d-tools

System Integration Software

System Integrator

Custom Reports

Training Guide

Presented by:

D-Tools Professional Services Group

psg@d-tools.com

www.d-tools.com

Table of Contents

Purpose

To inform D-Tools users about our reports as well as provide an understanding of the report designer, report data and methods for customizing project-based reports.

Overview

This document points the reader to resources that help facilitate understanding of our standard reports and provides a detailed explanation of our reporting interface, the layout of section-based reports, data sources and methods of customization including examples of various solutions targeting Proposal style reports.

Understanding Standard Reports

D-Tools SI reports are key to getting results with the platform. Based on the powerful Active Reports reporting engine, our standard reports provide an impressive array of information about SI projects. With this powerful reporting capability comes complexity. Understanding the process for writing custom reports is tied to understanding standard reports.

Many custom report questions are resolved by helping the D-Tools SI user understand the capabilities of standard reports. To be effective at writing custom reports you must understand the powerful SI reporting framework that includes standard reports and the various report settings that drive their behavior.

To that end, it is imperative that you spend time studying the standard reports and report capabilities in the software before moving forward. To get started, these links will point you to specific topics on the D-Tools documentation wiki:

Running Reports

Setting Cover Page Image

Arrange Items

Report Settings

Client Reports

Proposal Reports

Proposal Reports (Install Price)

Line Item Detail

Line Item Detail (Install Price)

Report Groups

After reviewing the previous topics you should have a good understanding of:

1. How to run reports.
2. How to create and modify report definitions.
3. How Report Settings affect reports.
4. The main difference between Proposal style reports and Line Item Detail reports.
5. How to create and use Report Groups.
6. Setting a Custom Cover Page Image.
7. Using the Arrange Items feature to sort items in reports.

No amount of reading about reports will replace using the software and testing the features. Have a few test projects that are setup specifically for testing report features and to help you understand the results you are getting when building custom reports. Having confidence in the areas listed above is critical as you embark on writing custom reports.

Getting Started with Custom Reports

As you get started with custom reports, there are several key points that will help you navigate your way:

1. D-Tools reports are developed using a report engine called ActiveReports.
2. Standard reports cannot be modified.
3. A custom report can be created based on an existing report. The existing report can be a standard report or a custom report.
4. All reports are stored locally on the Windows machine of each user that runs a D-Tools SI client.
5. There are two methods of sharing custom reports with other users:
    a. Publish the report to your SI server. This will cause other clients to be prompted to download the report at their next login.
    b. Export the report to file and send it to another user.
6. Most custom reports will require writing Visual Basic .NET code. If modifying or writing your own scripts sounds like something you do not want to take on, custom report writing may not be for you.
7. In the latest SI 2017 reports, even changing the look and feel of reports (colors, fonts, etc.) may require editing the report script (Visual Basic .NET code).

The Report Designer

Custom reports are created in the Report Designer interface. D-Tools has more than one Report Designer. For the purposes of this document, we will always refer to the "Standard Report Designer", which is provided to create custom reports that can be run in the Project Editor or from the Project Explorer.

See the Standard Report Designer article on our wiki for an introduction to the Report Designer:.

Also review the ActiveReports User Guide section introducing the Report Designer.

Opening the Report Designer

1. Starting from the SI Home Page, on the lower left, click Projects → Reports.
2. Then from the Report Explorer, click Standard in the section labeled Create.
3. The standard report designer will open.

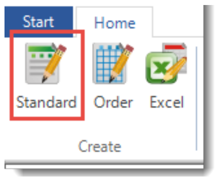
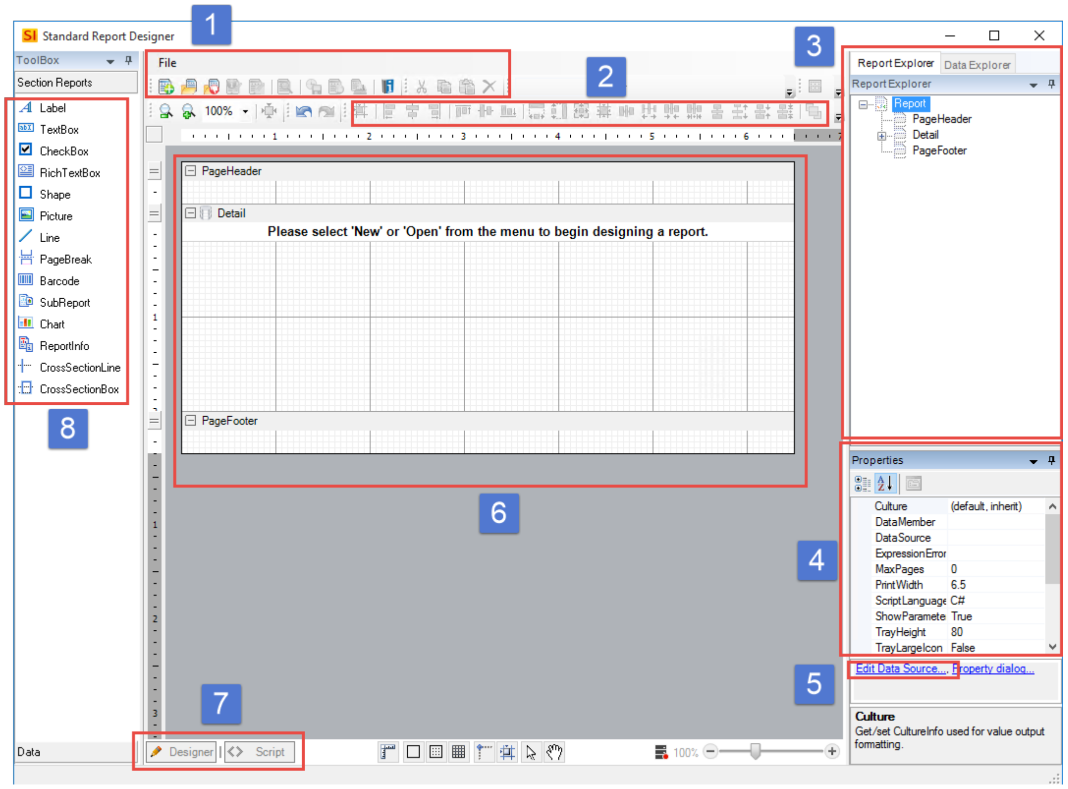


Figure 1. Open the Report Designer

Report Designer Layout

Understanding the Report Designer layout will help us be prepared to write custom reports.

1. The File menu and main toolbar include key tools for starting a new report, opening, saving, previewing and publishing reports.
2. The format toolbar offers a host of useful tools to help with alignment, sizing, and spacing.
3. The Report Explorer shows the hierarchy of report sections and elements. The Data Explorer provides a tree view of the XML data source (more on this later).
4. The Properties window shows details of the selected report element or details of the report when no element is selected.
5. The Edit Data Source link opens a dialog box that shows which XML node is selected.
6. The report sections and controls are shown in the center of the Report designer window.
7. The Designer and Script tabs allow toggling between the report layout and the report script.
8. The Toolbox contains the report controls that are available to add to the report.



Please select 'New' or 'Open' from the menu to begin designing a report.

Culture	(default, inherit)
DataMember	
DataSource	
ExpressionError	
MaxPages	0
PrintWidth	6.5
ScriptLanguage	C#
ShowParamete	True
TrayHeight	80
TrayLargeIcon	False

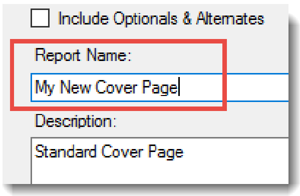
Culture
Get/set CultureInfo used for value output formatting.

Figure 2. Report Designer Overview

Our First Custom Report (Exercise 1)

For our first report we will create a custom cover page report. This report has a simple layout in comparison to most other reports which makes it easier to understand. To keep this example simple we will start by only changing the font (watch it here). To begin, after opening the Report Designer:

1. Click File → New.
2. The New Report Wizard opens and the 'New Report Based on Existing Report' radio button is selected. Click Next.
3. A list of reports is presented. Scroll through the list and find the 'Cover Page' report. Select it and click Next.
4. Now type a name for your new report in the 'Report Name' textbox. Click Next.



Include Optionals & Alternates
Report Name: My New Cover Page
Description: Standard Cover Page

Figure 3. Name your report

5. There are seven additional pages in the wizard including a summary et the end. For this exercise, no changes are needed on these pages. Click through them if you like, or click Finish to move on.
6. A copy of the original Cover Page report is created and loaded in the Report Designer. The phrase 'Unpublished report:' followed by the name of your report is listed in the upper left corner of the titlebar. It is worth noting that if you close this report prior to publishing it, you will have to open it from the list of unpublished reports (File → Open → Open Unpublished Report).
7. The Cover Page report has 12 text elements called controls. *'Control' is the generic term for the textboxes, labels, pictures, lines, shapes and other elements placed on the report page.* To update the font for each control, we first need to select them. This can be accomplished in one of two ways:
    a. While holding the Ctrl key, left-click on each text control.
    b. Or simply select the text items one at a time.
8. After making your selection, click on the Font Names drop-down list along the top of the Report deisgner toolbars. Choose a new font.

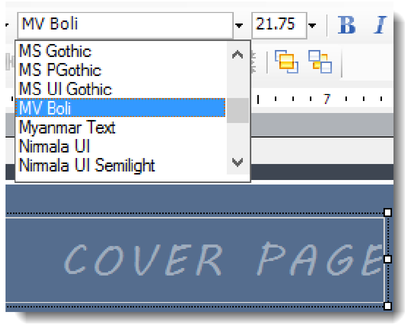


Figure 4. Select a font

9. Next, let us preview the changes. Click File → Preview or use the Preview toolbar button. The Preview window will open. Use the zoom tools to enlarge the preview and investigate your results. Close the preview by clicking the 'X' in the upper-right corner of the window.

Figure 5. Zoom in for detail

10. Now you are ready to publish the report. Click File → Publish Report or use the Publish Report toolbar button.
11. Now that your report is published, return to D-Tools and run the report on a project.

Report Organization

Before attempting more complex changes, it is important to develop a better understanding of report organization. D-Tools reports created in Active Reports are often highly structured documents called Section Reports. In many cases, reports called Subreports, are embedded in other reports which allows for reporting across unique sets of data within the same report output.

Section Report Structure

Section reports are divided into components called sections that help organize the flow of data within the report. There are four types of sections, which come in pairs except for the Detail section. The Detail section is often where the main portion of the data being expressed in the report is shown. The section types are:

1. Report Header/Footer – a report can have one Report Header/Footer pair and the output of these sections is only shown once in the report output. The header comes at the beginning of the report and the footer comes at the end. D-Tools uses the Report Header section to include the Cover Page on Proposal reports.
2. Page Header/Footer – a report can have one Page Header/Footer pair and the output of these sections can be shown once on each page of the report. They appear at the top and bottom of the page, respectively. Page titles, column headers and page numbers are examples of data shown on Page Header/Footer sections.
3. Group Header/Footer – a report can include single or nested groups, also coming in pairs. Many group Header/Footer pairs can be created within a report. Group sections are used extensively within D-Tools reports. Our dynamic grouping feature (By Location by System, etc.) uses Group sections. Other report sections like the Project Summary are located in Group sections. Group sections are located directly before and after the Detail section.
4. Detail Section – typically where the data pointed to by the data source is represented in the report. One instance of the section is created per data record.

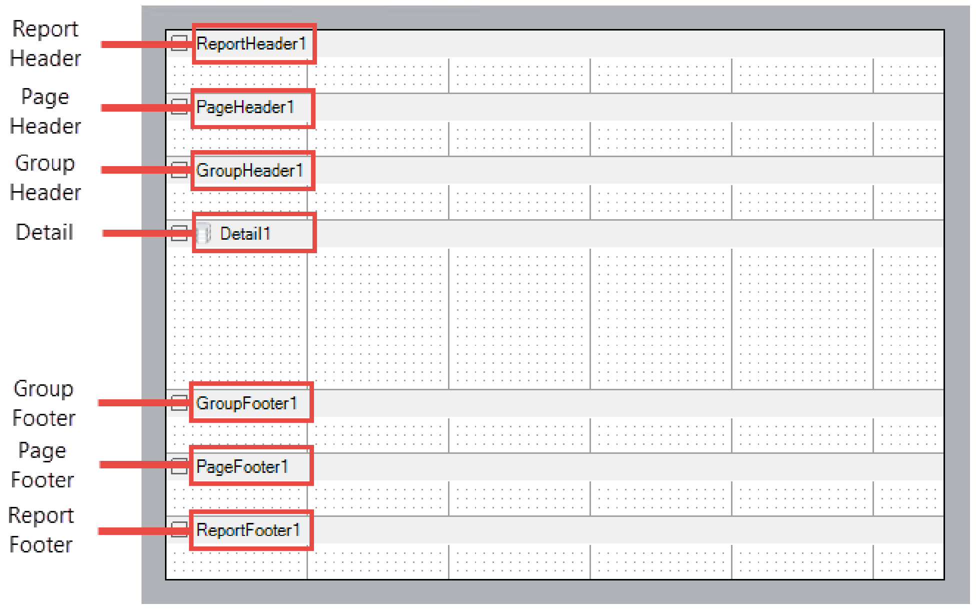


Figure 6. Report layout with section types

The ActiveReports User Guide provides additional details in the Section Report Concepts section.

Subreports

Subreports are no different than other reports. They serve a purpose which is to be embedded in reports. In the Report Designer's Report Information section, a report can be designated as a subreport. Subreports do not show up in your list of reports within the D-Tools Report Explorer. They are listed only in the Report Designer's Open Published Report or Open Unpublished Report windows.

Anytime a report needs to show more than one node within the XML data source, a subreport makes this possible. For example, Proposal style reports show Packages, Items and Accessories in context. The `dtr:ProposalItems/dtr:ProposalItem` node of the data source presents this hierarchy to the report. No other reports use this porton of the data source. The Project Summary section of the Proposal report is a subreport that iterates over the `dtr:Items/dtr:Item` data which allows it to easily summarize all project equipment, labor and adjustment values while ignoring the idea of Packages and Accessories. This node treats all data records as Items.

There are many cases where subreports allow reports to reflect diverse portions of a project's data. they are often a key component of getting good results with custom reports. We will work with subreports in our next exercise.

Getting Prepared: Custom Report Examples

In the next section we will create a custom Proposal report. Before we begin, take some time to read about and work through a few custom report examples. Found on our support wiki, there are three different report examples. The reports include a custom contract, wire label report and an example of adding a calculated field to display a maintenance fee. The three examples highlight different techniques that will prove useful on custom reports you may develop.

See the Custom Report Examples wiki page to try these exercises.

Creating a More Complex Custom Report (Exercise 2)

Next, we will create a custom Proposal report. While the changes we make are rather trivial, due to the complexity of proposal style reports, the process is surprisingly involved. Our goal is to create a custom proposal report that uses a different font. This task is essentially the same as creating the custom cover page except that it requires us to create the Proposal report, modify the font property for all text elements and repeat this for a group of subreports that are embedded in the main report. In total we must create, modify, publish and link together eight reports. While the task is not difficult, it is tedious.

Let's take a look at the structure of the standard Proposal report. The report is divided into sections that process different elements of the data. There is a cover page section, dynamic group sections that allow us to manage different data groupings (showing project items by location, system, etc.), sections and subreports that handle the hierarchy of items (packages, items and accessories), a subreport that includes miscellaneous items and two subreports that manage the project summary. The following graphic outlines the hierarchy of the main report and subreports.

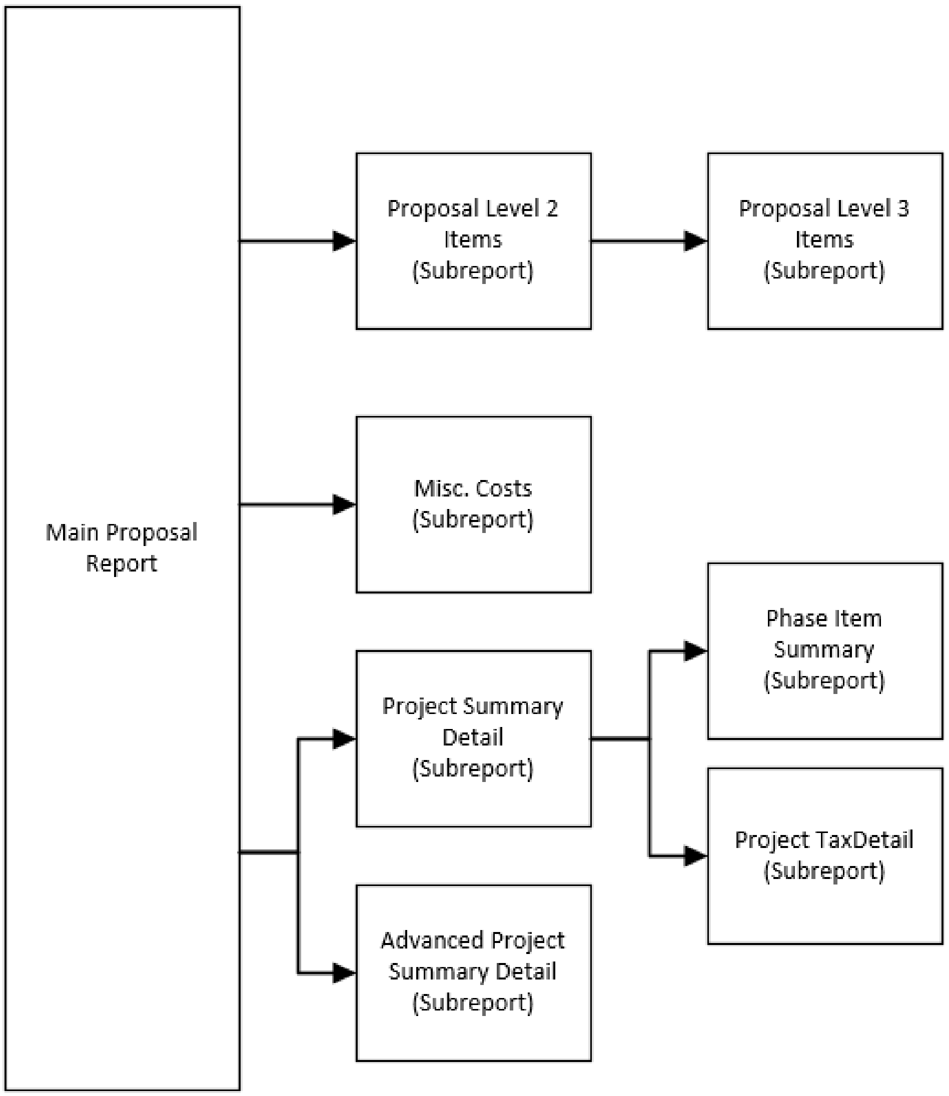

Main Proposal Report
Proposal Level 2 Items (Subreport)
Proposal Level 3 Items (Subreport)
Misc. Costs (Subreport)
Phase Item Summary (Subreport)
Project Summary Detail (Subreport)
Project TaxDetail (Subreport)
Advanced Project Summary Detail (Subreport)

Figure 7. Proposal report hierarchy of subreports

Each item in Figure 7 is a different report that must be created, updated and published. In a few cases, subreports have other subreports embedded in them as well. Notice that 'Proposal Level 3 Items' is referenced from 'Proposal Level 2 Items' and the 'Project Summary Detail' subreport contains two subreports. To speed the process we will start with the subreports shown to the right side of Figure 7 and work back towards the main report (watch the whole process here). Let us get started:

Level 2 and Level 3 Items Subreports

1. Starting from the Report Designer, click File → New.
2. The New Report Wizard opens and the 'New Report Based on Existing Report' radio button is selected. Click Next.
3. A list of reports is presented. Scroll through the list and find the 'Proposal Level3 Items' report. Select it and click Next.

4. Now type a name for your new report in the 'Report Name' textbox (typically the same as the original name with your company initials as a prefix). Click Finish.
5. The new report opens. It may not be clear but there are seven textboxes that need to be selected. The 'price' and 'quantity' textboxes are actually both doubled up. Two textboxes are stacked in each of those locations. Start by selecting the 'Alternates' textbox in the 'grpAlternate' secton and update the font by making a selection from the toolbar.
6. Next, drag a window across the textboxes in the 'grpAccessories' section (See Figure 8). Again, update the font by choosing a new font from the toolbar.

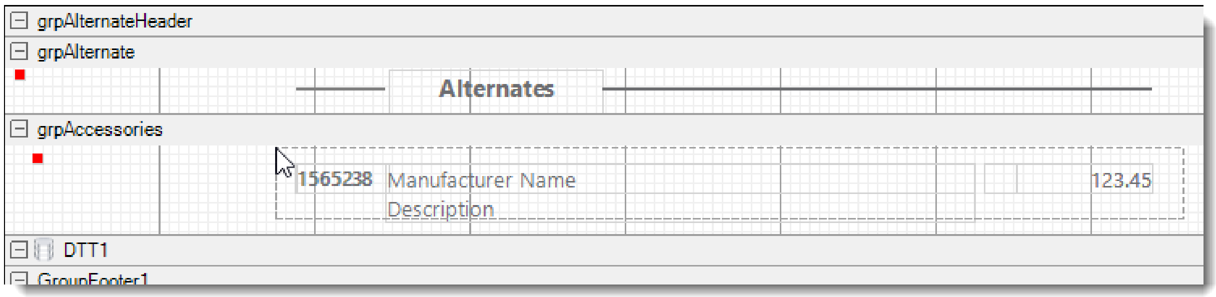


Figure 8. Select textboxes by dragging a window around them

7. Preview the changes. Click File → Preview or use the Preview toolbar button. Verify that your report looks as expected with new a new font. Close the preview by clicking the 'X' in the upper-right corner of the window.
8. Publish the report by clicking File → Publish Report or use the Publish Report toolbar button.
9. Repeat the first six steps for the 'Proposal Level2 Items' subreport. Before previewing and publishing this subreport we need to embed our custom 'Proposal Level3 Items' report. This subreport gets embedded in the 'grpItemFooter' report section. It is the only control in that section. Select the subreport (See Figure 9) and then right-click on it. Choose 'Bind to D-Tools Report'. In the dialog box find your custom 'Proposal Level3 Items' report and select it.

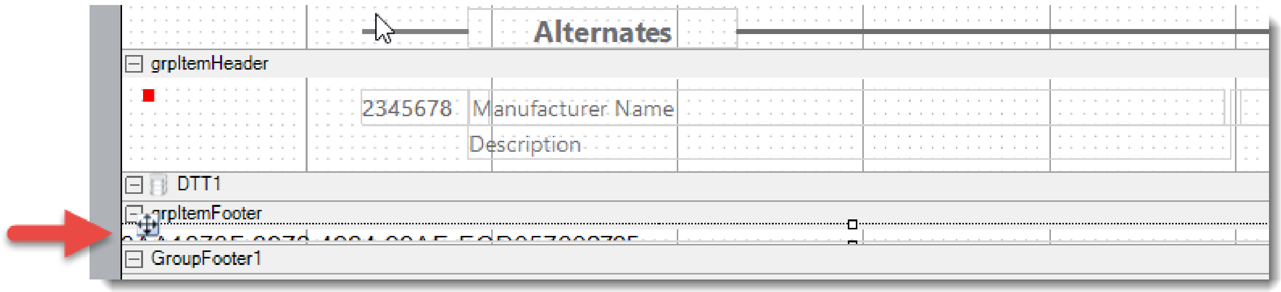


Figure 9. Select the embeded subreport

10. Preview the report and publish.

Miscellaneous Costs Subreport

1. Starting from the Report Designer, click File → New.
2. In the New Report Wizard, leave the default values and click Next.
3. In the list of reports that opens, find the 'Misc. Costs' report. Select it and click Next.
4. Give your new report a name and click Finish.

5. When the report opens, notice it has eight textboxes to update. Try holding Ctrl and clicking on each textbox. Choose a new font.
6. Preview the report and publish.

Project Summary Detail with Subreports

1. Starting from the Report Designer, click File → New.
2. In the New Report Wizard, leave the default values and click Next.
3. In the list of reports, find the 'Phase Item Summary' report. Select it and click Next.
4. Give this report a name and click Finish.
5. This report only has two text items to update. Select them and choose a new font.
6. Preview the report and publish. Remember this subreport is going to be embedded in the 'Project Summary Detail' subreport. We will bind them together later.
7. Repeat steps 1-5 for the 'Project TaxDetail' subreport. Note that this report also has two text items to update.
8. After choosing a note font, preview the report and publish.
9. Now repeat steps 1-5 for the 'Project Summary Detail' subreport. This report should have 21 text items to update. Note that the 'ReportHeader1' section has five text items. Two pricing textboxes are stacked on each other. Try selecting them individually from the Report Explorer and updating the font (See Figure 10).

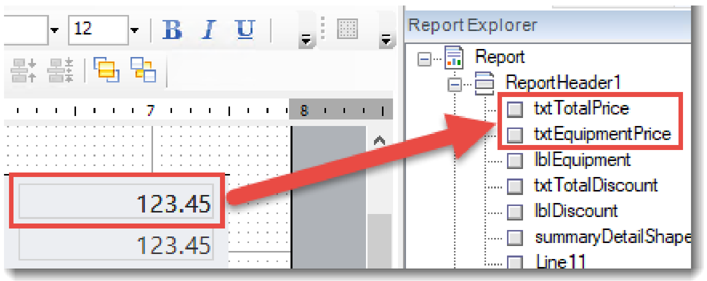


Figure 10. Select textboxes from the Report Explorer

10. The next step is to embed our custom 'Phase Item Summary' and 'Project TaxDetail' reports. The 'Phase Item Summary' subreport gets embedded in the 'grpLaborDetail' section. Select the subreport and right-click on it. Choose 'Bind to D-Tools Report'. In the dialog box find your custom 'Phase Item Summary' report and select it. Repeat this for the 'Project TaxDetail' subreport which is found in the 'grpTaxDetails' section.
11. After both embedded subreports have been replaced, preview the report and publish.

Advanced Project Summary Detail Subreport

1. Starting from the Report Designer, click File → New.
2. In the New Report Wizard, leave the default values and click Next.
3. In the list of reports, find the 'Advanced Project Summary Detail' report. Select it and click Next.
4. Give this report a name and click Finish.
5. This report has 51 text items to update. Select them and choose a new font. This might be easier to accomplish one section at a time.
6. Preview the report and publish.

Create the Proposal Report and Embed Subreports

1. Starting from the Report Designer, click File → New.
2. In the New Report Wizard, leave the default values and click Next.
3. In the list of reports, find the 'Proposal' report. Select it and click Next.
4. Give this report a name and click Finish.
5. This report has approximately 68 text items to update. Select them and choose a new font. This might be easier to accomplish one section or one item at a time.
6. Now we need to update the subreports to complete this report. Find each subreport, selecting them one at a time, right-click, choose 'Bind to D-Tools Report', and select the appropriate report. Below is a list of subreports and the section to find them:
    a. Section 'grpItemFooter', your custom 'Proposal Level2 Items' report
    b. Section 'grpMiscCosts', your custom 'Misc. Costs' report
    c. Section 'grpSummaryDetail' with two subreports stacked over each other:
        i. Your custom 'Project Summary Detail' report (the control name in the Report Explorer is 'subSummaryDetail')
        ii. Your custom 'Advanced Project Summary Detail' report (the control name in the Report Explorer is 'subAdvProjSummary')
7. After updating all of the subreports, preview the report and publish.

Now you should have a fully functional Proposal report with a custom font. In the next section we will consider adjustments to report controls.

Modifying Report Controls

Reports created with Active Reports are built with a combination of controls. Many controls share common properties like name, size, location and visibility. The most common control used is a texbox. Textbox controls have additional properties including font, color, text justification, vertical alignment and the data they represent.

Control Name

Every control must have a name. When controls are created, the Report Designer gives them a default name based on the type of control. A new textbox will be given a name `TextBox1` where th digits increment for each textbox added. A common naming convention we use for a textbox is 'txt' followed by a word describing the use of the textbox. A textbox representing the model of a product would be `txtModel`.

Control Location

Every control has a location property that is typically expressed as an ordered pair (x, y). You can type the ordered pair value simply as two numeric values separated by a comma or type an X and Y value into individual properties. The X value represents the distance from the left report margin to the left edge of the control. The Y value represents the distance from the top report margin to the top edge of the control. The location property of lines are unique among controls, having four values, X1, Y1, X2 and Y2 representing the location of each end of the line.

Control Size

Most controls have a size property that is described with a Width and Height propert. These values can be entered individually or as an ordered pair (width, height).

Control Visbility

Controls have a visibility property the determines whether the control is visible on the page where it should be rendered. This property is often used to hide a textbox that contains a data value being used as part of another calculation but does not need to be viewed directly. Later we will review how to capture the value of a hidden textbox and use it as part of a formula. Control visibility can also be set at runtime to show or hide elements of a report for different scenarios.

Control Color

Many controls have a color property. Color is often picked from a dropdown list in the property. A hexadecimal or RGB color value can also be typed directly into the property. When using a hex value simply type in a six character value (FFFFFF). For RGB colors type the value as an ordered triple (255,255,255). Some controls, like textboxes, have a ForeColor and a BackColor property. The ForeColor is the color of the text and the BackColor is the color of the overall textbox. In most cases the BackColor is transparent. Other color property variations exist in different controls.

TextBox Borders

Textbox controls have border properties that work in a similar fashion to Micrsoft Excel and other applications. In the Report Designer, right-click on a textbox control after selecting it and you will see a Format Border option.

Control Layout Adjustments

The Report Designer has an entire toolbar dedicated to alignment, size and spacing. These tools are some of the most useful available to quickly adjust controls in relation to other controls.

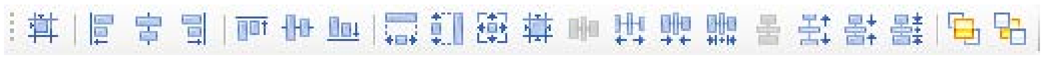


Figure 11. Alignment, size and spacing tools

Control DataField

Textbox controls have a DataField property that determines what the textbox represents in the report. For instance, the value shown in the DataField property the manufacturer of an item is `dtr:Manufacturer`. Quantity is shown as `dtr:Quantity`. Values with the 'dtr' prefix are part of the report data source. Some DataField values might not have the prefix which means they have been added to the report in the script. We will look more closely at data in a later section.

Modifying Report Sections

Report sections provide a great deal of flexibility in our reports. Sections played an important part in our seoncd custom report exercise. Each subreport was placed in separate section. This gives us control to show or hide subreports by simply setting the visibility property of the section where the subreport is located. Remember that section reports have four different section types. Review the Report Organization section of this guide to help reinforce the basic uses and features of the different section types. Sections share some properties with other elements of our reports. They have Height and Visibility properties. Sections can also be set to grow or shrink to allow them to properly show the data in their controls.

Section Name

Every report section must have a name. When sections are created, the Report Designer gives them a default name based on the type of section. A new Group Header/Footer pair will be given a name `GroupHeader1` and `GroupFooter1` where the digit increments for each section added. A common naming convention we use for a Group section is 'grp' followed by a word describing the purpose of the section. A section including a contract subreport might be named `grpContract`.

Section Height

Every section has a height property that is expressed as a single decimal number. This property sets the default height of the section.

Section CanGrow/CanShrink

Sections have a CanGrow and CanShrink property. If the default height of a section is such that some control elements may get cut off, CanGrow set to True will allow the section to automatically resize to fi the controls. CanShrink works similarly in that if controls do not fill the entire section, it can resize to not leave extra blank space. Of course, if extra blank space is desired in a section, be sure to set the CanShrink property to False.

Section Visibility

The section visibility property is often used to hide subreports. For instance, remember our custom Proposal report includes two Project Summary subreports. There is a parameter that can turn off the section where these subreports are located. The same goes for the Cover Page in the Proposal report. A parameter can be set to show or hide the section that houses the Cover Page.

Section KeepTogether

A section's KeepTogether property attempts to stop a section from being split across multiple pages. If a section has KeepTogether set True and the section will not fit on the current page, the section will be pushed to the next page. If the section will not completely fit on the next page, then the KeepTogether setting is ignored.

Section NewPage

The NewPage property acts like a page break. It can be set to None, Before, After or BeforeAfter. This allows a new page to be created before a section, after a section or both.

Section Report Events

The flow of events will help you to understand report behaviors that can seem very strange. As you take on more advanced custom report projects, understanding the event sequence will be critical to successfully writing reports.

In most custom reports, the Format event of a section will be where much of the scripting is done. In some cases, the BeforePrint event will be important. Keep in mind, all Format events fire for all sections before the BeforePrint events fire. Remember that some sections are processed many times within a report which means the sections Format event is fired many times. All of this happens before a single BeforePrint event fires. The point is that it can be tricky to use data captured in one event, like Format, and use it in a later event, like the BeforePrint event. Proceed cautiously when attempting to solve logic-based problems across events.

The following content is largely duplicated from the ActiveReports User Guide "Section Report Events" material available at:
http://help.grapecity.com/activereports/webhelp/AR11/index.html#ReportEvents.html. Where the information provided in the User Guide does not pertain to Section Reports it has been omitted.

Single-Occurrence Events

The following events are all of the events that are raised only once during a Section report's processing. These events are raised at the beginning or at the end of the report processing cycle.

ReportStart

Use this event to initialize any objects or variables needed while running a report. This event is also used to set any Subreport control objects to a new instance of the report assigned to the Subreport control.

DataInitialize

This event is raised after ReportStart. Use it to add custom fields to the report's Fields collection. Custom fields can be added to a bound report (one that uses a Data Control to connect and retrieve records) or an unbound report (one that does not depend on a data control to get its records). In a bound report the dataset is opened and the dataset fields are added to the custom fields collection, then the DataInitialize event is raised so new custom fields can be added. The DataInitialize event can also be used to make adjustments to the DataSource or to set up database connectivity.

ReportEnd

This event is raised after the report finishes processing. Use this event to close or free any objects that you were using while running a report in unbound mode, or to display information or messages to the end user. This event can also be used to export reports.

Multiple-Occurrence Events

The following events are raised multiple times during a Section report's processing.

FetchData

This event is raised every time a new record is processed. The FetchData has an EOF parameter indicating whether the FetchData event should be raised. This parameter is not the same as the Recordset's EOF property and is defaulted to True. When working with bound reports (reports using a DataControl), the EOF parameter is automatically set by the report; however, when working with unbound reports this parameter needs to be controlled manually.

Use the FetchData event with unbound reports to set the values of custom fields that were added in the DataInitialize event or with bound reports to perform special functions, such as combining fields together or performing calculations. The FetchData event should not have any references to controls on the report.

If you need to use a value from a Dataset with a control in the Detail section, set a variable in the FetchData event and use the variable in the section's Format event to set the value for the control. Please note that this method of setting a variable in the FetchData event and using it to set a control's value is only supported in the Detail_Format event.

Also use the FetchData event to increment counters when working with arrays or collections.

PageStart

This event fires before a page is rendered. Use this event to initialize any variables needed for each page when running an unbound report.

PageEnd

This event is raised after each page in the report is rendered. Use this event to update any variables needed for each page when running an unbound report.

When Bound and Unbound Data Values Are Set

1. The Fields collection is populated from the dataset that is bound to the report after the DataInitialize event is raised. (In an unbound report, the Fields collection values are not set to anything at this point.)
2. The FetchData event is raised, giving the user a chance to modify the Fields collection.
3. Any fields that are bound have the values transferred over.
4. The Format event is raised.

Note: *D-Tools reports largely use bound data from the data source. In some cases we create unbound data fields. These unbound fields are created in the DataInitilaize event. Later these fields are populated with data in the FetchData event. Remember FetchData fires for every record in the data set. This means you can calculate a value for an unbound field for every item that your report processes.*

Events that Occur for Each Instance of Each Section

In a Section report, regardless of the type or content of the various sections, there are three events for each section: **Format**, **BeforePrint** and **AfterPrint**.

Because there are many possible report designs, the event-raising sequence is dynamic in order to accommodate individual report demands. The only guaranteed sequence is that a section's Format event is raised before the BeforePrint event, which in turn occurs before the AfterPrint event but not necessarily all together. Reports should not be designed to rely on these events being raised in immediate succession.

Format

ActiveReports raises this event after the data is loaded and bound to the controls contained in a section, but before the section is rendered to a page.

The Format event is the only event in which you can change the section's height. Use this section to set or change the properties of any controls or the section itself.

Also use the Format event to pass information, such as an SQL String, to a Subreport.

If the CanGrow or CanShrink property is True for the section or any control within the section, all of the growing and shrinking takes place in the Format event. Because of this, you cannot obtain information about a control or section's height in this event.

Because a section's height is unknown until the Format event finishes, it is possible for a section's Format event to be raised while the report is on a page to which the section is not rendered. For example, the Detail Format event is raised but the section is too large to fit on the page. This causes the PageFooter events and the PageEnd event to be raised on the current page, and the PageStart, any other Header events, and possibly the FetchData event to be raised before the section is rendered to the canvas on the next page.

BeforePrint

ActiveReports raises this event before the section is rendered to the page.

The growing and shrinking of the section and its controls have already taken place. Therefore, you can use this event to get an accurate height of the section and its controls. You can modify values and resize controls in the BeforePrint event, but you cannot modify the height of the section itself.

Also use this event to do page-specific formatting since the report knows which page the section will be rendered to when this event is raised. Once this event has finished, the section cannot be changed in any way because the section is rendered to the canvas immediately after this event.

AfterPrint

ActiveReports raises this event after the section is rendered to the page.

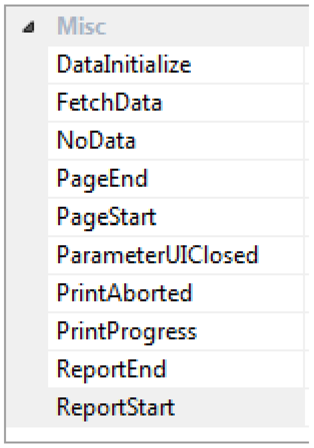
Although AfterPrint was an important event prior to ActiveReports Version 1 Service Pack 3, it is rarely used in any of the newer builds of ActiveReports. This event is still useful, however, if you want to draw on the page after text has already been rendered to it.

Event Sequence

Multi-threaded, single-pass processing enables Section reports to surpass other reports in processing and output generation speed. ActiveReports processes and renders each page as soon as the page is ready. If a page has unknown data elements or its layout is not final, it places the page in cache until the data is available.

Summary fields and KeepTogether constraints are two reasons why a page might not render immediately. The summary field is not complete until all the data needed for calculation is read from the data source. When a summary field such as a grand total is placed ahead of its completion level, such as in the report header, the report header and all following sections are delayed until all of the data is read.

There are ten report events in the code behind a Section report, or seven in a ActiveReport script.



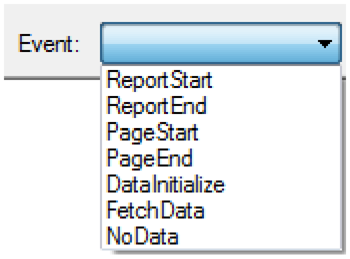


Figure 12. Report Events

Because there are so many ways in which you can customize your reports, not all reports execute in the same way. However, when you run a report, this is generally what happens:

1. ActiveReports raises the **ReportStart** event. The report validates any changes made to the report structure in ReportStart. In some cases, data source properties raise the **DataInitialize** event.
2. Printer settings are applied. If none are specified, the local machine's default printer settings are used.
3. If the **DataInitialize** event was not already raised, ActiveReports raises it and opens the data source.
4. If the data source contains Parameters with unset values and the **ShowParameterUI** property is set to **True**, ActiveReports displays a parameters dialog to request values from the user.
5. Closing the dialog raises the **ParameterUIClosed** event. If the report is a subreport that requires parameters, ActiveReports binds the subreport parameters to any fields in the parent report.
6. ActiveReports raises the **FetchData** event.
7. If there is no data, the **NoData** event is raised.
8. The **PageStart** event raises, and then raises again after each **PageEnd** event until the final page.
9. Group sections are bound and sections begin rendering on pages.
10. ActiveReports raises Section Events to process sections in (roughly) the following order:
    - Report header
    - Page header
    - Group header
    - Detail
    - Group footer
    - Page footer
    - Report footer
11. After each event, ActiveReports checks the **Cancel** flag to ensure that it should continue.
12. Other events may raise, depending on the report logic.
13. The **PageEnd** event raises after each page becomes full, and the **PageStart** raises if the report has not finished.
14. Finally, ActiveReports raises the **ReportEnd** event.

Events that May Occur

These events occur in response to user actions, or when there is no data for a report.

DataSourceChanged

This event occurs if the report's data source is changed. This is mainly useful with the end-user designer control.

NoData

This event occurs if the report's data source returns no records.

ParameterUIClosed

This event occurs when the user closes the parameter dialog.

PrintAborted

This event occurs when the user cancels a print job.

PrintProgress

This event occurs once for each page while the report document is printing.

Fully understanding how t owork with events

Accessing Project Data

D-Tools reports have a data source defined, which in our case is a version of the project data. While most data that is needed for a report is available, certain data or relationships between data elements may not be directly (or easily) accessible. When working on a custom report, make sure to determine the data source so that it is clear which fields are available to the report.

Data for Proposal Reports

Earlier, when subreports were introduced, we briefly covered the data source of these reports. Remember, Proposal style reports show Packages, Items and Accessories in context. To do this, these reports use the `dtr:ProposalItems/dtr:ProposalItem` node of the data source because it includes this hierarchy.

To process Packages, Items and Accessories in context, adds a great deal of complexity to Proposal reports. Items in these reports are shown in three levels. The first level is in the main report within the 'grpItemHeader' section. If the second level is required, a subreport located in the 'grpItemFooter' section runs. This subreport represents 'Level2' items. Within that subreport, if the third level is required, another subreport runs that shows 'Level3' items. Figure 7 demonstrates this at a high level.

When processing report data there are different scenarios that determine how the three levels of items are used. Let us review a few scenarios that describe how this works.

Level One Items (Main Report)

When report items are being processed, two types of items can be shown in Level One; a Package or an individual item.

Level Two Items

At Level Two we also show two types of items. If a Package in Level One is set to show the items inside of it, they show up in Level Two. Also, an individual item in Level One, if it is set to show Accessories, they show up in Level Two.

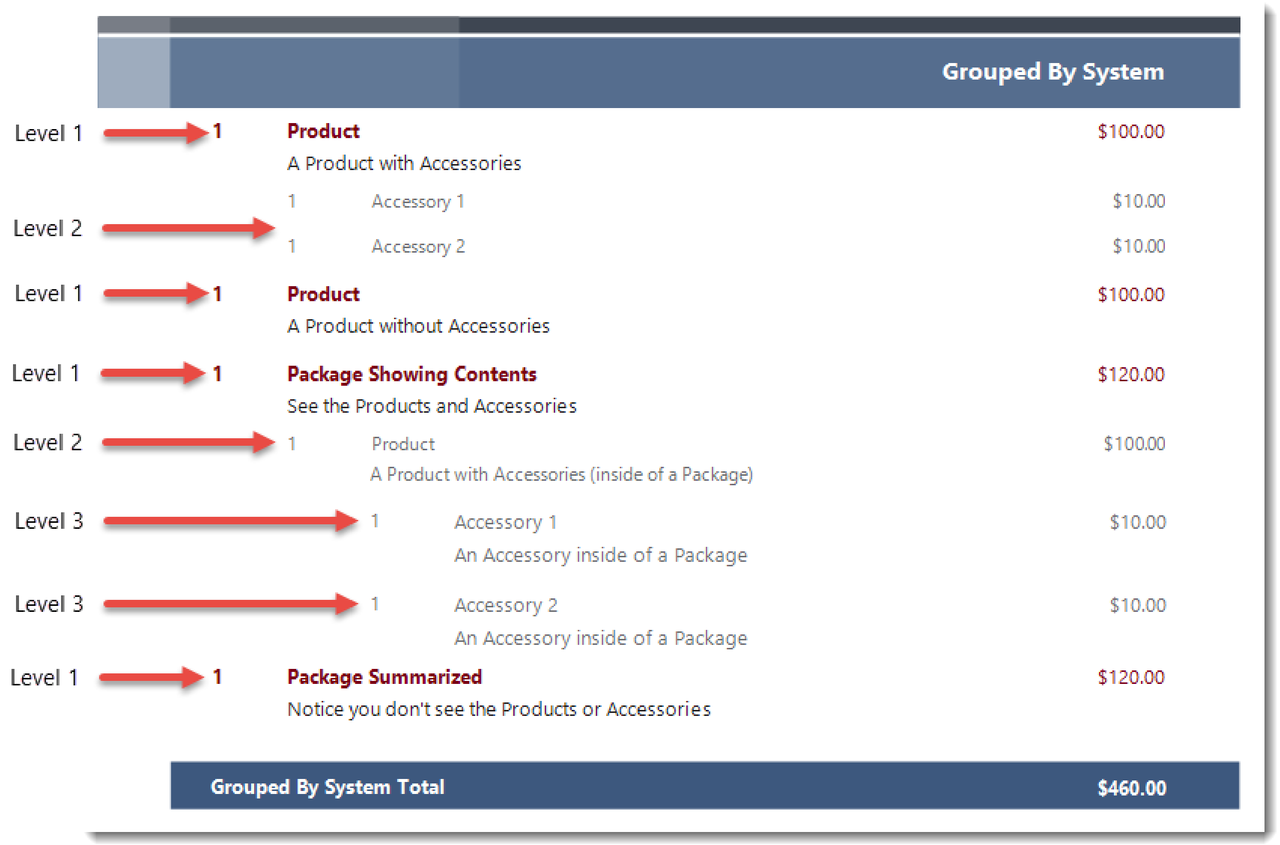
Level Three Items

At Level Three we only show one type of item. Remember, if a Package in Level One is set to show the items inside of it, they show up in Level Two. If the Package's items in Level Two are set to show Accessories, those Accessories are shown in Level Three.

Figure 13 provides examples of the different levels and items they can represent.

A Few Reasons to Use Proposal Reports

- Communicating in terms of Packages instead of the parts in a Package
- Include Accessories with parent items and hide them to simplify the narrative



Grouped By System

1	**Product** A Product with Accessories		$100.00
1		Accessory 1	$10.00
1		Accessory 2	$10.00
1	**Product** A Product without Accessories		$100.00
1	**Package Showing Contents** See the Products and Accessories		$120.00
1		Product A Product with Accessories (inside of a Package)	$100.00
1		Accessory 1 An Accessory inside of a Package	$10.00
1		Accessory 2 An Accessory inside of a Package	$10.00
1	**Package Summarized** Notice you don't see the Products or Accessories		$120.00

Grouped By System Total **$460.00**

Figure 13. Examples of records at each level

Data for Line Item Detail Reports

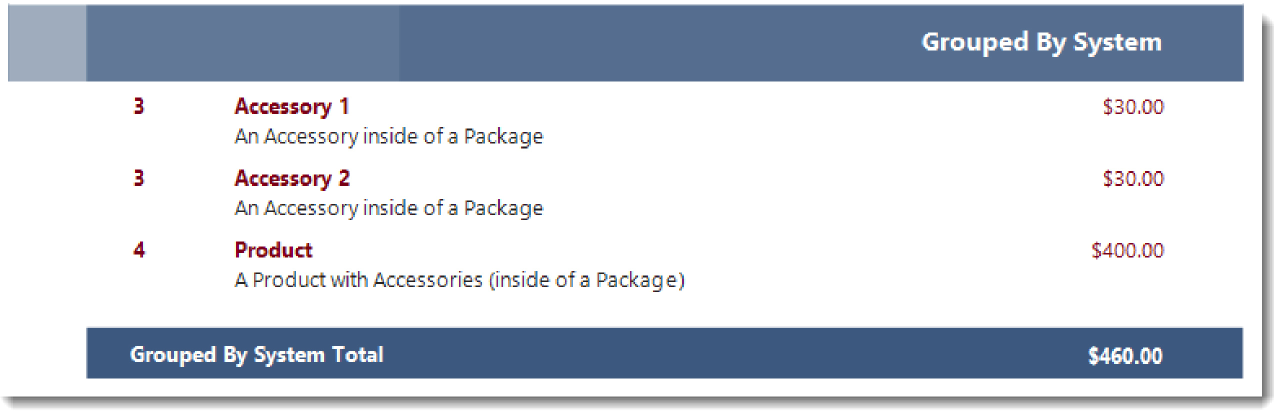
Line Item Detail style reports ignore the idea of Packages and Accessories altogether. They only represent report items as a simple aplhatbetical list (grouped however the report is run). Line Item Detail reports us the `dtr:Items/dtr:Item` node of the data source. This node is simply a list of report items. There is no concept of Packages or Accessories within the list.

If you do not use Packages in your projects and/or you do not need to communicate in your reports in a fashion like that demonstrated in Figure 13, then Line IItem reports might make more sense for your custom report purposes. These reports are easier to work with. They have less subreports to manage. Overall, Line Item reports tend to have a more streamlined output that is easier to follow.

A Few Reasons to Use Line Item Reports

- Easier to show items in straight columns and even add gridlines
- The math is always easier to understand because items are never hidden in Packages and Accessories are never hidden in Items
- Some D-Tools users need a simple output that shows a simple list of parts. Line Item reports give you that list.

Figure 14 is an example of the same project data from Figure 13, but shown in a Line Item Detail report. Notice the Product and Accessory records are all treated like top level items. No Packages are shown and there is no hierarchy of Products to Accessories. It is simply an alphabetical list.

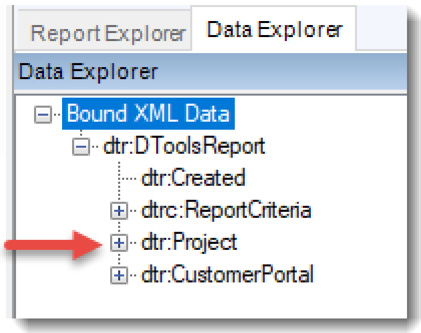


		Grouped By System	
3	**Accessory 1** An Accessory inside of a Package		$30.00
3	**Accessory 2** An Accessory inside of a Package		$30.00
4	**Product** A Product with Accessories (inside of a Package)		$400.00
	Grouped By System Total		**$460.00**

Figure 14. Line Item Detail report example

Understanding the Data Explorer

The Data Explorer provides a tree view of the XML data source. It provides a view of the entire set of data fields that are available to the report. After determining the specific node that the report is pointing to, it becomes clear what data is easily accessible. When viewing the Data Explorer, opening the `dtr:Project` node (Fig. 16) exposes the data used in our reports. Now we begin to see how to access useful data for our reports. Inside the `dtr:Project` node we find a large number of fields. This is a list of some important nodes and fields from Figure 15:

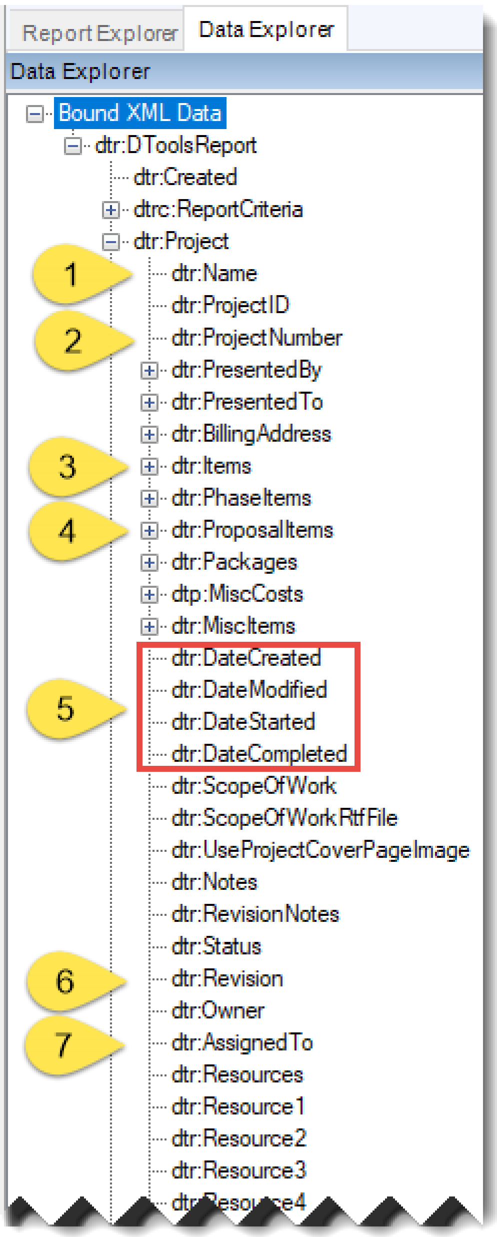


```
Report Explorer | Data Explorer
Data Explorer
⊟ Bound XML Data
   ⊟ dtr:DToolsReport
        dtr:Created
      ⊞ dtrc:ReportCriteria
      ⊞ dtr:Project
      ⊞ dtr:CustomerPortal
```

Figure 16. Opening the project node

1. Project Name as `dtr:Name`
2. Project Number as `dtr:ProjectNumber`
3. The list of Items as `dtr:Items`
4. The list of Proposal Items (for Proposal reports) as `dtr:ProposalItems`
5. Several date fields related ot the project
6. The project revision number as `dtr:Revision`
7. The name of the D-Tools user assigned to the project as `dtr:AssignedTo`. This is typically the salesperson.

Remember, the report Detail section will process records from the node that is set in the data source (see Fig. 2).



```
Report Explorer | Data Explorer
Data Explorer
⊟ Bound XML Data
   ⊟ dtr:DToolsReport
        dtr:Created
      ⊞ dtrc:ReportCriteria
      ⊟ dtr:Project
           dtr:Name
           dtr:ProjectID
           dtr:ProjectNumber
         ⊞ dtr:PresentedBy
         ⊞ dtr:PresentedTo
         ⊞ dtr:BillingAddress
         ⊞ dtr:Items
         ⊞ dtr:PhaseItems
         ⊞ dtr:ProposalItems
         ⊞ dtr:Packages
         ⊞ dtp:MiscCosts
         ⊞ dtr:MiscItems
           dtr:DateCreated
           dtr:DateModified
           dtr:DateStarted
           dtr:DateCompleted
           dtr:ScopeOfWork
           dtr:ScopeOfWorkRtfFile
           dtr:UseProjectCoverPageImage
           dtr:Notes
           dtr:RevisionNotes
           dtr:Status
           dtr:Revision
           dtr:Owner
           dtr:AssignedTo
           dtr:Resources
           dtr:Resource1
           dtr:Resource2
           dtr:Resource3
           dtr:Resource4
```

Figure 15. Inside the project node

Textbox Control Summary Fields

Textbox controls have a native capability for calculating a summary value. Summary values such as subtotals or grand totals are common. Subtotals might be calculated for a value in a group section. An example such as the total for a location or system in a project is a typical use of the summary properties.

Using Subtotals

In D-Tools reports we use subtotals in dynamic groups. In our reports, 'dynamic groups' are what allow us to change the grouping of a report without using a completely different report. In report definitions we can set a report's data to be grouped 'By Location', 'By System' or some combination of groupings.

An example is when a report is grouped 'By Location' and we want to populate each location's subtotal in the group footer section. To do this we need to set a few properties in the textbox where we want to show the value.

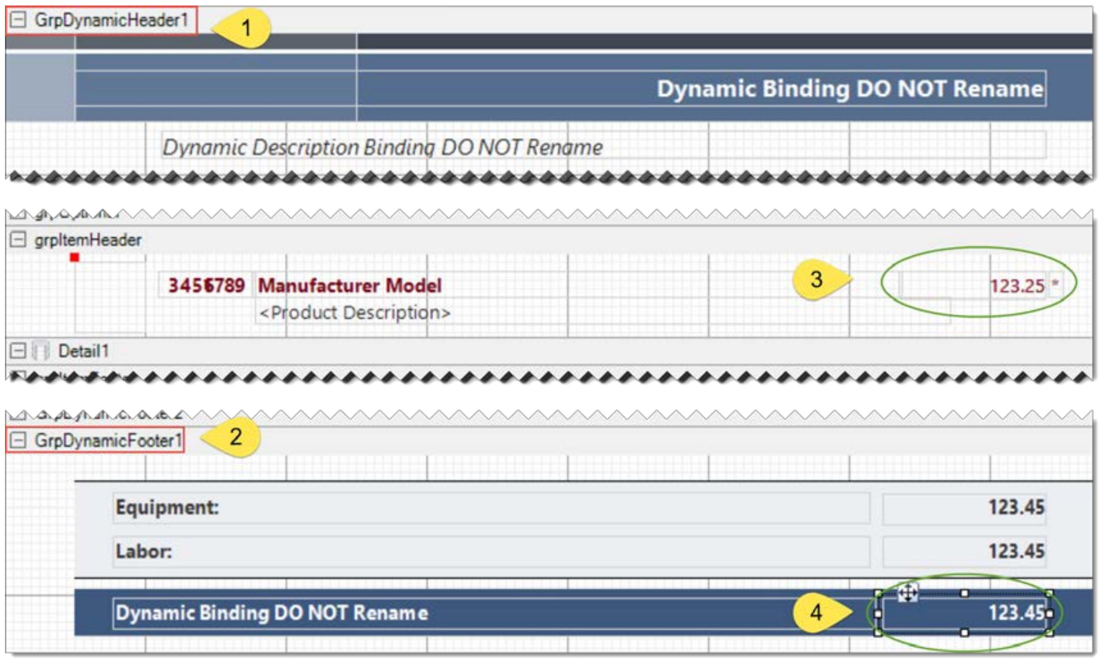
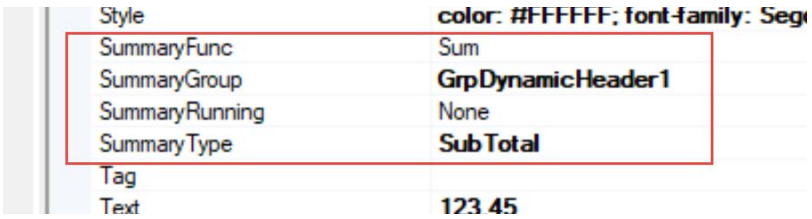

GrpDynamicHeader1
1
Dynamic Binding DO NOT Rename
Dynamic Description Binding DO NOT Rename
grpItemHeader
3456789 Manufacturer Model
<Product Description>
3
123.25
Detail1
GrpDynamicFooter1
2
Equipment:
123.45
Labor:
123.45
Dynamic Binding DO NOT Rename
4
123.45

Figure 17. Group Subtotals

Let us follow the subtotal logic In Figure 17:

1. The section header 'GrpDynamicHeader1' would be where we show the name of a location.
2. The section 'GrpDynamicFooter1' is where we place our subtotal value.
3. In this case, the value we want to provide a subtotal for is likely the product price.
4. Our subtotal function is setup in the textbox and it sums every instance of the datafield in question that occurs between sections one and two (the group header and footer).

Figure 18 demonstrates the summary properties of the textbox necessary to generate a subtotal. The 'SummaryFunc' property is set to Sum. The 'SummaryGroup' is set to GrpDynamicHeader1 which is the section where the sum is reset. That means that when this subtotal needs to be calculated, the value is reset whenever the GrpDynamicHeader1 section is processed. In our example that means the reset happens for each Location. We do not use 'SummaryRunning' for this example. The 'SummaryType' is SubTotal.



Style	color: #FFFFFF; font-family: Seg
SummaryFunc	Sum
SummaryGroup	GrpDynamicHeader1
SummaryRunning	None
SummaryType	SubTotal
Tag	
Text	123.45

Figure 18. Summary Function Settings

Using Grand Totals

Calculating grand total values is simple. For example, we use Grand Total summary functions in our Project Summary reports. The most common way to calculate a Grand Total is to set the 'SummaryType' property of a textbox to 'GrandTotal'. This will process each instance of the datafield referenced by the textbox and sum it.

A Brief Introduction to Report Scripting

The ActiveReports Report Designer offers a robust scripting engine that is largely responsible for the flexibility and options that are available in D-Tools reports. The Report Designer supports both C# (C Sharp) and VB.NET script languages. D-Tools reports typically use VB.NET. Providing an in-depth explanation of VB.NET script syntax, libraries and semantics is beyond the scope of this document. The following information should point the aspiring custom report developer in the right direction.

How to Learn VB.NET

To get started learning some VB.NET touside of the context of D-Tools reports, download a copy of Microsoft's Visual Studio software. Visual Studio is known as an Integrated Development Environment or IDE. Browse to www.visualstudio.com and in the Downloads section look for a 'Community' edition of the IDE. This is a free application that can be used to write VB.NET code and learn the basics. The web is loaded with resources to help you.

A Few Pointers to Get Started

The following topics serve to highlight some important ideas about VB.NET and writing scripts. For a more formal introduction to Visual Basic see Microsoft's Visual Basic Guide online.

Use Comments

What are comments you say? Writing code is like telling a story. Sometimes the story gets confusing. We use comments to help us read between the lines. In VB.NET you only need to type an apostrophe before a line of text to get the system to ignore it and not treat it like code.

```
' writing a comment before a line of code is useful
Dim variableNAme As Decimal ' but placing the comment after the code works too
' adding an apostrophe before a line of code causes it to be ignored
' this is referred to as "commenting out the code"
' Dim ignoredVariable As String
' thoughtful comments make code easier to understand and help you look like a pro
```

Classess, Objects and Methods

VB.NET is an object-oriented language. This means that it uses concepts called classes, objects and methods. Think of a class like a definition. An example of a class might be Vehicle. Car and Truck are classes that inherit preoprties from Vehicle. Car and Truck are similar to Vehicle but they are in ways unique. An object is a specific truck created from the class called Truck. The truck object might be identified using Make, Model, year or VIN. Methods are the things the truck object can do, like honk, accelerate, stop, turn. An object called `blueTruck` might be created from a Class called Truck. To stop the truck you might call a method called stop, like this: `blueTruck.stop`. In D-Tools reports, common use of objects and methods allow us to ask for information like the project grand total or the equipment discount amount. We have access to many methods to help us get the information we need.

Declaring and Using Variables

When we want to store information to use in our script, we declare variables. To create a variable we declare it in the script, with a `Dim` statement. A `Dim` statement declares and allocates memory for a variable. There are many options available when declaring variables but that is a more advanced topic. A simple variable that stores a decimal number representing the project grand total could be declared using `Dim grandTotal As Decimal`. We call `Dim` followed by a variable name followed by an `As` cause and ending with a data type. Now we can store a decimal value and access it using `grandTotal`.

To store a value in `grandTotal` we could write code like `grandTotal = 150.50`. Later in our code we could access the value stored in `grandTotal` by referring to it. Later, we will review how to do just that.

Global Variables

Sometimes we need to store a value in one part of our script but use it in another part. To do this we use global variables. To create a global variable we declare it directly in the script, before we call any events int the report. We can us the same `Dim` statement to create the global variable.

Accessing Variables

You might be thinking, aren't all variables global? I should be able to access a variable from any part of my code once I create it. But this simply is not true. Global variables, created outside of any events, classes or procedures in your code, are available.

An example of a variable that is not always available would be a variable that is part of a Class. Remember a Class is like a definition. We create an object from the Class, use methods to store and retrieve data, and behind the scenes, the object looks to the Class to determine how this is done. The Class may have variables defined where data gets stored. It is not accessible directly but only through the methods of the Class.

Just keep in mind that context matters when we are attempting to store and retrieve data. Variables are generally available within the local context of your code; that is to say, within an event or procedure in the code. Some examples a bit later will shed light on this topic.

Libraries and Methods

VB.NET is more of a framework than a language. The language is Visual Basic (VB) and the libraries and capabilities we will take advantage of are known as .NET. Together they make

VB.NET. The VB.NET ecosystem is broad and powerful. It does much of the heavy lifting and problem solving for you.

VB.NET does this by including libraries for you to use. You can include a library with an `Imports` statement. In some reports we need to write code that handles XML. In VB.NET we can access a wealth of tools made for working with XML by importing the `System.XML` library. To import this library the proper statement is `Imports System.XML`.

Once a library has been imported, it can be used to create Objects. Those Objects typically offer Methods to store and retrieve data. Methods such as these are called Setters and Getters. Setters are used to store values and Getters are used to retrieve them.

Introduction to API Methods

The Report Designer Script tab provides access to the powerful VB.NET framework and it also includes access to the `ReportUtilities` API. The `ReportUtilities` API has a large number of Methods that make report writing more efficient. Using an API Method often helps accomplish otherwise difficult tasks with a single line of code.

Getting and Setting Textbox Values

A common procedure in reports is to get the value of a textbox, store it in a variable and use it for a calculation. And sometimes we want to use a calculated value and place it in a textbox. We have two Methods that are made to do exactly this.

Use the following code to retrieve a value from a textbox and store it in a variable:

```
Dim variableName As String

variableName = ReportUtilities.ReturnTextBoxValue(rpt, “sectionName”, “textBoxName”)
```

In this code we declared a variable called `variableName` that stores String values. Then we used the assignment operator (=) to set a value for the variable. The value is returned from the `ReturnTextBoxValue` Method. This Method requires three arguments. Arguments are the values we pass to the Method so that it can provide us with a proper response. The `ReturnTextBoxValue` Method needs a reference to the report, which has been defined for us as `rpt`. Then we need to pass the name of the report section where the textbox is located followed by the name of the textbox. Notice the section name and textbox name are both in quotes. With this information the Method returns the value stored in the textbox.

The code to set a value in a textbox looks similar:

Use the following code to retrieve a value from a textbox and store it in a variable:

```
Dim variableName As String = “Hello World!”

ReportUtilities.SetTextBoxValue(rpt, “sectionName”, “textBoxName”, variableName)
```

To set a value in a textbox we use the `SetTextBoxValue` Method. This Method accepts four arguments. We pass the `rpt` object, the name of the report section where the textbox is located followed by the name of the textbox and finally we have to pass the value we want to set. In this case we are using the variable that we declared, `variableName`.

Applying Data Source Filters

Applying filters to the data source can be a quick way of getting the results you need. In a report that only needs to show equipment, you might want to filter out labor items. Purchase Order Request reports are unlikely to need to show OFE parts. The API contains many filters to help make sense of the data.

`ApplyAlternateItemFilter`	Excludes "Alternate" items
`ApplyLaborItemFilter`	Excludes "Labor" items
`ApplyNonBillableItemFilter`	Excludes "Non-billable" items
`ApplyOFEItemFilter`	Excludes "OFE" items
`ApplyOptionalAndAlternateItemFilter`	Excludes "Alternate" and "Optional" items
`ApplyOptionalItemFilter`	Excludes "Optional" items

Each of these Methods is called like this: `ReportUtilities.ApplyXXXFilter(rpt)`

Methods Providing Project Totals

It is often helpful to be able to get a "total" value directly from the API. The grand total, equipment total, labor total, the total of adjustments, etc. These are all useful Methods. In a Project Summary report, we can quickly fill in values using Methods that return project totals.

`TotalEquipmentAdjustment`	Total Product Adjustment for the project
`TotalEquipmentPrice`	Total Product price for the project (after adjustments)
`TotalLaborAdjustment`	Total Labor Adjustment for the project
`TotalLaborPrice`	Total Labor price for the project (after adjustments)
`TotalMiscCosts`	Total Misc Costs for the project
`TotalMiscParts`	Total Misc Parts for the project
`TotalProjectPrice`	Total Price of the project
`TotalProjectTax`	Total Tax for the project

These Methods are only an introduction to the `ReportUtilities` class. Many more Methods and Properties exist. Review the latest available ReportUtilities Class Library (need link) for additional information.

Solutions to Common Reporting Scenarios

Combine a Proposal Report with a Contract

In the exercise we will start with a standard Proposal report, adjust the size of a section and then embed a standard contract report in the new section.

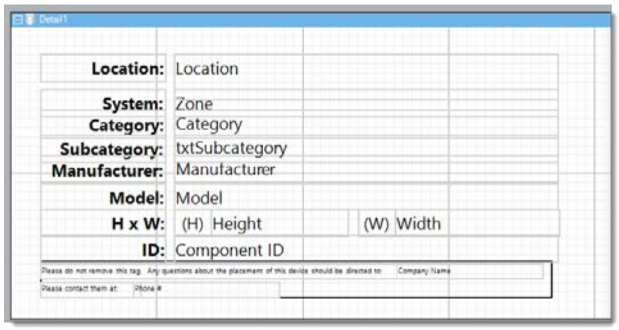
1. Starting from the Report Designer, click File → New.
2. In the New Report Wizard, leave the default values and click Next.
3. In the list of reports, find the 'Proposal' report. Select it and click Next.
4. Give this report a name like 'Proposal with Contract' and click Finish.

5. When the report opens, scroll to the bottom of the report and select the section header named 'grpSummaryDetail'. In the Properties section, increase the Height property from 3.709 to 4.009. This will provide some working room at the bottom of the section.
6. Select the PageBreak control from the ToolBox on the left of the interface. Below the last textbox control, in the space created in the previous step, begin to drag a line across the page. A PageBreak control will be placed in the section. Make sure it is below the existing controls.
7. Select the SubReport control from the ToolBox. Now place and configure the control:
    a. Starting below the new PageBreak control, drag a rectangle from left to right across the report. A SubReport control will be drawn.
    b. Select the new control and set the X value of the Location property to zero. Do not change the Y coordinate unless necessary.
    c. For the Size property, set the value 7.9, 0.375.
    d. Right-click on the SubReport control and choose 'Bind to D-Tools Report'. Select the 'Contract' report and click OK.
8. Preview the report and check for appropriate formatting and spacing.
9. Publish the report.

Create a Custom Product Label Report with Unique QR Codes

In the exercise we will modify an Avery label report and add a custom QR code that represents a unique value for each label printed. This could be useful to tag items in a warehouse.

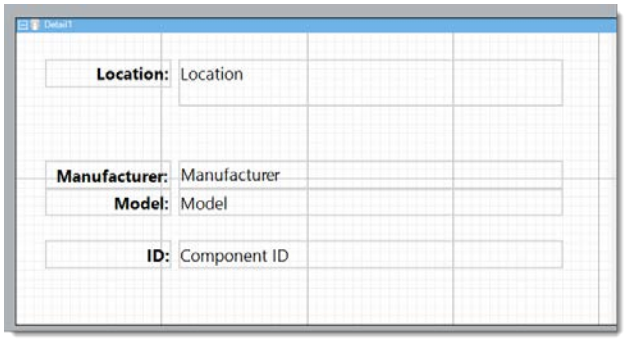
1. Starting from the Report Designer, click File → New.
2. In the New Report Wizard, leave the default values and click Next.
3. In the list of reports, find the 'Avery Walk Thru - 5263' report. Select it and click Next.
4. Give this report a name like 'Equip Labels with QR Code - 5263' and click Finish.
5. When the report opens, note that several of the textboxes do not apply in this scenario (see Figure 19). Deleted textboxes until the report detail resembles Figure 20.



Location:	Location
System:	Zone
Category:	Category
Subcategory:	txtSubcategory
Manufacturer:	Manufacturer
Model:	Model
H x W:	(H) Height (W) Width
ID:	Component ID

Please do not remove this tag. Any questions about the placement of this device should be directed to: Company Name
Please contact them at: Phone #

Figure 19. Default Report



Location:	Location
Manufacturer:	Manufacturer
Model:	Model
ID:	Component ID

Figure 20. Several TextBoxes removed

6. Adjust the locations of the existing labels and TextBox controls towards the top of the report. This will provide some working room at the bottom of the section.

7. Select the Barcode control from the ToolBox. Drag a control on the bottom-right corner of the report. Select the new control and set the Size property to 1,1. The results should look similar to Figure 21.
8. Select the Barcode control. In the Style property dropdown select 'QRCode'.
9. Since the Component ID is always unique, we will use it for the QR Code value. In the DataField property, type 'dtr:ComponentID'.

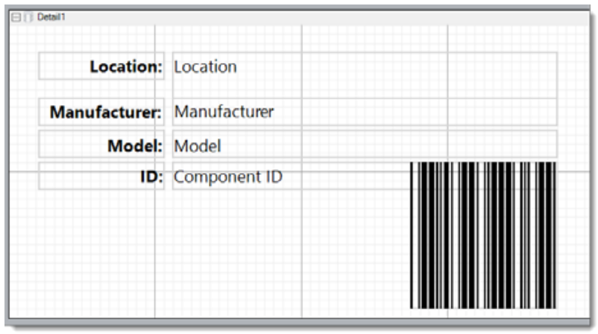


Figure 21. Barcode added

10. Preview the report. Note that each QR Code is unique.
11. Publish the report.

Use the Company Profile Feature with Embedded Data

In the exercise we will add a Company Profile RTF document with embedded data tags. The Company Profile will represent a canned letter to the customer with client and project data added at runtime.

1. Open WordPad and copy/paste the following sample text:

```
Thank you for the opportunity to be involved with your upcoming project!

We have assembled the following proposal for the project located at
[!ADDRESS] in [!CITY], [!STATE].

We look forward to the successful completion ...
```

2. Save the file on your Desktop as 'CompanyProfile.rtf'.
3. In SI, navigate to the Control Panel → Company Information → Profile (RTF). Click 'Attach' and browse to your RTF document. Save and Close.
4. Now, from the Report Designer, click File → New.
5. In the New Report Wizard, leave the default values and click Next.
6. In the list of reports, find the 'Company Profile' report. Select it and click Next.
7. Give this report a name like 'My Company Profile' and click Finish.
8. Notice the RTF control is located in the section labeled 'Detail1'. Select the TextBox control and draw a small TextBox, in the 'Detail1' section along the left side of the report. Select the new control and:
    a. Set the Visibility property to False.
    b. Change the (name) property to 'txtAddr'.
    c. Set the DataField to 'dtr:PresentedTo/dtp:Address1'.
9. Repeat step 8 for two additional TextBox controls.
    a. Set the Visibility property to False for each TextBox.
    b. Change the (name) property to 'txtCity' and 'txtState'.

    c. Set the DataField to ‘dtr:PresentedTo/dtp:City’ and ‘dtr:PresentedTo/dtp:State’ respectively.

10. Place your cursor on the ‘Detail1’ section header and double-click. This will take you to the Script portion of the report and place your cursor in the appropriate code section.
11. Find the ‘Sub Detail1_Format’ event. Below the existing code add the following lines, so the event looks like:

```
Sub Detail1_Format
       ReportUtilities.SetCompanyProfile(rpt, "Detail1", "txtCompInfoRTF")
       ' cpature the street address
       Dim addr As String = ReportUtilities.ReturnTextBoxValue(rpt,
"Detail1", "txtAddr")
       txtCompInfoRTF.ReplaceField("ADDRESS", addr)
       ' capture the city
       Dim city As String = ReportUtilities.ReturnTextBoxValue(rpt,
"Detail1", "txtCity")
       txtCompInfoRTF.ReplaceField("CITY", city)
       ' capture the state
       Dim state As String = ReportUtilities.ReturnTextBoxValue(rpt,
"Detail1", "txtState")
       txtCompInfoRTF.ReplaceField("STATE", state)End Sub
```

12. Preview the report. Note that the sample report data contains no address so the embedded data will be blank.
13. Publish the report and test against your project data.

Create a Custom Cover Page the includes a Project Resource

In the exercise we will start with a standard Cover Page report, add a TextBox for the salesperson’s name that populates using the Roles/Resources settings.

1. In SI, navigate to the Control Panel → Users. Editing your User Account, click the Roles dropdown list. Select ‘Sales Rep’, then Save and Close. Close the ‘Manage Users’ window.
2. From the Project Explorer click Reports → Report Settings → Data. Under ‘Resources’ set ‘Resource 1’ to ‘Sales Rep’.
3. Starting from the Report Designer, click File → New.
4. In the New Report Wizard, leave the default values and click Next.
5. In the list of reports, find the ‘Cover Page’ report. Select it and click Next.
6. Give this report a name like ‘My Cover Page’ and click Finish.
7. Towards the bottom-left of the report, select the Label containing the text ‘Presented By:’. Narrow the TextBox control by changing the X value of the Size property from 2.35 to 1.
8. Select the TextBox control from the ToolBox and draw a TextBox to the right of the previous Label. Use the alignment and sizing tools to set an appropriate size for the new control.

9. Select the new TextBox control and set the DataField property to 'dtr:Resource1'.
10. Preview the report. Note that no resource value is shown. The test report data has no resources assigned. Publish the report.
11. Open the Information tab of a project and click on the Resources option. Click Assign and pick an appropriate resource from the list.
12. Run the report. The 'Sales Rep' name should be listed above the company name.

Embed an RTF Document into a Report Section (Content under development)

Coming in a future installment of this guide

Create a Custom Purchase Order Report for Excel Export (Content under development)

Coming in a future installment of this guide

Proposal Report with Embedded Scope of Work for individual Systems (Content under development)

Coming in a future installment of this guide

Read Custom Excel Tables and Create Tables in Reports (Content under development)

Coming in a future installment of this guide