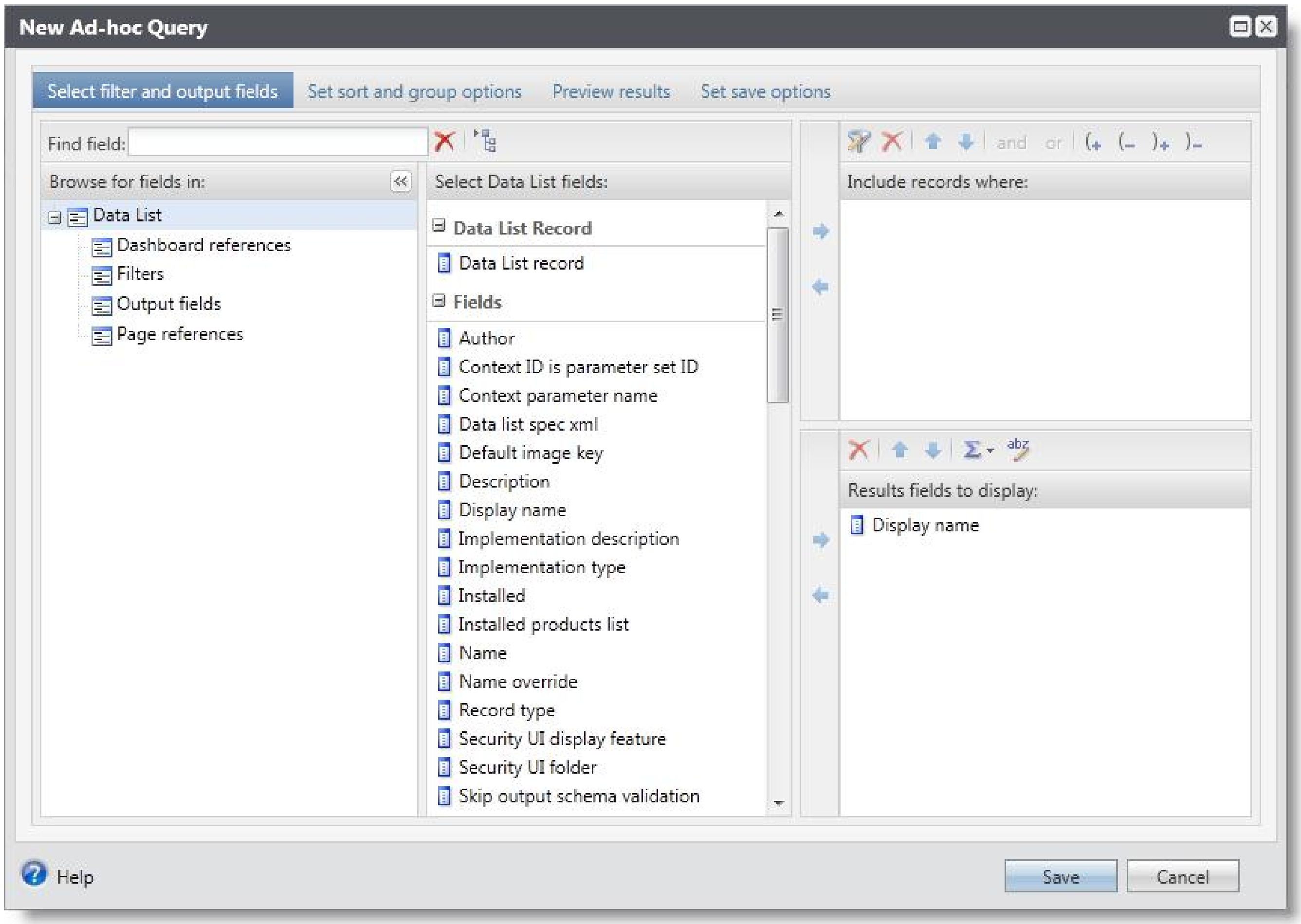
Page 1 of 2

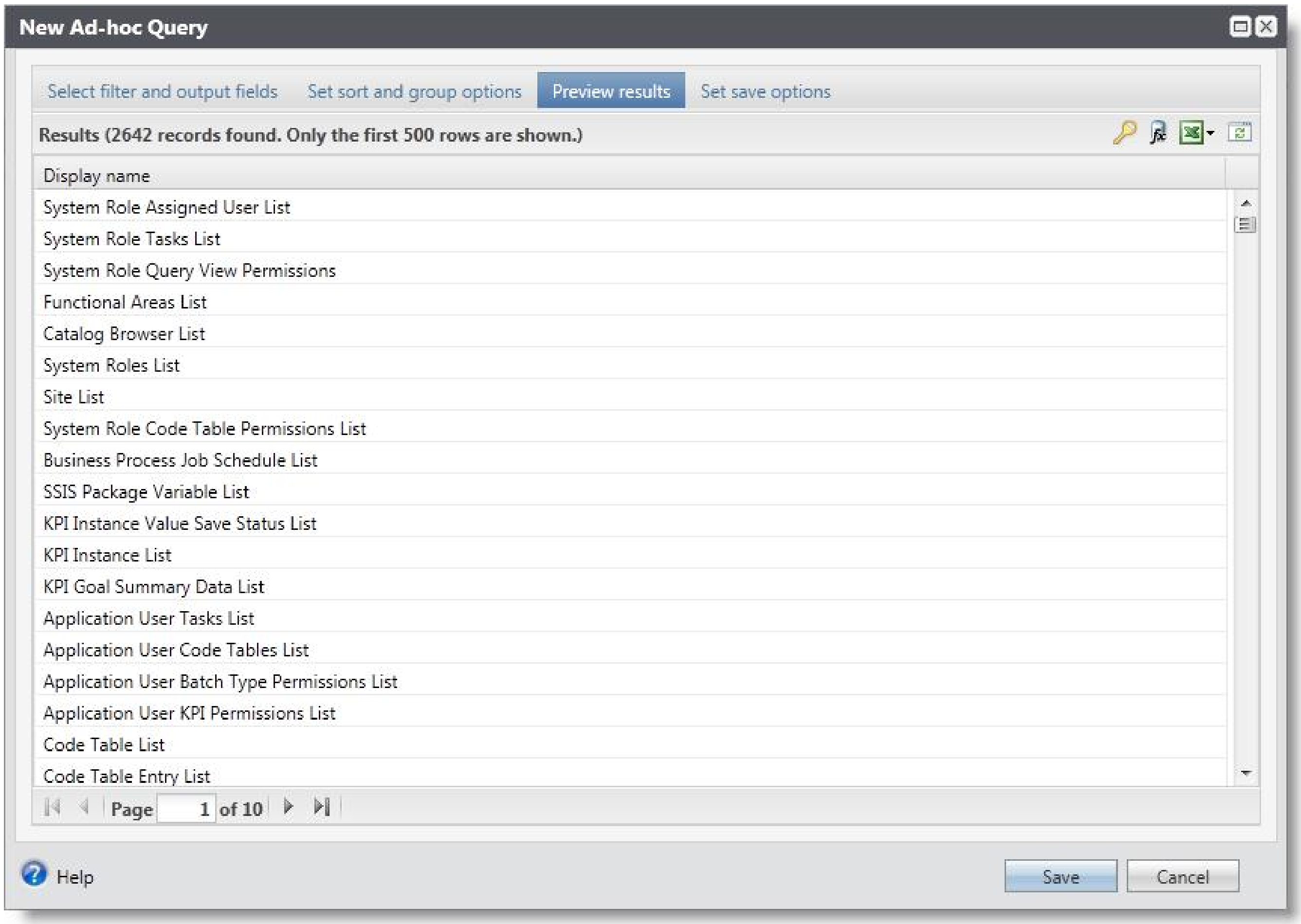
Name	Description	Author	Record type	Date added	Installed
Closed Drawer Ap...	Approve the close...	Blackbaud Product...	Reconciliation	9/21/2011	Yes
Closed Drawer Mar...	Mark the closed dr...	Blackbaud Product...	Reconciliation	9/21/2011	Yes
Test RE7 Credentia...	Test credentials to...	Blackbaud Product...		9/21/2011	Yes
Test RE7 Connecti...	Test connectivity t...	Blackbaud Product...		9/21/2011	Yes
Unsettled Credit C...	Remove an unsettl...	Blackbaud Product...	Unsettled Credit C...	9/21/2011	Yes
Mark Credit Card T...	Marks the designat...	Blackbaud Product...	Credit Card Payme...	9/21/2011	Yes
Mark Credit Card T...	Marks the designat...	Blackbaud Product...	Credit Card Payme...	9/21/2011	Yes
Microsite Event On...	Approve registere...	Blackbaud Product...	Event	9/21/2011	Yes
OLAP Role Record...	Removes the data...	Blackbaud Product...	Data Mart	9/21/2011	Yes
System Role Group...	Load users from gr...	Blackbaud Product...	System Role	9/21/2011	Yes
Application User Al...	This record operati...	Blackbaud Product...	Application User Al...	9/20/2011	Yes
ScheduleHeartbeat...	Ensures the sched...	Blackbaud Product...	Schedule	6/17/2011	Yes
Appeal Email List R...	Refreshes an appe...	Blackbaud Product...	Email List	6/17/2011	Yes
Archived Appeal M...	Delete archived ap...	Blackbaud Product...	Archived Appeal M...	6/17/2011	Yes
Acknowledgement...	Deletes acknowled...	Blackbaud Product...	Marketing Acknow...	6/17/2011	Yes
Acknowledgement...	Deletes acknowled...	Blackbaud Product...	Marketing Acknow...	6/17/2011	Yes

New Ad-hoc Query Views

In addition to creating these pages, We have also created new ad-hoc query views to allow querying over our platform feature metadata. You can now create queries of data forms, data lists, record operations, smart fields, etc and use the full power of our query tool semantics to mine the features in the system.

For example, want to know which data lists in the system have the most number of output fields? Use the Data List query and include the `COUNT(OutputFields\ID)` sorted by the count descending:


New Ad-hoc Query
Select filter and output fields
Set sort and group options
Preview results
Set save options
Find field:
Browse for fields in:
Data List
Dashboard references
Filters
Output fields
Page references
Select Data List fields:
Data List Record
Data List record
Fields
Author
Context ID is parameter set ID
Context parameter name
Data list spec xml
Default image key
Description
Display name
Implementation description
Implementation type
Installed
Installed products list
Name
Name override
Record type
Security UI display feature
Security UI folder
Skip output schema validation
and
or
Include records where:
Results fields to display:
Display name
Help
Save
Cancel


New Ad-hoc Query

Select filter and output fields | Set sort and group options | Preview results | Set save options

Results (2642 records found. Only the first 500 rows are shown.)

Display name
System Role Assigned User List
System Role Tasks List
System Role Query View Permissions
Functional Areas List
Catalog Browser List
System Roles List
Site List
System Role Code Table Permissions List
Business Process Job Schedule List
SSIS Package Variable List
KPI Instance Value Save Status List
KPI Instance List
KPI Goal Summary Data List
Application User Tasks List
Application User Code Tables List
Application User Batch Type Permissions List
Application User KPI Permissions List
Code Table List
Code Table Entry List

Page 1 of 10

Help | Save | Cancel

This suite of functionality should go a long way towards helping you manage the vast amount of features that make up the system.

Create a Report Spec

To create a new Report Spec, add a new item to your ***Microsoft Visual Studio*** solution's Blackbaud AppFx project:

1. Right-click the project.
2. Click **Add** > **New item**.
3. Select **Blackbaud AppFx Catalog** as the category of the item.
4. Select **Report Spec**.

After a Report Spec has been added to the project, you will notice the spec contains a `RDLFileName`, `Folder`, and `DataRetrieval` element. Within the `DataRetrieval` element, a stored procedure has been stubbed out for you in the `CreateSQL` element. It also attempts to name the report and `.rdl` file based on the filename selected.

```
<ReportSpec
       xmlns="bb_appfx_report"
       xmlns:common="bb_appfx_commontypes"
       ID="d0d55376-82cb-4176-8268-35910164175f"
       Name="FoodBankTransactionTotals Report"
       Description="REPLACE_WITH_DESCRIPTION"
       Author="Blackbaud Product Development"
       >

       <RDLFileName>FoodBankTransactionTotals.rdl</RDLFileName>
       <Folder>System Reports/Misc Reports</Folder>

       <DataRetrieval>
               <CreateSQL ObjectName="dbo.USP_REPORT_xxx" ObjectType="SQLStoredProc">
                       <![CDATA[
create procedure dbo.USP_REPORT_xxx
(
<list any report parameters here>
)
as
       <build the report SQL here>
                       ]]>
               </CreateSQL>
       </DataRetrieval>

</ReportSpec>
```

When the Report Spec is loaded, it will create the stored procedure specified in the database as well as load the `.rdl` file specified into Reporting Services.

Exploring a Report Spec

Report Specs are composed of four main elements:

- RDLFileName
- Folder
- DataRetrieval
- DataSource (optional)

RDLFileName

The `RDLFileName` element contains the name of the .rdl file for the report itself. The `.rdl` file will be uploaded to the Report Server when the Report Spec is loaded. The `.rdl` file itself must also be included in the same Blackbaud AppFx project as an Embedded Resource.

Note: The Report Designer for `.rdl` files is included in the SQL Server install as part of the Business Intelligence Development Studio for SQL Server 2008 or SQL Server 2008 R2 and is not part of the standard ***Visual Studio*** installation. If your Blackbaud AppFx project is in ***Visual Studio 2010***, you will likely want to create/edit the report itself in a separate ***Visual Studio 2008*** project and then copy the `.rdl` file into the ***Visual Studio 2010*** project.

SQL Server 2012 includes the Report Designer as part of SQL Server Data Tools and should allow you to create the Blackbaud AppFx project and the `.rdl` file in ***Visual Studio 2010***.

Folder

The `Folder` element contains the folder that the report will be deployed to on the Reporting Server. This should be a relative path from `Blackbaud\AppFx\<DatabaseName>`. For example `<Folder>Sample Reports\Food Bank</Folder>` would deploy the report to something like `Black-baud\AppFx\BBInfinity\Sample Reports\Food Bank` on the Report Server.

DataRetrieval

The `DataRetrieval` element can contain one or more `CreateSQL` elements that are used to either create Transact-SQL objects or grant permissions to Transact-SQL objects.

For example, the following would create the stored procedure `dbo.USR_USP_REPORT_FOOD-BANKTRANSACTIONTOTALS` as well as grant the `BBAPPFXREPORTROLE` rights to execute the stored pro-cedure. The report (rdl) itself would then include a Dataset that used the stored procedure.

```
                <CreateSQL ObjectName="dbo.USR_USP_REPORT_FOODBANKTRANSACTIONTOTALS" Object-
Type="SQLStoredProc">
                       <![CDATA[
        create procedure dbo.USR_USP_REPORT_FOODBANKTRANSACTIONTOTALS as

          select
            c.[KEYNAME] as [FOODBANK],
            f.[DESCRIPTION],
            fh.[FOODBANKTXTYPE] as [TRANSACTIONTYPE],
            fi.[NAME] as [FOODITEM],
            sum(ft.[FOODITEMAMOUNT]) as [TOTALFOODITEMAMOUNT],
            sum(ft.[QUANTITY]) as [TOTALFOODITEMQUANTITY]
          from dbo.[USR_FOODBANK] f
          left join dbo.[CONSTITUENT] c on c.[ID] = f.[CONSTITUENTID]
          inner join dbo.[USR_FOODBANKTXHEADER] fh on fh.[FOODBANKID] = f.[ID]
          left join dbo.[USR_FOODBANKTXDETAIL] ft on ft.[FOODBANKTXHEADERID] =
fh.[ID]
          left join dbo.[USR_FOODITEM] fi on fi.[ID] = ft.[FOODITEMID]
          group by
            c.[KEYNAME],
            f.[DESCRIPTION],
            fh.[FOODBANKTXTYPE],
            fi.[NAME]
                       ]]>
               </CreateSQL>
```

Alternatively, the report may contain a Dataset using embedded Transact-SQL. In this case, the report author would include the Transact-SQL for the report in the `.rdl` file itself. The `BBAPPFXREPORTROLE` will still need to be granted rights to the individual tables referenced in the Dataset. This can also be done using the `Crea-teSQL` element of the Report Spec. Here is a sample that would grant `SELECT` permissions to the `BBAPPFXRE-PORTROLE` for the `USR_FOODBANK` table.

```
<CreateSQL ObjectName="USR_FOODBANK" ObjectType="SQLTable"/>
```

DataSource

The `DataSource` element is optional for the Report Spec. Many reports will pull their data directly from the Infinity database. In this case, the `DataSource` element is not needed as the data source within the report will be updated to use the Infinity data source (`BBAppFxDB`) when the `Report Spec` is loaded.

However, reports may leverage any data source that is available in Reporting Services by specifying the relative path to the data source in the `DataSource` element. For example, the below will update the data source in the report to use the Blackbaud Data Warehouse.

```
  <DataSource>
    <DataSourceRelativePath>Blackbaud OLAP Reports/Blackbaud OLAP SQL data
source</DataSourceRelativePath>
  </DataSource>
```