THE EXPERT'S VOICE® IN SQL SERVER

Business Intelligence with SQL Server Reporting Services

DELIVERING POLISHED BI WITH SQL SERVER

Adam Aspin

Apress®

Contents at a Glance

Introduction

Business intelligence (BI) means different things to different people. For some, it means advanced analytics. For others, it is self-service report creation. A few see it as a prelude to Big Data. For many users, however, BI is much more accessible than any of these things. It is quite simply the regular production of clear and meaningful reports and dashboards that translate data into actionable information.

In my experience, all that a majority of business people ask for is up-to-date data that they can access when they want in a ready-to-use format. They do not always have the time or the skills to develop their own flashy output. Most end users have no desire to delve deeply into vast realms of data to track processes, sales, and targets. They want only to identify anomalies and react swiftly to changing business conditions.

So, despite all the hype, there is still a place for good old corporate business intelligence. This is the mixture of data, technology, and IT skills that creates and distributes “canned” reports in all shapes and sizes to a multitude of users across an organization. Despite being out of the spotlight (thanks to the glare of self-service BI), corporate business intelligence still solves the vast majority of enterprise reporting needs. It delivers reliable and trusted data to users who have only to select a few parameters to get the results they need.

This book is about energizing this forgotten, but fundamental sector of the BI universe. More specifically, it aims to show you how you can capitalize on a technology that you probably already have–SQL Server Reporting Services–to push your corporate BI to the next level. After reading this book you will be able to produce designer dashboards that integrate gauges, maps, charts, and text to deliver targeted information. You will learn to exploit the subtleties of a variety of visualizations (KPIs, sparklines, trellis charts, bubble charts, and many others) to give your reports real pizzazz. Specifically, you will learn how to make important trends stand out, while providing clear visual alerts to draw your attention to significant thresholds and deviations from targets. All this can be done via a solid and reliable toolkit that is used regularly by a vast developer community around the world.

In this book you will also learn how to revamp the appearance of your reports to make them more user-friendly. You will learn how to add slicers and other interactive tweaks to make your output better adapted for delivery to mobile devices such as tablets and cellphones.

This book is not meant as an introduction to SSRS. It presumes that you already have some experience of producing basic reports. The aim here is to help you to build on your experience to develop more advanced scorecards and dashboards and then to adapt the output for mobile devices.

In the book you will see dozens of visualizations that can enhance the ways in which business intelligence is presented. All of them-and more-are available on the Apress web site in an SSDT project. Feel free to take them as the basis for your own reports and dashboards, and to adapt them to your specific needs.

CHAPTER 1

SQL Server Reporting Services as a Business Intelligence Platform

Not a month goes by without some shiny, new, all-singing, all-dancing business intelligence product being announced as the Next Big Thing. A few months later, no one can remember it ever existed, except your CEO, who wants to know what her organization is using to deliver corporate business intelligence (BI). She wants a solid, stable, and mature platform that can deliver flawless reports to users automatically and regularly. She does *not* want her staff wasting time developing their own reports and learning corporate data structures. Secretly, many users agree with her. She *does*, however, want a certain level of interactivity. Of course, she expects BI on mobile devices and a tangible level of "wow factor." At least, that is what you thought she said, before your eyes glazed over in fear as you wondered which product was going to be forced on your lovingly engineered SQL Server universe.

The product quite simply could be SQL Server Reporting Services.

SQL Server Reporting Services (SSRS) has been around for over 15 years. Longevity like this, fortunately, can have its advantages, and in the case of SSRS they are the following:

- SSRS is a mature, stable, and efficient platform.
- SSRS can accept data from many different sources, most often relational as well as dimensional and tabular.
- SSRS handles the details of scheduled report creation, snapshots, and automated report delivery with consummate ease.
- SSRS slots into a complete SQL Server BI stack efficiently and smoothly.
- SSRS reports can now also be made available in the cloud using Power BI.

Often underestimated, SQL Server Reporting Services is an extremely powerful environment for delivering clean, clear, and polished reports to users. If you add a final layer of BI polish to the mix (and that is what this book is all about), then you have an incredibly wide-ranging and effective business intelligence tool at your disposal.

SSRS is *not*, however, designed to be a self-service BI tool. It exists to deliver corporate BI, where prebuilt reports are what the user wants. Indeed, in this world the user specifically does *not* want to develop his or her own reports. They may want to change some core parameters or drill down or across through data, but they definitely do not want to spend (they might even say "waste") their time formatting their own reports. These users prefer to have an MI (Management Information) department prepare their reports for them.

So, if self-service BI is what you are looking for, then Power BI is probably a place to start, and my book *High Impact Data Visualization with Power View, Power Map, and Power BI* (Apress, 2014) should be your first port of call.

However, if you are looking at delivering pre-built, polished, and interactive business intelligence to your users, where timely, accurate, and attractive information delivery is the major requirement, then SSRS is probably the tool that you need. It can provide the following:

- KPIs
- Scorecards
- Dashboards

These features can be delivered to schedule or displayed on demand. The data can be updated according to your business requirements. Not only that, but SQL Server Reporting Services, if you are using SQL Server 2012 SP1 or later, can deliver your BI not only to browsers, but also to smartphones and tablets.

While users cannot generate reports, and slice and dice the data as they can in the self-service world, they can

- Set multiple parameters to define the data that will be displayed.
- Drill down into report hierarchies.
- Drill across to other reports

This book exists to show how you can use SSRS to provide corporate business intelligence that delivers all that is described above, plus add the "wow" factor that your boss wants. The trick is to learn how.

Business Intelligence Concepts

As this book is about delivering business intelligence using SQL Server Reporting Services, it will help to clarify a few basic concepts from the outset. When delivering corporate BI, you will mostly be developing key performance indicators, scorecards, and dashboards. Let's begin by defining these as succinctly as possible.

Key Performance Indicators

Key Performance Indicators (KPIs) have been an essential part of business intelligence for many years. They display a set of selected fundamental measurements that managers use to make productive decisions. A classic KPI will probably contain the following:

- A *value*: This is the figure returned from a business system.
- A *goal*: This is the figure that should have been attained, and comes from a budget document that has been loaded (somehow) into an accessible data source.
- A *status*: This is nearly always shown as a symbol that indicates how aligned the value is to the metric (in other words, how well you are doing). In some cases, it can be the color of a display element.
- A *trend*: This shows how the metric being measured has evolved over time and it can be displayed as a symbol or a small graph called a sparkline.

Put simply, a KPI will let you display a result, how it maps to target, and where it is going. It lets you see anything that can be measured that is of strategic or tactical importance.

Scorecards

Scorecards are collections of KPIs. They might all relate to the same area of business, or give a view across multiple business areas. Scorecards might themselves become part of a dashboard.

Dashboards

There are many overlapping–and sometimes conflicting–definitions of what makes up a dashboard. I have no intention of getting caught up in arcane theoretical disputes, so I will consider a dashboard to be a visual overview of essential corporate data. It can display many possible data snapshots of multiple aspects of a business, or show a highly specific set of metrics relating to a business area or department. I consider that it can include targets and objectives, but that these are not compulsory.

A dashboard can be made up of multiple elements, which many people also call "widgets" or visualizations. SSRS lets you use the following as the building blocks of your dashboards:

- KPIs
- Scorecards
- Tables
- Charts
- Gauges
- Sparklines
- Maps
- Images

These core elements can often be combined to produce multiple variations on a theme. How they can be added to reports and combined to deliver BI is the subject of the book.

SSRS for Business Intelligence, Practically

Now that you have had a rapid overview of the theory, it is time to get practical. You need to see more generally how SQL Server Reporting Services can deliver BI elements to your users. Specifically, you need to see what you will be doing when developing your BI solutions.

With SSRS for Business Intelligence, you will be developing the following:

- *Reports*: This is a more generic term that I use to describe an SSRS `.rdl` file, whatever the type of presentation it contains.
- *Visualizations*: BI dashboards, scorecards, and reports are normally made up of several elements. They could be charts, gauges, tables, maps, images, or combinations of these elements. I use the term *visualization* to describe a single element, even if it is composed of different elements from the SSRS toolbox. For instance, a KPI containing a sparkline, some metrics, and an indicator will make up a single visualization.
- *Widgets*: A widget is any element (or object) that can be stored independently and reused. In most cases it is really only a synonym for a visualization.

Delivering BI is not just a technical matter. It involves the following three basic threads that have to be melded into a final delivery:

- Data
- Design
- Platform

Let's take a look at these three.

Getting the Data Right

Making sure that the data is perfect is the key to delivering effective business intelligence. The data must be not "good enough," not "nearly right," but absolutely one hundred percent accurate. Obviously, this book is not the place to discuss data architecture, data cleansing, data quality, master data, or the definition of metrics and processing for BI. However, in the real world, you will have to deal with all of this.

As we are only dealing with the presentation layer in this book, I will simplify the learning curve by using a single database whose data we will presume is perfect (and thus ready for use in SSRS) to underpin the visualizations and dashboards that you will be creating. I will also make this single database the repository for all ancillary information that you may need when developing your outputs. By "ancillary" I mean data such as

- *Budget data*, used in KPIs, for instance
- *Geographical data*, used when creating maps or analyzing data by country or region
- Any other data that is *not* habitually sourced from an OLTP or OLAP system, such as certain types of reference data

However, having the data accurate and accessible is only the first part of the story. How you then present the data to SSRS is also fundamental. There are several aspects to this.

Prepare the Data

As you could well be sourcing data from multiple tables (for instance, you could be aggregating sales data from one table or view and comparing it with budget data from another set of tables), I generally advise that you mash up the data in the database *before* you present it to SSRS. There are several advantages to this approach, compared to filtering, joining and calculating data in SSRS itself:

- You are using data from a database, so it seems more logical to use the unrivalled power of the database language (T-SQL) rather than the reporting tool. SSRS is necessarily more limited where data manipulation is concerned.
- The heavy lifting takes place where the data is kept: in the database.
- SQL Server provides an amazing toolset to help you collect, rationalize, and output data. So do not be afraid to use temporary tables, CTEs, table variables, and all the other tools in the T-SQL toolkit to shape your data.

So what should your aims and objectives be when preparing datasets in T-SQL? Everyone has their own opinions, but I suggest that you take the following points into consideration when setting up your data for reports:

- Output only the precise data that is needed by your report. Extra rows and columns "in case they might be useful one day" slow down report processing and weigh on network transmission needlessly.
- Similarly, your T-SQL may include intermediate columns used for calculations. Often these are not displayed in the final report, so you may not need to output them to SSRS. They will likely only cause confusion when you (or your team) are developing or maintaining the report.
- Do not make code re-use into a target. Dashboards and reports change all the time, so trying to develop "one size fits all" stored procedures or scripts that underlie a group of reports can be a waste of time. Be prepared to create a stored procedure for each different visualization. This way, when the requirement for a minor change comes in, you will not have to worry about its ramifications across a whole range of dashboards. Alternatively, in the cases where you have a clear group of reports that are all based on the same data, reusable stored procedures can be a boon.
- Do document your code to remind yourself of any common elements across stored procedures. This way you know that if you change something for one chart (say), then you will need to modify other stored procedures too.
- Try where possible–and this may be a contentious point–to provide all the required data for a visualization in a single data set. This may seem obvious, so let me explain further. Some visualizations require data that is not a metric, such as the maximum for a gauge pointer or the thresholds used in ranges. These are only single, scalar figures. However, it can be easier sometimes to add them as columns to a dataset (even if the same figure is repeated identically in every record) and use them as the source data. This often only adds a few columns to a datasource, and has the advantage of keeping all the data in the same place for a visualization, and not cluttering up the report with added datasets containing only one or two values.
- Sometimes, however, you may want or need to reuse data such as thresholds or maximum values in different visualizations in the same report. In this case, do use separate datasets in a report to access these elements.
- Remember that in some cases (the second KPI in Chapter 2 is an example) you will have to create several datasets for a single visualization. Experience will tell you when you have to use multiple datasets and subsequently link them inside SSRS to produce the required result.

Of course, you may disagree strongly with these ideas. Feel free to do so; the essential thing is that you have thought through the reasons for how and why you have set up the data in the way that you have chosen. If you can justify them to yourself, then you can insist on them for your team–and defend your decisions to your boss.

Use Views and Stored Procedures

While it is perfectly possible to custom code every piece of SQL (or MDX or DAX) that feeds data into SSRS, it is frequently a much better practice to aim for some reusability. This nearly always means adding views and stored procedures to your data layer.

Views

It can often be worth the effort to prepare views that you can use and re-use as a conformed source of source data. You might be able to use views directly in certain reports, or (more likely) use them as the basis for specific processing and output using a stored procedure. If you will repeatedly be joining tables or aggregating data, then a view will save you a lot of repetitive coding. Of course, in most situations you will never really know what data you will be using before you start building visualizations. However, it can be worth refactoring your initial SQL queries into views earlier rather than later in the project lifecycle.

Stored Procedures

SSRS will let you use both T-SQL code blocks and stored procedures to assemble and deliver data as a dataset. However, I am a firm believer in using stored procedures rather than freeform code blocks in SSRS. My reasons include the following:

- Data manipulation is centralized in the database. You will not spend time opening multiple reports and then digging through datasets and scrolling through code in a tiny dialog in SQL Server Data Tools.
- Parameters are created automatically and correctly when you specify a stored procedure as a data source for a data set.
- Large and complex processes delivered as stored procedures will not exceed the character limit of the SSRS code box.
- You work using a suitable UI, rather than trying to code in a text box or (possibly worse) copying and pasting back and forth between SSRS and SQL Server Management Studio.
- You are a data professional and so probably feel happier using a more structured and coherent method of working.
- You are minimizing the risk of SQL injection attacks.

Some Ideas on Source Data Definition

I believe strongly that preparing the data for a visualization is a fundamental part of successful BI report creation with SSRS. This is why for every example in this book I always not only give the code used to deliver the required data, but show sample output and then explain the reasoning behind the T-SQL. I do this to draw the reader's attention to some of the many ways in which you can prepare data for BI reporting. As you will see, I tend to use temporary tables and CTEs quite regularly. I also tend to break down separate logical elements into small elements of code, rather than aim for monolithic structures. These are personal choices, and all I want to do is to provide some hints and suggestions for code use–and re-use. How you write your code is your choice. All I can suggest is that you try to make it as efficient as possible.

There are many design decisions that you will have to make when defining the datasets that feed into your reports. You may prefer to create many datasets–possibly not only one, but several for each widget. Alternatively, you might try to cram everything into a single datasource for all the elements on a dashboard, and then apply filters to apply only the relevant data to specify charts or gauges individually.

My approach is usually to find a common ground between these two extremes. In most cases, I find that a data source can be created (and subsequently filtered) for associated groups of elements–gauges, charts, or KPIs that make up a visualization. Sometimes an element may need several datasets. In yet other cases you may need multiple datasets because you are using disparate data sources.

Anyway, in this book you will find examples of most of these approaches. This will not mean that a certain way to prepare the data is best suited to a certain type of visualization, just that I am taking advantage of a scenario to show some of the ways that datasets can be used–and combined–in SSRS.

On a more general note, I find that spending time defining the data requirements at the start of a project–or even for a single BI element–can help me to focus on what each visualization is trying to achieve. It can also save a lot of development time as you will minimize the number of times that you modify your T-SQL code, and subsequently add, rename, and delete columns of data in SSRS datasets and objects. So I can only advise you to think through the data before you start building a chart, gauge, or map. To encourage good practice, therefore, I begin every example with the model that you are aiming to build and the code needed to produce the data that is required. This way you can see what is required for each visualization, why you need it, and what is, in my opinion, the best way to deliver it.

Of course, my SQL programming style may not be the same as yours. After all, we all have our own way of delivering BI to our users. So I am not saying “this is how you must do it,” but merely “here are some ideas based on my experience.” How you prepare the data for your dashboards and reports is completely up to you.

Real-World Data

In the real world, you will use data from many different sources in a variety of formats. SQL Server might not be the only relational database you are using. Relational data may not be the only data format; you could be using dimensional data from a SQL Server Analysis Services database or even data in the Analysis Services tabular format.

SSRS will allow you to use data from these varied platforms easily and simply. However, I decided not to use all these potential data sources in this book and stick to SQL Server relational data for the following reasons:

- Not everybody is using all the potential data sources that SSRS can handle.
- Many users are likely to be federating data in a SQL Server reporting layer (using techniques such as linked servers or ETL processes using SQL Server Integration Services).
- All readers are likely to know T-SQL, which makes it a comprehensible lingua franca for explaining how data is produced.
- Using SQL Server as the data layer means that readers can concentrate on SSRS rather than getting distracted by other languages such as MDX or DAX.

However, preparing the data is not the end of the process. You need to be able to reuse data in certain circumstances, and almost certainly you will need to cache both data and reports to ensure that your users enjoy a top-class BI experience. The final chapter of this book will outline some of the techniques that can help you enhance the user experience and deliver output smoothly–and quickly.

Designing SQL Server Reports for Business Intelligence

Once your data is in place, you can move on to designing the output. This is admittedly the fun part, and is often where the real challenge resides. You need to remember this, especially when faced with seemingly impossible demands from management or users, or apparently insane demands for stylistic pirouettes from the Corporate Style Police.

Presentation

Your first choices will inevitably concern the presentation approach that your reports will be taking. They could be

- Text-based
- Graphical
- Hybrid

You could be aiming for a traditional look and feel, or attempting something different. The choice is yours, as long as you remember that variety should not mean distraction, and novelty can wear off faster than you think.

Design

This is the area where your choice of backgrounds, borders, images, and text will come into play. Color and style are vast areas, and ones that are probably best left out of the hands of technical people like me. Nonetheless, a few comments are necessary.

The French have a saying that translates as "there is no discussing taste and color." This usually means that everyone knows that they (and only they) are right in matters of good taste–especially where report design is concerned. Moreover, any discussion with these people is pointless as it is clear (to them, at least) that *they* (whoever they are) are right. I agree: they (or even you) are always right. Or more probably, your boss, or another department somewhere in the organization, is inevitably right when it comes to the choice of colors and report presentation generally.

Consequently, to placate these people, I have adopted a color scheme that avoids garish colors and uses shades of gray for text and borders. This approach lets the information speak louder than the color scheme. You, and your internal clients, may prefer other color palettes and design styles, and this is up to you. The steps describing color and style are only there to indicate how to apply colors, not necessarily the color that you must use.

In some cases, I have thrown these principles out of the window, as style can also be a function of the delivery method. So, specifically when defining output for smartphones, I tend to use bold primary colors or even white-on-black displays. This is because on a phone a competing distraction is only a swipe away, so I want to grab the user's attention.

One thing that you will have to make abundantly clear to your users–and in-house style gurus–is that SSRS is not a design application. It cannot deliver the kinds of currently fashionable interfaces that certain other products can. What it can do (and this is the mantra you will need to repeat) is deliver BI in a fraction of the time and cost of many of the alternatives. Not only that, but it integrates perfectly into your SQL Server ecosystem, does not require expensive training, and is easy to maintain. Oh, and you will not need the licensing costs of third-party software. Or the extra consultants to get it up and running. Or to explain exactly why all of your reporting infrastructure needs to be replaced, yet again.

Layout

Having admitted that there are limitations to what SSRS can deliver visually, I now want to compensate by saying that it can nonetheless produce some really slick and impressive output. To get the most out of SQL Server Reporting Services to deliver visually compelling BI, you need to know a set of techniques–and a few tricks–that help you get the job done. These include

- Aligning visualizations
- Enclosing objects in other objects

- Controlling the size and growth of objects
- Applying images as backgrounds to many objects

Ideas for these aspects of report creation are discussed in Chapter 7.

Interface and Interactivity

You probably do not need me to remind you that the SSRS web interface (Report Manager) will never win any prizes for ergonomics. With a little effort, however, you can revamp and even replace this tired old UI to give the user a more pleasant experience. This revamping also covers ways to enhance the (admittedly limited) interactivity that is on offer.

This revamping is mainly based on using the following features:

- *"Postback" and hidden variables*: This means using the `Action` property of an item on a report to re-display the same report where some parameters are changed and others kept the same.
- *SSRS Expressions*: Expressions are the key to enhancing most aspects of SSRS, and interactivity is no exception. You will see many examples of expressions in this book. Fortunately, you do not have to learn an entire programming language, as only a handful of Visual Basic keywords are essential when revamping reports. So, if you are prepared to learn to use `If()`, `Switch()`, `Lookup()` and one or two others, then you can bring the SSRS interface into the 21st century.

Chapters 8, 9, and 10 will show you some of the techniques that can be applied when revamping the report interface.

Multi-Purposing

Many of the techniques and most of the widgets that you learn to build in this book can be used in a plethora of circumstances. Just because you learn to apply a technique to tablet BI does not mean that it cannot be used effectively in reports that are read on a laptop. Certain techniques explained in the context of output for smartphones can be used on a PC. So please do not think that just because I may explain an approach to business intelligence delivery in the context of a certain output device that you can only use it on that specific device. Indeed, I encourage you to experiment and to discover which techniques work best in which circumstances and on which devices.

The Sample Database

The sample database that is used throughout this book as the basis for all reports contains the sales data for a small English car reseller called Brilliant British Cars Ltd. It has been going since 2012 and now exports a small set of English luxury and sports cars to Europe and North America.

Preparing Your Environment

If you intend to follow the examples in this book, you will need to set up a SQL Server Data Tools (SSDT) environment that is correctly configured so that you do not waste any time and can create your own version of the reports that are detailed in the various chapters. If you have copied the CarSalesReports solution from the Apress web site and installed it as described in Appendix A, then you already have an environment that contains all the examples. However, as it can be fruitful to build your own reports from scratch, here is how to set up and configure an empty SSDT project.

Creating an SSDT Project

The first thing to do is to create an SSDT project.

1. Run SQL Server Data Tools (Start ➤ All Programs ➤ SQL Server 2014 ➤ SQL Server Data Tools).
2. Click New Project.
3. Click Report Server Project. Set the name and select a location such as `C:\BIWithSSRS`. Uncheck Create directory for solution.
4. Click OK.

■ **Note** I am presuming that you are using the 2014 version of SQL Server. However, 2012 will work just as well. Indeed, for everything except output to mobile devices, SQL Server 2008 R2 will work just as well.

Adding a Shared Datasource

You now need to add a shared datasource that connects to the source database. I am presuming that you have downloaded and restored the `CarSales_Reports` database as described in Appendix A.

1. Display the Solution Explorer window (Ctrl+Alt+L).
2. Right-click Shared Data Sources and select Add New Data Source from the context menu.
3. In the Shared Data Source Properties dialog, enter the following:
   a. *Name*: CarSales_Reports.rds
   b. *Type*: Microsoft SQL Server
   c. *Connection string*: `Data Source=YourServer\YourInstance; Initial Catalog= CarSales_Reports`
4. Click OK.

■ **Note** You can, of course, click Edit and configure the data source to point to another database using a different security configuration.

Add Shared Datasets

You now need to add the shared datasets that are used in the examples in this book. Before adding them, it is best to know what they are and what they do.

- `CurrentMonth`: This gives the current month that is passed as the default to the `ReportingMonth` parameter of most reports.
- `CurrentYear`: This gives the current year that is passed as the default to the `ReportingYear` parameter of most reports.
- `ReportingFullMonth`: This gives the month names and month number. It is used in a couple of reports to display the full month name as a report parameter.
- `ReportingMonth`: This gives the months in the year. It is used in the `ReportingMonth` parameter (found in most reports) to allow the user to choose the month for which data will be displayed.
- `ReportingYear`: This gives all the years for which there are data in the database. It is used in the `ReportingYear` parameter of most reports to allow the user to choose the year for which data will be displayed.

You add a shared dataset like this, using `ReportingYear` as an example.

1. Right-click Shared Datasets (in the Solution Explorer window) and select Add New Dataset from the context menu.
2. Enter a dataset name (`CurrentMonth` in this example) and set the Data source to `CarSales_Reports`.
3. Set the Query type to Stored Procedure.
4. Select (or paste) the stored procedure `Code.pr_ReportingYear` as the stored procedure name.
5. Click OK.

You can now create the five other shared datasets, using the following properties:

Dataset	**Property**	**Value**
`CurrentMonth`	Query type	Stored Procedure
	Stored procedure name	`Code.pr_CurrentMonth`
`CurrentYear`	Query type	Stored Procedure
	Stored procedure name	`Code.pr_CurrentYear`
`ReportingYear`	Query type	Stored Procedure
	Stored procedure name	`Code.pr_ReportingYear`
`ReportingFullMonth`	Query type	`Text`

(*continued*)

Dataset	Property	Value
	Query	`SELECT 1 AS ReportingMonth, 'January' AS MonthName` `UNION` `SELECT 2 AS Expr1, 'February' AS Expr2` `UNION` `SELECT 3 AS Expr1, 'March' AS Expr2` `UNION` `SELECT 4 AS Expr1, 'April' AS Expr2` `UNION` `SELECT 5 AS Expr1, 'May' AS Expr2` `UNION` `SELECT 6 AS Expr1, 'June' AS Expr2` `UNION` `SELECT 7 AS Expr1, 'July' AS Expr2` `UNION` `SELECT 8 AS Expr1, 'August' AS Expr2` `UNION` `SELECT 9 AS Expr1, 'September' AS Expr2` `UNION` `SELECT 10 AS Expr1, 'October' AS Expr2` `UNION` `SELECT 11 AS Expr1, 'November' AS Expr2` `UNION` `SELECT 12 AS Expr1, 'December' AS Expr2`
`ReportingMonth`	Query type	Stored Procedure
	Stored procedure name	`Code.pr_ReportingMonth`

Configuring Parameters

Nearly every report used in this book will use two parameters to choose the month and year for which data is displayed. These parameters are

- `ReportingYear`
- `ReportingMonth`

Here is how to add them to a report.

1. With a report open (or freshly created) right-click Parameters in the Report Data window and select Add Parameter from the context menu.
2. In the General tab, add the parameter name (`ReportingYear`, in this example).

3. Set the data type to Integer.
4. Click Available Values on the left.
5. Select Get values from a query.
6. Set the following:
    a. Dataset: ReportingYear
    b. Value field: ReportingYear
    c. Label field: ReportingYear
7. Click Default Values on the left and set the following:
    a. Dataset: CurrentYear
    b. Value field: CurrentYear
8. Click OK.

You can now do exactly the same for a second parameter named CurrentMonth. The only differences are that the Dataset, Value field, and Label field will use ReportingMonth for the available values, and the default values will use CurrentMonth for the dataset and Value field.

■ **Note** Even if you do not add these parameters to a report that uses them, they will be added by the stored procedures that return the data (in most cases). However, you will still have to configure them as described above for the reports to function correctly.

You can now use this project to build the examples from this book without any risk of overwriting the examples in the sample project. Indeed, you can then compare your work with the sample report files in the sample project, or even copy and paste items between projects if you want to speed up the learning curve.

When you are following the examples in this book, you will see that I am giving each SSRS report the name of the example that you can find in the CarSalesReports project. This way you can compare your work with the model that was used to produce each image that you can see for each example. You can name your reports anything you want, of course.

Code and Stored Procedures

All the code examples in this book are shown as stand-alone code snippets that you can run as-is in an SSDT query window (assuming that you have downloaded the `CarSales_Reports` database from the Apress web site, restored it to your computer, and are running the query against this database). When it comes to creating the actual reports, however, I will presume that you have either made the code into a stored procedure, or are using the referenced stored procedure from the `CarSales_Reports` database. As I am assuming that you know how to create and use stored procedures I will not be explaining how to tweak the code to add a ‘sproc header.

Reusability

BI reports and dashboards can be immensely varied. They can use a multitude of different types of visualization. Not only that, but each type can vary in a myriad of subtle ways.

One of the objectives of this book is to provide you with a starter kit of reusable widgets that you can then tailor to suit your specific requirements. These widgets are not, however, designed to be generic. They are designed to illustrate the techniques that can be brought to bear to solve BI reporting challenges using SSRS.

However, you should be able to adapt most of the widgets described in this book, or taken from the source code download page on the Apress web site, to your own requirements. One hint that I can give you is that if you need to change, for instance, field names in a widget to suit your data, the best approach is to follow these steps.

1. Display the Solution Explorer window. Ctrl+Alt+L is one way of doing this.
2. Right-click the report file that you want to modify and select View code from the context menu.
3. Select Edit ➤ Find and Replace ➤ Quick Replace.
4. Enter the text that you want to find and the replacement text. This could be a file name, for instance.
5. Click Replace All.
6. Save the file and close it.
7. Reopen the report normally.
8. Preview and test the report.

If all has gone well, you will have taken a major step towards adapting the widget to your specific requirements.

I hope also that the code samples will be reusable and customizable to some extent. Obviously, they cannot cover more than a fraction of the needs of a reporting system, yet many of the overall approaches that are taken can hopefully provide you with some ideas and prototypes.

The CarSales_Reports Database

If you are following the examples in this book, you must install the `CarSales_Reports` database as described in Appendix A. This database is deliberately simple, so that you can concentrate on building dashboards and BI reports rather than struggle to understand an overly complex mound of data.

As a nod to the real world, the source data is in a series of relational tables, but the reports source their sales data from a view over the OLTP tables. This avoids you having to create the same joins again and again, as well as reminding you that reporting data is often prepared in ways like this.

The sample database also contains three user-defined functions that round up output. They are used for charts and gauges to set the upper limit of scales in many cases. The reason for this is that if you set a scale limit to a “rounded” figure, the scale increments (and gridlines and tick marks, if appropriate) will be much easier to read. If you need to adapt these functions to set other increments to round up by, you can use them as a starting point.

Book Audience

This is not a book for total SSRS novices. If you are using this book, then I am presuming that you can already create reports and are at ease with the concepts and practice of

- Data sources
- Datasets

- Tables
- Matrixes
- Basic charts
- Images

You will need a basic familiarity with these elements because BI with SSRS requires you to build on these foundations and to add

- Gauges
- Complex charts
- Sparklines
- Indicators
- Data bars

Not only will you be using these more "BI-oriented" elements, but you will be learning to combine most of the tools that SSRS has to offer. Moreover, you will be using all these tools together to create dashboards and output for mobile devices.

This book has no pretentions about being exhaustive. There are simply too many ways to use SSRS for business intelligence to cover in one small volume. Consequently, you will find only a selection of techniques and approaches that my experience has shown to be useful. There are plenty more ways to develop and use gauges and charts, for instance, than are explained in this book. Indeed, I hope that you will want to take the models that you find here and further enhance and develop them to create your own visualizations.

You are welcome to begin your BI development using this book as a starting point if you wish. However, if you feel that you need to revise some core knowledge, then *Pro SQL Server 2012 Reporting Services* by Brian McDonald, Shawn McGehee, and Rodney Landrum (Apress, 2012) is a perfect place to get the information that you might need.

As I am assuming that you are an intermediate-to-advanced SSRS user, I have had to make some presumptions about your level of knowledge. Nonetheless, I do not want to leave near beginners completely out in the cold, as there are simply too many cool things that can be done even if your knowledge of SSRS is only rudimentary. My approach is to start many chapters with an initial example that explains many of the core techniques and tools that you will be using in greater detail further on in the chapter. This first item isolates and explains each step in the process of developing a BI visualization. I hope that this approach will give novices a rapid introduction to the relevant techniques. From then on in the chapter, all other items are explained a little more succinctly. Specifically, the multiple properties that most elements require to be set are given as a table of properties that you have to set from one of the requisite dialogs.

SSRS allows you to set many properties in two or even three ways. I will try always to explain how to set properties using the dialogs that appear after you right-click an option in the context menu. This does not mean that you cannot use the properties window (or any other available method) if you prefer.

Some SSRS objects–charts spring to mind–are quite complex to configure. It follows that defining all the required properties for an arresting visual element can take quite a few clicks. This can also mean traversing a multitude of dialogs, panes, and windows full of settings.

It can also be difficult to select the appropriate part of a gauge or chart when you want to modify it. Admittedly, this becomes second nature after a little time, but until then you may need a little practice to get the job done.

Nearly all the examples in this book tend to use virtually the same basic template where the only parameters are the year and month. Inevitably, your dashboards and visualizations will be more complex than this, but I prefer to stick to a simple core report rather than cause confusion by altering the parameters for each report. This way you can concentrate on the elements that make up a dashboard rather than the mechanics of building and configuring different parameter sets. The report `__BaseReport.rdl` can be used as a template for nearly all reports if you wish to save time when it comes to adding the shared data source and shared datasets, and configuring the core parameters. Just remember to make a copy of the file first, so that you can reuse it easily for other reports.

How Best to Use This Book

This book is designed to be read in a linear fashion, from start to end. Indeed, some later chapters build on visualizations developed in previous chapters. To avoid this becoming a constraint there is nothing to stop you from merely taking the completed report or widget from the sample application and using it as a basis for development if you are jumping in at a later chapter. Equally (and if you are already a proficient SSRS developer) you can certainly leap in to any chapter or example to glean some useful information and tips.

Chapters 2-7 are designed to be read as a coherent unit. They explain how to build dashboards from component units. The basic components are described in Chapters 2-5, as follows:

- *Chapter 2* is an introduction to KPIs and some of the ways (from classic to more adventurous) of presenting your Key Performance Indicators.
- *Chapter 3* introduces gauges in business intelligence for SSRS. You will see some fairly standard gauges alongside some of the more interesting things that you can do with gauges.
- *Chapter 4* looks at some of the chart types that my experience has led me to believe are suited to BI. This is not a complete introduction to all the chart types that SSRS offers, but a trip into some of the less traditional aspects of chart design and creation.
- *Chapter 5* introduces maps and geographical data in SSRS. In a break with the format of this book, it provides a more complete introduction and does not presume prior experience with maps in SSRS.

Chapters 6 and 7 show how to assemble these components to build sample dashboards.

- *Chapter 6* shows how to use and manipulate images to standardize presentation in dashboards.
- *Chapter 7* shows how to assemble the composite parts of a dashboard into a coherent and attractive report.

Chapters 8 and 9 show how to customize and revamp the SSRS user interface and add interactivity.

- *Chapter 8* shows how to extend interactivity in SSRS using slicers and highlighting. You will also see how to create a kind of popup menu to enhance the user experience.
- *Chapter 9* shows how to enhance the user experience through adding tiles to the SSRS report interface. You will even see how to imitate Power BI carousels to scroll through lists of data as well as developing controlled paged datasets. Finally, you will see how to create “tabbed” reports.

The final three chapters cover mobile output and a series of techniques to deliver BI efficiently and quickly:

- *Chapter 10* gives you a series of tips and tricks that you will find useful when preparing reports that are destined to be viewed on tablets and smartphones.
- *Chapter 11* explains how to work more efficiently through standardizing certain methods and techniques.
- *Chapter 12* concludes the book with approaches that help you deliver data faster to the output devices and consequently enhance the user experience.

■ **Note** In this book I intend to be agnostic as to the publishing platform, so you can apply nearly every technique explained in the book to native Reporting Services just as you can to SSRS in SharePoint.

Conclusion

Now that you have seen what this book is about, it is time to get practical and to start using SSRS to create and deliver business intelligence to your users. The first stop on the road to the next level of SSRS is producing KPIs. This is the subject of the next chapter.

CHAPTER 2

KPIs and Scorecards

Key Performance Indicators, or KPIs, are a core element in most business intelligence. SSRS has been able to display KPIs for some time now, and they can be invaluable when it comes to delivering essential performance data in a succinct and meaningful way.

A set of KPIs is often referred to as a scorecard. Here I will not be delving into the management theory of what makes up a "balanced" scorecard, but will only refer to a scorecard as being a collection of KPIs.

Nonetheless, there are many ways to display KPIs. You may be used to a more traditional style of presentation where each KPI is on a separate row in a table, with visual elements to alert the user to deviations from an expected result. However, there are many other ways to deliver KPIs that need not involve tabular data. These can include using gauges to present the information. They may also include using text and colors to highlight changes in status and showing trends with the aid of sparklines. This chapter will begin with traditional KPIs, both to explain the concepts and to demonstrate the techniques. Then I will move on to some examples of less traditional KPIs. The underlying principle will always be the same, nevertheless. I will be comparing a metric to a target and highlighting any deviation from the objective, be it positive or negative. Similarly, I will display a trend over time.

In any case, I hope that you will discover that SSRS can deliver visually arresting KPIs in many ways, and I also hope that the examples in this chapter will be of use in your organization.

What Are Key Performance Indicators?

Before you can start using SSRS to display KPIs, you need to define what they are. Without wishing to get lost in layers of management-speak, let's say that Key Performance Indicators are a way of measuring progress towards a defined organizational objective. In practical terms, this usually means displaying the following elements:

- *A goal*: This is the target you are measuring an outcome against. This will be a figure, perhaps a budget or a sales target.
- *A value*: This is the actual data that will be compared to the target.

These two elements are the core of any KPI. However, they can be extended with one or both of the following elements:

- The *status* of the value compared to the goal. This indicates how well you are doing.
- The *trend* (over time, inevitably) of how well you are doing.

What you want to display in a KPI is up to you, or rather, up to the business you are working in. I consider that as long as you have a target figure and valid data to compare to the target, along with some indication of progress, then you have a KPI that you can use to convey meaningful information to your audience.

KPI Value

These metrics are often the easiest to understand. They are the figures that represent the business reality. In practice, however, delivering high-level data can require a lot of work preparing the source data. So you must be prepared to aggregate, calculate, filter, and rationalize your data in order to produce a meaningful set of KPI values.

KPI Goal

It probably sounds a little obvious, but you need to have goals (or targets) for a KPI to function. This means getting your business users to specify exactly what the targets are; they can be simple values (such as budgetary data) or they can be calculated values such as a defined percentage increase or decrease. In any case, you need to store and/or calculate the goal data for each metric. In practice, this can be tougher than it sounds. The difficulties are often more operational than technical, because

- Business users do not always have clearly defined quantifiable goals.
- Budgetary values, where they exist, are often in separate systems, or (worse in some cases) in spaghetti-like spreadsheets, which present difficulties such as:
    - Loading and updating the data can be painful, as it is not structured.
    - Mapping the budget data to the data from line of business systems is difficult.

Resolving these problems is outside the scope of this book. In the following examples, you will be using a set of budgetary data from the sample database that has been pre-crunched to map to the business data and that allows you to create meaningful KPIs.

KPI Status

Status is probably best thought of as a visual signal of how well you are doing. Frequently a status indicator is a symbol like a traffic light. Alternatively, it can be an image that indicates success or failure. It may even be a symbol that changes color to indicate good, average, or bad (or even under-achievement, on target, or over-achievement). As these examples imply, most status indicators will only display three values. You can extend this to five or even more values if you want. Just remember that you can easily lose the visual effect and clarity if you make it too complex for instant and intuitive understanding by the user.

What you need to retain is that the KPI status is essentially a flag indicator that has to be calculated. So you will need to know three things:

- The value (as for any KPI)
- The target (also as for any KPI)
- The thresholds at which a status flag switches from one value to another

KPI Status Thresholds

Let's take a simple example of a status flag in a KPI. Suppose that you have a target of 100. The business has said that any result under 80 is bad news, whereas anything over 125 will win a vacation. This means that you have the target and the thresholds. Once you have the value, you can then divide the result by the target. Under 80 percent is bad news, between 80 percent and 125 percent is expected, and anything over 125 percent is good news indeed. It is that easy. (In the real world, threshold calculations can be more complex than this, but I want to demonstrate the principle.) Of course, the thresholds can be absolute values rather than percentages; it will all depend on the business requirements. In the case that I just mentioned, the example only has three status flags; you can have as many status flags as you wish in your visualizations. However, I will never exceed three status indicators in the examples in this book as I firmly believe that using more than three status flags obscures the information rather than clarifies it.

Threshold Flags

If you are carrying out any status calculations in a database (relational or dimensional), I generally advise that you convert the status to a simple indicator, such as 1 for bad, 2 for OK, and 3 for good. This is because

- It is often easier to have the business logic in one place.
- You can standardize the threshold flags across your entire reporting suite, or better still across the enterprise.
- It is simpler to maintain reports if the business logic is in a structured (and hopefully annotated) environment.
- It is generally easier to concentrate on the way that an indicator value is *displayed* in SSRS (the choice of symbols or images and the color selection, for instance) if you are not having to mix this in with the calculation and the business logic in the report at the same time.

This threshold value will then be used by SSRS to display a more visual and meaningful indicator.

You can, of course, perform calculations and apply business logic in SSRS, and there are many valid reasons for this approach, too. However, as I mentioned in Chapter 1, in this book I will always place business logic at a lower layer of the solution.

KPI Trends

KPIs can also give a visual indication of how the current results compare to past outcomes. Here again, exactly how you define the time-based comparison will depend on the business. For example, you may need to

- Compare the current month's sales with the previous month's sales
- Compare the current month's sales with the same month of the previous year
- Visualize the last few months or year's data as a sparkline

KPI Trend Thresholds and Flags

The principles that I outlined for status thresholds and flags apply in a very similar way to trend thresholds and flags. You need the current value and the value you are comparing it with. Then you need to compare the two and indicate what the trend is on a simple scale. Interestingly, trend indicators in KPIs tend to show at least five values, and can show more; though, once again, clarity is essential, and a trend indicator that has nine different levels can be hard to read. So I will stick to a maximum of five in this book. Later I will show you how to use sparklines to display trends in a different type of detail.

As for the indicators, I feel that trend indicators are best calculated at the database layer and a simple flag (say 1 through 5) is sent to the report to be displayed in a more visual and intuitive way. This is what you will do in the next few examples.

A Simple KPI

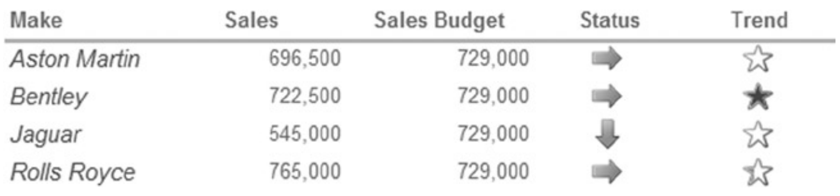
I suspect that you have had enough theory and want to get down to delivering KPIs with SSRS. So let's move on to a simple KPI that will let you see how the principles can be applied in practice. This KPI takes sales data for the selected year (up to the selected month) and does the following:

- Displays the sales data.
- Displays the forecast data.
- Compares the sales data with the sales forecast and indicates the status on a scale of 1 to 3. The status is displayed as a traffic light.
- Compares the current sales up to the selected month with the sales for the same period in the previous year and returns a trend flag on a scale of 1 to 5. The trend is displayed as five pointed star that ranges from empty (a threshold indicator of 1) to full (an indicator of 5).

You will use a single stored procedure to gather the data and apply business logic. Status and trend indicators will be delivered in the same dataset as the business data and target metrics. At the risk of belaboring the point, I prefer to centralize the business logic and source data in a single place (a stored procedure in a reporting database) wherever possible. Of course, the reporting suite that you develop may take another approach, and there is nothing to prevent you carrying out the calculations and applying business logic in SSRS.

As this is the first example in this chapter (indeed in the whole book) I will be explaining it in more detail than the subsequent KPIs in this chapter. So this is an excellent place to start if you have a basic grasp of SQL Server Reporting Services and need to consolidate your knowledge before moving on to some of the more advanced techniques that you will find later in this chapter.

Figure 2-1 shows the KPI that you are trying to create. It displays sales and budget figures alongside the status and trend indicators for all the makes of car sold in 2014 up to the end of June.



Make	Sales	Sales Budget	Status	Trend
Aston Martin	696,500	729,000	→	☆
Bentley	722,500	729,000	→	★
Jaguar	545,000	729,000	↓	☆
Rolls Royce	765,000	729,000	→	☆

Figure 2-1. *A KPI showing sales by make for the first half of 2014*

The Source Data

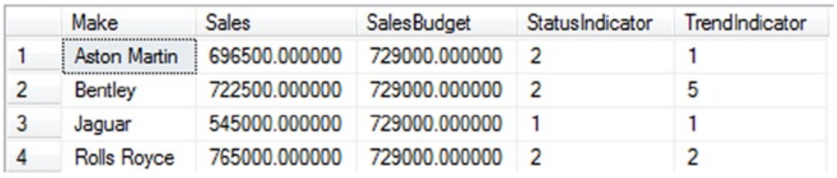
First, you need some data to work with. The following code snippet (available as `Code.pr_CarSalesYearToDateKPISimple` in the `CarSales_Reports` database) gives you the data that you need for June 2014:

```
DECLARE @ReportingYear INT = 2014
DECLARE @ReportingMonth TINYINT = 6

IF OBJECT_ID('Tempdb..#Tmp_KPIOutput') IS NOT NULL DROP TABLE Tempdb..#Tmp_KPIOutput
```

```
CREATE TABLE #Tmp_KPIOutput
(
ReportingYear INT
,Make NVARCHAR(80) COLLATE DATABASE_DEFAULT
,Sales NUMERIC(18,6)
,SalesBudget NUMERIC(18,6)
,PreviousYear NUMERIC(18,6)
,StatusIndicator SMALLINT
,TrendIndicator SMALLINT
)

INSERT INTO #Tmp_KPIOutput
(
ReportingYear
,Make
,Sales
)

SELECT     ReportingYear
          ,Make
          ,SUM(SalePrice)
FROM      Reports.CarSalesData
WHERE     ReportingYear = @ReportingYear
          AND ReportingMonth <= @ReportingMonth
GROUP BY  ReportingYear
          ,Make
-- Previous Year Sales
;
WITH SalesPrev_CTE
AS
(
SELECT     ReportingYear
          ,Make
          ,SUM(SalePrice) AS Sales
FROM      Reports.CarSalesData
WHERE     ReportingYear = @ReportingYear - 1
          AND ReportingMonth <= @ReportingMonth
GROUP BY  ReportingYear
          ,Make
)

UPDATE    Tmp
SET       Tmp.PreviousYear = CTE.Sales
FROM      #Tmp_KPIOutput Tmp
          INNER JOIN SalesPrev_CTE CTE
          ON Tmp.Make = CTE.Make

;
WITH Budget_CTE
AS
```

```
(
SELECT     SUM(BudgetValue) AS BudgetValue
          ,BudgetDetail
          ,Year
FROM      Reference.Budget
WHERE     BudgetElement = 'Sales'
          AND Year = @ReportingYear
          AND Month <= @ReportingMonth
GROUP BY  BudgetDetail
          ,Year
)

UPDATE    Tmp
SET       Tmp.SalesBudget = CTE.BudgetValue
FROM      #Tmp_KPIOutput Tmp
          INNER JOIN Budget_CTE CTE
          ON Tmp.Make = CTE.BudgetDetail
          AND Tmp.ReportingYear = CTE.Year

-- Internal Calculations
-- Year on Year Delta
-- TrendIndicator

UPDATE  #Tmp_KPIOutput
SET     TrendIndicator =
           CASE
               WHEN ((Sales - PreviousYear) / Sales) + 1 <= 0.7 THEN 1
               WHEN ((Sales - PreviousYear) / Sales) + 1  > 1.3 THEN 5
               WHEN ((Sales - PreviousYear) / Sales) + 1  > 0.7
                             AND ((Sales - PreviousYear) / PreviousYear) <= 0.9 THEN 2
               WHEN ((Sales - PreviousYear) / Sales) + 1  > 1.1
                             AND ((Sales - PreviousYear) / PreviousYear) <= 1.3 THEN 4
               WHEN ((Sales - PreviousYear) / Sales) + 1  > 0.9
                             AND ((Sales - PreviousYear) / PreviousYear) <= 1.1 THEN 3
               ELSE 0
           END

-- StatusIndicator

UPDATE    #Tmp_KPIOutput
SET       StatusIndicator =
              CASE
                WHEN ((Sales - SalesBudget) / Sales) + 1 <= 0.8 THEN 1
                WHEN ((Sales - SalesBudget) / Sales) + 1  > 1.2 THEN 3
                WHEN ((Sales - SalesBudget) / Sales) + 1  > 0.8
                     AND ((Sales - SalesBudget) / SalesBudget) <= 1.2 THEN 2
                ELSE 0
              END
-- Output

SELECT   Make, Sales, SalesBudget, StatusIndicator, TrendIndicator
FROM     #Tmp_KPIOutput
```

Running this snippet gives the output in Figure 2-2.



	Make	Sales	SalesBudget	StatusIndicator	TrendIndicator
1	Aston Martin	696500.000000	729000.000000	2	1
2	Bentley	722500.000000	729000.000000	2	5
3	Jaguar	545000.000000	729000.000000	1	1
4	Rolls Royce	765000.000000	729000.000000	2	2

Figure 2-2. *The data for a simple KPI*

How the Code Works

This T-SQL snippet is quite simple, really; it does the following:

- Prepares a temporary table to hold the output that will be sent to SSRS. This table contains only the essential data used by the KPI. This means the make, the (sales) value, the (budgetary) target, and the status indicator are on a scale of 1-3 and the trend indicator is on a scale of 1-5.
- Groups the sales per make of car for the selected year (2014) up to and including the selected month (June) and adds them to the output table.
- Updates the table with the corresponding figures for the previous year.
- Updates the table with the budget figures for the current year.
- Calculates the status and trend indicators using a predefined and hard-coded logic. This sets the status from 1 through 3 and the trend from 1 through 5 using a predefined business rule that we will imagine has been chosen by the CFO.

■ **Note** I have got into the habit of defining status indicators as 1-3 and trend indicators as 1-5. This is far from the only way of doing this, and many other approaches exist. You could (for status) use -1, 0, and 1, for instance. My only advice is to set a standard and stick to it across all of your KPIs.

Building the KPI

Now that you have your source data, you can get into the fun bit: building the KPI itself. As this is not only the first example in this chapter, but the first in the entire book, I will try to make the explanation reasonably comprehensive so that SSRS novices are not left floundering. In any case (and whatever your level of SSRS knowledge), remember that you can refer back to Figure 2-1 at any time when you are building this KPI if you need to check that you are doing things the right way.

1. Create a new SSRS report named `KPI_Basic.rdl`.
2. Add the shared data source `CarSales_Reports`. Name it `CarSales_Reports` (and not `Datasource1`, which is the default).

3. Add the following four shared datasets (ensuring that you also use the same name for the dataset in the report):

    a. `CurrentYear`

    b. `CurrentMonth`

    c. `ReportingYear`

    d. `ReportingMonth`

4. Add the parameters `ReportingYear` and `ReportingMonth`, and set their properties as defined in Chapter 1.

5. Add a new stand-alone (i.e., not shared) dataset named `CarSalesYearToDateKPISimple`. Have it use the `CarSales_Reports` data source and the stored procedure `Code.pr_CarSalesYearToDateKPISimple` that you saw earlier.

6. Drag a table from the SSRS toolbox onto the report body.

7. Select the table and display the properties window unless it is already visible (pressing F4 is one way to do this; another is to select View ➤ Properties Window).

8. Find the `DataSetName` property and click the pop-up list to the right. Select the dataset `CarSalesYearToDateKPISimple`.

9. Select all the textboxes (or cells if you prefer) in the table. Expand BorderStyle in the Properties window and set the Default property to None. This will ensure that the table will only display the borders that you have specifically added.

10. Add two new columns to the table. One way to do this is to right-click a text box anywhere inside a table. You then select Insert Column ➤ Right (or left) from the context menu. You can add these columns anywhere. Your table will now have five columns.

11. Add the following three fields to the `Details` row of the leftmost three columns, in this order: `Make`, `Sales`, `SalesBudget`. You can either drag the fields into the detail (second) row from the dataset `CarSalesYearToDateKPISimple`, or click the tiny table symbol that appears when you hover the mouse pointer over a textbox and select the field from the pop-up list.

12. Drag an indicator (from the SSRS toolbox) to the fourth column of the details row. The Select Indicator dialog will appear. Select the three-gray-arrows indicator shown in Figure 2-3.

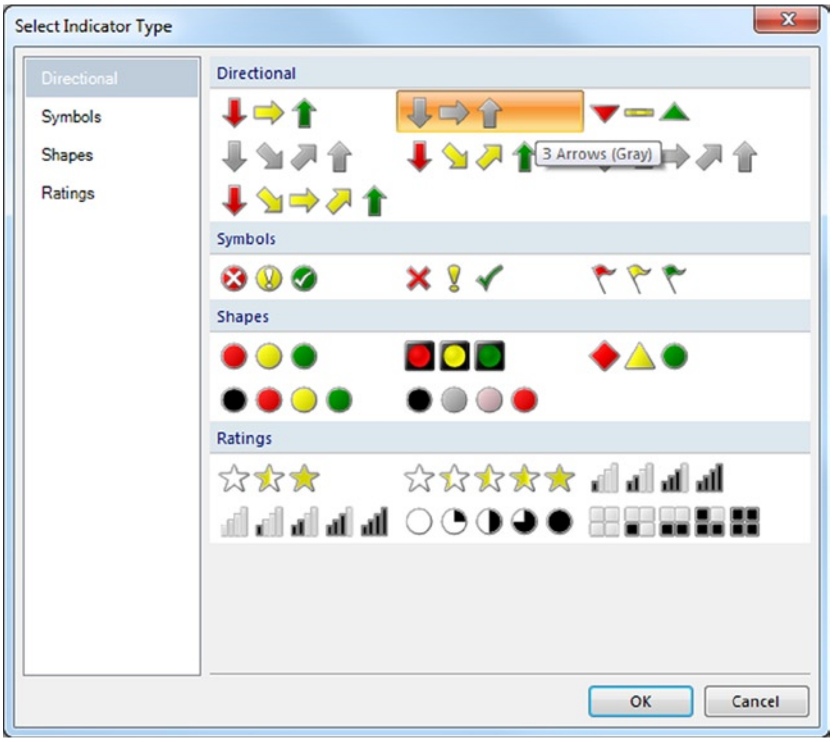


Figure 2-3. *The Select Indicator Type dialog*

13. Click OK.
14. Right-click the indicator that you can now see in the fourth column and select Indicator Properties from the context menu.
15. Click the Values and States option on the left to display the Change Indicator Value pane of the dialog.
16. Set the Value to `[Sum(StatusIndicator)]`.
17. Select Numeric as the States measurement unit.
18. Leaving the icon images as they are, set the following start and end attributes for the three icons (in this order from top to bottom):
    a. Down-facing arrow: Color: Gainsboro, Start and End: 1.
    b. Right-facing arrow: Color: Silver, Start and End: 2.
    c. Up-facing arrow: Color: Dim Gray, Start and End: 3.

The dialog should look like Figure 2-4.