**DAL – Sharding of RDBMS**

<http://blogs.msdn.com/b/windowsazure/archive/2013/09/05/dal-sharding-of-rdbms.aspx>

From <http://social.technet.microsoft.com/wiki/contents/articles/17987.cloud-service-fundamentals.aspx>

**Editor's Note:** This post was written by [Shaun Tinline-Jones](http://social.msdn.microsoft.com/profile/shauntj/), and [Chris Clayton](http://social.technet.microsoft.com/profile/chris%20clayton%20(azurecat)/?ws=usercard-inline).  Both Senior AzureCAT Program Managers in the Cloud and Enterprise Group.

The "[Cloud Service Fundamentals](http://code.msdn.microsoft.com/Cloud-Service-Fundamentals-4ca72649)" application, referred to as "CSFundamentals," demonstrates how to build database-backed Azure services.  This includes a description of the scenario, implementation architecture and reusable components for logging, configuration and data access.  The code base is intended to be a tangible exploration of best practices for delivering scalable, available services on Azure based on production deployments by the Windows Azure Customer Advisory Team.

With the majority of companies these days looking to move their cloud initiatives forward, the business drivers for specific solutions vary greatly, from lowering costs to significantly increasing agility and scale.  When solutions are trying to attain “cloud scale”, the strategy of “vertical scalability”, increasing capacity by upgrading the hardware, is replaced with “horizontal scalability”, increasing the number of machines that share a specific task.  A great example of this tradeoff is the creation of web farms where many servers serve the same web site content versus having a single monolithic machine try to handle the load.

Most people that start down this path plan for horizontal scalability on the compute nodes but bypass the more complex and potentially more critical state tiers, such as the relational database management system (RDBMS) and caches.  These services are often IO-intensive and bound by a single instance.  One technique to implement horizontal scalability in the state tier is known as *sharding*.  Sharding is when you logically separate your RDBMS data into multiple databases, typically with the same schema. For example, an employee table could be split across three distinct databases where each database stores a different department’s employees.

The benefits of sharding assists in far more than just capacity related scenarios.  For the purposes of this post we will focus on sharding an RDBMS that is implemented in Azure SQL Database platform, and primarily serves OLTP scenarios.  Some example scenarios that could benefit from a sharded database structure include:

* Throttling thresholds or throughput limits are hit too often.
* Size of the database becomes unwieldy (index rebuilds, backups).
* A single unavailable database affects all users (as opposed to a single shard).
* A database that has difficulty scaling up and down gracefully in respond to demand.
* Certain business models, such as multi-tenant or Software as a Service offerings.

When using a multi-tenant database as a service solution, such as Windows Azure SQL Database, there are typically Quality of Service (QOS) controls put in place that throttle clients under various conditions. Throttling typically occurs when resource pressure climbs.  Sharding is a key strategy to help reduce resource pressure by taking the load that would typically affect a single server and spreading it across as multiple servers that each contain a shard. For example, assuming an even distribution, creating five shards reduces the load to approximately twenty percent on each database.

As with anything that grants greater power, there are sacrifices that must be made.  Sharding increases the complexity of several key areas, requiring more planning. These include:

* Identity columns should be globally unique across all shards in case future business needs necessitate the reduction in shard count.  If the identity is not unique across all shards, merging two shards can result in conflicts.
* Referential integrity cannot reference or enforce relationships to rows in other shards as they are independent databases.
* Queries that cross shards should be avoided if possible, because they require querying each shard and merging the results.  The need to do “fan out” queries across the shards is not only costly from a performance point of view but increases the complexity of the sharding framework that must support it.  If cross-shard queries are necessary, the typical strategy is to query each shard asynchronously.  However, there are times where a synchronous approach offers more control of the resultset size.

In most cases, sharding is a Data Access Layer (DAL) concept, abstracting most of the intricacies from the higher-level application logic.

How you define a “tenant” is one of the most crucial decisions that you can make when building a sharded architecture.  A tenant is the largest unique categorization of data that is guaranteed to be on the same shard.  Queries that are constrained to a single tenant typically perform better as they do not need to execute fan-out operations during normal operational states.  Some of the factors that influence the decision of the appropriate tenant definition include:

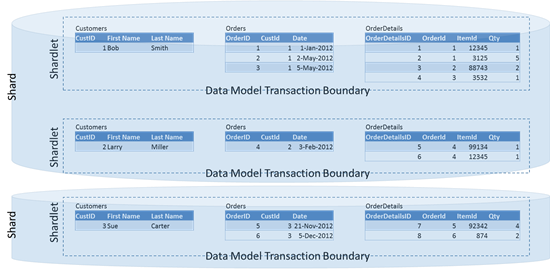
* The level of awareness of the higher-level application code of the identifier.
* The ability of most core business transactions to be performed at this level.
* The ability to avoid throttling in general day-to-day operations at the tenant granularity.

In an effort to surface these concepts and considerations at a high level, the Windows Azure Customer Advisory Team has built out a basic sharding Data Access Layer (DAL) in the Cloud Services Fundamentals (CSF) package (<http://code.msdn.microsoft.com/Cloud-Service-Fundamentals-4ca72649>).

In CSF the tenant is defined as an individual user.  Some of the factors that led to selecting this tenant were:

* Most of the core business requirements do not require queries across multiple users.
* An unavailable shard only impacts a defined set of users, leaving all others to continue their typical usage of the system.  The quantity of users that are on a single shard can be controlled to a number that is tolerable to a business.

The tenant is defined and implemented such that cross-database transactions are not required.  In Figure 1, we refer to this set of data as a shardlet, a Data Model Transaction Boundary.

[](http://blogs.msdn.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-01-13-25/2783.data-model-transaction-boundary.png)

*Figure 1 - Data Model Transaction Boundary*

When the user connects to the database for the first time in the session they have a series of simple queries that they can execute to understand if any features are unavailable to them due to offline shards.

In an effort to simplify the sharding techniques being demonstrated in CSF, we decided to create a set of shards that have enough storage to fulfill capacity needs for the foreseeable future. By selecting this size, it eliminated the need to demonstrate increasing and reducing the number of shards, which would also include actions like tenant movement.  An integer is generated by hashing the tenant’s name and this id is used to lookup a matching range in a “shard map”.  CSF uses a range-based mechanism, where a range of these numbers are assigned to a specific shard (captured in the “shard map”).

In the event that shards need to be added or removed from the shard set, it requires that tenants become unavailable until they are transitioned into their new shards. Due to this significant limitation, it is expected that the shard set will be significantly over provisioned when it is first created to reduce or eliminate the need to do complex shard management.

This solution requires that the Data Access Layer (DAL) is aware of the tenant id to determine the placement of the tenants in the shard set.  If a query is executed that includes a shard that is unavailable the entire query will fail.  In the event that the DAL does not include the tenant id all shards will have to be queried, increasing the chance of failure and reducing performance.

There is currently some preliminary work underway to offer further code samples demonstrating more advanced sharding techniques.  The sample will offer improvements in the following areas:

* Reactive and proactive shard management.
* Global uniqueness and identity management.
* Migration of tenants between shards within the set.
* Expansion and contraction of the shard set.
* Improvements in queries that are not tenant aware.

In conclusion, the Cloud Services Fundamentals code sample is a great way to start to explore the basic concepts of sharding, which is an important technique for creating “cloud scale” applications.

<http://blogs.msdn.com/b/windowsazure/archive/2013/06/21/building-blocks-of-great-cloud-applications.aspx>

**Building Blocks of Great Cloud Applications**

Rate This  
http://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-off.png

[http://i1.social.s-msft.com/profile/u/avatar.jpg?displayname=craig_kitterman&size=largeCraig\_Kitterman](http://blogs.msdn.com/502003/ProfileUrlRedirect.ashx)

Craig\_Kitterman

8,325 Points 2 1 0

Recent Achievements

New Blog Commentator Blogger II New Blogger

[View Profile](http://social.msdn.microsoft.com/profile/craig_kitterman/?ws=usercard-hover)

21 Jun 2013 4:00 PM

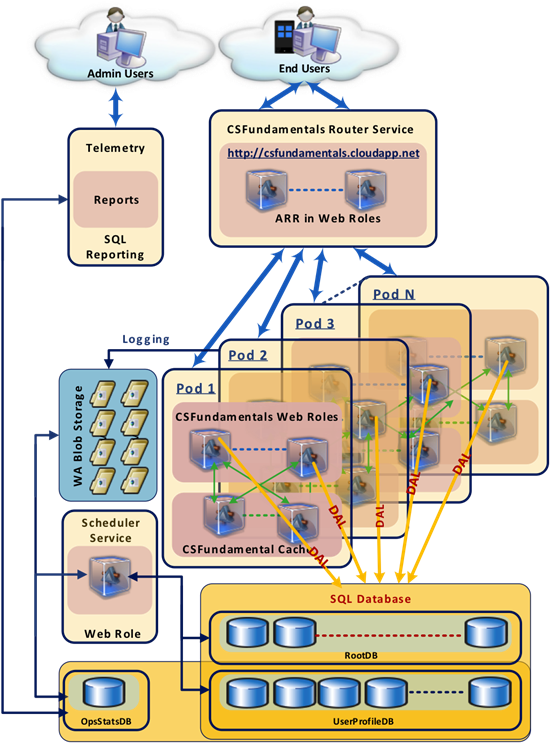
* Comments [1](http://blogs.msdn.com/b/windowsazure/archive/2013/06/21/building-blocks-of-great-cloud-applications.aspx#comments)

**Editors note**: this post was written by Michael Thomassy, Principal Program Manager, Windows Azure Customer Advisory Team

Following the blog on [Designing Great Cloud Applications](http://blogs.msdn.com/b/windowsazure/archive/2013/05/07/writing-a-cloud-application-is-easy-writing-a-good-cloud-application-is-harder.aspx), the Azure CAT team is planning to give more detail and technical explanation to the components found in the code project [Cloud Service Fundamentals in Windows Azure](http://code.msdn.microsoft.com/Cloud-Service-Fundamentals-4ca72649) posted on MSDN Code Gallery. This starts the series of blogs and tech articles to describe the use of these fundamental build blocks which we’ll refer to as components.  Over the course of the next several months, we will be publishing a series of blogs every other Thursday with detailed technical notes that walk through the individual components of Cloud Service Fundamentals.

Over the years we’ve worked with Windows Azure customers, within and outside of Microsoft, with many deep discussions about what is needed in their Windows Azure services.  We’ve seen firsthand how answering some basic questions about implementing cloud services can grow quickly in complexity.  For example, rather than giving just a piece of sharding code, we need a data access layer.  Followed by resiliency of the data access layer that require developing retry logic as well as solid guidance for logging errors at scale.  Not to mention building an ops store you can query for reports and generate alerts.  You can see how the discussion progresses with each component as they depend and build on one another. These discussions and implementations resulted in the code project [Cloud Service Fundamentals in Windows Azure](http://code.msdn.microsoft.com/Cloud-Service-Fundamentals-4ca72649) that ties together a number of basic components into a working cloud application.

This code project was a challenge for the CAT team as we were focused on enabling complex, database backed services on Windows Azure for some of our largest customers.  It’s based on work that we did with actual Windows Azure customers to solve specific problems.  These problems often required best practices beyond the basic samples when we combined many of the requirements of large scale cloud services including elastic scale, partitionable workloads, availability, business continuity, large number of distributed users, and high volume, low latency requests.  You can see the architecture for the Cloud Service Fundamentals code project below.



Our technical series will detail the components in the code project, including:

1. Telemetry – The basics for instrumentation and logging of application services through asynchronous mechanisms at scale implemented in a data pipeline.  Effectively leveraging the telemetry data is critical in troubleshooting a service and determining the health of a service.  The code project implements a scheduler using a background worker role to collect telemetry data periodically from the application, perf counters, IIS logs, event logs and the sharded SQL Database DMVs. The data is written to a custom ops store database in Windows Azure SQL Database.  The data collected by the scheduler can be viewed by reports hosted in SQL Reporting.
2. Data Access Layer – The layer accessing the multiple databases in Windows Azure SQL Database efficiently and reliably.  The code project has data access wrappers for both single-database and sharded solutions, and demonstrates techniques such as parallel fan-out queries across shards.
3. Caching – By using Azure Caching, user data may be stored and retrieved more efficiently from a dedicated cache when combined with the Data Access Layer.
4. Configuration – Configuration files are key to help make managing your application seamless whether configuration parameters are in web.config or the service config – this should be transparent to the application.
5. Application Request Routing – Implementation of cookie based routing & affinitization of users to multiple hosted services and sharded databases leveraging the ARR (Application Request Routing) technology in IIS to enable scale-out at the application service level with sharded databases.
6. Deployment – Methods to deploy your custom configuration with multiple hosted services, variable number of instances and configuring the number of shards through the use of Windows Azure Cmdlets in PowerShell.

We’ll post technical blogs and publish the details on the [TechNet Wiki](http://social.technet.microsoft.com/wiki/contents/articles/17987.cloud-service-fundamentals.aspx).  Looking forward to your comments and contributions.

## Cloud Service Fundamentals in Windows Azure – application at:

<http://code.msdn.microsoft.com/Cloud-Service-Fundamentals-4ca72649>

**Cloud Service Fundamentals in Windows Azure**

The "Cloud Service Fundamentals" application, referred to as "CSFundamentals," demonstrates how to build database-backed Azure services.  This includes a description of the scenario, implementation architecture and reusable components for logging, configuration and data access.  The code base is intended to be a tangible exploration of best practices for delivering scalable, available services on Azure based on production deployments by the Windows Azure Customer Advisory Team.

**Creators:**Michael Thomassy, Rafael Fernandez, Mark Simms, Christian Martinez, Ewan Fairweather, James Podgorski, Paolo Salvatori, Rama Ramani, Shaun Tinline-Jones, Silvano Coriani, Valery Mizonov, Gus Apostol

Note: The complete version of this "Description" page is contained in the Microsoft Word document titled "Exploring CSFundamentals in WA.docx" as part of the "C# Download" linked above. The purpose of the rest of this page is to highlight each section in the document and describe the topics covered in those sections.

Details of indiviudal components are also documented on the [Cloud Service Fundamentals wiki collection](http://social.technet.microsoft.com/wiki/contents/articles/17987.cloud-service-fundamentals.aspx).

**Scenario Description**

Many modern cloud services integrate social experiences, based on a foundation of registering and managing per-user information.  The intent of the CSFundamentals code reference is to demonstrate several best practices and experiences for implementing modern data-driven applications on Azure in context.  The code base implements the following functionality:

* [MVC4](http://www.asp.net/mvc/mvc4) web application providing scalable user registration and login against a sharded database with distributed cache integration
* Multi-cloud service deployment, using Application Request Routing (ARR) to transparently leverage multiple cloud services for additional scale and reliability
* Queryable operational data store, with scheduled tasks for collecting and integrating application and server performance/health metrics.

**Prerequisites**

The reference implementation was developed using Visual Studio 2012 Ultimate; however, there shouldn't be issues with other editions.  The following items need to be installed:

* Windows Azure SDK v2.0. Recommended to install through the [Web Platform Installer](http://www.microsoft.com/web/downloads/platform.aspx).
* [Microsoft® SQL Server® 2012 Express](http://www.microsoft.com/en-us/download/details.aspx?id=29062) (for the sqlcmd.exe utility).
* For SQL Reporting, install [Microsoft SQL Server Data Tools - Business Intelligence for Visual Studio 2012](http://www.microsoft.com/en-us/download/details.aspx?id=36843).

**Visual Studio Project Layout**

The CSFundamentals applications incorporates several projects in one solution and three cloud service definitions for use by developers.  This section provides an overview of these artifacts, the directory structure and third party libraries.

**Visual Studio Solution**

The code base is contained in one Visual Studio solution, **CloudServiceFundamentals.sln**.  Open this Visual Studio solution to work with the application code, reusable libraries, and publish the application services to Azure.

**Visual Studio Projects**

The overall solution consists of several projects; with reusable and scenario specific code separated.

* *Common:*  Reusable components.
  + Microsoft.AzureCat.Patterns.Common.  
    Reusable baseline aspects of the codebase, including configuration, logging, serialization and helper base classes.
  + Microsoft.AzureCat.Patterns.Web.  
    An implementation of the MVC4 providers and configuration utilities designed to work against a scale-out relational data service
* *Data*:  Libraries for working with data stores.
  + Microsoft.AzureCat.Patterns.CacheClient.AzureCaching.  
    Provides a wrapper around the Azure Caching client SDK, including the use of a pluggable binary serializer (in this case [protobuf-net](http://code.google.com/p/protobuf-net/))
  + Microsoft.AzureCat.Patterns.Data.Common.  
    Helper classes for working with data storage, relational and non-relational, including retry and telemetry logic for working with Windows Azure SQL Database.
  + Microsoft.AzureCat.Patterns.Data.SqlAzureDalSharded.  
    Helper classes for implementing a scale-out relational database solution, including partitioning,  connection affinity and metadata management.
* *Database:*  Database schema definitions.
  + Microsoft.AzureCat.Patterns.Data.SqlAzure.OpsStatsDB.  
    The schema for the operations database, used to centralize telemetry information from compute, storage and SQL.
  + Microsoft.AzureCat.Patterns.Data.SqlAzure.RootDB.  
    The database project for the user metadata and shard management databases.
  + Microsoft.AzureCat.Patterns.Data.SqlAzure.UserProfileDB.  
    The database project for the user profile and information databases.
* *Logic:*
  + CSFundamentals.Logic. Application specific logic and data schema.
* *Roles:*  Cloud service web and worker roles.
  + CSFundamentalsCaching.WorkerRole.  
    Hosting the Azure Caching web roles providing caching for the CSFundamentalsService.WebRole.
  + CSFundamentalsService.WebRole.  
    The public web site exposed by the MVC front-end web application as an Azure Web Role.
  + RouterService.WebRole.  
    The worker role hosting the cookie based application request routing (ARR).
  + SchedulerService.WorkerRole.  
    The worker role hosting the [Quartz.NET](http://quartznet.sourceforge.net/) scheduler
* *Router:*
  + Microsoft.AzureCat.Router.LoadBalanceThenCustomCookieProvider.  
    Load Balance requests based on the cookie contents routing traffic to multiple  
    cloud services (multiple CSFundamentalsServices)
* *SQLReporting:* Telemetry Reports
  + OpsStatsDBReporting.
  + Parameterized reports (.rdl) using SQL Reporting from the OpsStatsDB database. The reports show query stats, database size & growth, wait times and errors across all sharded databases.
* *Tasks:*
  + Microsoft.AzureCat.Patterns.Tasks.SqlAzureMgmt.  
    Collection of scheduled tasks using the Quartz scheduler to collect and load  
    telemetry data from different sources into a Windows Azure SQL Database for  
    exploration and charting.
* *CSFundamentalsService:* Azure web service hosting the front-end CSFundamentalsService.WebRole  
  and CSFundamentalsCaching.WorkerRole.
* *RouterService:* Azure web service hosting the RouterService.WebRole.
* *SchedulerService:* Azure web service hosting the SchedulerService.WorkerRole.

**Shared Libraries**

The reference implementation makes use of multiple shared libraries, managed through NuGet packages. The Microsoft libraries distributed through NuGet used by this solution include:

* Reactive Extensions (library for supporting asynchronous and push-based operations)
* Enterprise Library’s Transient Fault Handling Application Block

The non-Microsoft libraries used by this solution:

* Dapper (micro-ORM, used in the database access code)
* Quartz.NET (scheduler library, used for managing recurring tasks)
* Common.Logging (used by Quartz.NET)
* NLog (application logging library)
* Protobuf-net (binary serializer)

**Reusable Components**

There are several reusable aspects bundled with the CSFundamentals application.  This document provided describes the components, explains their design, configuration and potential applicability to your own projects. These components include:

* Configuration to integrate managing multiple configuration files
* Logging using NLog to capture application diagnostic information
* Data Access to sharded databases using Windows Azure SQL Database
* Caching using the Azure Caching in Windows Azure SDK 1.8
* Scheduling using the Quartz.NET scheduler to manage Telemetry collection
* Application request routing to set affinitization to a hosted service
* Reports on Telemetry data stored in the OpsStats database using Windows Azure SQL Reporting of the application Logging information collected by the Scheduler

**Social Scenario Implementation**

The CSFundamentals application implements a custom membership provider against a scaled-out relational store based on Windows Azure SQL Database.  For simplicity, the CSFundamentals code sample does not focus on functional capabilities but does provide four very simple social networking features in the MVC Web UI for purposes of demonstrating the key Azure implementation concepts.

1. User Registration
2. User Login
3. Add Comments
4. Retrieving Comments

 The focus of the implementation is on these core concepts:

* Database Design: a scaled-out relational data store based describing the design, partitioning model and scale-out aware data access layer implemented by the CSFundamentals application.
* Shard Map: the application's data access layer supports sets of sharded databases called the "Root" databases and the "UserProfile" databases with the shard map contained in the file “RootUsersShardMap.xml".
* Implemented Features: a description of the implementation for registration, login, add comment and get comment using the scale-out database design, shard map and code snippets from the data access layer.
* Telemetry Monitor: details of the task-driven telemetry collection and transformation service and how the SchedulerService aggregates data from multiple sources (Windows Azure Diagnostics, the Root and UserProfile sharded Windows Azure SQL Database, etc.) into a consolidated destination database.

**Configuration and Deployment**

In the Configuration and Deployment section, a description is provided for each step to configure, build and deploy the code sample to Windows Azure.  These steps include:

1. Get an Azure subscription and create services for the CSFundamentals services: Hosted Services in Windows Azure, Azure SQL Database server and databases, Windows Azure Storage account
2. Download the CloudServiceFundamentals solution and C# source code from the link above
3. Create Certificates - management certificates, certificate for Remote Desktop usage and creating SSL certificate (the SSL certificate will be added to the RouterService project)
4. Change the placeholder “csfundamentals\_” settings in the appropriate project files to ***your*** Windows Azure service values including modifying the QuartzJobs.xml and the RootUsersShardMap.xml
5. Package and deploy the 3 hosted services in the CloudServiceFundamentals Visual Studio solution:
   1. CSFundamentalsService
   2. RouterService
   3. SchedulerService

For these steps, refer to the word document provided. This details the configuration files and project files that require modification and customization for your subscription's services.

**Building the CloudServiceFundamentals Solution**

To build the solution, the CloudServiceFundamentals.sln file relies on enabling NuGet packages to be automatically downloaded by Visual Studio. These packages include Microsoft and 3rd party libraries previously listed. Typically, the Release BUILD configuration is selected which is recommended to use the provide Powershell scripts for configuration and deployment.

**Configuration and Deployment from Visual Studio**

The configuration and deployment sections describe how to modify the files to enable the solution and services to be deployed either directly by Visual Studio or through the use of Powershell scripts using the Windows Azure cmdlets. When using Visual Studio for deployment, there 9 files that require manual changes before the projects can be built and the service packages created.  The Powershell scripts help simplify modifying these files programmatically.

**Configuration and Deployment from Powershell**

The Powershell scripts provided for configuring and deploying the CloudServiceFundamentals solution to Windows Azure are located under the “PowerShellDeployment” directory. The scripts are organized and numbered in the order they should be executed along with a description of the steps.  There are manual steps involved too that must be followed in the order described which include uploading the management certificate to the Azure portal, modifying the RoutingService project (selecting your SSL certificate) and creating the deployment packages for each service from Visual Studio.

**More Resources**

For additional reading and background, please refer to the following resources:

* [Building Big: Lessons learned from Windows Azure customers - Part I (Scale)](http://channel9.msdn.com/Events/Build/2012/3-029)
* [Building Big: Lessons learned from Windows Azure customers - Part II (Availability)](http://channel9.msdn.com/Events/Build/2012/3-030)
* [Best Practices for the Design of Large-Scale Services on Windows Azure Cloud Services](http://msdn.microsoft.com/en-us/library/windowsazure/jj717232.aspx)
* [Failsafe: Guidance for Resilient Cloud Architectures](http://msdn.microsoft.com/en-us/library/windowsazure/jj853352.aspx)

**Version History**

* v1.10: April 12, 2013
  + Renamed the code gallery project from "ContosoSocial" to "Cloud Service Fundamentals" with short names CSFundamentals and CloudServiceFundamentals.
  + Added SQL Reporting (.rdl) reports project.  These reports are provided for use against the OpsStatsDB Telemetry database.
* v1.20: July 30, 2013
  + Updated projects to support Windows Azure SDK 2.0
  + Updated retry logic support using Enterprise Library’s Transient Fault Handling Application Block