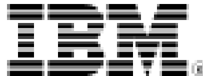


developerWorks.

Make dashboards with XQuery

Present business data with a Web-based dashboard

Skill Level: Intermediate

James R. Fuller (jim.fuller@webcomposite.com)
Technical Director
FlameDigital Limited & Webcomposite s.r.o.

31 Mar 2009

Many digital dashboards that cropped up in the 1980s were horrible (if not unsubtle) analogs to a car's dashboard. Very few presented business data in a compelling manner. Today, Web-based dashboards try to achieve the same thing. Discover what makes a good dashboard, and learn to identify and leverage key performance indicators (KPIs) for more effective digital dashboards. Finally, build a Web dashboard using the eXist XML database and XQuery.

Some years ago, I worked for a client that generated a broad range of sports-related data feeds. We needed to come up with a way to demonstrate the breadth and depth of the client's data services product and decided to use a scoreboard analogy that could be placed on the client's Web site. The idea was that showing key sports statistics like "football goals scored," "horse races covered," or "number of cricket runs"—all kept current in real time—would have the desired effect and convey the quality of the client's data services to potential customers.

How this specific scorecard was developed is not the aim of this article, however: I retell it only because a peculiar thing occurred after the widget was deployed.

Frequently used acronyms

- API: application programming interface
- CSS: Cascading Stylesheet
- HTML: Hypertext Markup Language
- HTTP: Hypertext Transfer Protocol

© Copyright IBM Corporation 2009. All rights reserved.

- UI: User Interface
- URL: Uniform Resource Locator
- XML: Extensible Markup Language
- XSLT: Extensible Stylesheet Language Transform

After the scorecard was in use, the client's senior management started to ask questions about the figures generated. Initially, these questions revolved around the correctness of our summations, then became more introspective, such as, "We should have more football goals showing" or "Why is the update speed on busy sports days slower than expected?" The various figures that we generated turned out to be effective benchmarks of the quality and overall health of the client's sports-related data feeds.

As the scorecard application updated every second or so, it relayed a heartbeat of the company to senior management, who were quick to pick up on any irregularities. Of course, what we had done was nothing new—just a reformulation of the concept of developing a dashboard of KPIs—but it took a marketing initiative to create it.

Dashboards overview

In enterprise development circles, during one period of time, dashboards were the subject of considerable hype, so let me try to provide an informal definition for their use in this article:

> A *dashboard* is a visual display of important information needed to achieve one or more objectives, consolidated and arranged preferably on a single screen, allowing the information to be comprehended at a glance.

The objectives and types of data can be broken down into three categories:

- **Strategic.** Strategic objectives tend to be those things that interest executives and key decision-makers of a company, who are forever trying to gain a better understanding of what is happening on the shop floor so that they have a more intuitive sense of overall company health.
- **Analytical.** Objectives that revolve around inferring and identifying trends are more analytical in nature.
- **Operational.** Operational data crops up when directly monitoring the state of a process; examples of this data type are "Is the Web site running?" or "Is our contact e-mail form working?"

© Copyright IBM Corporation 2009. All rights reserved.

Apart from the above categories, dashboards that I have worked on or seen tend to have a few commonalities, such as how often they update themselves or whether they present quantitative data, qualitative data, or both—some become a jumping-off point to navigate and drill down into the detail.

Examples of KPIs

Here are a few examples of KPIs that might be appropriate for your business:

- **Web site.** Number of hits, number of visitors, duration
- **Technical support.** Support requests, customer happiness, call duration
- **Human Resources (HR).** Employee turnover, count of empty positions, number of pending performance reviews
- **Information Technology (IT).** System downtime, network performance, bug resolution
- **Sales.** Anticipated sales, number of orders, billings
- **Marketing.** Customer demographics, campaign success, market share
- **Finance.** Profit, expenses, revenues

What makes a good KPI?

Dashboards tend to present things that are measurable, which are typically denoted in business circles as *key performance indicators.* Any metric that can be measured and that says something about the state or health of the company's business processes is a KPI. A good KPI, therefore, should be good at measuring the success or failure of any strategic, analytical, or operational objectives within your organization.

The sidebar, "Examples of KPIs," provides a few examples of KPIs that you might use in your business. These KPIs are generic to a whole range of companies: I espouse always trying to identify (as in the scorecard example earlier) a KPI that succinctly characterizes the health of processes specific to your businesses . The more related to whatever your company does, the more appropriate the metric.

Try to realize that your KPIs may experience volatility in their definition as you learn through time more appropriate ways to define them as well as what questions you want to answer with them. Ultimately, it's the questions you want to ask and answer of your business for which using KPIs makes the most sense.

Past that, don't be afraid of KPIs: They can be as advanced or as simple as you choose to make them, and plenty of reference materials are available on the Web to help you craft your own. I guess the most important thing to allow for in their creation

© Copyright IBM Corporation 2009. All rights reserved.

is to ensure that you have the data available.

What makes a good dashboard?

A good Web dashboard strives to present KPIs in the clearest manner possible, which maximises data and reduces as much as possible any graphical noise extraneous to the presentation of information. Organising display elements along the lines of strategic, operational, and analytical objectives is a good starting point for how to group items. When grouping items that you want to compare, check that the comparison is relevant, valid, and provides some useful feedback to the business rather than explaining what happened in the past. Saying how a particular thing occurred in the past can be of service, but remember that a dashboard's job is to explain the present and current state of things. If you require a historical record to help explain the present, then build your dashboard so that it or its data can be saved at regular intervals.

A dashboard that initially shows data in its most condensed form, then progressively discloses more information can turn into a tool for navigating information. Although, if you do provide elements like data drill-down or a navigation UI, you will also need to be aware of the need to match the dashboard to a specific user role. It's best practice to try to match a single dashboard to a specific role.

Finally, a good Web dashboard eschews over-elaborate graphs. The classic example is the use of things like three-dimensional (3D) pie charts, where the added depth actually makes it more difficult to visually determine the contribution of the individual slices to a whole. Going even further, plenty of quality discussions (see Resources for links to more information) renounce the use of the pie chart altogether.

A bad dashboard exhibits the negation of all the advice I have just given. Here, I list some stereotypical traits that I have found to be invariably present in poorly designed dashboard applications:

- **Overuse of color.** Organizing data based on color is good, but most people seem to go overboard.
- **Requires the user to scroll or use tabs.** Going over one page of data makes the user's brain work harder.
- **Too much detail.** Not summarising can make it nearly impossible for the person to synthesize data that results in concrete actions.
- **Excessive precision.** Too many decimal places in a number or too many data labels on a graph are examples.
- **Critical exceptions go unnoticed.** Not highlighting exceptions, such as overshooting some target or not attaining some goal, makes the

© Copyright IBM Corporation 2009. All rights reserved.

dashboard less effective as a call to action in the current time frame.

- **Employing metrics that are incomplete or deficient.** Ensuring that the KPI is appropriate and correctly formulated might seem obvious but is as common as the previous bad characteristics. Either manually generate the metric or use a spreadsheet to assist in testing some example calculations before you enshrine it within your dashboard application.

The fundamental idea of data ink encapsulates another way to describe what makes a good or bad dashboard. A basic principle of information and data design, *data ink* is literally the total ink used for anything that represents data (for example, numbers or data lines on a graph) on a printed page. The same is true for a computer screen, where the data pixels are analogous to data ink and everything else is not. Maximising data pixels in information design means that you remove as many extraneous graphical elements as possible, use muted color schemes, and avoid fussy fonts.

Installing the sample dashboard

Before I show you how to build the example dashboard, you might want to get the source files from Download and install the dashboard. To do this, extract the .zip file and follow the README file instructions included.

When you have everything installed, you should be able to view the example dashboard by accessing dashboard.xq from the eXist database using a Web browser and the URL:

```
http://localhost:8080/exist/rest/db/dashboards_with_xquery/xquery/dashboard.xq
```

If you installed the XQuery files somewhere else in eXist, adjust the URL accordingly.

Also be aware that the example dashboard uses a lot of test data and credentials to online Web services, all of which exist on the Internet. These resources should remain available, but if you do encounter problems, I suggest that you replace example data with your own credentials and test data.

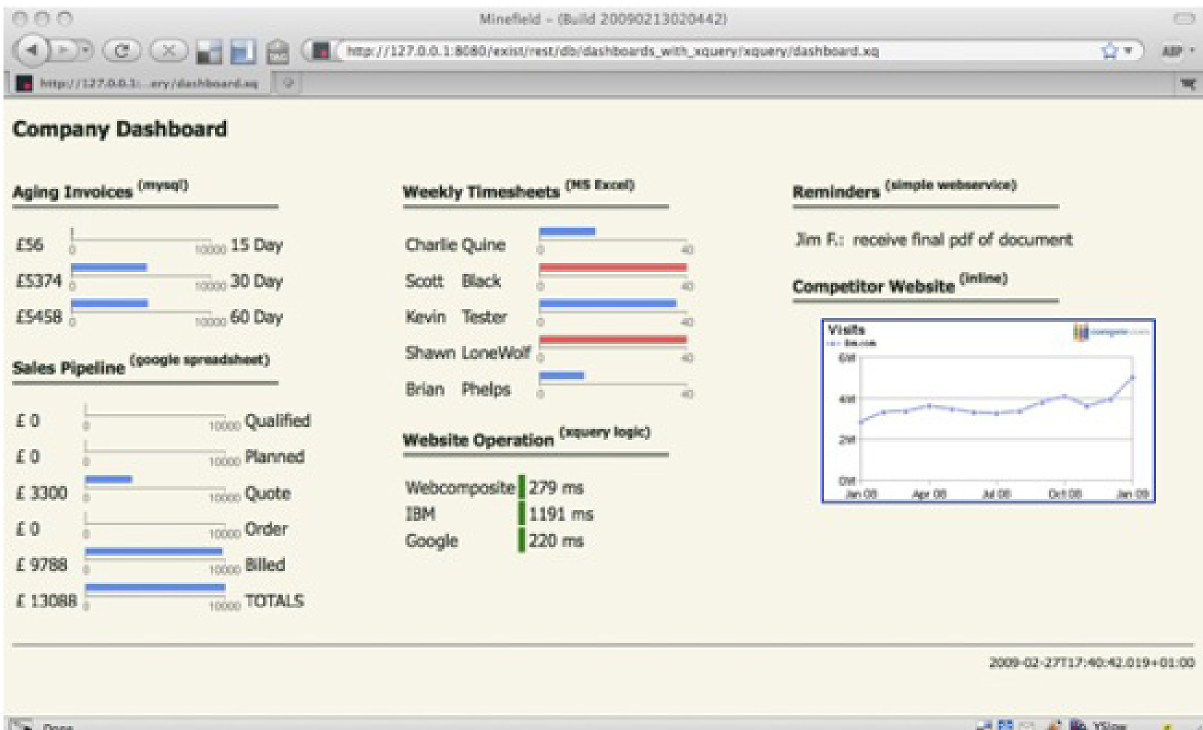
Building a company dashboard

Now that you know what dashboards are and that their main goal is to present KPIs in an engaging manner that helps you ask and answer important business questions, you can move on to actually creating the sample dashboard. Using XML technologies like the eXist XML database and XQuery, you can aggregate and consume XML data, then create a HTML representation of a dashboard.

© Copyright IBM Corporation 2009. All rights reserved.

Figure 1 shows the sample dashboard in its final form.

Figure 1. The sample dashboard



This Web page has few extraneous design elements competing for the eye's attention, which hopefully means that I adhered to the prior sections' advice of increasing data pixels over all else. The CSS file defines an off-color background and font that I think make it easy to read.

I chose KPIs that were readily available to show how well XQuery integrates data:

- **Aging Invoices.** When a company charges for their services, payment typically occurs on a standard set of terms (for example, the number of days for the customer to pay).
- **Sales Pipeline.** A company categorizes sales opportunities between a series of stages, which defines how likely it is to actually win a job and bill for it.
- **Weekly Timesheets.** Employees' weekly time sheets are displayed, with emphasis on identifying when individuals work past the standard 40-hour work week.
- **Website Operation.** This monitor indicates whether a Web site is in operation.
- **Reminders.** These are a dated series of tasks for the company.

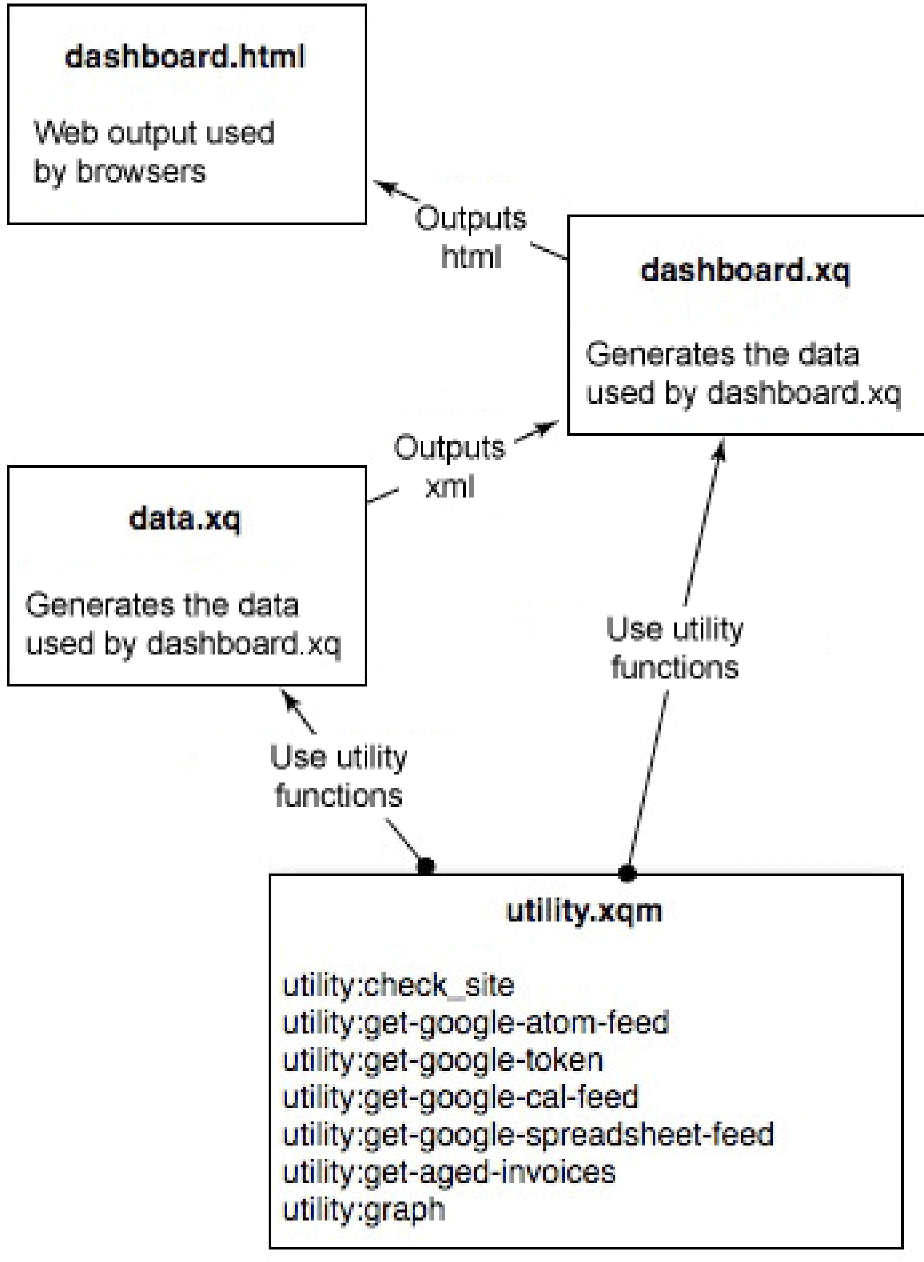
© Copyright IBM Corporation 2009. All rights reserved.

- **Competitor Websites.** This chart indicates the performance of Web sites in comparison with competitors.

The dashboard is more mash-up than full-blown application, which takes advantage of functionality specifically found within the eXist XML database implementation of XQuery. The dashboard you will create is implemented using just three XQuery files, one of which is an XQuery module used by the others, as in Figure 2.

Figure 2. Dashboard XQuery components

© Copyright IBM Corporation 2009. All rights reserved.



The utility.xqm XQuery module contains functions that both data.xq and dashboard.xq use. The data.xq XQuery file's job is to generate an XML document containing all KPI-related data. The dashboard.xq file's role is to present this data, creating an HTML file that you can view in a browser.

Data source first

When I develop software, I try to flesh out the data layer of an application before

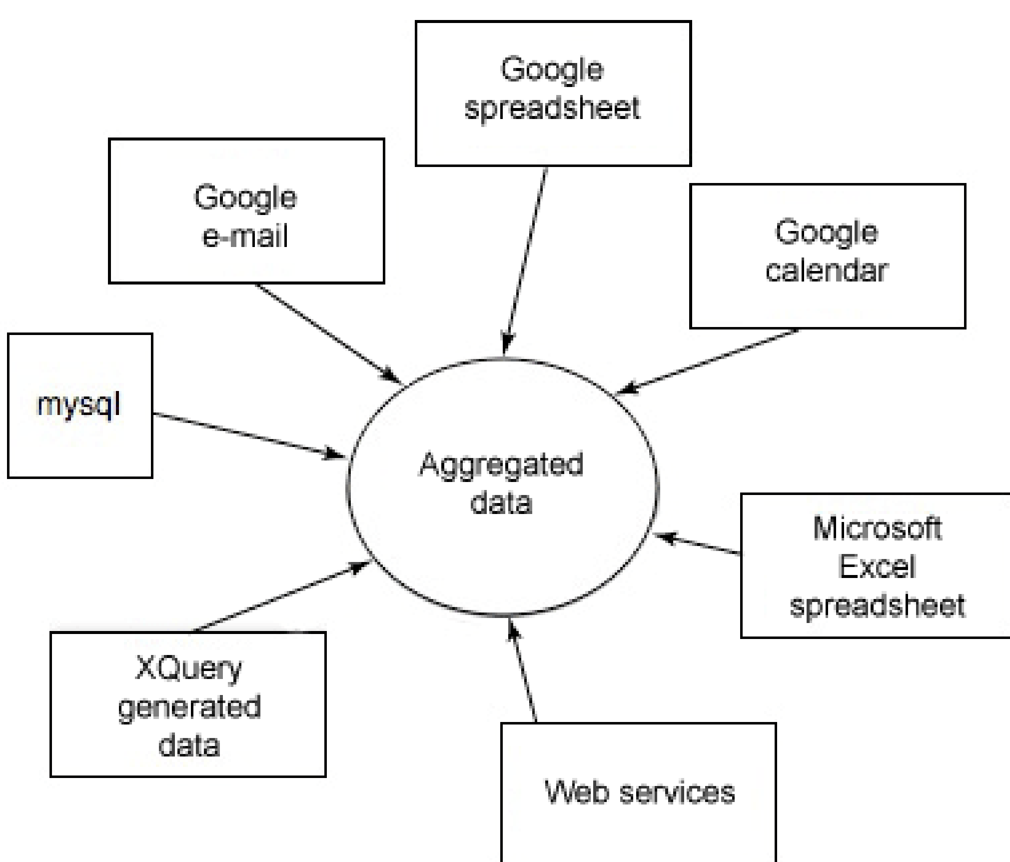
© Copyright IBM Corporation 2009. All rights reserved.

doing anything else: it is unusual for a useful application *not* to depend on some sort of data. As you create a dashboard that represents data, it behooves you to identify data sources first.

The goal of a dashboard is to present KPIs, so it follows that data sources need to be integrated that are either KPIs themselves or contribute to their calculation. Regardless of whether a KPI is qualitative or quantitative in nature, all KPIs need to say something useful about the running of the business. For this sample dashboard, I use publicly available data to illustrate how XQuery handles integration scenarios. Additionally, I chose a range of popular Web services relevant to bringing in all that XML data generated internally, externally, and at the edge of the business.

The diagram in Figure 3 shows a selection of data sources, available either from popular Web services such as Google™ or from commonplace technologies like relational databases.

Figure 3. Diagram of possible data sources



Each data source requires some sort of integration with a Web service, XML data, or XQuery-generated data:

© Copyright IBM Corporation 2009. All rights reserved.

- **Website Operation.** Shows that XQuery is an expansive language in its own right
- **Aging Invoices.** Illustrates how you can access MySQL from XQuery
- **Sales Pipeline.** Shows how you can access Google Docs spreadsheets from XQuery
- **Weekly Timesheets.** Show how to access a local Microsoft® Office Excel® spreadsheet
- **Reminders.** Integrates with the popular Backpack Web service
- **Competitor Web sites.** A chart indicating performance of Web sites in comparison with competitors

The result of processing data.xq should be a single XML document that contains all the required data. The easiest way to integrate data within XQuery is to use the `doc()` function, which retrieves XML from the site at the Web URL. If you supply this function with a URL that points to a resource that returns data that is not well-formed (in the XML sense), then you will get an error.

The code sections in Listing 1 show this processing in action in data.xq.

Listing 1. Processing in data.xq

```
<sales_pipeline desc="example accessing published Google spreadsheet">
{doc("http://spreadsheets.google.com/feeds/list/
pZDPqHJcLzxKntsQv2tuIMQ/1/public/basic")}
</sales_pipeline>

...

<backpack desc="example accessing simple web service">
{doc("http://dashboardwithxquery.backpackit.com/
69babcbce8aa212fc83464088b45f7a97a9f3dce/reminders.xml")}
</backpack>
...

<timesheet desc="example accessing local MS Excel file">
doc("file:///Users/jimfuller/Source/Writing/1_articles/1_ibm/
3_dashboards_with_xquery/working/src/data/MSOFFICE-timesheet.xls")
</timesheet>
```

The `sales_pipeline` element accesses a Google document that I published and made accessible by a URL. Google documents are exposed in this manner as an Atom XML feed (see Resources for a link). The Backpack URL returns an XML feed that represents reminders that I set up at the URL http://www.backpackit.com. The `timesheet` element does something different and accesses a locally hosted Excel file. As of a few years ago, it is possible to use Excel spreadsheets that are generated in a Microsoft-specific XML format.

© Copyright IBM Corporation 2009. All rights reserved.

The next step in data integration is to use XQuery to create a set of helper functions, all of which are located in the utility.xqm XQuery module. Here are short descriptions of the functions, with particular aspects of the code highlighted:

- **utility:check_site().** This bit of XQuery code checks whether a Web URL is accessible and how long it took for the Web server to respond. Listing 2 shows this function.
  Listing 2. The utility:check_site() function

```
declare function utility:check_site($uri) {
let $start := util:system-time()
let $response := httpclient:get(xs:anyURI($uri),false(),())
let $end := util:system-time()
let $response-time := (($end - $start) div xs:dayTimeDuration('PT1S')) * 1000
let $status-code := string($response/@statusCode)
return <test xmlns=""
        ts="{current-dateTime()}"
        status="{$status-code}"
        response="{$response-time}" />
};
```

  The function returns a test element that dashboard.xq uses to display the status of a Web site.

- **utility:get-aged-invoices().** Because an unimaginable amount of data is encoded and stored within relational databases, it seems appropriate to show an example of integrating an eXist XML database to interact with a MySQL® community server. XQuery has no native built-in relational database management system (RDBMS) functionality, but many of the popular XQuery processors have extension functions. The eXist XML database has Structured Query Language (SQL) extension functions available in the form of an optional module, which you must enable by uncommenting the appropriate section in the eXist conf.xml file (read the README file included in sample code.) Listing 3 shows the `utility:get-aged-invoices()` function.
  Listing 3. The utility:get-aged-invoices() function

```
declare function utility:get-aged-invoices(){
let $connection := sql:get-connection("com.mysql.jdbc.Driver",
"jdbc:mysql://localhost/test", "root",
"")
let $data := sql:execute($connection, "select * from invoices;", fn:true())
return <invoices>
<total>{sum($data/sql:row/sql:invoice_amount)}</total>
<age amt="15">
    {sum($data/sql:row/sql:invoice_amount[../sql:invoice_terms='15'])}</age>
<age
amt="30">
    {sum($data/sql:row/sql:invoice_amount[../sql:invoice_terms='30'])}</age>
<age
amt="60">
    {sum($data/sql:row/sql:invoice_amount[../sql:invoice_terms='60'])}</age>
```


© Copyright IBM Corporation 2009. All rights reserved.

```
</invoices>
};
```

The function first establishes a Java™ Database Connectivity (JDBC)-style connection using the eXist SQL extension module's `sql:get-connection()` function. Then, you define an SQL statement to select the data from the database, which is executed using `sql:execute`. The results of this function are saved in the `$data` variable, which you then use to pick out relevant data using XPath.

Here is a series of functions that integrate with the various Google-based Web services (see Resources for a link to more information). Many of Google's Web services use similar techniques to expose and connect to them, so I have included a few additional examples. (They are not themselves used in the sample dashboard, but you might find them instructive or useful.)

- **utility:get-google-token($Email,$Passwd,$accountType,$source,$service).** Google Web services have a few different authentication mechanisms, depending on how deeply you want to integrate. I chose its `ClientToken` version (in Listing 4) to create a one-time authentication token appropriate for purposes of this example.
  Listing 4. The Google utility:get-google-token($Email,$Passwd,$accountType,$source,$service) service

```
declare function
        utility:get-google-token($Email,$Passwd,$accountType,$source,$service){
let $params := concat("Email=",$Email,
                      "&amp;Passwd=",$Passwd,
                          "&amp;source=",$source,
                          "&amp;accountType=",$accountType,
                          "&amp;service=",$service)
let $uri := concat("https://www.google.com/accounts/ClientLogin?",$params)
let $response := httpclient:get(xs:anyURI($uri),false(),())
let $token := substring-after(xmldb:decode($response),"Auth=")
return
        string($token)
};
```

  Please be advised that this function returns a string and not XML. In XQuery, you can constrain a function's inputs and outputs and give them a data type.

- **utility:get-google-spreadsheet-feed($Email,$Passwd).** This function (in Listing 5) returns the list of Google documents, as an Atom feed, from within your own (or someone else's) Google documents. Because I used a publicly available Google spreadsheet, I thought some developers would like to know how to connect and use Google documents through

© Copyright IBM Corporation 2009. All rights reserved.

XQuery.

Listing 5. The utility:get-google-spreadsheet-feed($Email,$Passwd) function

```
declare function utility:get-google-spreadsheet-feed($Email,$Passwd){
let $accountType :=  "HOSTED_OR_GOOGLE"
let $source := "Dashboards-XQUERY-Example"
let $service := "wise"
let $token := utility:get-google-token($Email,$Passwd,$accountType,$source,$service)
let $headers := <headers> <header name="Authorization"
value="GoogleLogin auth={$token}"/>
</headers>
let $uri :=
        xs:anyURI('http://spreadsheets.google.com/feeds/spreadsheets/private/full')
return
        httpclient:get($uri, false(), $headers)
};
```

This function crafts the proper HTTP request to retrieve a list of documents in Google documents, which in turn can be used to open the said documents.

- **utility:get-google-atom-feed().** Another interesting data source isa Google e-mail address. The data is exposed in Gmail as an Atom data feed, as in Listing 6. (This example dashboard does not actually use this source, but it's useful to note as an example of a data source to manipulate e-mail.)

Listing 6. The utility:get-google-atom-feed() function

```
declare function utility:get-google-atom-feed($user,$pass,$label){
let $token := util:string-to-binary(concat($user,':',$pass))
let $headers := <headers>
<header name="Authorization" value="Basic {$token}"/>
</headers>
let $uri := xs:anyURI(concat('https://mail.google.com/mail/feed/atom/',$label))
return
httpclient:get($uri, false(), $headers)
};
```

- **utility:get-google-cal-feed($Email,$Passwd).** Another popular Google service is Google Calendar (not used in data.xq) to show calendar event data. Listing 7 shows the corresponding function.

Listing 7. The utility:get-google-cal-feed($Email,$Passwd) function

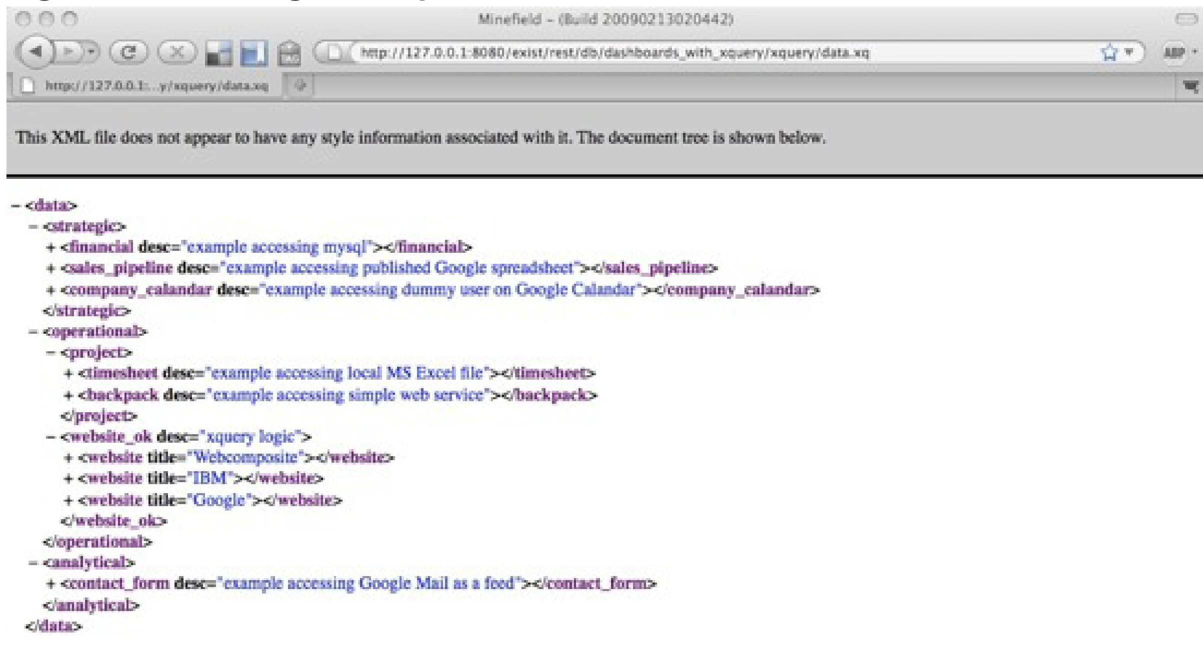
```
declare function utility:get-google-cal-feed($Email,$Passwd){
let $accountType := "HOSTED_OR_GOOGLE"
let $source := "Dashboards-XQUERY-Example"
let $service := "cl"
let $token := utility:get-google-token($Email,$Passwd,$accountType,$source,$service)
let $headers := <headers>
<header
        name="GData-Version"
```


© Copyright IBM Corporation 2009. All rights reserved.

```
        value="2"/>
<header name="Authorization"
        value="GoogleLogin
        auth={$token}"/>
</headers>
let $uri := xs:anyURI(
        concat('http://www.google.com/calendar/feeds/',
                        string($Email),
                        '/private/full')
        )
return
httpclient:get($uri, false(), $headers)
};
```

These functions are used by data.xq to create a massive XML document that is the source document for the digital dashboard. If you have code installed, I suggest that you access data.xq through your browser: You should see something similar to Figure 4.

Figure 4. Retrieving data.xq



This markup's structure is arbitrary, and you can choose all manner of alternatives.

Dashboard presentation

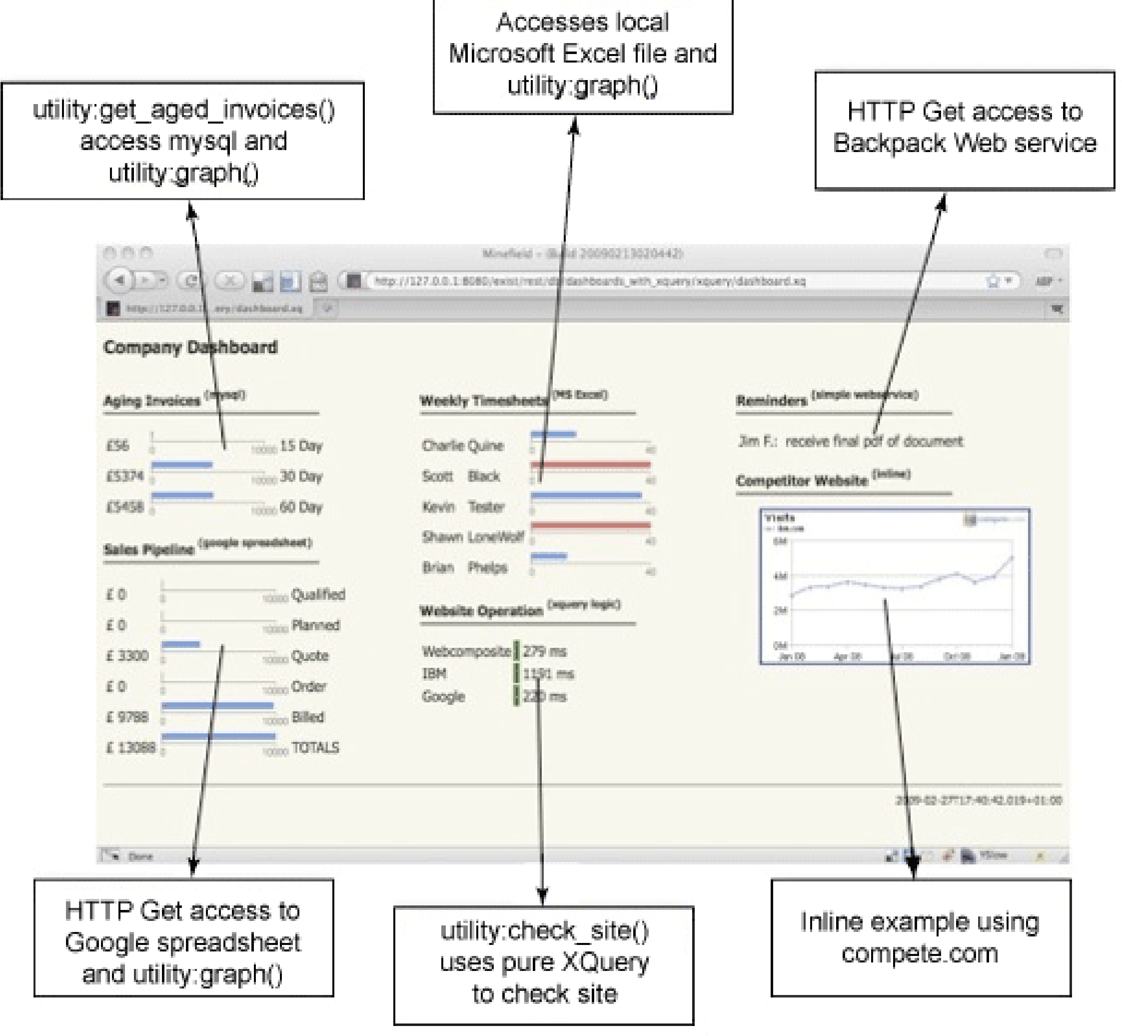
The dashboard.xq XQuery file generates output—an HTML document—based on eXist XML database-specific serialisation options that represent the dashboard:

© Copyright IBM Corporation 2009. All rights reserved.

```
declare option exist:serialize
        "method=xhtml media-type=text/html indent=yes omit-xml-declaration=no";
```

To create the example dashboard, take the aggregated XML document and use XQuery functions, defined in utility.xqm, to construct each specific area of the Web page, as in Figure 5.

Figure 5. Relate functions to the dashboard



Google Charting API
The bar chart at http://chart.apis.google.com/chart?cht=bhs&chco=4D89F9,CCCCCC&chs=150x50&chl=0|100&chd=t:50 is a real example of how you can use the Google Charting API to dynamically generate bar graphs. The above chart uses the following parameters:

© Copyright IBM Corporation 2009. All rights reserved.

- `cht=bhs` is the chart's type.
- `chco=4D89F9` is the chart's color.
- `chs=150x50` is the chart's size.
- `chl=0|100` is the chart's markers.
- `chd=50` is the data that the chart uses.

For more information about the Google Charting API, see Resources.

The KPIs are shown using a mixture of bar charts and HTML to show the status of any particular metric. In areas where you use bar charts, I opted to employ the Google Charting API. This clever API creates the charts used in the sales pipeline, aging invoices, and time sheets areas:

- **Aging Invoices.** Shows the status of unpaid invoices by using three separate bar charts representing standard terms (15-, 30-, and 60-day).
- **Sales Pipeline.** Uses six bar charts to indicate how many potential sales exist in each state of the sales workflow. The bar chart range is indicates desired targets for each phase of the pipeline.
- **Timesheets.** A bar chart is generated for each individual, with the bar turning red if an individual works more than 40 hours.

To help make it easier to create charts, I created an XQuery function, `utility:graph()`, which is defined in utility.xqm (see Listing 8).

Listing 8. The utility:graph() function

```
(: Wrap up Google Charting API :)
declare function utility:graph($type,$colors,$size,$markers,$data,$alt){
let $src
        :=concat('http://chart.apis.google.com/chart?chf=bg,s,F7F5E6&amp;chco=',$colors,
'&amp;chs=',$size,
'&amp;cht=',$type,
'&amp;chl=',$markers,
'&amp;chd=t:',$data
)
return <img alt="{$alt}"src="{$src}"/>
};
```

The `utility:graph()` function constructs the URL and returns the resultant `img` element. To learn more about Google Charting API options, see Resources.

The example dashboard needs data to do anything, so it brings in aggregated data generated by data.xq using the `doc()` function, saving the results into a `$data` variable:

© Copyright IBM Corporation 2009. All rights reserved.

```
let $data :=
doc('http://localhost:8080/exist/rest/db/dashboards_with_xquery/xquery/data.xq')
```

Placing the XML document in the `$data` variable means that the metrics are all measured at the same time. This is not necessary, but it does mean that you have a consistent "heartbeat" across all metrics—useful if you intend to regularly save results and view historical performance.

The listings that follow describe each display area and explain the main principles of what XQuery does. I highlighted the XQuery code in **bold** to help you pick it out from HTML markup.

Aging Invoices

The XQuery code in Listing 9 displays three bar charts that must be normalised so that they can be compared. As you use a range of 10,000, this means dividing values by the same amount, then multiplying by 100 to obtain a percentage of the bar used.

Listing 9. Bar chart code for Aging Invoices

```
<h3>Aging Invoices<sup>(mysql)</sup></h3>
<table>
<tr>
<td>£{$data/data/strategic/financial/invoices/age[@amt='15']}</td>
<td>
{ let $amt :=(xs:float($data/data/strategic/financial/invoices/age[@amt='15'])
        div 10000) * 100
return
        utility:graph("bhs","4D89F9",
                "150x30", "0|10000",$amt, '15 day')
}
</td>
<td>15 Day</td> </tr>
<tr>
<td>£{$data/data/strategic/financial/invoices/age[@amt='30']}</td>
        <td>{ let $amt :=
(xs:float($data/data/strategic/financial/invoices/age[@amt='30'])
div 10000) * 100
return
        utility:graph("bhs",
        "4D89F9",
        "150x30",
        "0|10000",
        $amt,
        '30 day')
} </td>
<td>30 Day</td> </tr>
<tr>
<td>£{$data/data/strategic/financial/invoices/age[@amt='60']}</td>
<td>{ let $amt := (xs:float($data/data/strategic/financial/invoices/age[@amt='60'])
        div 10000) * 100
return
utility:graph("bhs","4D89F9", "150x30",
        "0|10000", $amt, '60 day') } </td>
<td>60 Day</td> </tr>
</table>
```


© Copyright IBM Corporation 2009. All rights reserved.

Each table row needs to select the correct data, which is achieved by selecting the `amt` attribute, which represents the total amount. Remember that this data was originally generated from the MySQL database, so you need to ensure that the MySQL server is running and loaded with the example data set. You will also need to uncomment the data.xq code, as explained within the README installation instruction file.

Sales Pipeline

The sales pipeline data is obtained from a publicly viewable Google Docs spreadsheet. That means that you select the data effectively from an Atom feed. The only oddity when you present this data is that the Atom feed (from the Google spreadsheet) supplies data in rows containing delimited data, as in Listing 10.

Listing 10. The Sales Pipeline code

```
<h3>Sales Pipeline
<sup>(google spreadsheet)</sup></h3>
<table>
{for $item in $data/data/strategic/sales_pipeline/atom:feed/atom:entry/atom:content
return
        let $cols := tokenize($item,',')
        return
<tr>
<td>£{substring-after($cols[2],':')}</td>
<td>{
let $amt := (xs:float(substring-after($cols[2],':'))
       div 10000) * 100 return
utility:graph("bhs",
"4D89F9,C6D9FD",
"150x30",
"0|10000",
$amt,
substring-after($cols[1],':'))
}</td>
<td>{
substring-after($cols[1],':')
}</td>
</tr> }
</table>
```

The mixture of XML containing delimited data is somewhat odd, but XQuery gives you plenty of options to parse it. I use the `tokenize()` function to separate each column, then the `substring-after()` function gets to the data value.

Weekly Timesheets

Sometimes, the best way to capture time sheet data is just to open a local spreadsheet. Now that Excel supports an XML format, you just need to know its structure to start playing with it, as with the code in Listing 11.

Listing 11. Manipulating Excel spreadsheet data

© Copyright IBM Corporation 2009. All rights reserved.

```
<h3>Weekly Timesheets <sup>(MS Excel)</sup></h3>
<table> {
for $item in $data/data/operational/project/timesheet/
ss:Workbook/ss:Worksheet/ss:Table/ss:Row[not(position()=1)]
return
<tr> <td>{$item/ss:Cell[1]}</td>
<td>{$item/ss:Cell[2]}</td> <td>
{
let $hours := xs:decimal(($item/ss:Cell[3] div 40 ) * 100)
return
        if ($hours &lt; 100) then
                utility:graph("bhs",
                "4D89F9,C6D9FD",
                "150x30",
                "0|40",
                $hours,
                $item/ss:Cell[3])
                else
                utility:graph("bhs",
                "FF5858,C6D9FD",
                "150x30",
                "0|40",
                $hours,
                $item/ss:Cell[3]) }
</td> </tr>
}
</table>
```

XQuery lets you avoid using the first row, which contains column labels. The only other functionality you need to account for is displaying the bar chart in red if the employee goes over a 40-hour work week. The `if|then|else` statement in XQuery lets you choose a bar graph based on testing for the `$hours` variable being greater than 40.

Microsoft's foray into providing open standards-based XML source formats is a departure from the many years of using binary formats. These formats are indeed quicker, but being closed means that it is difficult to achieve the kind of integration shown here. The Excel XML format that represents a spreadsheet is just about rational, but I will leave any discussion about the merits of its specific structure for others to decide.

Website Operation

The data.xq-generated test elements tell you whether the Web site is online and how long it took for the Web server to respond. Instead of using anything fancy, I simply ran a test on the `status` attribute, as in Listing 12.

Listing 12. Testing the Web site

```
<h3>Website Operation <sup>(xquery logic)</sup></h3>
<table> {
for $item in $data//website_ok/website
return
        <tr>
```


© Copyright IBM Corporation 2009. All rights reserved.

```
        <td>{$item/@title/string()}</td>
<td>
{
if ( $item/test/@status = '200' ) then
        attribute style{'background-color:green'}
else
attribute style{'background-color:red'
}
}  </td>
<td>{

$item/test/@response/string()

} ms</td>
</tr>
}
</table>
```

It is imaginable to use the same kind of functions to go further than just testing for existence. For example, you might test whether a contact form was working.

Reminders

Taking data from another Web application (http://www.backpackit.com/), I just iterate through an XML Web service using an XQuery FLWR (for, let, where, and return) expression, as in Listing 13.

Listing 13. Iterating through the XML Web service

```
<h3>Reminders <sup>(simple webservice)</sup></h3> <table> {
for $item in $data/data/operational/project/backpack/reminders/reminder
        return
<tr>
<td>{
string($item/creator/@name)
}:</td>
<td> {
$item/content
}</td>
</tr>
}
</table>
```

Competitor Website

To inspire readers, I thought it would be fun to include external widgets. In this sense, a dashboard becomes more of a mash-up. The compete.com Web site does a good job of displaying the performance of several Web sites in comparison.

Conclusion

I built the example dashboard to demonstrate how to use XQuery first rather than to create something speedy or highly reusable. Here are some optimisation

© Copyright IBM Corporation 2009. All rights reserved.

suggestions:

- Generate dashboard.html by scheduling execution of dashboard.xq using `cron` (and `wget` or `curl`).
- Alternatively, use the scheduler of the eXist XML database to schedule creation of dashboard.html.
- Store results from each data source independently inside the XML database.
- Modularise each KPI—perhaps using XSLT to manage the presentation of data.

With three straightforward XQuery files, you implemented a Web dashboard that shows the power of XQuery and XML databases to aggregate and present data.

© Copyright IBM Corporation 2009. All rights reserved.

Downloads

Description	Name	Size	Download method
Source files for the example dashboard	src.zip	13KB	HTTP

Information about download methods

© Copyright IBM Corporation 2009. All rights reserved.

Resources

Learn

- Information Dashboard design: Read Stephen Few's excellent and concise overview of what one should do when designing dashboards.
- XQuery 1.0: An XML Query Language: See the core document outlining the XQuery language.
- XML Path Language (XPath) 2.0: Read the related XPath 2.0 specification.
- eXist XML Database: Explore this open source database management system that stores XML data according to the XML data model and features efficient, index-based XQuery processing.
- Atom Feed Specification: Read the Internet Engineering Task Force (IETF) specification of the Atom feed XML markup format.
- Google API Web services: See the main listing of all Google APIs.
- Google ClientLogin authentication: Learn how ClientLogin authentication is requested and used within most Google APIs.
- Google Calendar Web service: Read the API documentation for the Google Calendar service.
- Google Docs Web service: Read the API documentation for the Google Docs listing service.
- Google Spreadsheet Web service: Read the API documentation for the Google Spreadsheet service.
- General Google Atom feed: Read an overview of Google Web services that emit Atom feeds.
- Google Chart API: Read the API documentation for the Google Chart service.
- 37 Signals' Backpack API Web service: See 37 Signals' API for their popular Backpack service, which is intranet groupware.
- Data Ink: Explore the Data-Ink Ratio, a concept first conceived by Tufte in 1983. It revolves around the concept of leveraging data over all else within information design.
- Information Dashboard design: Read Stephen Few's good discussion on why you should reconsider using pie charts.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of

© Copyright IBM Corporation 2009. All rights reserved.

technical articles and tips, tutorials, standards, and IBM Redbooks.

- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

Get products and technologies

- MySQL Community Database Server: Download and try the example dashboard integrates with a MySQL database.
- MySQL JDBC connector: Download the JDBC connector, required by the eXist database.
- IBM product evaluation versions: Download or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- XML zone discussion forums: Participate in any of several XML-related discussions.
- developerWorks blogs: Check out these blogs and get involved in the developerWorks community.

About the author

James R. Fuller
Jim Fuller has been a professional developer for 15 years, working with several blue-chip software companies in both his native USA and the UK. He has co-written a few technology-related books and regularly speaks and writes articles focusing on XML technologies. He is a founding committee member for XML Prague and was in the gang responsible for EXSLT. He spends his free time playing with XML databases and XQuery. Jim is technical director for a few companies (FlameDigital, Webcomposite s.r.o.) and can be reached at jim.fuller@webcomposite.com.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business

© Copyright IBM Corporation 2009. All rights reserved.

Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of IBM trademarks.
Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft and Excel are trademarks of Microsoft Corporation in the United States, other countries, or both.
Other company, product, or service names may be trademarks or service marks of others.

© Copyright IBM Corporation 2009. All rights reserved.