Microsoft

Microsoft® Visual Studio® LightSwitch™ 2011

What is LightSwitch?

Visual Studio LightSwitch Technical White Paper

Author: Andrew Brust, Blue Badge Insights

Published: August, 2011

Applies to: Visual Studio LightSwitch 2011

Summary: This is the first in a series of white papers about Microsoft® Visual Studio® LightSwitch™ 2011, Microsoft's new streamlined development environment for designing data-centric business applications. We'll provide an overview of the product that includes analysis of the market need it meets, examination of the way it meets that need relative to comparable products in the software industry, concrete examples of how it works, and discussion of why it's so important.

Copyright

White Papers in this Series

1. What is LightSwitch?
2. Quickly Build Business Apps
3. Get More from Your Data
4. Wow Your End Users
5. Make Your Apps Do More with Less Work

Contents

Introduction

This is the first in a series of white papers about Microsoft® Visual Studio® LightSwitch™ 2011, Microsoft's new streamlined development environment for designing data-centric business applications. In this first paper, we'll provide an overview of the product that includes:

- analysis of the market need it meets
- examination of the way it meets that need relative to comparable products in the software industry
- concrete examples of how it works
- discussion of why it's so important

LightSwitch is a very exciting project, and it's easy enough to use that you might wish to dive right in. However, some background and context on the product may help you use it more effectively. So let's take a look at the need LightSwitch solves, and how it does so differently from other business application development tools.

What is LightSwitch and Why is it Here?

When it comes to custom business software, virtually *all* applications are data-centric applications. Successive generations of software development tools, including dBase, Visual FoxPro® and Visual Basic® have sought to make data-centric application development easier. Within the nearly 10-year history of .NET this is also true. Considering the range of technology from Windows Forms and ASP.NET to WPF and Silverlight, each .NET UI technology has offered its own data binding technology. Visual Studio has offered important tooling including the various wizards, the Dataset designer, the Data Sources window, and the Entity Data Model designer to provide assistance in the rapid development of data-centric applications.

Complexity has become unavoidable because of these designers, wizards, data access technologies, and data binding conventions. Also, general purpose software development platforms are not, and perhaps *should not* be, data-centric in approach. Ironically, the multitude of data tools and technologies at our disposal makes it hard to develop data-centric applications in a quick and easy manner.

To address the need for streamlined development of data-centric business applications, Microsoft has introduced Visual Studio LightSwitch. LightSwitch applications use the modern .NET stack of technologies, and wrap them in an abstraction layer optimized for data management and maintenance. LightSwitch makes it possible to build data-centric applications quickly, through visual means. With LightSwitch, you won't be writing the same code repeatedly to provide data access functionality or the user interface needed around it. You can write no code, a little code, or a significant amount – and you can rest assured that it will be high-value code, rather than mere "plumbing. "

Going Deeper

In broad strokes, LightSwitch is a new edition of Visual Studio, which includes special Visual C# and Visual Basic .NET project types, and unique designers. LightSwitch allows developers to design

databases, screens around the tables in those databases, and the logic and rules that bind all of it together.

Screen designs can be inferred and generated from table schemas, or can be built using a hybrid approach where the screens are generated and then lightly or heavily customized. Data rules can be specified declaratively in property windows, or they can be expressed imperatively through code. Coding can be invoked on a sliding scale of complexity. VB .NET and C# can be used as if they were merely expression languages, or complete methods and classes can be developed and integrated into the application.

LightSwitch business applications are multi-tiered, featuring a client application and a combination of LINQ, WCF RIA Services and the Entity Framework to implement the application services tier. Unlike other tools that provide their own application environments, LightSwitch produces applications based on standard components from the .NET stack. In effect, it provides an abstraction layer over application development best practices, yet it eliminates laborious and repetitive plumbing code required to construct a properly architected database, data access tier, and user interface (UI) framework.

LightSwitch has an extensions model that is simultaneously attractive to advanced developers and to Independent Software Vendors (ISVs). Advanced developers can extend the product's basic functionality using their .NET development skills. ISVs can offer their extensions to a market of business application developers eager to integrate advanced functionality into their LightSwitch applications, but who may lack the skills or time to implement such functionality on their own.

A Fresh Approach

LightSwitch works on the familiar principle of helping developers specify a data model and the build screens around it. Within that widely accepted paradigm, LightSwitch works in an innovative manner, allowing so much detail to be expressed within the data model itself that a fully functional UI can be interpreted and generated automatically. But unlike various other tools and frameworks which do this, LightSwitch also allows for customizations by the developer, over a custom code event model and a wide range of extensibility points.

LightSwitch's screen design is hierarchical and declarative, rather than physical and imperative. Instead of working in a WYSIWYG form designer wherein controls are physically placed, LightSwitch developers specify what data should be displayed and edited, how the controls that manage that data are to be configured, and then how the general layout scheme should be applied within and between sections of the screen. LightSwitch ably takes care of the rest, lifting the developer out of the minutiae of screen design, but without the typical accompanying sacrifice of precision control and customization.

Customize What You Want

LightSwitch offers a spectrum of customization to the developer on the coding side as well. Entire LightSwitch applications can be created that:

- have no code at all
- have code in their data models but not in their screens
- have substantial code within all parts of the application

Code can be inserted to handle specific events, but standalone methods and classes can also be created and called from event handling code.

Screens can have their own parameters (values that drive data queries) and can override the basic queries in the data model with extra selection criteria and sorting specifications. Security permissions, roles and users can be configured without any programming. Simple, single lines of code can be used to enable or disable whole screens, specific options within screens or user interface behaviors, based on a user's identity and corresponding role membership.

For some, the mere ability to write code within the context of the application is not enough, and access to the full .NET Framework is required. LightSwitch keeps these options open to those developers who wish to explore them. Through a robust extensions model, LightSwitch applications can use custom controls, business types, data sources, screen templates, themes and shells authored by internal IT and/or commercial component vendors.

We'll discuss all of these LightSwitch capabilities in this 5-part white paper series but for now, let's take a quick look at why the product is so important, and explore, at a high level, how it works.

LightSwitch's Importance

While the operating system and database technology underlying most business applications has changed quite a bit since the first generation of PCs became prevalent in the workplace, the requirements of these applications remain stunningly similar. To a great extent, most business applications are oriented around structured data (customers and orders, portfolios, or securities and trades for example) and must accommodate the creation, updating, inter-relating, querying and reporting of that data. These business application scenarios must typically serve a few related, but distinct audiences:

- end-users who need to view and edit the data pertinent to them and their business units
- administrative users who need to establish and maintain certain master data (customer information, transaction types, trading partners, and so forth)
- Super-users who must be granted access to most or all of the data and functionality in the system

This basic business application scenario need has stayed with us no matter how many operating systems, Graphical User Interfaces, development platforms, desktop, web, or service-oriented

architectures have come along. As software platforms mature, they service these scenarios more capably. Then as the technology churns, the scenarios are accommodated less directly, leaving productivity to suffer.

The phenomenon of consistent business application scenario needs accompanied by the cycle of technology churn and the resulting ebb and flow of platforms accommodating the need productively constitutes a real sore spot in software development. The phenomenon has created a challenging environment for companies' personnel and consultants to implement business applications in an economically feasible manner. This has turned business application development work and its fees into a tax (of sorts) for business users everywhere, and has driven work away from the customer to markets with lower development costs. That in turn, has made the premise of developing business applications more complex and risky for the business units that need them. This is not a good situation for the industry, or its customers.

Modeling the Application

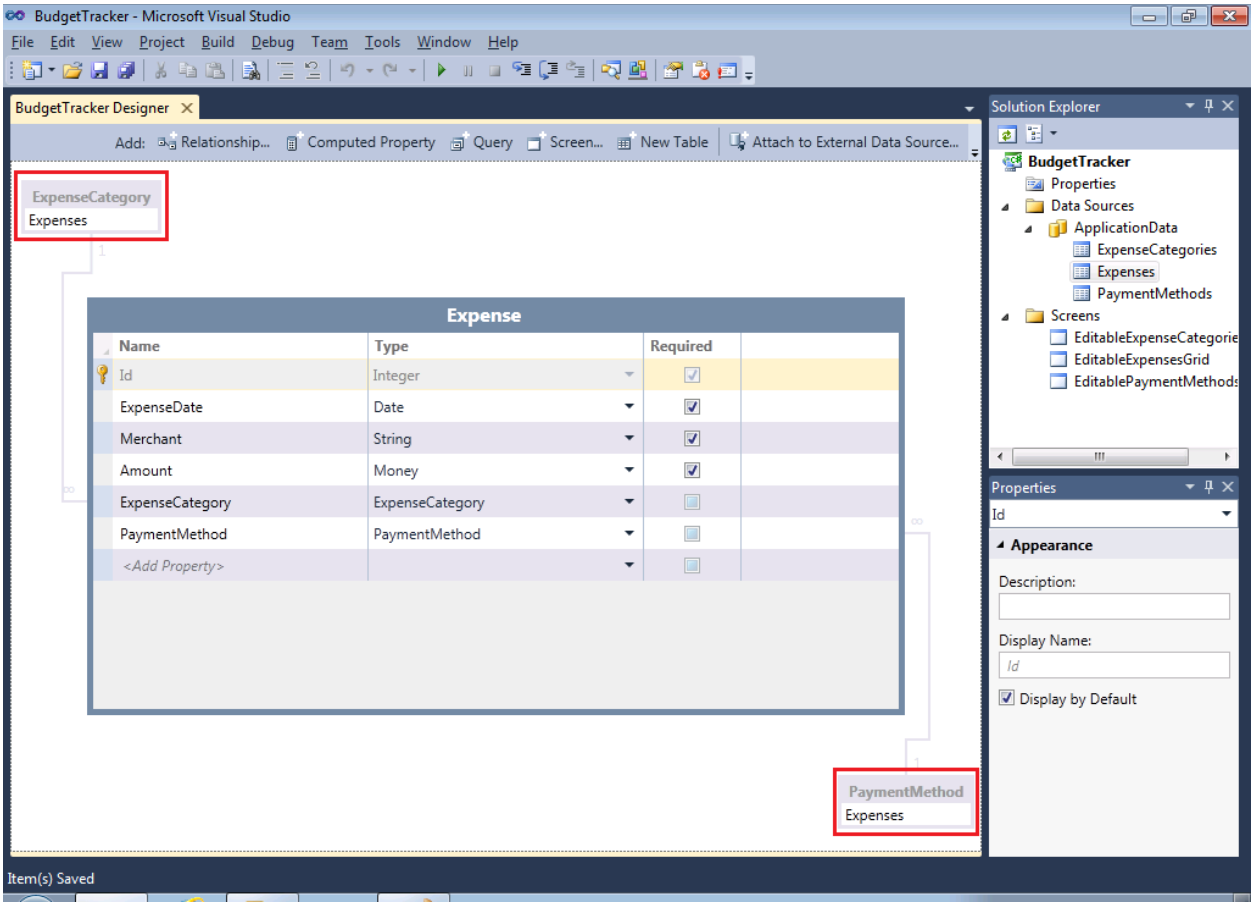
Two solutions are needed to contend with this situation. First, we need a good business application development tool for the current popular platform. Second, we need that tool to use an approach that transcends the platform for which it was designed, making it portable to new platforms or platform versions. LightSwitch tackles the current platform challenge admirably, but what about platform independence?

As developers use LightSwitch to design an application, the product codifies the design in a technology-independent manner by capturing the model in a special ApplicationDefinition file that stores details of the application's tables and screens. Using a technology-agnostic model as part of the LightSwitch approach lays encouraging groundwork for adapting to new technologies.

A Show-and-Tell LightSwitch Overview

We've had a lot of contextual discussion and analysis at this point, so let's now take a quick look at how easy LightSwitch makes the basics of data-centric application development. We'll demonstrate a scenario where we create a budgeting application. Our database and application would need to track expenses and budgets as well as master data like expense categories and payment methods. We might start by designing our *Expense* data entity, which would be used as the basis for both a database table and a data maintenance screen. **Figure 1** shows what the properties of that entity might look like in LightSwitch's Data Designer.

Figure 1: The LightSwitch Data Designer



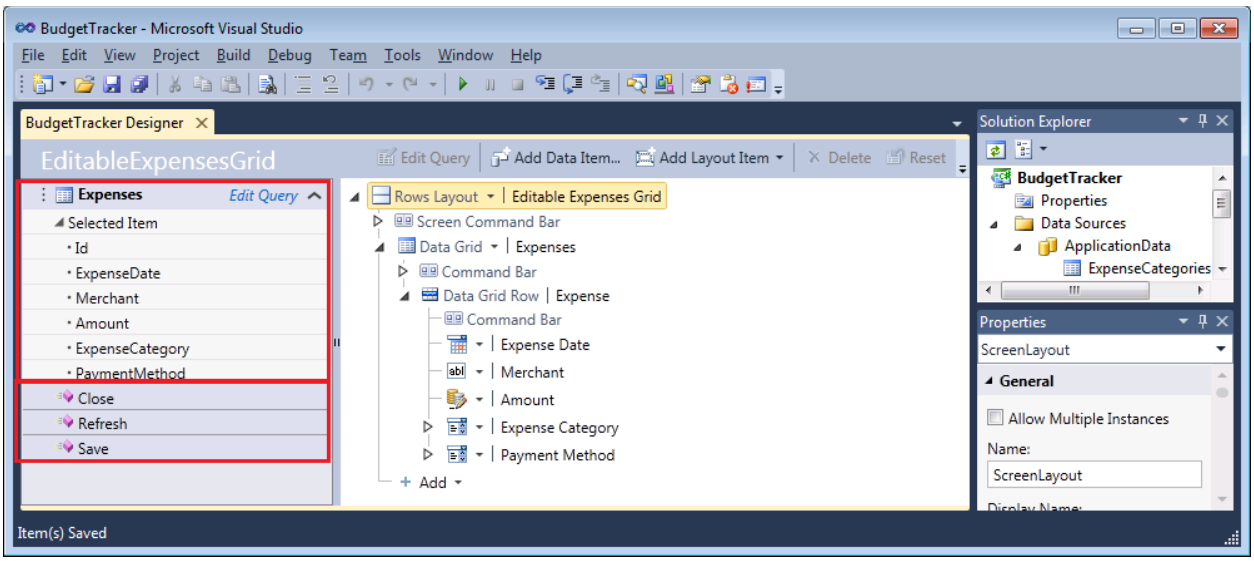
Notice the highlighted thumbnail depictions of the related *ExpenseCategory* and *PaymentMethod* entities in the upper-left- and lower-right-hand corners, respectively. (Don't worry about those for now; we'll explain them in more detail in the next paper in this series.) The *Expense* entity has properties pointing to each of these related entities and an automatically-generated **ID** field. The *ExpenseDate*, *Merchant*, and *Amount* properties are created simply by entering their names in the left-most column of the property grid. For *ExpenseDate* and *Amount*, data types of *Date* and *Money* are selected. The *Merchant* property's type was left at the default setting of *String*.

Business Types and Generated UI

If the LightSwitch developer automatically generated a screen from this entity and then opened the screen in design view, he'd see something similar to what's shown in **Figure 2**.

Before we discuss how this schematic, hierarchical view of the screen works and is used, let's talk a bit more about the *Amount* field. As we mentioned, its data type is *Money*, which is a business-oriented categorization. If LightSwitch were less business application development-oriented, we might have had to set the data type to something like *Double* or *Float*. The reason we didn't have to do so is because LightSwitch supports the notion of Business Types. Other built-in Business Types include *Email* and *PhoneNumber*, and partners can create new Business Types, packaged as LightSwitch extensions.

Figure 2: The LightSwitch Screen Designer



Business Types allow LightSwitch developers to specify, in an implicit, declarative fashion, important data entry and/or business rules around their data. For example, typing a field as *Email* or *PhoneNumber* instead of *String* immediately implies specific input masks and does so more naturally than a more technological approach such as specifying a regular expression.

It's not that regular expressions are bad, or that they are too difficult for business application developers, but they do require a change of context and mindset from the application domain to the programming domain. That shift can disrupt the business application developer's train of thought, separate him from the business problem and impede productivity and time-to-market for the application.

User interface behavior and requirements are also implied by Business Type designation. For example, in the screen in **Figure 2**, the control type associated with the *Amount* field is a *Money Editor*, which is a specialized control for displaying, entering and editing currency data. The developer doesn't have to change to that control type from a more naïve default (such as a generic textbox). LightSwitch observes the business type selected and acts on it. This further removes friction from the business application development process.

Iterative Screen Design

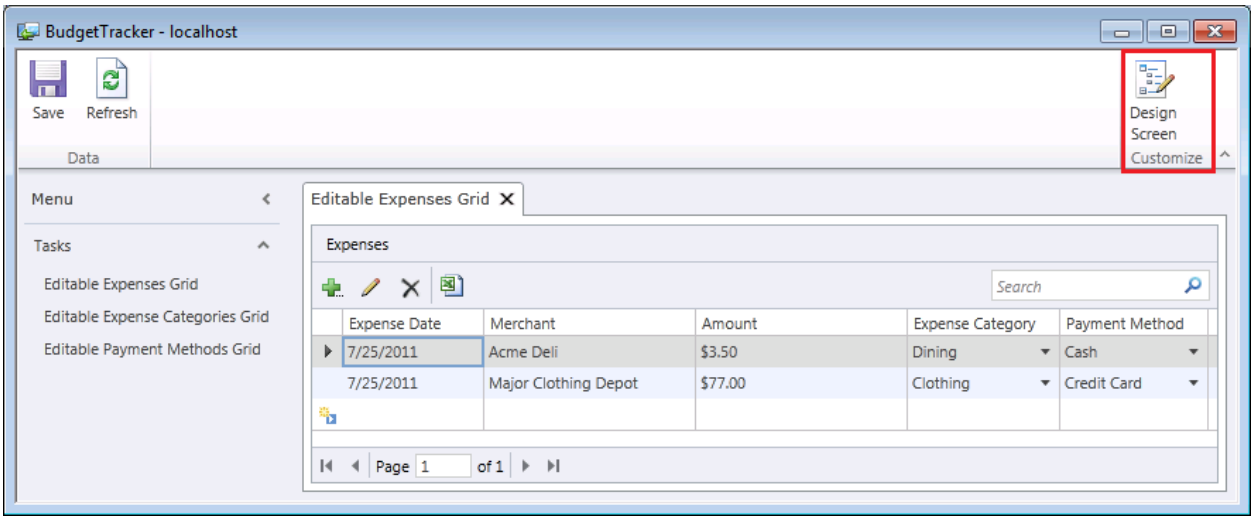
Let's review how the other fields are mapped to UI elements:

- date time editor is used for *ExpenseDate*
- a text box is used for *Merchant*
- auto-complete boxes are used for *PaymentMethod* and *ExpenseCategory*.

In **Figure 2**, notice the highlighted *ExpenseItems* query on the left that represents a database query against the corresponding table and the standard *Close*, *Refresh,* and *Save* methods beneath it.

Figure 3 shows what the screen looks like when fully rendered in the application.

Figure 3: A running screen

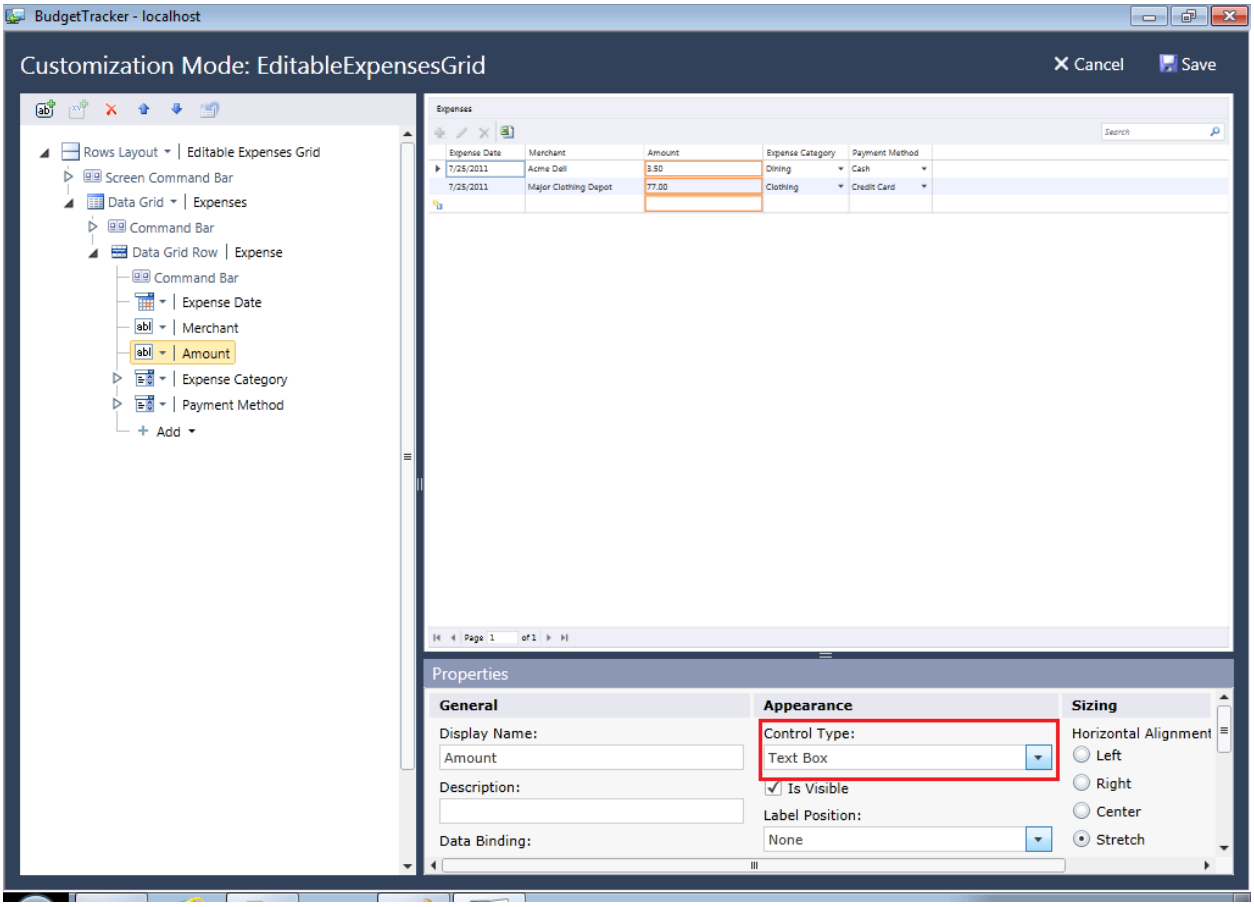


Design-While-Executing

The separation between configuring a screen at design time and executing it to confirm proper rendering needn't be a hard dichotomy. See the highlighted **Design Screen** button on the far-right of the application's ribbon in **Figure3**? It appears only when the application is executed in Debug mode, and it allows the screen to be edited interactively while the application is running. Interactive editing is shown in **Figure 4**.

In this live customization mode, the control tree appears on the left, the **Properties** window on the bottom, and the remainder of the screen is occupied by a live, running copy of the screen, which will update to reflect changes made in either of the other two panes. For demonstration's sake, **Figure 4** shows the control type for the *Amount* field changed from *Money Editor* to *Text Box*. In the preview pane, the *Amount* data (highlighted by LightSwitch in orange outline) is no longer currency-formatted, appearing without a dollar sign. Clicking the **Save** button on the upper-right-hand corner would exit the live design mode and make the control type change permanent. Clicking the **Cancel** button to its left would discard the change and revert to the original screen design with a *Money Editor* control being used to maintain the *Amount* field data.

Figure 4: Screen design during application execution



Business Rules and Validation

Not only is this intellectually straightforward, it's just plain fast. The number of clicks and typed characters used to generate this functionality is minimal. However, even though LightSwitch achieves a large amount of code reduction, some coding, albeit extremely streamlined, will be necessary.

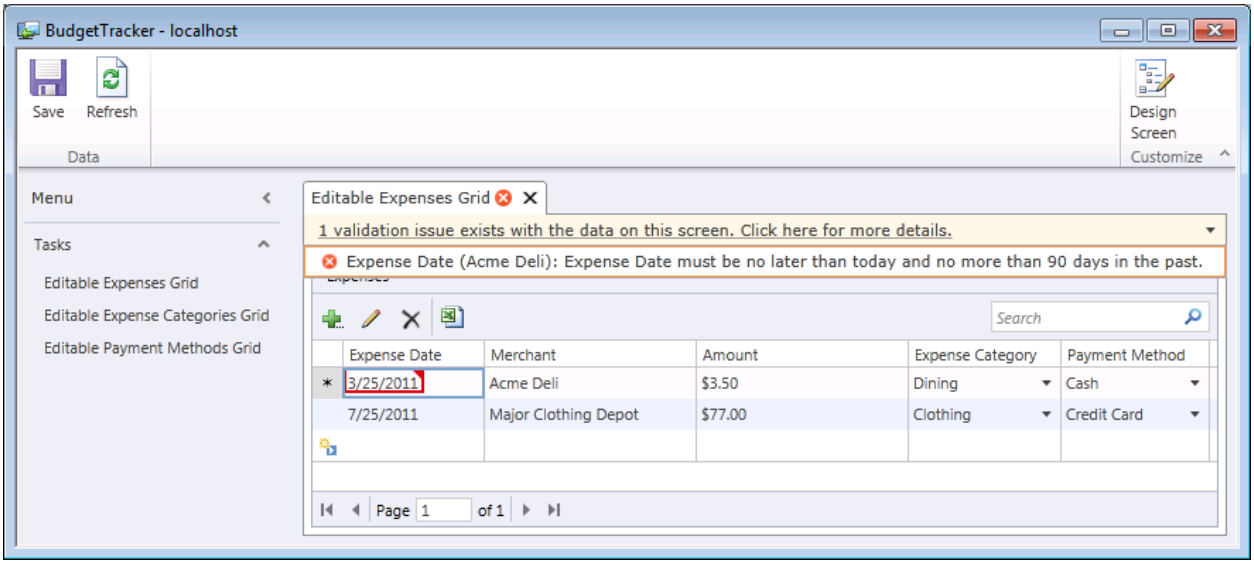
For example, imagine a business rule in our application that says an expense entry's *ExpenseDate* cannot be in the future, nor can it be more than 90 days old. We'd need to make certain that rule were enforced in our data layer and implemented in our UI in such a way that a warning message were displayed in any screen that allowed expense entries to be added or edited. LightSwitch makes this surprisingly simple to do. The *ExpenseDate* field in the *Expense* data entity has a *Validate* event that we can code to. **Listing 1** shows the code necessary to implement the rule.

Listing 1: A validation rule for the Expense Date field

```
partial void ExpenseDate_Validate(EntityValidationResultsBuilder results)
{
    if (!((ExpenseDate <= DateTime.Now) && (ExpenseDate >= DateTime.Now.AddDays(-90))))
    {
        results.AddPropertyError("Expense Date must be no later than today and no more
than 90 days in the past.");
    }
}
```

Not only is the code terse and straightforward to write, but it covers both the data layer and UI validation requirements that we outlined above. **Figure 5** demonstrates this by depicting the error message displayed (automatically) when the rule is broken.

Figure 5: Error message display, caused by our validation rule



It is impressive and convenient that a couple lines of code are all that's required to get this working. Ironically though, LightSwitch's real value is almost obfuscated by this kind of functionality. Because it handles certain things so easily, LightSwitch can appear to developers as yet another tool that forces tradeoffs that invalidate its value proposition. Despite appearances though, that is not the case at all.

Flexibility

LightSwitch doesn't dumb things down. Instead, it speeds them up. As such, it helps business users build their own applications while letting highly-trained developers build these same types of applications very quickly. LightSwitch also lets IT organizations augment the standard capabilities of the product with their own set of extensions that assure compliance with corporate standards, while still allowing LightSwitch to function as users expect:

- You can custom-code the UI if you want.
- You can add code in the screen to augment what's in the data entity definition.

- You can validate using simple expressions, or the full power of the Visual Basic .NET or C# programming languages.
- You can create custom controls and embed sophisticated behaviors there.

A good productivity tool is one that eliminates common, time-consuming coding tasks, but that gives you power to do specialized development when that is required. When a tool has a layered set of advanced capabilities that you can iteratively explore and master, it becomes truly useful in an enterprise setting, and not just in the business unit shadows.

Extensibility

We mentioned LightSwitch's extensions model, but let's address it more directly as it is a very important part of what makes LightSwitch special. Many business application development tools provide productivity, but do so at the cost of functionality. Although many tools accommodate the basic "CRUD" (create, read, update and delete) data maintenance use case well, they don't go far beyond that. That may be understandable, but it also imposes a real blocker on users of those tools.

What Microsoft has done with LightSwitch, conversely, is to achieve reasonably ambitious goals in provision of built-in functionality and then engineer a complete and well-documented framework for other parties to extend that functionality. This lets domain experts implement functionality specific to their industry or specialty, and it lets others in that community avail themselves of that functionality for their own applications.

Any company in the developer component space is familiar with this model. Because LightSwitch developers will be so focused on productivity they are likely to be more appreciative consumers of LightSwitch extensions than enterprise developers are of developer component libraries. Ironically, the .NET component market has its roots in the Visual Basic (VB) custom controls space. The VB control market had the same productivity-centric underpinnings as those of LightSwitch and it gave birth to the developer component industry. VB custom controls had a file extension of .VBX, which stood for Visual Basic eXtensions; the fact that LightSwitch's add-ons share the "extensions" designation is a good omen for the value and potential of the business opportunity.

LightSwitch takes the component industry back to the core fundamental value of empowerment – where the extension takes the developer from nowhere to done. This brings a compelling value proposition to the end user, and thus to the component vendor as well.

Deployment

We've seen so far that LightSwitch adds a lot of value, and does so while allowing graduated levels of customization with good old-fashioned programming. It is useful, nuanced, and will solve many issues.

From the screenshots we have seen so far, you might suspect that LightSwitch produces desktop applications exclusively. However, LightSwitch offers a much more versatile set of deployment options and application types than the Windows Office-like UI portrayed in the screenshots.

LightSwitch works in sync with .NET and the entire Microsoft stack so that it can produce desktop applications easily. And because these desktop applications' entire installation packages can be pulled down and executed implicitly by navigating to a URL, they deploy with minimal friction.

In addition to running as desktop applications that execute with significant permissions and desktop integration, LightSwitch can also produce applications that run entirely in the browser, and in the browser's security sandbox. The options don't end there. Beyond applications which run completely on the desktop, LightSwitch can also produce multi-tier, distributed applications whose application services execute on a central server.

Perhaps LightSwitch applications' most exciting deployment scenario is that of running in the cloud, using the Microsoft® Windows® Azure™ Platform as a Service (PaaS) cloud environment and its SQL Azure Database as a Service environment. The excitement stems not just from the power of running as a cloud-deployed application, but also because LightSwitch allows developers to take advantage of the cloud with fewer barriers and greater simplicity than other Microsoft application development environments. As long as you have your Azure subscription ID and storage account information handy and have met a few other setup prerequisites, deploying your LightSwitch is as simple as running a wizard. This effectively creates a feedback mechanism where LightSwitch amplifies the value of the cloud and Azure amplifies the value of LightSwitch.

LightSwitch's range of deployment options means you have choice and you have portability. The application that you run on a few desktops today is one you could re-deploy to the cloud next month. And since LightSwitch applications are built on the most modern components of the Microsoft stack (including the ADO.NET Entity Framework, WCF RIA Services, and SQL Server™/SQL Azure), the scalability is in the technology to make such migrations realistic. LightSwitch deployment doesn't just give you options, it gives you assurance that each option will *work* and that LightSwitch applications can scale up and down to conform to each deployment target.

Conclusion

We've seen in this overview that LightSwitch has brought to the .NET stack time-tested approaches to meeting the timeless requirements of data-centric business applications. But LightSwitch goes further, by lifting the customizability ceiling typically imposed by business application development products, and allowing a complete spectrum of custom coding. LightSwitch has also modernized the business application development paradigm to accommodate layered/distributed application architectures, cloud computing, and Rich Internet Application (RIA) technology.

In the remaining papers in this series, we'll go through each of these points in more depth. In the next paper, we'll take a more complete look at developing a LightSwitch application. Subsequent papers will examine advanced data techniques; customizing your applications' look and feel, and mastering their deployment; and taking advantage of LightSwitch extensions to give your applications industrial-strength power.

Even if you were to stop reading here, you would have enough information – and hopefully more than enough motivation – to start building LightSwitch applications for your own business application needs. You can get productive with LightSwitch very quickly and there's no penalty for experimentation early in the learning process. In fact, readers may wish to build a simple LightSwitch application right now and then continue with the next paper. Either way, we're now ready to delve further into LightSwitch's features and capabilities.

For more information:

Visual Studio LightSwitch Website: http://www.microsoft.com/lightswitch

Visual Studio LightSwitch Dev Center: http://msdn.microsoft.com/lightswitch